FOUFNE

PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW
MERA-PIAP
Al. Jerozolimskie 202 02-222 Warszawa Telefon 23-70-81

~~Ośrodek Automatyzacji Procesów Produkcji~~
Pracownia Oprogramowania Mikroprocesorowych Urządzeń Automatyki i Robotów

074

A

Główny wykonawca

Wykonawcy mgr inż. A. Aderek, mgr inż. K. Czarnomski, mgr M. Dembek mgr inż. M. Porada

Konsultant

Poufne, egz nr 1

Nr zlecenia UR 01.03.01 L

Asembler skrośny dla mikroprocesora 8086 na minikomputer SM - 4.
Etap 6: Wykonanie konsolidatora w wersji docelowej.

SKROŚNY KONSOLIDATOR RELOKUJĄCY QRL-86/SM /listing/

Zleceniodawca problem węzłowy 06.6

Pracę rozpoczęto dnia 86.01.01 **zakończono dnia** 86.06.30

Kierownik pracowni
w.z. A. Ustaszewski
mgr M. Dembek

Z-ca Dyrektora ds. Automatyki
dr inż. T. Gałązka

Kierownik ośrodka
mgr inż. A. Aderek

Praca zawiera:

stron 1
rysunków
fotografii
tabel
tablic
załączników 1

Nr rejestr. 5617

Rozdzielnik - ilość egz:

Egz. 1 BOINTE
Egz. 2 OAP - A
Egz. 3 OAP - A
Egz. 4
Egz. 5
Egz. 6

POUFNE

1

Analiza deskryptorowa OPROGRAMOWANIE + ASEMBLER + MIKROPROCESOR

Analiza dokumentacyjna

Praca zawiera listing programu konsolidatora skrośnego w wersji docelowej. Opis programu i instrukcja użytkowania znajdują się w sprawozdaniu z etapu 5. p.n. "Opracowanie dokumentacji powykonawczej".

Tytuły poprzednich sprawozdań

681.32:621.377-181.48 Mikroprocesor

681.322-181.4 Minikomputer

UKD

PIAP-252/82-6000

Opis ogólny

Niniejsza praca została wykonana w ramach zlecenia p.n. "Asembler skrośny dla mikroprosesora 8086 na minikomputer SM-4".

Konsolidator jest programem współpracującym z asemblerem. Przetwarza moduł wynikowy zawierający skompilowany przez asembler program na moduł ładowalny z kodem programu gotowym do ładowania i uruchomienia na mikroprocesorze 8086. Jeżeli moduł ładowalny składa się z wielu modułów wynikowych, konsolidator konsoliduje je ze sobą. Wszystkim adresom relokowalnym, nie określonym przez asembler w fazie kompilacji, zostają przyporządkowane wartości absolutne, przez co dokonuje się lokacja programu przygotowywanego na mikroprocesor 8086.

Program będący przedmiotem sprawozdania odpowiada funkcjonalnie połączeniu dwóch procesorów należących do pakietu MCS-86 Software Development Package produkcji Intel:

- QRL86,
- OH86.

Program QRL86 jest odpowiednikiem połączonych ze sobą konsolidatora LINK86 i lokatera LOC86. Program OH86 przekształca kod absolutny programu zgodnie z heksadecymalnym formatem taśmy papierowej.

Dla produktu PIAP przyjęto nazwę: skrośny konsolidator relokujący QRL-86/SM.

Dokładny opis programu wraz z instrukcją obsługi znajduje się w sprawozdaniu p.n. "Asembler skrośny dla mikroprocesora 8086 na minikomputer SM-4. Etap 5.: Opracowanie dokumentacji powykonawczej".

```
LABEL
  1;
CONST
  dlw=80;          (* DLUGOSC LINII DYREKTYWY      *)
  dls=32;          (* DLUGOSC TABLICY NA NAZWE     *)
  at=343B;         (* MASKA ALIGN TYPE             *)
  atat=0;          (* ALIGN TYPE = AT              *)
  ati=1;           (* ALIGN TYPE = INPAGE          *)
  atbt=40B;        (* ALIGN TYPE = BYTE            *)
  atwo=100B;       (* ALIGN TYPE = WORD            *)
  atpr=140B;       (* ALIGN TYPE = PARA            *)
  atpg=200B;       (* ALIGN TYPE = PAGE            *)
  atinot=376B;     (* MASKA ZEROWANIA INPAGE       *)
  ct=36B;          (* MASKA COMBINE TYPE           *)
  ctno=0;          (* COMBINE TYPE = NO            *)
  ctpub=10B;       (* COMBINE TYPE = PUBLIC        *)
  ctmem=4B;        (* COMBINE TYPE = MEMORY        *)
  ctcom=30B;       (* COMBINE TYPE = COMMON        *)
  ctsta=24B;       (* COMBINE TYPE = STACK         *)
  maxm=30;         (* MAX ILOSC MODULOW WE         *)
  ms=30;           (* MAX ILOSC NAZW SEGMENTOW I KLAS W MODULE WE  *)
  maxe=20;         (* MAX ILOSC EXTERNALI W MODULE WE      *)
  maxs=20;         (* MAX ILOSC SEGMENTOW W MODULE WE      *)
  maxob=32;        (* MAX ILOSC BAJTOW W REKORDZIE DANYCH  *)
TYPE
  wiersz=ARRAY[0..dlw] OF char;
  nazwa=ARRAY[1..dls] OF char;
  order=(boot, res, smcs, sm, cs, no, mem);
  lonum=0..65535;
  liczba=ARRAY[1..5] OF integer;
  tablbin=ARRAY[1..20] OF integer;
  inpufil= RECORD     (* ZAWIERA NAZWE PLIKU WEJSCIOWEGO      *)
             spec: nazwa;
             next: ^inpufil
           END;
  adres = RECORD
            par: lonum;
            off:lonum
          END;
  pozycja=RECORD  (*  ZAWIERA DANE SEGMENTU (KLASY) PRZETWARZA-
                      NEGO PRZEZ LOCATER *)
            nus: integer;    (* NUMER SEGMENTU W MODULE WE   *)
            nug: integer;    (* NUMER MODULU WE*)
            seg: nazwa;      (* NAZWA SEGMENTU *)
            kl: nazwa;       (* NAZWA KLASY    *)
            typs: integer;   (* OPIS COMBINE-TYPE I ALIGN-TYPE SEGMENTU
                                POSTACI TAK JA NA WY ASSEMBLERA (REKORD 98) *)
            zord: order;     (* ZLECENIE ORDER ODNOSZACE SIE DO POZYCJI *)
            nord: integer;   (* NUMER POZYCJI WG ORDER       *)
            aabs: order;     (* ZLECENIE ADDRESSES ODNOSZACE SIE DO POZ *)
            pusty:  boolean;(* CZY DO POZYCJI UTWORZONEJ NA PODST ORDER
                               LUB ADDRESSES WPISANO SEGMENT Z PLIKU WE  *)
            ap: adres;       (* ADRES POCZATKOWY *)
            dl: lonum;       (* DLUGOSC POZYCJI (BAJTOW)     *)
            ssc: char;       (* RODZAJ ZMIANY SEGSIZE:
                                @ = BRAK ZMIAN
                                = EXACT
                                + = ZWIEKSZ O
                                - = ZMNIEJSZ O          *)
            ssv: lonum;      (* WARTOSC ZMIANY SEGSIZE       *)
            ptr: ^pozycja     (* WSKAZNIK NASTEPNEJ POZYCJI. POZYCJE
                                 TWORZA TABLICE TYPU WSKAZNIKOWEGO  *)
            /* DLA POZYCJI res ADRES POCZATKOWY OBSZARU RES = ap
             ADRES KONCOWY              = dl, ssv (PARAGRAF, OFFSET)     */
          END;
  global=RECORD              (* ZAWIERA DANE NAZW PUBLIC I EXTERNAL  *)
           nrg: integer;   (* NR MODULU WE W KTORYM JEST DANY PUBLIC  *)
```

```
            nrs: integer;   (* NR SEGMENTU- ATRYBUTU SEGMENT PUBLICA   *)
            extern: boolean;(* ETYKIETA EXTERN      *)
            public: boolean;(* ETYKIETA PUBLIC      *)
            etykieta: nazwa;(* ETYKIETA PUBLICA - EXTERNALA         *)
            offset: lonum;  (* WARTOSC PUBLICA      *)
            gptr: ^global
          END;
  rass=RECORD              (* OPIS DYREKTYWY ASSUME        *)
         nrm: integer;    (* NUMER MODULU Z SEGMENTEM ASSUMED     *)
         nrms: integer;   (* NUMER MODULU ASSUMED W SEGMENCIE nrm         *)
         ex: boolean;     (* ASSUME WG EXTERNALA          *)
         nex: integer     (* NUMER EXTERNALA - GDY ex     *)
       END;
  posptr= ^pozycja;
  globalptr = ^global;
VAR
  listing          (* ZLECENIE LISTINGU W DYREKTYWIE *),
  startctrl        (* OBECNOSC CONTROL START *),
  startboot        (* OBECNOSC CONTROL BOOTSTRAP *),
  memory           (* OBECNOSC SEGMENTU MEMORY *),
  listopen         (* OTWARTY PLIK LISTINGOWY *),
  wyopen           (* OTWARTY PLIK WYJSCIOWY *)    :boolean;

  plik             (* NAZWA BIEZACEGO PLIKU WE *),
  plwy             (* NAZWA PLIKU WYJSCIOWEGO *),
  pllst            (* NAZWA PLIKU LISTINGOWEGO *),
  startpub (* NAZWA ZMIENNEJ 'PUBLIC' ZADEKLAROWANEJ JAKO ADRES CSTARTOWY *),
  error1, error2, error3  (* NAZWY PRZEKAZYWANE DO errorcom *)    : nazwa;

  tlin, tlip: ARRAY[0..20] OF ^wiersz;
  (* TABLICE WSKAZNIKOW DO KOLEJNYCH WIERSZY DYREKTYWY:
   TLIN - WIERSZE NIENORMOWANE, TLIP - WIERSZE NORMOWANE   *)

  plikptr          (* POINTER DO BIEZACEGO PLIKU WE *),
  obiekty          (* TABLICA Z NAZWAMI PLIKOW WE *)  : ^inpufil;
  (* POSTAC TABLICY OBIEKTY: /NAZWA PLIKU WE, WSK NASTEPNEGO REKORDU/,
   /.../,....,/NAZWA PLIKU WE, NIL/     *)

  s0clist          (* LISTA POZYCJI Z LINII WYWOLANIA I Z WE *),
  ltd              (* LISTA SEGMENTOW I OBSZAROW RES ULOKOWANYCH *) : posptr;
  (* POSTAC TABLICY Z SEGMENTAMI:
   /DANE SEGMENTU(KLASY), WSK NASTEPNEJ POZYCJI/, /.../,.....,
   /NIEZDEFINOWANE, NIL/     *)

  startadr         (* DANE ADRESU STARTOWEGO *),
  lowad, highad    (* DANE DLA ERRORCOM *)       : adres;

  glob: ARRAY['?'..'_'] OF globalptr;
  (* TABLICA ROZPROSZONA Z NAZWAMI GLOBALI *)

  externals: ARRAY[1..maxm,1..maxe] OF globalptr;
  (* TABLICA WSKAZNIKOW DO EXTERNALI W FUNKCJI
   OD NRMODWE, NR EXTERNALA W MODULE (REKORDZIE 8C)        *)

  segments: ARRAY[1..maxm,0..maxs] OF posptr;
  (* TABLICA WSKAZNIKOW DO POZYCJI SEGMETOW
   W FUNKCJI OD NRMODWE, NR KOLEJNY SEGMENTU W MODULE WE   *)

  assume: ARRAY[0..3] OF rass;
  (* TABLICA ZAWIERAJACA NUMERY SEGMENTOW, RTORYCH ADRESY SA
   ASSUMED W REJESTRACH SEGMENTOWYCH, ODPOWIEDNIO: CS, DS, ES, SS  *)

  nrmodwe (* NUMER AKTUALNIE PRZETWARZANEGO MODULU WE *),
  len      (* WSKAZNIK POPRAWNOSCI WYKONANIA RESET  *),
  nrerror (* NUMER BLEDU PRZEKAZYWANY DO errorcom  *),
  chksum  (* SUMA KONTROLNA WYPROWADZANEGO REKORDU HEXADECYMALNEGO *): integer;

  we       (* PLIK ZAWIERAJACY AKTUALNIE PRZETWARZANY MODUL WE *),
```

```
wy      (* PLIK ZAWIERAJACY WYPROWADZANE REKORDY HEXADECYMALNE *),
lst     (* PLIK LISTINGOWY *),
ti      (* PLIK MONITORA OPERATORA *): text;

dat1    (* INFORMACJE LICZBOWE DLA PROCEDURY ERRORCOM *),
usba    (* AKTUALNY UPPER SEGMENT BASE ADDRESS *): lonum;
```

```
PROCEDURE skok1(nrerror: integer); EXTERNAL;
(**E+*)

        (*************************************)
        (*           OUTBYTE                 *)
        (*************************************)

(*
 + PROCEDURA WYPROWADZ BAJT BINARNY W POSTACI 2 ZNAKOW HEXADECYMALNYCH NA
 + PLIK wy I MODYFIKUJE SUME KONTROLNA chksum.
 + WE: int = WARTOSC BINARNA DO WYDRUKOWANIA
 *)
PROCEDURE outbyte(int: integer);

        (*************************************)
        (*           OUTPAR                  *)
        (*************************************)

  (*
   * PROCEDURA WYPROWADZA ZNAK HEXADECYMALNY Z BITEM PARZYSTOSCI NA PLIK wy.
   * WE: hex = WARTOSC HEXADECYMALNA ZNAKU DO WYPROWADZENIA
   *)
  PROCEDURE outpar(hex: integer);
  VAR
    c, d: char;
   BEGIN
     IF hex<10
      THEN c:=chr(hex+ord('0'))
      ELSE c:=chr(hex-10+ord('A'));
     CASE c OF
       '0':    d:=chr(60B);
       '1':    d:=chr(261B);
       '2':    d:=chr(262B);
       '3':    d:=chr(63B);
       '4':    d:=chr(264B);
       '5':    d:=chr(65B);
       '6':    d:=chr(66B);
       '7':    d:=chr(267B);
       '8':    d:=chr(270B);
       '9':    d:=chr(71B);
       'A':    d:=chr(101B);
       'B':    d:=chr(102B);
       'C':    d:=chr(303B);
       'D':    d:=chr(104B);
       'E':    d:=chr(305B);
       'F':    d:=chr(306B);
      END;
     write(wy, d)
   END;
  (*      OUTPAR  *)

 BEGIN
   int:=int AND 377B;
   outpar(int DIV 16);
   outpar(int MOD 16);
   chksum:=(chksum+int) AND 377B
 END;
(*      OUTBYTE         *)

        (*************************************)
        (*           SHIFTR                  *)
        (*************************************)

(*
 * FUNKCJA SHIFTR PRZESUWA 16-BITOWA LICZBE X O C BITOW W PRAWO.
 * Z LEWEJ STRONY SA WPROWADZANE ZERA.
 *)
```

```
FUNCTION shiftr(x, c: integer): integer;
 BEGIN
   (*$C
    TST     C (%6)          ; SPRAWDZAMY O ILE BITOW TRZEBA
                            ; PRZESUNAC
    BEQ     1$              ; JESLI LICZBA BITOW DO PRZESUNIECIA
                            ; JEST ROWNA ZERU TO KONIEC
    2$:     CLC             ; ZERUJEMY CARRY ABY Z LEWEJ STRONY
                            ; WSZEDL BIT ZEROWY
    ROR     X (%6)          ; PRZESUWAMY LICZBE X O JEDEN BIT W PRAWO
                            ; Z LEWEJ STRONY WCHODZI BIT ZEROWY
    DEC     C (%6)          ; ZMNIEJSZAMY O JEDEN LICZBE BITOW DO
                            ; PRZESUNIECIA
    BNE     2$              ; JESLI LICZBA BITOW DO PRZESUNIECIA
                            ; JEST ROZNA OD ZERA TO PRZESUWAJ DALEJ
    1$:     NOP             ; KONIEC PRZESUWANIA
    *)
   shiftr:= x
 END;
(* SHIFTR *)

        (************************************************)
        (*              S H I F T L                     *)
        (************************************************)

FUNCTION shiftl (x, c: integer): integer;

  (*      PROGRAM PRZESUWA LICZBE X 16-BITOWA (TRAKTOWANA JAKO LICZBA
   *       BEZ ZNAKU) O C BITOW W LEWO. NAJSTARSZE BITY Z LEWEJ STRO-
   *       NY, KTORE WYJDA POZA OBREB SLOWA, SA TRACONE. *)

VAR
  temp1, temp2: integer;

 BEGIN  (* SHIFTL *)
   /*$C
    MOV     %0, TEMP1 (%6)          ; PRZECHOWANIE R0
    MOV     X (%6), %0              ; ZALADOWANIE LICZBY X DO R0
    ASH     C (%6), %0              ; PRZESUNIECIE W LEWO O C BITOW
    MOV     %0, TEMP2 (%6)
    MOV     TEMP1 (%6), %0          ; ODTWORZENIE R0
    */
   shiftl := temp2
 END  (* SHIFTL *);

        (****************************************)
        (*           LOBIN                      *)
        (****************************************)

(*
 * PROCEDURA PRZEKSZTALCA LICZBE TYPU LONUM NA LICZBE BINARNA TYPU
 * TABLBIN, 20 BITOWA, W TABLICY, KTOREJ KOLEJNE ELEMENTY
 * ODPOWIADAJA KOLEJNYM BITOM
 * WE: ld - LICZBA TYPU LONUM
 * WY: lb - LICZBA BINARNA W TABLICY
 */
PROCEDURE lobin (ld: lonum; VAR lb: tablbin);
VAR
  i: integer;
 BEGIN
   FOR i:=1 TO 20 DO
    BEGIN
      lb[i]:=ld AND 1;
      ld:=shiftr (ld, 1)
    END
 END;
/* LOBIN */
```

```
        (*************************************)
        (*           BINLON                  *)
        (*************************************)

(*
 * PROCEDURA PRZEKSZTALCA LICZBE BINARNA TYPU TABLBIN NA LICZBE
 * TYPU LONUM. BITY 17 - 20 LICZBY TYPU TABLBIN SA IGNOROWANE.
 */
PROCEDURE binlon (VAR lbin: tablbin; VAR ldl: lonum);
VAR
  i: integer;
 BEGIN
   ldl:=0;
   FOR i:=16 DOWNTO 1 DO
    BEGIN
      ldl:=shiftl (ldl, 1);
      ldl:=ldl+lbin[i]
    END
 END;
/* BINLON */

        (*************************************)
        (*           SUMBIN                  *)
        (*************************************)

(*
 * PROCEDURA SUMUJE 2 LICZBY BINARNE TYPU TABLBIN.
 * WE: l1, l2 - SKLADNIKI
 * WY: suma - WYNIK SUMOWANIA
 */
PROCEDURE sumbin (l1, l2: tablbin; VAR suma: tablbin);
VAR
  i, carry: integer;
 BEGIN
   carry:=0;
   FOR i:=1 TO 20 DO
   .BEGIN
      suma[i]:=(l1[i]+l2[i]+carry) MOD 2;
      carry:=(l1[i]+l2[i]+carry) DIV 2;
    END;
   IF carry<>0
    THEN
    skok1 (37)
 END;
/* SUMBIN */

        (*************************************)
        (*           SUB                     *)
        (*************************************)

(*
 * PROCEDURA REALIZUJACA ODEJMOWANIE LICZB TYPU TABLBIN.
 * WE: l1 - ODJEMNA
 *     l2 - ODJEMNIK
 * WY: l - ROZNICA
 *     carry - BIT PRZENIESIENIA
 */
PROCEDURE sub (VAR l1, l2, l: tablbin; VAR carry: integer);
VAR
  i: integer;
 BEGIN
   carry:=0;
   FOR i:=1 TO 20 DO
    BEGIN
      IF (l1[i] = 1) AND (carry = 1)
       THEN
        BEGIN
          carry:=0;
```

```
          l1[i]:=0
        END;
      l[i]:=l1[i]+carry-l2[i];
      IF l[i]<0
       THEN
        BEGIN
          carry:=1;
          l[i]:=1
        END
    END
 END;
/* SUB */

        (*****************************************)
        (*            SUBBIN                     *)
        (*****************************************)

(*
 * PROCEDURA ODEJMUJE LICZBY BINARNE TYPU TABLBIN.
 * WE: l1 - ODJEMNA
 *     l2 - ODJEMNIK
 * WY: l - ROZNICA
 * GDY WYNIK MNIEJSZY OD 0 WYJSCIE Z BLEDEM
 */
PROCEDURE subbin (l1, l2: tablbin; VAR l: tablbin);
VAR
  carry: integer;
 BEGIN
   sub (l1, l2, l, carry);
   IF carry<>0
    THEN
    skok1 (37)
 END;
/* SUBBIN */

        (*****************************************)
        (*            SUBSBIN                    *)
        (*****************************************)

(*
 * PROCEDURA ODEJMUJE LICZBY BINARNE TYPU TABLBIN.
 * WE: l1 - ODJEMNA
 *     l2 - ODJEMNIK
 * WY: l  - ROZNICA
 *     ok - POPRAWNOSC WYNIKU
 * WYNIK POPRAWNY JEST W ZAKRESIE (-32K, 32K-1)
 */
PROCEDURE subsbin (l1, l2: tablbin; VAR l: tablbin; VAR ok: boolean);
VAR
  i, carry: integer;
 BEGIN
   sub (l1, l2, l, carry);
   ok:=true;
   FOR i:=17 TO 20 DO
   ok:=ok AND (l[i]=l[16])
 END;
/* SUBSBIN */

        (*****************************************)
        (*            ADDRBIN                    *)
        (*****************************************)

(*
 * PROCEDURA PRZETWARZA ADRES POSTACI PARAGRAF, OFFSET NA LICZBE
 * BINARNA TYPU TABLBIN.
 * WE: par, off - ADRES W POSTACI WEJSCIOWEJ
 * WY: lb - ADRES W POSTACI BINARNEJ TABLBIN.
 */
```

```
PROCEDURE addrbin (par, off: lonum; VAR lb: tablbin);
VAR
  l1, l2: tablbin;
  i: integer;
 BEGIN
   lobin (par, l2);
   FOR i:=1 TO 16 DO
   l1[i+4]:=l2[i];
   FOR i:=1 TO 4 DO
   l1[i]:=0;
   lobin (off, l2);
   sumbin (l1, l2, lb)
 END;
/* ADDRBIN */

        (*****************************************)
        (*           DIGIT                       *)
        (*****************************************)

(*
 * PROCEDURA DRUKUJE ZNAK HEXADECYMALNY
 * WE: i = WARTOSC ZNAKU DO WYDRUKU
 *)
PROCEDURE digit(VAR lst: text; i: lonum);
 BEGIN
   IF i<10
    THEN
    write(lst,chr(i+ord('0')))
    ELSE
    write(lst,chr(i-10+ord('A')))
 END;
(*      DIGIT   *)

        (*****************************************)
        (*           NUM4H                       *)
        (*****************************************)

(*
 * PROCEDURA DRUKUJE HEXADECYMALNIE LICZBE Z ZAKRESU 0..65535
 * WE: i = DANA LICZBA
 *)
PROCEDURE num4h(VAR lst: text; i: lonum);
 BEGIN
   digit(lst, shiftr(i, 12));
   i:=i AND 7777B;
   digit(lst, shiftr(i, 8));
   i:=i AND 377B;
   digit(lst, shiftr(i, 4));
   i:=i AND 17B;
   digit(lst, i)
 END;
(*      NUM4H   *)

        (*****************************************)
        (*           NUM 4                       *)
        (*****************************************)

(*
 * PROCEDURA DRUKUJE HEXADECYMALNIE LICZBE Z ZAKRESU 0.. 65535 Z LITERA 'H'
 * WE: i = DANA LICZBA
 *)
PROCEDURE num4(VAR lst: text; i: lonum);
 BEGIN
   num4h(lst, i);
   write(lst,'H')
 END;
(*      NUM4    *)
```

```
        (*************************************)
        (*           NUM 5                   *)
        (*************************************)

(*
 * PROCEDURA DRUKUJEHEXADECYMALNIE LICZBE Z ZAKRESU 0..1048585
 * WE: i = DANA LICZBA DIV 16
 *     k = DANA LICZBA MOD 16
 *)
PROCEDURE num5(VAR lst: text; i: lonum; k: integer);
 BEGIN
   num4h(lst, i);
   digit(lst, k);
   write(lst,'H')
 END;
(*      NUM5    *)

        (*************************************)
        (*           INC                     *)
        (*************************************)

(*
 * FUNKCJA ODCZYTUJE ZNAK Z PLIKU we, PRZETWARZA GO NA BAJTOWY INTEGER
 * (STARSZY BAJT = 0) I MODYFIKUJE SUME KNTROLNA sum GDY NIE MOZE ODCZYTAC
 * ZNAKU BO EOF - BLAD.
 * WE: WSK PLIKU we USTAWIONY NA CZYTANY ZNAK I USTAWIONY nrerror
 * WY: ch = ODCZYTANY ZNAK
 *)
FUNCTION inc(VAR ch: char; VAR sum: lonum): integer;
VAR
  i: integer;
 BEGIN
   IF eof(we)
    THEN skok1(12);
   IF eoln(we)
    THEN
    readln(we);
   read(we,ch);
   i:=ord(ch) AND 377B;
   sum:=(sum+i) AND 377B;
   inc:=i
 END;
(*      INC     *)

        (*************************************)
        (*           LONGREC                 *)
        (*************************************)

(*
 * PROCEDURA ODCZYTUJE DLUGOSC REKORDU we. MODYFIKUJE sum. GDY EOF - BLAD.
 * WE: WSK PLIKU NA 1-SZY BAJT DLUGOSCI REKORDU
 * WY: WSK PLIKU NA 1-SZY BAJT ZA BAJTAMI DLUGOSCI REKORDU
 *     dlr = DLUGOSC REKORDU
 *)
PROCEDURE longrec(VAR dlr, sum: lonum);
VAR
  ch: char;
 BEGIN
   dlr:=(inc(ch,sum) AND 377B) OR shiftl (inc(ch,sum), 8)
 END;
(*      LONGREC         *)

        (*************************************)
        (*           REWREC                  *)
        (*************************************)

(*
 * PROCEDURA PRZEWIJAJACA REKORD NA PLIKU we. SPRAWDZA CZY sum PO PRZEWINIE-
```

```
 * CIU PLIKU = 0. GDY sum <> 0 LUB EOF PRZED KONCEM REKORDU - BLAD.
 * WE: WSK PLIKU USTAWIONY NA 1-SZY BAJT DLUGOSCI REKORDU
 *     sum = SUMA KONTROLNA PO BAJCIE TYPU PLIKU
 * WY: WSK PLIKU USTAWIONY NA BAJT TYPU NASTEPNEGO REKORDU LUB EOF.
 *)
PROCEDURE rewrec(VAR sum: lonum);
VAR
  i, k, dlr: lonum;
  ch: char;
 BEGIN
   longrec(dlr,sum);
   FOR k:=1 TO dlr DO
   i:=inc(ch,sum);
   IF sum<>0
    THEN skok1(12)
 END;
(*      REWREC  *)

        (*****************************************)
        (*           EQU                         *)
        (*****************************************)

(*
 * FUNKCJA POROWNUJE 2 ADRESY 20-TO BITOWE POSTACI: PARAGRAF, OFFSET.
 * = TRUE GDY ADRESY ROWNE
 * = FALSE GDY ADRESY ROZNE
 *)
FUNCTION equ ( VAR adres1, adres2: adres): boolean;
 BEGIN
   equ:=(adres1.par=adres2.par) AND (adres1.off=adres2.off)
 END;
/* EQU */

        (*****************************************)
        (*           GT                          *)
        (*****************************************)

(*
 * FUNKCJA POROWNUJE 2 ADRESY 20-TO BITOWE POSTACI: PARAGRAF, OFFSET
 * ZNORMALIZOWANE T.J. OFFSET < 16.
 * = TRUE GDY adres1 > adres2
 * = FALSE GDY adres1 <= adres2
 *)
FUNCTION gt ( VAR adres1, adres2: adres): boolean;
 BEGIN
   gt:=(adres1.par>adres2.par)
   OR ((adres1.par=adres2.par) AND (adres1.off>adres2.off))
 END;
/* GT */

        (*****************************************)
        (*           SUMATOR                     *)
        (*****************************************)

/*
 *       FUNKCJA SUMUJE DWIE LICZBY 0..65535 I SPRAWDZA CZY WYNIK NIE
 *       PRZEKRACZA ZAKRESU (65535). GDY JEST PRZEKROCZENIE
 *       NASTEPUJE WYJSCIE Z PROCEDURY Z BLEDEM nr.
 */
FUNCTION sumator (x, y: lonum; nr: integer): lonum;
VAR
  z: lonum;
 BEGIN
   z:=x+y;
   IF (z<x) OR (z<y)
    THEN
    skok1(nr);
   sumator:=z
```

```
 END;
/*      SUMATOR     */

        (*************************************)
        (*           ALITADDR                *)
        (*************************************)

/*
 * PROCEDURA OKRESLA ODSTEP DO NAJBLIZSZEGO ADRESU SPELNIAJACEGO
 * DANY ALIGN TYPE. ADRES PODANY NA WE NIE MUSI BYC W POSTACI
 * ZNORMALIZOWANEJ.
 * WE: ad    = ADRES BIEZACY OD KTOREGO LICZONY JEST ODSTEP
 *     typ   = TYP SEGMENTU (TAK JAK W REKORDZIE DANYCH
 *             O SEGMENCIE W MODULE WE)
 * WY: delta = OBLICZONY ODSTEP
 */
PROCEDURE alitaddr (ad: adres; typ: integer; VAR delta: lonum);
VAR
  tp, mask: integer;
 BEGIN
   tp:=typ AND at;
   CASE tp OF
     atbt: mask:=0;
     atwo: mask:=1;
     atpr: mask:=17B;
     atpg: mask:=377B;
    END;
   delta:=(shiftl (ad.par, 4) + ad.off) AND mask;
   IF delta<>0
    THEN
    delta:=mask+1-delta;
 END;
/* ALITADDR */
```

```
procedure num4(VAR lst: text; i: lonum); external;
procedure num5(VAR lst: text; i: lonum; k: integer); external;
(**E+*)

        (*************************************)
        (*          ERRORCOM                 *)
        (*************************************)

/*
+ PROCEDURA DRUKUJE KOMUNIKAT O BLEDZIE NA PLIKU LISTINGOWYM lst (JEZELI
+ OTWARTY) I NA MONITORZE OPERATORA (PLIK ti)
*/

procedure errorcom(i: integer);
var
segment: array[1..12] of char;
class: array[1..10] of char;
filet, name: array[1..9] of char;

        (*************************************)
        (*          KOMUNIKAT                *)
        (*************************************)

(*
* PROCEDURA DRUKUJE KOMUNIKAT O BLEDZIE NA PLIKU kom.
* WE: kom = PLIK NA KTORYM JEST DRUKOWANY KOMUNIKKAT
*     n = NUMER BLEDU
*     error1, error2, error3 = NAZWY DO WYDRUKOWANIA W TEXCIE KOMUNIKATU
*     dat1, lowad, highad = WARTOSCI DO WYDRUKOWANIA W TEXCIE KOMUNIKATU
*)

procedure komunikat(n: integer; var kom: text);
begin
if (n=13) or (n=16) or (n=26) or (n=28) or (n=31) or (n=32)
  or (n=36) or (n=38) or (n=49)
   then
    write(kom,'   WARNING ')
   else
    write(kom,'   ERROR ');
write(kom,n:2, ':   ');
case n of
2:      writeln(kom,'INVALID SYNTAX');
3:      writeln(kom,'MISSING INPUT FILE NAME');
6:      writeln(kom,'INVALID KEY WORD');
7:      writeln(kom,'NUMERIC CONSTANT LARGER THEN 20 BITS');
8:      writeln(kom,'NON NUMERIC  CHARACTER IN NUMERIC CONSTANT');
9:      writeln(kom,'NUMERIC CONSTANT LARGER THEN 16 BITS');
10:     begin
        writeln(kom,'TOO MANY NAMES IN INPUT MODULE');
        writeln(kom,  filet, error1);
        end;
11:     begin
        writeln(kom,'SEGMENT COMBINATION ERROR');
        writeln(kom, filet, error1);
        writeln(kom, segment, error2);
        writeln(kom, class, error3);
        end;
12:     writeln(kom,'INVALID INPUT MODULE');
13:     begin
        writeln(kom,'MORE THEN ONE SEGMENT WITH MEMORY ATTRIBUTE');
        writeln(kom, segment, error1);
        end;
16:     begin
        writeln(kom,'DIFFERENT VALUES FOR');
        writeln(kom, filet,  error1);
        writeln(kom,'   SYMBOL: ', error2);
        end;
```

```
17:     writeln(kom,'TOO MANY INPUT MODULES');
18:     begin
        writeln(kom,'TOO MANY SEGMENTS IN INPUT MODULE');
        writeln(kom, filet, error1);
        end;
20:     begin
        writeln(kom,'INVALID NAME');
        writeln(kom, name, error1);
        end;
22:     begin
        writeln(kom,'SEGMENT SIZE OVERFLOW; OLD SIZE + CHANGE > 16 BITS');
        writeln(kom, segment, error1);
        end;
23:     begin
        writeln(kom,'SEGMENT SIZE UNDERFLOW; OLD SIZE - CHANGE < 0');
        writeln(kom, segment, error1);
        end;
24:     writeln(kom,'INVALID ADDRESS RANGE');
26:     begin
        writeln(kom,'DECREASING SIZE OF SEGMENT');
        writeln(kom, segment, error1);
        end;
27:     begin
        writeln(kom,'SPECIFIED SEGMENT IS ABSOLUTE');
        writeln(kom, segment, error1);
        end;
28:     begin
        writeln(kom,'PAGE RESIDENT SEGMENT CROSSES PAGE BOUNDARY');
        writeln(kom, segment, error1);
        end;
29:     begin
        writeln(kom,'TOO MANY EXTERNAL SYMBOLS IN INPUT MODULE');
        writeln(kom, filet, error1);
        end;
31:     begin
        writeln(kom,'UNRESOLVED EXTERNAL REFERENCE TO NAME NEAR SPECIFIED ADDRESS
        writeln(kom, name, error2);
        writeln(kom, segment, error1);
        write(kom,'   NEAR: ');
        num4(kom,dat1);
        writeln(kom);
        end;
32:     writeln(kom,'UNRESOLVED SYMBOLS');
33:     begin
        writeln(kom,'SEGMENT OVERFLOW');
        writeln(kom, segment, error1);
        writeln(kom, class, error2);
        end;
34:     begin
        writeln(kom,'SPECIFIED CLASS NOT FOUND IN INPUT MODULES');
        writeln(kom, class, error2);
        end;
35:     begin
        writeln(kom,'SPECIFIED SEGMENT NOT FOUND IN INPUT MODULES');
        writeln(kom, segment, error1);
        end;
36:     begin
        writeln(kom,'SEGMENTS OVERLAP');
        writeln(kom, segment, error1);
        writeln(kom, segment, error2);
        write(kom,'   LOW OVERLAP ADDRESS: ');
        num5(kom, lowad.par, lowad.off);
        writeln(kom);
        write(kom,'   HIGH OVERLAP ADDRESS: ');
        num5(kom, highad.par, highad.off);
        writeln(kom);
        end;
37:     writeln(kom,'INPUT MODULES EXCEED 8086 MEMORY');
```

```
38:     begin
        writeln(kom,'SEGMENT WITH MEMORY ATTRIBUTE NOT PLACED HIGHEST IN MEMORY')
        writeln(kom, segment, error1);
        end;
39:     begin
        writeln(kom,'NO MEMORY BELOW SEGMENT FOR SPECIFIED SEGMENT');
        writeln(kom, segment, error1);
        writeln(kom, segment, error2);
        end;
40:     writeln (kom, 'OFFSET FIXUP OVERFLOW');
41:     begin
        writeln(kom,'SPECIFIED CLASS OUT OF ORDER');
        writeln(kom, class, error1);
        end;
42:     begin
        writeln(kom,'SPECIFIED SEGMENT OUT OF ORDER');
        writeln(kom, segment, error1);
        end;
43:     begin
        writeln(kom,'ADDRESS FOR CLASS SPECIFIED MORE THEN ONCE');
        writeln(kom, class, error1);
        end;
44:     begin
        writeln(kom,'SEGMENT ADDRESS PREVIOUSLY SPECIFIED IN INPUT MODULE OR COMM
        writeln(kom, segment, error1);
        end;
45:     begin
        writeln(kom,'SEGMENT SPECIFIED MORE THEN ONCE IN ORDER');
        writeln(kom, segment, error1);
        end;
46:     begin
        writeln(kom,'CLASS SPECIFIED MORE THEN ONCE IN ORDER');
        writeln(kom, class, error2);
        end;
47:     writeln(kom,'SPECIFIED SEGMENT NOT IN SPECIFIED CLASS');
48:     writeln(kom,'INVALID COMMAND LINE');
49:     writeln(kom,'SEGMENT ALIGNMENT NOT COMPATIBLE WITH ASSIGNED ADDRESS');
50:     writeln(kom,'INVALID COMMAND LINE; TOKEN TOO LONG');
51:     writeln (kom, 'REFERENCED LOCATION IS OUTSIDE 64K FRAME OF REFERENCE');
53:     begin
        writeln(kom,'CAN NOT ALLOCATE CLASS AT SPECIFIED ADDRESS');
        writeln(kom, class, error1);
        end;
end
end;    (*      KOMUNIKAT       *)
/*++++++++++++++++++++++++ POCZATEK PROCEDURY ERRORCOM ++++++++++++++++++++++++*/
begin
  segment:='   SEGMENT: ';
  class:='   CLASS: ';
  name:='   NAME: ';
  filet:='   FILE: ';
  komunikat(i,ti);
  if listopen then
    komunikat(i,lst);
end;    (* ERRORCOM     *)
```

```
(**E+*)
FUNCTION gt(VAR adres1, adres2: adres): boolean; EXTERNAL;
FUNCTION shiftl (x, c: integer): integer; EXTERNAL;
PROCEDURE errorcom(i: integer); EXTERNAL;
PROCEDURE skok1(nrerror: integer); EXTERNAL;

        (*************************************)
        (*          DYREKTYWA                *)
        (*************************************)

(*
 + PROCEDURA ODCZYTUJE LINIE WYWOLANIA WRAZ Z EW. KONTYNUACJAMI,
 + SPRAWDZA POPRAWNOSC SYNTAKTYCZNA, ANALIZUJE I PRZETWARZA DANE
 * PODANE PRZEZ OPERATORA
 *)
PROCEDURE dyrektywa;
VAR
  curchar, charaddr, charsegs, i: integer;
  over: boolean;
  num: liczba;
  sep: SET OF char;
  filptr: ^inpufil;
  curptr: posptr;
  txtnull: nazwa;
  namecon: ARRAY[0..8] OF char;

        (*************************************)
        (*          CZYTAJ                   *)
        (*************************************)

  (*
   * PROCEDURA ODCZYTU I NORMALIZACJI LINII DYTREKTYWY. ODCZYTYWANIE
   * JEST POWTARZANE DOPOKI NIE ZOSTANIE NAPISANA LINIA NIEPUSTA.
   * JEZELI ZOSTANIE ZAPISANY ZNAK KONTYNUACJI '&' NASTEPNE ZNAKI
   * DO KONCA LINII SA IGNOROWANE I JEST ODCZYTYWANA LINIA NASTEPNA,
   * ZACZYNAJACA SIE OD '**'.
   * NORMALIZACJA LINII POLEGA NA ZAMIANIE MALYCH LITER NA DUZE
   * I USUNIE CIU SPACJI.
   *)
  PROCEDURE czytaj;
  VAR
    linenum, charnum, nrlnor, nrcnor, i: integer;
    continue: boolean;
  BEGIN
     REPEAT  (*      ODCZYT DYREKTYWY        *)
      linenum:=0;
      nrlnor:=0;
      nrcnor:=0;
      new(tlip[0]);
      write('%C86)'); /* PISZ PROMPT */
       REPEAT
        readln;
        new(tlin[linenum]);
        charnum:=0;
        WHILE NOT(eoln OR (charnum>dlw)) DO /* CZYTAJ LINIE */
         BEGIN
           read(tlin[linenum]^[charnum]);
           charnum:=charnum+1
         END;
        FOR i:=charnum TO dlw DO /* UZUPELNJ LINIE */
        tlin[linenum]^[i]:='%';
        continue:=false;
        FOR i:=0 TO dlw DO /* NORMALIZUJ LINIE ZNAK PO ZNAKU */
         BEGIN
           IF tlin[linenum]^[i]='&' /* DO NASTEPNEJ LINII */
            THEN
             BEGIN
```

QRL

```
                    continue:=true;
                    EXIT
                  END;
                IF (tlin[linenum]^[i]>='a') AND (tlin[linenum]^[i]<='z')
                 THEN /* ZAMIANA MALYCH LITER NA DUZE */
                  BEGIN
                    tlip[nrlnor]^[nrcnor]:=
                    chr(ord(tlin[linenum]^[i])+ord('A')-ord('a'));
                    nrcnor:=nrcnor+1
                  END
                 ELSE
                 IF tlin[linenum]^[i]='%' /* KONIEC LINII */
                  THEN EXIT
                  ELSE
                  IF (tlin[linenum]^[i]<>' ') AND (tlin[linenum]^[i]<>chr(11B))
                   THEN /* PRZEPISZ ZNAK <>LITERA, ODSTEP */
                    BEGIN
                      tlip[nrlnor]^[nrcnor]:=tlin[linenum]^[i];
                      nrcnor:=nrcnor+1
                    END;
                IF nrcnor>dlw /* OTWARCIE NOWEJ LINII ZNORMALIZOWANEJ */
                 THEN
                  BEGIN
                    nrlnor:=nrlnor+1;
                    new(tlip[nrlnor]);
                    nrcnor:=0
                  END
              END;
             IF continue
              THEN /* OTWARCIE LINII KONTYNUACJI */
               BEGIN
                 linenum:=linenum+1;
                 IF linenum>20
                  THEN
                  skok1(50)
                  ELSE
                  write('**')
               END
            UNTIL NOT continue;
           tlip[nrlnor]^[nrcnor]:='%' /* ZAKONCZENIE LINII ZNORMALIZOWANEJ */
         UNTIL nrlnor+nrcnor>0 /* POWTORZ PROMPT GDY LINIA WYWOLANIA PUSTA */
  END;
(*      CZYTAJ  *)

      (*****************************************)
      (*           LINE                        *)
      (*****************************************)

(*
 * FUNKCJA POBIERAJACA ZNAK Z TEKSTU DYREKTYWY.
 * WE: i = NR ZNAKU W ZNORMALIZOWANYM TEKSCIE (T.J. BEZ SPACJI, DUZE
 *         LITERY, BEZ KOMENTAZY I ZNAKOW KONTYNUACJI)
 * WY: line = ZNAK
 *)
FUNCTION line(i: integer): char;
VAR
  ilw,ilz: integer;
 BEGIN
   ilw:=i DIV (dlw+1);
   ilz:=i MOD (dlw+1);
   IF tlip[ilw]=NIL
    THEN
    line:='%'
    ELSE
    line:=tlip[ilw]^[ilz];
 END;
(*      LINE    *)
```

```
      (*************************************)
      (*           CHCTRL                  *)
      (*************************************)

 (*
  * PROCEDURA KONTROLUJE CZY BIEZACY ZNAK W LINII WYWOLANIA JEST
  * DANYM ZNAKIEM. JEZELI NIE - NASTEPUJE WYJSCIE Z BLEDEM NR 2.
  * JEZELI TAK - PRZEJSCIE DO NASTEPNEGO ZNAKU.
  *)
 PROCEDURE chctrl(c: char);
  BEGIN
    IF line(curchar) <> c
     THEN
     skok1(2);
    curchar:=curchar+1;
  END;
 /* CHCTRL */

      (*************************************)
      (*           PLIK                    *)
      (*************************************)

 (*
  * FUNKCJA SPRAWDZA WSTEPNIE CZY DANY TEKST ODPOWIADA
  * DLUGOSCIA (DO DELIMITERA ) SPECYFIKACJI PLIKU I CZY ZAWIERA TYP
  * SPECYFIKACJA JEST PRZEPISYWANA DO TABLICY pl. JEZELI SPECYFIKACJA
  * NIE ZAWIERA TYPU PLIKU JEST DODAWANY TYP "'typ'86". SPECYFIKACJA
  * PLIKU W TABLICY pl JEST UZUPELNIANA SPACJAMI.
  * WE: curchar - NUMER PIERWSZEGO ZNAKU BADANEGO TEKSTU
  *     typ - ZNAK W EW. DODAWANYM TYPIE
  * WY: plik = TRUE GDY TEKST DLUGOSCIA ODPOWIADA SPECYFIKACJI PLIKU
  *           FALSE W PRZECIWNYM WYPADKU
  *     pl - TABLICA ZE SPECYFIKACJA PLIKU EW. UZUPELNIONA TYPEM
  *     curchar - JEZELI plik = TRUE - NUMER PIERWSZEGO ZNAKU ZA
  *                                    SPECYFIKACJA PLIKU
  *               JEZELI plik = FALSE - NIE ZMIENIONE
  *)
 FUNCTION plik(VAR pl: nazwa; typ:char): boolean;
 LABEL
   1;
 VAR
   konto, ext: boolean;
   i,j: integer;

      (*************************************)
      (*           SETEXT                  *)
      (*************************************)

   (*
    | PROCEDURA SPRAWDZA CZY MOZE DOPISAC TYP DO SPECYFIKACJI PLIKU - CZY
    | DOTYCHCZASOWA CZESC SPECYFIKACJI NIE JEST ZA DLUGA.
    | PROCEDURA DOPISUJE DO SPECYFIKACJI PLIKU TYP "'typ'86" POCZYNAJAC
    | OD ZNAKU NR j W TABLICY ZAWIERAJACEJ SPECYFIKACJE
    *)
   PROCEDURE setext(long: integer);
    BEGIN
      IF long>22
       THEN GOTO 1;
      pl[j]:='.';
      pl[j+1]:=typ;
      pl[j+2]:='8';
      pl[j+3]:='6';
      j:=j+4;
      ext:=true
    END;
   /* SETEXT */

 BEGIN
```

```
        konto:=false;
        ext:=false;
        plik:=false;
        j:=1;
        FOR i:=curchar TO curchar+dls-1 DO
         BEGIN /* PRZEPISZ ZNAK SPECYFIKACI PLIKU */
           IF line(i)='['
            THEN konto:=true;
           IF line(i)=']'
            THEN konto:=false;
           IF (line(i) IN sep) AND (NOT konto OR (line(i)<>','))
            THEN (* ZNAK "," NIE KONCZY SPECYFIKACJI GDY WYSTEPUJE
                  W NUMERZE KONTA *)
            EXIT;
           ext:=ext OR (line(i)='.');
           IF NOT ext AND (line(i)=';')
            THEN /* DODAJ TYP PRZED NUMEREM WERSJI */
            setext(i-curchar);
           pl[j]:=line(i);
           j:=j+1;
           IF j>dls
            THEN EXIT
         END;
        IF (i=curchar) OR (i>curchar+dls-1) OR NOT(line(i) IN sep)
         THEN /* BLAD DLUGOSCI SPECYFIKACJI PLIKU */
         GOTO 1;
        IF NOT ext
         THEN /* DODAJ TYP NA KONCU GDY NIE MA NUMERU WERSJI */
         setext(j);
        curchar:=i;
        plik:=true;
        FOR i:=j TO dls DO
        pl[i]:=' ';
1:
     END;
    (* plik *)

        (*****************************************)
        (*           CLASSES                     *)
        (*****************************************)

    (*
     * FUNKCJA SPRAWDZA, CZY TEKST ZACZYNAJACY SIE OD BIEZACEGO ZNAKU
     * (curchar-TEGO) JEST SLOWEM KLUCZOWYM CLASSES LUB CS I CZY ZA NIM
     * NASTEPUJE "(".
     * WY: classes = TRUE GDY JEST SLOWO KLUCZOWE
     *             = FALSE W PRZECIWNYM WYPADKU
     *     curchar = NR PIERWSZEGO ZNAKU ZA SL. KLUCZ. I '(' GDY classes
     *               NIEZMIENIONE W PRZECIWNYM WYPADKU
     *)
    FUNCTION classes :boolean;
    VAR
      l1: ARRAY[0..7] OF char;
      i: integer;
     BEGIN
       FOR i:=0 TO 7 DO l1[i]:=' ';
       FOR i:=0 TO 7 DO
        BEGIN
          l1[i]:=line(curchar+i);
          IF l1[i] IN sep
           THEN EXIT
        END;
       IF l1='CLASSES('
        THEN
         BEGIN
           classes:=true;
           curchar:=curchar+8;
         END
```

```
    ELSE
    IF l1='CS(     '
     THEN
      BEGIN
        classes:=true;
        curchar:=curchar+3;
      END
     ELSE
     classes:=false
 END;
(*  classes  *)

      (*************************************)
      (*          SEGMENTS                 *)
      (*************************************)

(*
 * FUNKCJA SPRAWDZA, CZY TEKST ZACZYNAJACY SIE OD BIEZACEGO ZNAKU
 * (curchar-TEGO) JEST SLOWEM KLUCZOWYM SEGMENTS LUB SM I CZY ZA
 * NIM NASTEPUJE "(".
 * WY: segments = TRUE GDY JEST SLOWO KLUCZOWE
 *              = FALSE W PRZECIWNYM WYPADKU
 *     curchar  = NR PIERWSZEGO SLOWA ZA SL. KLUCZ. I "("
 *                GDY segments
 *              = NIEZMIENIONE W PRZECIWNYM WYPADKU
 *)
FUNCTION segments :boolean;
VAR
  l1: ARRAY[0..8] OF char;
  i: integer;
 BEGIN
   FOR i:=0 TO 8 DO l1[i]:=' ';
   FOR i:=0 TO 8 DO
    BEGIN
      l1[i]:=line(curchar+i);
      IF l1[i] IN sep
       THEN EXIT
    END;
   IF l1='SEGMENTS('
    THEN
     BEGIN
       segments:=true;
       curchar:=curchar+9;
     END
    ELSE
    IF l1='SM(     '
     THEN
      BEGIN
        segments:=true;
        curchar:=curchar+3;
      END
     ELSE
     segments:=false
 END;
(*  segments  *)

      (*************************************)
      (*          COPY                     *)
      (*************************************)

(*
 * PROCEDURA KOPIUJE NAZWE SEGMENTU (KLASY, ZMIENNEJ) POCZYNAJAC
 * OD curchar-TEGO ZNAKU DO SEPARATORA. MAX ILOSC KOPIOWAYCH
 * ZNAKOW dls.
 * curchar JEST USTAWIANA PIERWSZYM ZNAKU ZA NAZWA (NA SEPARATORZE).
 * JEZELI NAZWA NIE WYSTEPUJE - WYJSCIE Z BLEDEM NR 2.
 *)
PROCEDURE copy(VAR t: nazwa);
```

```
VAR
  i,j: integer;
 BEGIN
   i:=1;
   WHILE(i(=dls) AND NOT (line(curchar-1+i) IN sep) DO
    BEGIN /* PRZEPISZ NAZWE */
      t[i]:=line(curchar-1+i);
      i:=i+1
    END;
   FOR j:=i TO dls DO /* UZUPELINJ NAZWE SPACJAMI */
   t[j]:=' ';
   IF i = 1 /* BRAK NAZWY */
    THEN skok1(2);
   curchar:=curchar-1+i;
   WHILE NOT (line(curchar) IN sep) DO /* OMIN POZOSTALE ZNAKI NAZWY */
   curchar:=curchar+1
 END;
(*  copy  *)

      (****************************************)
      (*           NUMBER                     *)
      (****************************************)

(*
 * PROCEDURA WYKONUJE KONWERSJE LICZBY W LINII DYREKTYWY. LICZBA
 * MOZE BYC PODANA W KODZIE DZIESIETNYM, HEXADECYMALNYM, OKTALNYM
 * LUB BINARNYM. MOZE BYC POPRZEDZONA DOWOLNA ILOSCIA  ZER.
 * OSTATNIA LITERA WSKAZUJE NA TYP KODU, ODPOWIEDNIO: 0-9, H,
 * O LUB Q, B. LICZBA W HEX MUSI NA POCZATKU MIEC CYFRE.
 * ZAKRES LICZB: 0..1 MEGA (20 BITOW)
 * JEZELI ZAPIS LICZBY NIE POPRAWNY - WYJSCIE Z BLEDEM NR 8.
 * WE: curchar = NR PIERWSZEGO ZNAKU KODU LICZBY
 * WY: over = PRZEKROCZENIE ZAKRESU PONAD 1 M
 *     num =  ZDEKODOWANA LICZBA, KOLEJNE ELEMENTY TABLICY ZAWIERAJA
 *            WARTOSCI KOLEJNYCH CYFR (OD NAJNIZSZEGO RZEDU) LICZBY
 *            W ZAPISIE HEX
 *     curchar = NR PIERWSZEGO ZNAKU ZA KODEM LICZBY GDY NOT over
 *          .       NIEOKRESLONE GDY over
 *)
PROCEDURE number;
LABEL
  1;
VAR
  lastchar, i, j: integer;
  bit: ARRAY[1..20] OF integer;

      (****************************************)
      (*           DEC                        *)
      (****************************************)

  (*
   | PROCEDURA WYKONUJE KONWERSJE LICZBY ZAPISANEJ W KODZIE
   | DZIESIETNYM
   | WE: curchar = NUMER PIERWSZEGO ZNAKU ZNACZACEGO W KODZIE LICZBY
   |     lastchar = NUMER OSTATNIEGO ZNAKU KODU LICZBY
   | WY: over= P. OPIS PROCEDURY number
   |     bit = TABLICA ZAWIERAJACA ZDEKODOWANA LICZBE W PSTACI
   |           BINARNEJ: KOLEJNE ELEMENTY TABLICY SA ROWNE KOLEJNYM
   |           BITOM (OD NAJNIZSZEGO) LICZBY
  *)
  PROCEDURE dec;
  VAR
    i, j, carry: integer;
    data: ARRAY [1..7] OF integer;
   BEGIN
     FOR i:=curchar TO lastchar DO /* KONTROLA POPRAWNOSCI LICZBY */
     IF NOT (line(i) IN ['0'..'9'])
      THEN
```

```
      skok1(8);
     over:=true;
     IF lastchar-curchar>6
      THEN /* ZA DLUGA LICZBA */
      GOTO 1;
     FOR i:=1 TO 20 DO /* ZERUJ TABLICE */
     data[i]:=0;
     FOR i:=curchar TO lastchar DO /* data:=WARTOSCI KOLEJNYCH CYFR */
     data[i-curchar+1]:=ord(line(i))-ord('0');
     FOR j:=1 TO 20 DO /* PRZETWORZ NA LICZBE BINARNA */
      BEGIN
        carry:=0;
        FOR i:=1 TO lastchar-curchar+1 DO
         BEGIN
           data[i]:=data[i]+10*carry;
           carry:=data[i] MOD 2;
           data[i]:=data[i] DIV 2
         END;
        bit[j]:=carry
      END;
     carry:=0; /* KONTROLA PRZEKROCZENIA WIELKOSCI LICZBY */
     FOR j:=1 TO lastchar-curchar+1 DO
     carry:=carry+data[j];
     over:=carry>0;
     IF over
      THEN
      GOTO 1;
  END;
 (*      dec     *)

     (*****************************************)
     (*           OCTAL                       *)
     (*****************************************)

 (*
  | PROCEDURA WYKONUJE KONWERSJE LICZBY ZAPISANEJ W KODZIE OKTALNYM
  | WE, WY - P. OPIS PROCEDURY dec
  *)
 PROCEDURE octal;
 VAR
   i, j: integer;
  BEGIN
    FOR i:=curchar TO lastchar-1 DO /* KONTROLA POPRAWNOSCI */
    IF NOT(line(i) IN ['0'..'7'])
     THEN
     skok1(8);
    over:=lastchar-curchar>7;
    IF over
     THEN /* ZA DLUGA LICZBA */
     GOTO 1;
    FOR i:=0 TO lastchar-curchar-1 DO
     BEGIN /* PRZETWORZ NA LICZBE BINARNA */
       j:=ord(line(lastchar-i-1))-ord('0');
       bit[3*i+1]:=j MOD 2;
       j:=j DIV 2;
       bit[3*i+2]:=j MOD 2;
       j:=j DIV 2;
       IF (i=6) AND (j>0)
        THEN
         BEGIN /* ZA DUZA WARTOSC */
           over:=true;
           GOTO 1
         END;
       bit[3*i+3]:=j MOD 2
     END;
  END;
 (*      OCTAL   *)
```

```
    (*************************************)
    (*           BIN                     *)
    (*************************************)

(*
 | PROCEDURA WYKONUJACA KONWERSJE LICZBY ZAPISANEJ W KODZIE
 | BINARNYM
 | WE I WY - P. OPIS PROCEDURY dec.
 *)
PROCEDURE bin;
VAR
  i, j: integer;
 BEGIN
   FOR i:=curchar TO lastchar-1 DO /* KONTROLA POPRAWNOSCI */
   IF NOT (line(i) IN ['0'..'1'])
    THEN
    skok1(8);
   IF lastchar-curchar>20 /* KONTROLA WIELKOSCI LICZBY */
    THEN
     BEGIN
       over:=true;
       GOTO 1;
     END
    ELSE
    over:=false;
   FOR j:=1 TO lastchar-curchar DO /* PRZETWORZ NA LICZBE BINARNA */
   bit[j]:=ord(line(lastchar-j))-ord('0');
 END;
(*      bin     *)

    (*************************************)
    (*           HEX                     *)
    (*************************************)

(*
 | PROCEDURA WYKONUJE KONWERSJE LICZBY ZAPISANEJ W KODZIE
 | HEXADECYMALNYM
 | WE - P. OPIS PROCEDURY dec
 | WY: over = P. OPIS PROCEDURY dec
 |      num = P. OPIS PROCEDURY number
 *)
PROCEDURE hex;
VAR
  i: integer;
  c: char;
 BEGIN
   IF NOT (line(curchar) IN ['1'..'9']) AND (line(curchar-1)<>'0')
    THEN /* PIERWSZY ZNAK - CYFRA */
    skok1(8);
   FOR i:=curchar+1 TO lastchar-1 DO /* KONTROLA NASTEPNYCH ZNAKOW */
   IF NOT (line(i) IN ['0'..'9','A'..'F'])
    THEN
    skok1(8);
   over:=lastchar-curchar>5; /* KONTROLA WIELKOSCI LICZBY */
   IF over
    THEN
    GOTO 1;
   FOR i:=lastchar-1 DOWNTO curchar DO
    BEGIN /* PRZETWORZ NA LICZBE HEXADECYMALNA num */
      c:=line(i);
      IF c IN ['0'..'9']
       THEN
       num[lastchar-i]:=ord(c)-ord('0')
       ELSE
       num[lastchar-i]:=ord(c)-ord('A')+10
    END;
 END;
(*      hex     *)
```

```
  (*----------   POCZATEK PROCEDURY NUMBER ------------------------------*)
 BEGIN
   WHILE line(curchar) ='0' DO /* OPUSC POCZATKOWE ZERA */
   curchar:=curchar+1;
   FOR j:=1 TO 20 DO /* ZERUJ TABLICE */
   bit[j]:=0;
   FOR j:=1 TO 5 DO
   num[j]:=0;
   lastchar:=curchar;
   WHILE NOT (line(lastchar) IN sep) DO /* SZUKAJ OSTATNIEGO ZNAKU */
   lastchar:=lastchar+1;
   lastchar:=lastchar-1;
   IF line(lastchar) ='H' /* W ZALEZNOSCI OD OSTATNIEGO ZNAKU */
    THEN
     BEGIN
       hex;
       curchar:=lastchar+1;
       GOTO 1
     END
    ELSE
    IF line(lastchar) IN ['0'..'9']
     THEN dec
     ELSE
     IF line(lastchar)='B'
      THEN bin
      ELSE
      IF line(lastchar) IN ['0'..'Q']
       THEN octal
       ELSE skok1(8);
   curchar:=lastchar+1;
   FOR i:=4 DOWNTO 0 DO /* PRZETWORZ LICZBY BIN NA HEX W TABLICY num */
   FOR j:=4 DOWNTO 1 DO
   num[i+1]:=2*num[i+1]+bit[4*i+j];
1:
 END;
(*      number  *)

      (****************************************)
      (*           SETS0C                     *)
      (****************************************)

(*
 * PROCEDURA DOPISUJE POZYCJE SEGMENTU (KLASY) O PODANYCH PARAMETRACH
 * NA KONCU LISTY s0clist NA POZYCJI WSKAZYWANEJ PRZEZ curptr.
 * TWORZY NOWA POZYCJE I PRZESUWA DO NIEJ curptr.
 *)
PROCEDURE sets0c(sgm, cls: nazwa; setord, setabs: order;
                 paragraph, offset, length, ssvalue: lonum; setss: char);
 BEGIN
   WITH curptr^ DO
    BEGIN /* WYPELNIJ BIEZACA POZYCJE */
      seg:=sgm;
      kl:=cls;
      zord:=setord;
      aabs:=setabs;
      pusty:=true;
      ap.par:=paragraph;
      ap.off:=offset;
      dl:=length;
      ssc:=setss;
      ssv:=ssvalue;
      new(ptr); /* UTWORZ NOWA POZYCJE */
      curptr:=ptr
    END;
   curptr^.ptr:=NIL /* WSTEPNIE WYPELNIJ NOWA POZYCJE */
 END;
/* SETS0C */
```

```
        (*************************************)
        (*           CONORDER                *)
        (*************************************)

(*
 * PROCEDURA ANALIZUJE CONTROL order. POZYCJE OKRESLONE NA PODST.
 * order WPISUJE DO TABLICY WSKAZYWANEJ PRZEZ curptr. SPRAWDZA
 * POPRAWNOSC SYNTAKTYCZNA order - BLAD POWODUJE WYJSCIE Z BLEDEM
 * NR 2.
 * WE: curchar = NR PIERWSZEGO ZNAKU ANALIZOWANEGO TEKSTU
 *               (PIERWSZEGO ZNAKU ZA 'order')
 *     curptr  = WSK PIERWSZEJ WOLNEJ POZYCJI W TABLICY
 *               DO KTOREJ BEDA ZAPISYWANE DANE Z order
 * WY: curchar = NR PIERWSZEGO ZNAKU ZA TEKSTEM
 *               NIEOKRESLONE GDY NOT popr
 *     curptr  = WSK PIERWSZEJ WOLNEJ POZYCJI W TABLICY
 *               PO ZAPISANIU DANYCH
 *)
PROCEDURE conorder;
VAR
  segment, klasa: nazwa;
 BEGIN
   chctrl('(');
   curchar:=curchar-1;
    REPEAT
     curchar:=curchar+1;
     IF segments /* ANALIZA TEKSTU ZA SLOWEM 'SEGMENTS' LUB 'SM' I '(' */
      THEN
       BEGIN
         curchar:=curchar-1;
          REPEAT
           curchar:=curchar+1;
           copy(segment);
           sets0c(segment, txtnull, sm, no, 0, 0, 0, 0, '@')
          UNTIL line(curchar)<>',';
         chctrl(')');
       END
      ELSE
      IF classes /* ANALIZA TEKSTU ZA SLOWEM 'CLASSES' LUB 'CS' I '(' */
       THEN
        BEGIN
          curchar:=curchar-1;
           REPEAT
            curchar:=curchar+1;
            copy(klasa);
            IF line(curchar)='('
             THEN /* PODANE NAZWY SEGMENTOW W KLASIE */
              BEGIN
                 REPEAT
                  curchar:=curchar+1;
                  copy(segment);
                  sets0c(segment, klasa, smcs, no, 0, 0, 0, 0, '@')
                 UNTIL line(curchar)<>',';
                chctrl(')')
              END
             ELSE
             sets0c(txtnull, klasa, cs, no, 0, 0, 0, 0, '@')
           UNTIL line(curchar)<>',';
          chctrl(')')
        END
       ELSE
       skok1(6)
    UNTIL line(curchar)<>',';
   chctrl(')')
 END;
(* conorder  *)
```

```
      (*************************************)
      (*           NUMBIN                  *)
      (*************************************)

(*
 * FUNKCJA OBLICZA WARTOSC BINARNA LICZBY ZALISANEJ NA 4 SPOSROD 5
 * ELEMENTOW TABLICY  num (OD 1 DO 4 LUB OD 2 DO 5).
 * WE: high = INDEX MSB ELEMENTU LICZBY
 *     low  = INDEX LSB ELEMENTU LICZBY
 *)
FUNCTION
  numbin(high, low: integer): lonum;
VAR
  i: integer;
  sum: lonum;
 BEGIN
   sum:=0;
   FOR i:=high DOWNTO low DO
   sum:=shiftl (sum, 4) OR (num[i] AND 17B);
   numbin:=sum
 END;
/* NUMBIN */

      (*************************************)
      (*           ADDRESSES               *)
      (*************************************)

(*
 * PROCEDURA ANLIZUJE CONTROL addresses. SPRAWDZA POPRAWNOSC SYNTAKTYCZNA
 * addresses - BLAD POWODUJE WYJSCIE Z BLEDEM.
 * DLA DANYCH SEGMENTOW (KLAS) WYKONUJE:
 * - JEZELI SEGMENT (KLASA) NIE WYSTEOUJE W 'order' TWORZY ODPOWIADAJACA
 *   MU POZYCJE W TABLICY s0clist,
 * - JEZELI SEGMENT (KLASA) WYSTEPUJE W CONTROL 'order' UZUPELNIA POZYCJE
 *   W TABLICY s0clist (JEZELI POZYCJA ZOSTALA JUZ UZUPELNIONA, T.J.
 *   NAZWA POWTARZA SIE W addresses - BLAD
 *   WE: curchar = NUMER PIERWSZEGO ZNAKU ANALIZOWANEGO TEKSTU
 *                 (PIERWSZEGO ZA 'addresses')
 *       curchar = NR PIERWSZEGO ZNAKU ZA TEKSTEM
 *)
PROCEDURE addresses;
VAR
  nameabs: nazwa;

      (*************************************)
      (*           SETABS                  *)
      (*************************************)

  (*
   | PROCEDURA ANALIZUJE DEKLARACJE ADRESOW SEGMENTOW LUB KLAS W CONTROL
   | addresses. WYKONUJE DZIALANIA OPISANE DLA PROCEDURY addresses
   | DLA KOLEJNYCH SEGMENTOW (KLAS) WYSTEPUJACYCH ZA SLOWEM 'SEGMENTS'
   | ('SM') LUB 'CLASSES' ('CS').
   | WE: segs = TRUE - DEKLARACJE SEGMENTOW
   |     css  = TRUE - DEKLARACJE KLAS
   |     namesm = ZMIENNA ZAPISYWANA JAKO NAZWA SEGMENTU
   |     namecs = ZMIENNA ZAPISYWANA JAKO NAZWA KLASY
   |              (PROCEDURA ODCZYTUJE NAZWE Z LINII WYWOLANIA
   |               DO ZMIENNEJ nameabs)
   |     tabs = TYP ZLECENIA ODNOSZACY SIE DO TWORZONYCH (UZUPELNIANYCH)
   |            POZYCJI
   |     nrerr = NUMER BLEDU Z JAKIM NALEZY WYJSC Z PROCEDURY
   |             GDY ADRES DLA SEGMENTU (KLASY) JEST PODANY WIELOKROTNIE
   *)
  PROCEDURE setabs(segs, css: boolean; VAR namesm, namecs: nazwa;
                   tabs: order; nrerr: integer);
  VAR
    same: boolean;
```

```
    curptr: posptr;
   BEGIN
     curchar:=curchar-1;
     REPEAT
      curchar:=curchar+1;
      copy(nameabs); /* NAZWA SEGMENTU (KLASY) */
      chctrl('('); /* ANALIZA ZAPISU ADRESU */
      number;
      IF over
       THEN skok1(7);
      chctrl(')');
      curptr:=s0clist; /* PRZESZUKAJ JUZ ZAPISANE SEGMENTY (KLASY) */
      same:=false;
      WHILE NOT same AND (curptr^.ptr()NIL) DO
      WITH curptr^ DO
      IF (segs AND (seg=namesm)) OR (css AND (kl=namecs))
       THEN
       same:=true
       ELSE
       curptr:=ptr;
      WITH curptr ^ DO
      IF same
       THEN /* ADRES PODANY DLA ZAPISANEGO SEGMENTU (KLASY) */
        BEGIN
          IF aabs()no
           THEN /* ZAPISANA POZYCJA JEST ABSOLUTNA */
            BEGIN
              error1:=nameabs;
              skok1(nrerr)
            END
           ELSE /* ZAPISZ ADRES DLA POZYCJI RELOKOWALNEJ */
            BEGIN
              aabs:=tabs;
              ap.par:=numbin(5,2);
              ap.off:=num[1];
            END
        END
       ELSE /* ADRES PODANY DLA NIEZNANEGO SEGMENTU (KLASY) */
       sets0c(namesm, namecs, no, tabs, numbin(5,2), num[1], 0, 0, '@')
     UNTIL line(curchar) () ',';
    chctrl(')')
   END;
  /* SETABS */

  /*************** POCZATEK PROCEDURY ADDRESSES ***************/
 BEGIN
   chctrl('(');
   curchar:=curchar-1;
    REPEAT
     curchar:=curchar+1;
     IF segments
      THEN /* TEKST ZA SL KLUCZ 'SEGMENTS' */
      setabs (true, false, nameabs, txtnull, sm, 44)
      ELSE
      IF classes
       THEN /* TEKST ZA SL KLUCZ 'CLASSES' */
       setabs (false, true, txtnull, nameabs, cs, 43)
       ELSE /* BLAD SLOWA KLUCZOWEGO */
       skok1(6)
    UNTIL line (curchar) () ',';
   chctrl(')')
 END;
/* ADDRESSES */

     (*****************************************)
     (*           RESERVE                     *)
     (*****************************************)
```

```
(*
 * PROCEDURA ANALIZUJACA TEKST CONTROL RESERVE. SPRAWDZA POPRAWNOSC
 * SYNTAKTYCZNA - BLAD POWODUJE WYJSCIE Z BLEDEM NR 2. ZAPISUJE POZYCJE
 * PRZEDSTAWIAJACE ZAREZERWOWANE POLA W TABLICY  s0clist. DLA TAKICH
 * POZYCJI:
 * ap.par = PARAGRAF ADRESU POCZATKOWEGO POLA
 * ap.off = OFFSET ADRESU POCZATKOWEGO POLA
 * dl  = PARAGRAF ADRESU KONCOWEGO POLA
 * ssv = OFFSET ADRESU KONCOWEGO POLA
 *)
PROCEDURE reserve;
VAR
  low, high: adres;
 BEGIN
   chctrl('(');
   curchar:=curchar-1;
    REPEAT
     curchar:=curchar+1;
     sep:=sep+['T']; /* PO 1-SZYM ADRESIE NASTEPUJE 'TO' */
     number; /* ANALIZA 1-GO ADRESU */
     sep:=sep-['T'];
     IF over
      THEN skok1(7);
     low.par:=numbin(5,2);
     low.off:=num[1];
     chctrl('T'); /* SLOWO 'TO' */
     chctrl('O');
     number; /* ANALIZA 2-GO ADRESU */
     IF over
      THEN skok1(7);
     high.par:=numbin(5,2);
     high.off:=num[1];
     IF gt( low, high )
      THEN /* 2-GI ADRES < 1-SZY ADRES */
      skok1(24);
     sets0c (txtnull, txtnull, no, res, low.par, low.off,
             high.par, high.off, '@')
    UNTIL line (curchar) <> '(';
   chctrl(')')
 END;
/* RESERVE */

      (*****************************************)
      (*           BOOTSTRAP                   *)
      (*****************************************)

(*
 * PROCEDURA OBSLUGUJACA CONTROL 'bootstrap'. ZAPISUJE POZYCJE
 * ODPOWIADAJACA SEGMENTOWI Z ROZKAZEM SKOKU DO STARTU PROGRAMU
 * I USTAWIA ZMIENNA GLOBALNA startboot.
 *)
PROCEDURE bootstrap;
 BEGIN
   startboot:=true;
   sets0c (txtnull, txtnull, no, boot, 177777B, 0, 4, 0, '@')
 END;
(*BOOTSTRAP     *)

      (*****************************************)
      (*           START                       *)
      (*****************************************)

(*
 * PROCEDURA ANALIZUJE TEKST ZA SL. KLUCZ. start. SPRAWDZ POPRAWNOSC
 * SYNTAKTYCZNA. ZAPISUJE DANE O ADRESIE STARTOWYM DO ZMIENNYCH GLOBALNYCH
 * WE: curchar = NR PIERWSZEGO ZNAKU TEKSTU
 * WY: startctrl = OBECNOSC PODANEGO CONTROLEM ADRESU STARTOWEGO
 *     startpub  = NAZWA ZMIENNEJ PUBLIC GDY MA BYC ONA ADRESEM STARTOWYM
```

```
 *                LUB startpub[1] = ' ' GDY ADRES JEST PODANY EXPLICITE
 *     startadr  = ADRES STARTOWY (GDY PODANY EXPLICITE)
 *     curchar   = NR PIERWSZEGO ZNAKU ZA TEKSTEM
 *)
PROCEDURE start;
 BEGIN
   chctrl ('(');
   IF line(curchar) IN ['0'..'9']
    THEN /* ADRES STARTOWY PODANY EXPLICITE */
     BEGIN
       number; /* ANALIZA BASE */
       IF over OR (num[5]<>0)
        THEN
        skok1(9);
       startadr.par:=numbin(4,1);
       chctrl(',');
       number; /* ANALIZA OFFSET-U */
       IF over OR (num[5]<>0)
        THEN
        skok1(9);
       startadr.off:=numbin(4,1);
       startpub:=txtnull; /* BRAK NAZWY ADRESU STARTOWEGO */
     END
    ELSE
    copy(startpub); /* NAZWA ADRESU STARTOWEGO */
   chctrl(')');
   startctrl:=true;
 END;
(*      START   *)

      (*****************************************)
      (*           SEGSIZE                     *)
      (*****************************************)

(*
 * PROCEDURA ANALICUJACA CONTROL segsize. SPRAWDZA POPRAWNOSC SYNTAKTYCZNA
 * DANE Z CONTROL ZAPISUJE DO TABLICY POZYCJI. MODYFIKUJE POZYCJE JUZ
 * ZAPISANA ( JEZELI MODYFIKACJA DOTYCZY SEGMENTU WYMIENIONEGO W LINII
 * WYWOLANIA) LUB TWORZY NOWA POZYCJE.
 *)
PROCEDURE segsize;
VAR
  name: nazwa;
  chng: char;
  same: boolean;
  curptr: posptr;
 BEGIN
   chctrl('(');
   curchar:=curchar-1;
    REPEAT
     curchar:=curchar+1;
     copy(name); /* NAZWA SEGMENTU */
     chctrl('(');
     IF line(curchar) IN ['+','-'] /* ANALIZA TYPU OKRESLENIA SEGSIZE */
      THEN
       BEGIN
         chng:=line(curchar);
         curchar:=curchar+1
       END
      ELSE
      chng:=' ';
     number; /* ANALIZA WARTOSCI DLA SEGSIZE */
     IF over OR (num[5]<>0)
      THEN
      skok1(9);
     chctrl(')');
     same:=false; /* SZUKAJ SEGMENTU POMIEDZY JUZ ZAPISANYMI */
     curptr:=s0clist;
```

```
        WHILE curptr^.ptr()NIL DO
        WITH curptr^ DO
         BEGIN
           IF name=seg
            THEN /* SEGSIZE ODNOSI SIE DO ZAPISANEGO WCZESNIEJ SEGM */
             BEGIN
               ssc:=chng;
               ssv:=numbin(4,1);
               same:=true;
               EXIT
             END;
           curptr:=ptr
         END;
        IF NOT same /* SEGSIZE DLA NIEZNANEGO SEGMENTU */
         THEN
         sets0c (name, txtnull, no, no, 0, 0, 0, numbin(4,1), chng)
       UNTIL line(curchar)()',';
      chctrl(')')
    END;
   (*      SEGSIZE          *)

        (****************************************)
        (*           REMEMBER                   *)
        (****************************************)

   (*
    * PROCEDURA ZAPAMIETYWANIA POCZATKU I OMIJANIA TEKSTU CONTROLA
    * PRZEZNACZONEGO DO POZNIEJSCEJ ANALIZY
    * WY: charno = NUMER 1-GO ZNAKU OMIJANEGO TEKSTU
    *)
   PROCEDURE remember(VAR charno: integer);
    BEGIN
      charno:=curchar;
      WHILE NOT (line(curchar) IN ['/', '%']) DO
      curchar:=curchar+1;
    END;

   (*+++++++++++++POCZATEK PROCEDURY DYREKTYWA++++++++++++++++++++++++++++++*)
  BEGIN
    (*      USTAWIENIE WARTOSCI POCZATKOWYCH *)
    sep:=[',', '/', '=', ')', '(', '%'];
    listing:=false;
    startboot:=false;
    startctrl:=false;
    FOR i:=1 TO dls DO
    txtnull[i]:=' ';
    plwy:=txtnull;
    (* ODCZYT LINII DYREKTYWY *)
    czytaj;
    curchar:=0;
    (* ANALIZA DEKLARACJI PLIKOW W WIERSZU DYREKTYWY  *)
    IF NOT (line(curchar) IN ['=', ','])
     THEN
     (* ANALIZA PLIKU WY  *)
     IF NOT plik(plwy, 'T')
      THEN
      skok1(48);
    IF line(curchar)=','
     THEN
      BEGIN       (* ANALIZA PLIKU LST  *)
        curchar:=curchar+1;
        IF NOT plik(pllst, 'M')
         THEN
         skok1(48);
        listing:=true;
      END;
    IF line(curchar)()'='
     THEN
```

```
 skok1(48);
curchar:=curchar+1;
(* ANALIZA PIERWSZEGO PLIKU WE   *)
new(obiekty);
filptr:=obiekty;
filptr^.next:=NIL;
IF NOT plik(filptr^.spec, 'O')
 THEN
 skok1(3);
 (*  OKRESL PLIK WY GDY NIE BYL WYSPECYFIKOWANY EXPLICITE    *)
IF plwy = txtnull
 THEN
  BEGIN
    i:=0;
     REPEAT
      i:=i+1;
      plwy[i]:=filptr^.spec[i]
     UNTIL plwy[i]='.';
    plwy[i+1]:='T';
    plwy[i+2]:='8';
    plwy[i+3]:='6';
  END;
WHILE line(curchar)=',' DO
 BEGIN (* ANALIZA NASTEPNYCH PLIKOW WE  *)
   curchar:=curchar+1;
   new(filptr^.next);
   filptr:=filptr^.next;
   filptr^.next:=NIL;
   IF NOT plik(filptr^.spec, 'O')
    THEN
    skok1(48);
 END;
(*  ANALIZA CONTROLS  *)
charaddr:=0;
charsegs:=0;
new(s0clist); /* INICJALIZUJ LISTE POZYCJI SEGMENTOW I KLAS */
curptr:=s0clist;
WITH curptr^ DO
 BEGIN
   ptr:=NIL;
   kl:=txtnull;
   seg:=txtnull;
 END;
WHILE line(curchar)='/' DO
 BEGIN /* ANALIZUJ KOLEJNE SLOWA CONTROLNE */
   curchar:=curchar+1;
   FOR i:=0 TO 8 DO namecon[i]:=' ';
   FOR i:=0 TO 8 DO
    BEGIN /* PRZEPISZ SLOWO KONTROLNE */
      IF line(curchar) IN sep
       THEN EXIT;
      namecon[i]:=line(curchar);
      curchar:=curchar+1
    END;
   IF (namecon='ADDRESSES') OR (namecon='AD       ')
    THEN remember (charaddr)
    ELSE
    IF (namecon='ORDER    ') OR (namecon='OD       ')
     THEN conorder .
     ELSE
     IF (namecon='RESERVE  ') OR (namecon='RS       ')
      THEN reserve
      ELSE
      IF (namecon='SEGSIZE  ') OR (namecon='SS       ')
       THEN remember (charsegs)
       ELSE
       IF (namecon='BOOTSTRAP') OR (namecon='BS       ')
        THEN bootstrap
```

```
            ELSE
            IF (namecon='START    ') OR (namecon='ST       ')
             THEN start
             ELSE skok1(2)
     END;
    IF line(curchar)<>'%' /* ZNAKI NIEPRAWIDLOWE */
     THEN
     skok1(48);
    IF charaddr<>0 /* ANALIZA CONTROL ADDRESSES PO INNYCH (PO ORDER) */
     THEN
      BEGIN
        curchar:=charaddr;
        addresses;
        IF NOT (line(curchar) IN ['/', '%'])
         THEN
         skok1(2)
      END;
    IF charsegs<>0 /* ANALIZA CONTROL SEGSIZE PO ORDER I PO ADDRESSES */
     THEN
      BEGIN
        curchar:=charsegs;
        segsize;
        IF NOT (line(curchar) IN ['/', '%'])
         THEN
         skok1(2)
      END;
 END;
(* DYREKTYWA  *)
```

```
PROCEDURE skok1(nrerror: integer); EXTERNAL;
FUNCTION gt(VAR adres1, adres2: adres): boolean; EXTERNAL;
(**E+*)

        (*************************************)
        (*           INVOCATIONCHECK           *)
        (*************************************)

/*
 + PROCEDURA KONTROLUJE POPRAWNOSC DANYCH W LINII WYWOLANIA PROGRAMU.
 + SPRAWDZA UNIKALNOSC NAZW KLAS I SEGMENTOW ORAZ MONOTONICZNOSC ADRESOW
 + DLA SEGMENTOW (KLAS) O OKRESLONEJ KOLEJNOSCI LOKOWANIA (order).
 */
PROCEDURE invocationcheck;
VAR
  curaddr: adres;
  currentptr, baseptr: posptr;
  smcstype, nameseg, nameclass: boolean;
 BEGIN
   currentptr:=s0clist; /* KONTROLA MONOTONICZNOSCI ADRESOW */
   curaddr.par:=0;
   curaddr.off:=0;
   WHILE currentptr^.ptr<>NIL DO
   WITH currentptr^ DO
   WITH curaddr DO
    BEGIN
      IF (aabs<>no) AND (zord<>no)
       THEN
       IF gt( curaddr, ap)
        THEN
         BEGIN
           error1:=seg;
           error2:=kl;
           IF aabs=sm
            THEN skok1(42)
            ELSE skok1(41)
         END
        ELSE
        curaddr:=ap;
     currentptr:=ptr
    END;
   baseptr:=s0clist; /* KONTROLA UNIKALNOSCI NAZW */
   WHILE baseptr^.ptr<>NIL DO
    BEGIN
      WITH baseptr^ DO
       BEGIN
         error1:=seg;
         error2:=kl;
         nameseg:=error1[1]<>' ';
         nameclass:=error2[1]<>' ';
         smcstype:=zord=smcs;
         baseptr:=ptr;
         currentptr:=ptr;
       END;
      WHILE currentptr^.ptr<>NIL DO
      WITH currentptr^ DO
       BEGIN
         smcstype:=smcstype AND (zord=smcs) AND (error2=kl);
         IF nameseg AND (error1=seg)
          THEN skok1(45);
         IF NOT smcstype AND nameclass AND (error2=kl)
          THEN skok1(46);
         currentptr:=ptr
       END
    END
 END;
/* INVOCATIONCHECK */
```

```
(**E+*)
PROCEDURE errorcom(i: integer); EXTERNAL;
FUNCTION inc(VAR ch: char; VAR sum: lonum): integer; EXTERNAL;
FUNCTION equ(VAR adres1, adres2: adres): boolean; EXTERNAL;
PROCEDURE longrec(VAR dlr,sum: lonum); EXTERNAL;
PROCEDURE rewrec(VAR sum: lonum); EXTERNAL;
PROCEDURE skok1(nrerror: integer); EXTERNAL;

        (*************************************)
        (*           MODUL                   *)
        (*************************************)

(*
 + PROCEDURA ODCZYTUJE MODUL WEJSCIOWY. SEGMENTY Z MODULU SA WSTEPNIE LINKO-
 + WANE I DOPISYWANE DO TABLICY  segwe. EXTERNALE I PUBLICI SA DOPISYWANE DO
 + TABLICY glob. POPRAWNOSC MODULU WE JEST FRAGMENTARYCZNIE SPRAWDZANA.
 + WE: nrmodwe = NUMER KOLEJNY MODULU WE
 +     we      = PLIK WE USTAWIONY NA POCZATEK
 +     plik    = NAZWA PLIKU WE
 *)
PROCEDURE modul;
VAR
  ch: char;
  i, sum, dlr: lonum;
  ir, iz: integer;
  nz: nazwa;
  idnt: ARRAY[2..ms] OF nazwa;
  glptr: globalptr;

        (*************************************)
        (*           INNAME                  *)
        (*************************************)

  (*
   * PROCEDURA ODCZYTUJE I SPRAWDZA POPRAWNOSC NAZWY Z MODULU we. MODYFIKUJE
   * sum I i (NR ZNAKU W REKORDZIE).GDY EOF - BLAD.
   * WE: WSK we = 1-SZY BAJT NAZWY
   * WY: WSK we = 1-SZY BAJT ZA NAZWA
   *     nz = NAZWA, OGRANICZONA DO dls ZNAKOW, EW. UZUPELNIONA SPACJAMI
   *)
  PROCEDURE inname;
  VAR
    k, j, m: integer;
   BEGIN
     k:=inc(ch,sum);          /* DLUGOSC NAZWY */
     i:=i+1;
     FOR j:=1 TO k DO /* ODCZYT NAZWY */
      BEGIN
        m:=inc(ch,sum);
        IF j<=dls
         THEN
         nz[j]:=ch;
        i:=i+1;
      END;
     IF NOT (nz[1] IN ['A'..'Z', '@', '?', '_']) /* KONTROLA POPRAWNOSCI */
      THEN skok1(12);
     FOR j:=k+1 TO dls DO /* UZUPELNIENIE NAZWY SPACJAMI */
     nz[j]:=' ';
   END;
  (*      INNAME  *)

        (*************************************)
        (*           IDENTIFIERS             *)
        (*************************************)

  (*
   * PROCEDURA ODCZYTUJE REKORD NAZW SEGMENTOW I KLAS. SPRAWDZA POPRAWNOSC
```

```
 * REKORDU (SUME KONTROLNA I DLUGOSC.
 * WE: WSK we NA 1-SZY BAJT ILOSCI ZNAKOW REKORDU
 *     sum = SUMA KONTROLNA PO ZNAKU TYPU REKORDU
 * WY: WSK we NA 1-SZYM BAJCIE NAST REKORDU
 *     idnt = TABLICA NAZW
 *)
PROCEDURE identifiers;
VAR
  n: integer;
 BEGIN
   longrec(dlr,sum);
   IF inc(ch,sum)<>0 /* 1-SZY ZNAK REKORDU = 0 */
    THEN skok1(12);
   i:=2;
   n:=2;
   WHILE i<dlr-1 DO /* ODCZYT KOLEJNYCH NAZW */
    BEGIN
      inname;
      IF n>ms
       THEN skok1(10);
      idnt[n]:=nz; /* NAZWA DO TABLICY idnt */
      n:=n+1
    END;
   n:=inc(ch,sum); /* SUMA KONTROLNA */
   IF (i<>dlr) OR (sum<>0)
    THEN skok1(12);
 END;
(*      IDENTIFIERS     *)

      (*****************************************)
      (*           SEGMENT                     *)
      (*****************************************)

(*
 * PROCEDURA ODCZYTUJE REKORD DANYCH SEGMENTU. OTRZYMANY SEGMENT WSTEPNIE
 * LINKUJE Z ODPOWIEDNIM SEGMENTEM Z TABLICY segwe (JEZELI TAKI ISTNIEJE)
 * LINKOWANIE: WSTAWIENIE POZYCJI LINKOWANEGO SEGMENTU ZA POZYCJA SEGMENTU
 * DO KTOREGO JEST LINKOWANY, Z seg[1]='*'.
 * JEZELI SEGMENT NIE JEST LINKOWANY JEGO POZYCJA JEST DOPISYWANA
 * DO TABLICY s0clist NA POZYCJE OKRESLONA WG CONTROLS LUB WG KLASY.
 * DLA WSTAWIANEJ POZYCJI nug:=nrmodwe I nus:=ir.
 * KONTROLA REKORDU we, SUMA KONTROLNA I WSK we - TAK JAK DLA identifiers.
 *)
PROCEDURE segment;
LABEL
  2;
VAR
  rex: pozycja;
  curptr, ssptr, lastptr, classptr: posptr;

     (*****************************************)
     (*           TAKENAME                    *)
     (*****************************************)

  (*
   | PROCEDURA OKRESLA NAZWE SEGMENTU (KLASY) NA PODSTAWIE JEJ NUMERU
   | I ZAPISUJE DO POZYCJI SEGMENTU W TABLICY s0clist. KASUJE ZAPIS NAZWY
   | W TABLICY idnt
   *)
  PROCEDURE takename(VAR name: nazwa);
   BEGIN
     IF NOT (iz IN [2..ms])
      THEN skok1(12);
     name:=idnt[iz];
     idnt[iz]:=nz;
   END;
  /* TAKENAME */
```

```
    (*************************************)
    (*           INSEGMENT               *)
    (*************************************)

(*
 | PROCEDURA ODCZYTUJE REKORD DANYCH SEGMENTU. DANE TE SA ZAPISYWANE
 | DO rex.
 KONTROLA REKORDU we, SUMA KONTROLNA I WSK we - TAK JAK DLA identifiers.
 *)
PROCEDURE insegment;
 BEGIN
   longrec(dlr,sum);
   FOR iz:=1 TO dls DO
   nz[iz]:=' ';
   WITH rex DO
    BEGIN
      typs:=inc(ch,sum); /* TYP SEGMENTU */
      IF typs AND at = atat /* ADRES SEGMENTU ABSOLUTNEGO */
       THEN
        BEGIN
          longrec(ap.par,sum);
          ap.off:=0;
          IF (inc(ch,sum)<>0) OR (dlr<>10)
           THEN skok1(12);
          aabs:=sm;
        END
       ELSE
        BEGIN
          ap.par:=0;
          ap.off:=0;
          aabs:=no;
          IF dlr<>7
           THEN skok1(12)
        END;
      longrec(dl, sum); /* DLUGOSC SEGMENTU */
      iz:=inc(ch,sum);
      takename(rex.seg); /* NAZWA SEGMENTU */
      iz:=inc(ch,sum);
      IF iz<>1
       THEN takename(rex.kl) /* NAZWA KLASY */
       ELSE
       kl:=nz;
      i:=inc(ch, sum); /* '1' W REKORDZIE */
      iz:=inc(ch, sum); /* SUMA KONTROLNA */
      IF (i<>1) OR (sum<>0)
       THEN skok1(12);
      IF (seg='??SEG                              ') AND (dl=0)
       THEN
       GOTO 2;
      zord:=no;
      pusty:=false;
      nus:=ir;
      nug:=nrmodwe;
      ssc:='@'
    END;
 END;
/* INSEGMENT */

    (*************************************)
    (*           TABSEGM                 *)
    (*************************************)

/*
 | PROCEDURA ZAPISUJE DANE Z POZYCJI rex DO TABLICY s0clist
 | UZUPELNIAJAC ODPOWIEDNIA POZYCJE LUB TWORZAC NOWA
 */
PROCEDURE tabsegm;
LABEL
```

```
  3;
VAR
  cursegs: boolean;

    (*****************************************)
    (*           LINKSM                      *)
    (*****************************************)

  (*
   : PROCEDURA LINKUJE SEGMENT Z we Z SEGMENTEM O TYCH SAMYCH seg I kl.
   : SPRAWDZA ZGODNOSC C.T., BRAK ZGODNOSCI - BLAD PRZERYWAJACY
   : LINKOWANIE: WSTAWIENIE POZYCJI LINKOWANEGO SEGMENTU ZA POZYCJA
   : SEGMENTU DO KTOREGO JEST LINKOWANY Z seg[1]='*'.
   *)
  PROCEDURE linksm;
  VAR
    ctrex, ctrob: integer;
    la /* NIEMOZNOSC LINKOWANIA SEGMENTOW JAKO COMMON Z MEMORY */,
    lb /* JEDEN Z SEGMENTOW ABSOLUTNY */ : boolean;
   BEGIN
     WITH curptr^ DO
     IF kl=rex.kl
      THEN
       BEGIN
         /* SPRAWDZ POPRAWNOSC C.T. I A.T. */
         ctrex:=rex.typs AND ct;
         ctrob:=typs AND ct;
         la:=NOT memory OR (ctrex()ctmem) AND (ctrex()ctcom);
         la:=la OR (ctrob()ctcom) AND ((ctrob()ctmem) OR (zord=mem));
         lb:=(typs AND at=atat) OR (rex.typs AND at=atat);
         IF (ctrex()ctrob) AND la OR lb OR (ctrex=ctno)
          THEN
           BEGIN
             error2:=rex.seg;
             error3:=rex.kl;
             skok1(11);
           END;
          /* WSTAW POZYCJE REX ZA POZYCJE ROB I KONTYNUACJE
           DO LISTY SEGMENTOW */
         WHILE (curptr^.ptr()NIL) AND (curptr^.ptr^.seg[1]='*') DO
         curptr:=curptr^.ptr;
         rex.ptr:=curptr^.ptr;
         rex.seg[1]:='*';
         new(curptr^.ptr);
         curptr^.ptr^:=rex;
         GOTO 2
       END
   END;
  (*      LINKSM          *)

    (*****************************************)
    (*           LOADPOZSM                   *)
    (*****************************************)

  (*
   : PROCEDURA WYPELNIA POZYCJE Z DYREKTYWY OD(AD) SM(SMCS) DANYMI
   : SEGMENTU Z we. GDY SEGMENT MA PODANE 2 ROZNE ADRESY ABSOLUTNE - BLAD.
   *)
  PROCEDURE loadpozsm;
   BEGIN
     WITH curptr^ DO
      BEGIN
        IF (kl()rex.kl) AND (kl[1]()' ')
         THEN skok1(47); /* NIEZGODNOSC KLAS */
        IF rex.aabs=sm /* SPRAWDZ ZGODNOSC ADRESOW */
         THEN
         IF ((aabs=sm)OR((zord=smcs)AND(aabs=cs)))AND NOT equ(rex.ap, ap)
          THEN /* 2 ADRESY PODANE DLA SEGMENTU ABSOLUTNEGO */
```

```
            BEGIN
              error1:=seg;
              skok1(27)
            END
           ELSE
            BEGIN
              aabs:=sm;
              ap:=rex.ap;
            END;
        nus:=rex.nus; /* WYPELNIJ POZYCJE curptr */
        nug:=rex.nug;
        kl:=rex.kl;
        typs:=rex.typs;
        pusty:=false;
        dl:=rex.dl;
        GOTO 2
      END;
 END;
(*      LOADPOZSM     *)

  (*************************************)
  (*           POZYSS                  *)
  (*************************************)

(*
 : PROCEDURA OBSLUGI POZYCJI SEGSIZE. POZYCJA JEST USUWANA Z TABLICY I
 : UZUPELNIANA DANYMI Z we.
 *)
PROCEDURE pozyss;
 BEGIN
   IF lastptr=NIL
    THEN
    s0clist:=s0clist^.ptr
    ELSE
    lastptr^.ptr:=curptr^.ptr;
   ssptr:=curptr;
   curptr:=curptr^.ptr;
   WITH ssptr^ DO
    BEGIN
      rex.ssc:=ssc;
      rex.ssv:=ssv;
    END;
   ssptr^:=rex;
   GOTO 3
 END;
(*      POZySS  *)

 /*------------------- POCZATEK PROCEDURY TABSEGM ----------------------*
BEGIN
  curptr:=s0clist;
  ssptr:=NIL;
  lastptr:=NIL;
  classptr:=NIL;
  WHILE curptr^.ptr<>NIL DO
  WITH curptr^ DO
   BEGIN
     cursegs:=pusty AND (zord=no) AND (aabs=no);
     IF (seg[1]='*') AND (classptr=lastptr)
      THEN          (* OMIJANIE SEGMENTOW LINKOWANYCH */
      classptr:=curptr
      ELSE
      IF rex.seg=seg
       THEN
        BEGIN
          IF cursegs
           THEN
           pozyss  /* POBIERANIE POZYCJI SEGSIZE */
           ELSE
```

```
                    IF pusty
                     THEN
                     loadpozsm /* WYPELNIJ POZYCJE NA SEGMENT */
                     ELSE
                     linksm /* LINKUJ SEGMENT DO JUZ ZAPISANEGO */
                   END
                  ELSE
                   IF NOT cursegs AND (rex.aabs<>sm) AND (rex.kl=kl)
                    AND (zord IN [smcs,cs,no]) AND (aabs IN [cs,no])
                    THEN /* USTALANIE POZYCJI KLASY SEGMENTU RELOKOWALNEGO */
                    classptr:=curptr;
                  lastptr:=curptr;
                  curptr:=ptr;
3:
                END;
               IF (memory OR (rex.typs AND ct<>ctmem)) AND (classptr<>NIL)
                THEN
                WITH classptr^ DO
                IF pusty AND (zord<>smcs)
                 THEN
                  BEGIN
                    /* LADOWANIE SEGMENTU RELOKOWALNEGO NA PUSTA POZYCJE KLASY */
                    nus:=rex.nus;
                    nug:=rex.nug;
                    seg:=rex.seg;
                    typs:=rex.typs;
                    pusty:=false;
                    dl:=rex.dl
                  END
                 ELSE
                 IF ssptr<>NIL
                  THEN
                   BEGIN
                     /* LADOWANIE WCZESNIEJ POBRANEJ I UZUPELNIONEJ
                      POZYCJI SEGSIZE NA KONCU KLASY */
                     ssptr^.zord:=zord;
                     ssptr^.aabs:=aabs;
                     ssptr^.ptr:=ptr;
                     classptr^.ptr:=ssptr
                   END
                  ELSE
                   BEGIN
                     /* TWORZENIE NOWEJ POZYCJI NA KONCU KLASY
                      I LADOWANIE TAM SEGMENTU */
                     rex.zord:=zord;
                     rex.aabs:=aabs;
                     rex.ptr:=ptr;
                     new(classptr^.ptr);
                     classptr^.ptr^:=rex
                   END
               ELSE
               IF ssptr<>NIL
                THEN
                 BEGIN
                   /* LADOWANIE NA KONCU TABLICY POBRANEJ WCZESNIEJ
                    I UZUPELNIONEJ POZYCJI SEGSIZE SEGMENTU MEMORY LUB ABSOLUTNEGO
                    LUB SEGMENTU NIE MAJACEGO SWOJEJ KLASY */
                   IF NOT memory AND (rex.typs AND ct=ctmem)
                    THEN
                     BEGIN
                       memory:=true;
                       ssptr^.zord:=mem
                     END;
                   ssptr^.ptr:=curptr;
                   IF lastptr=NIL
                    THEN
                    s0clist:=ssptr
                    ELSE
```

```
                lastptr^.ptr:=ssptr
              END
            ELSE
              BEGIN
                /* TWORZENIE NOWEJ POZYCJI NA KONCU TABLICY I LADOWANIE TAM
                 SEGMENTU ABSOLUTNEGO, MEMORY LUB NIE MAJACEGO SWOJEJ KLASY */
                IF NOT memory AND (rex.typs AND ct=ctmem)
                 THEN
                  BEGIN
                    memory:=true;
                    rex.zord:=mem
                  END;
                rex.ptr:=curptr;
                IF lastptr=NIL
                 THEN
                  BEGIN
                    new(s0clist);
                    s0clist^:=rex
                  END
                 ELSE
                  BEGIN
                    new(lastptr^.ptr);
                    lastptr^.ptr^:=rex
                  END
              END
          END;
         /* TABSEGM */
         /********************* POCZATEK PROCEDURY SEGMENT ********************/
        BEGIN
          insegment;
          IF ir>maxs
           THEN skok1(18);
          tabsegm;
2:
        END;
       (*      SEGMENT  *)

            (****************************************)
            (*           INSTALLNAME                *)
            (****************************************)

       (*
        * PROCEDURA ODCZYTUJE NAZWE GLOBALNA Z REKORDU we. SPRAWDZA, CZY NAZWA
        * JUZ WYSTAPILA , JEZELI NIE DOPISUJE JA DO ROZPROSZONEJ TABLICY glob.
        *)
       PROCEDURE installname;
       VAR
         a: boolean;
         searchptr: globalptr;
        BEGIN
          inname; /* nz:=NAZWA GLOBALNA */
          searchptr:=glob[nz[1]];
          a:=false;
          glptr:=NIL;
          WHILE NOT a AND (searchptr<>NIL) DO /* SZUKAJ NAZWY W glob */
           BEGIN
             a:=nz=searchptr^.etykieta;
             glptr:=searchptr;
             searchptr:=searchptr^.gptr
           END;
          IF NOT a /* DOPISZ NAZWE DO glob */
           THEN
            BEGIN
              IF glptr<>NIL
               THEN
                BEGIN
                  new(glptr^.gptr);
                  glptr:=glptr^.gptr
```

```
          END
         ELSE
          BEGIN
            new(glob[nz[1]]);
            glptr:=glob[nz[1]]
          END;
        WITH glptr^ DO
         BEGIN
           etykieta:=nz;
           public:=false;
           gptr:=NIL
         END
      END;
 END;
/* INSTALLNAME */

      (*****************************************)
      (*           EXTRNS                      *)
      (*****************************************)

(*
 * PROCEDURA ODCZYTUJE REKORD we Z NAZWAMI EXTERNALI. JEZELI DANA NAZWA
 * NIE WYSTEPUJE W TABLICY glob ZOSTAJE DO NIEJ DOPISANA.
 * KONTROLA REKORDU we, SUMA KONTROLNA I WSK we - TAK JAK DLA identifiers.
 *)
PROCEDURE extrns;
VAR
  m: integer;
 BEGIN
   longrec(dlr,sum);
   i:=1;
   m:=1;
   WHILE i<dlr-1 DO
    BEGIN
      installname; /* ODCZY NAZWY EXTERNALA */
      IF m>maxe /* GDY ZA WIELE NAZW (WYMIARY TABL. externals) */
       THEN skok1(29);
      externals[nrmodwe, m]:=glptr;
      m:=m+1;
    END;
   m:=inc(ch,sum); /* KONTROLA DLUGOSCI I SUMY KONTROLNEJ REKORDU */
   IF (i<>dlr) OR (sum<>0)
    THEN skok1(12)
 END;
(*      EXTRNS  *)

      (*****************************************)
      (*           PUBLICS                     *)
      (*****************************************)

(*
 * PROCEDURA ODCZYTUJE REKORD Z NAZWA PUBLIC. JEZELI DANY PUBLIC JUZ BYL W
 * glob - KOMUNIKAT O BLEDZIE I BIEZACA DEKLARACJA JEST IGNOROWANA. JEZELI
 * NAZWA WYSTAPILA JAKO EXTRN - JEJ POZYCJA ZOSTAJE UZUPELNIONA. JEZELI
 *  NAZWA NIE WYSTAPILA - W TABLICY glob ZOSTAJE UTWORZONA JEJ POZYCJA.
 * KONTROLA REKORDU we, SUMA KONTROLNA I WSK we - TAK JAK DLA identifiers.
 *)
PROCEDURE publics;
VAR
  nrsg, jeden: integer;
 BEGIN
   longrec(dlr,sum);
   IF inc(ch,sum)<>0 /* "0" W REKORDZIE */
    THEN skok1(12);
   nrsg:=inc(ch,sum); /* NR SEGMENTU PUBLICA */
   i:=3;
   installname; /* ODCZYT NAZWY PULICA */
   WITH glptr^ DO
```

```
    IF NOT public
     THEN
      BEGIN
        longrec(offset, sum);
        nrg:=nrmodwe;
        nrs:=nrsg;
        public:=true
      END
     ELSE
      BEGIN
        longrec(nrsg, sum);
        error2:=nz;
        errorcom(16)
      END;
    i:=i+3;
    jeden:=inc(ch, sum);
    nrsg:=inc(ch,sum);
    IF (jeden()1) OR (i()dlr) OR (sum()0)
     THEN skok1(12)
 END;
(*      PUBLICS                  *)

      (****************************************)
      (*           MODEND                     *)
      (****************************************)

 (*
  * PROCEDURA ODCZYTUJE REKORD KONCA MODULU.
  *)
 PROCEDURE modend;
  BEGIN
    rewrec(sum);
  END;
 (*      MODEND  *)
 (*++++++++++++++++++++POCZATEK PROCEDURY MODUL++++++++++++++++++++++++++++*)
BEGIN
  sum:=0;
  error1:=plik;
  FOR i:=2 TO ms DO
  idnt[i][1]:=' ';
  iz:=inc(ch,sum);
  IF iz()200B
   THEN skok1(12); (* 80H - REKORD NAZWY MODULU    *)
  rewrec(sum);
  iz:=inc(ch,sum);
  IF iz()210B
   THEN skok1(12); (* 88H - REKORD WERSJI ASM      *)
  rewrec(sum);
  iz:=inc(ch,sum);
  IF iz()226B
   THEN skok1(12); (* 96H - REKORD NAZW SEGMENTOW I KLAS   *)
  identifiers;
  ir:=1;
  iz:=inc(ch,sum);
  IF iz()230B
   THEN        (* 98H - REKORD DANYCH SEGMENTU         *)
   skok1(12);                  (* MUSI BYC CO NAJMNIEJ 1 REKORD 98H    *)
  WHILE iz=230B DO
   BEGIN  (* REKORDY DANYCH SEGMENTOW     *)
     segment;
     ir:=ir+1;
     iz:=inc(ch,sum)
   END;
  FOR ir:=2 TO ms DO      (* CZY WSZYSTKIE NAZWY OPISANE?         *)
  IF idnt[ir][1]()' '
   THEN skok1(12);
  IF iz=216B
   THEN
```

```
      BEGIN   (* 8EH - REKORD NIEZNANY        *)
        rewrec(sum);
        iz:=inc(ch,sum);
      END;
    IF iz=214B
     THEN
      BEGIN   (* 8CH - REKORD EXTERNALI       *)
        extrns;
        iz:=inc(ch,sum);
      END;
    WHILE iz=220B DO
     BEGIN  (* 90H - REKORDY PUBLIC-OW       *)
       publics;
       iz:=inc(ch,sum);
     END;
    WHILE (iz=240B) OR (iz=242B)    (* 0A0H,0A2H - REKORDY DANYCH I DUP *)
    DO
     BEGIN
       rewrec(sum);
       iz:=inc(ch,sum);
       IF iz=234B
        THEN
         BEGIN   (* 9CH - REKORD INF O RELOKOWALNOSCI*)
           rewrec(sum);
           iz:=inc(ch,sum)
         END
     END;
    IF iz<>212B
     THEN        (* 8AH - REKORD KONCA MODULU    *)
     skok1(12);
    modend;
  END;
(*      MODUL   *)
```

```
(*$E+*)
PROCEDURE errorcom(i: integer); EXTERNAL;
PROCEDURE skok1(nrerror: integer); EXTERNAL;
PROCEDURE alitaddr (ad: adres; typ: integer; VAR delta: lonum); EXTERNAL;
FUNCTION sumator (sk1, sk2: lonum; nrerror: integer): lonum; EXTERNAL;

        (*************************************)
        (*           LINKER                  *)
        (*************************************)

(* PROCEDURA SPRAWDZA ZGODNOSC DANYCH Z MODULOW WE Z CONTROLAMI INVOCATION
 + LINE, LINKUJE SEGMENTY ZAPISANE W TABLICY s0clist. WYPELNIA TABLICE
 +  segments. *)

PROCEDURE linker;
VAR
  a, b: boolean;
  curptr, lptr, cptr: posptr;
  c: char;
  glptr: globalptr;

        (*************************************)
        (*           LINK                    *)
        (*************************************)

  (* PROCEDURA OSTATECZNIE LINKUJE (LACZY) SEGMENTY WSTEPNIE LINKOWANE PRZEZ
   * modul. OKRESLA ALIGN TYPE POWSTALEGO SEGMENTU, JEGO DLUGOSC (EW. MODY-
   * FIKOWANA PRZEZ CONTROL SEGSIZE), JEZELI SEGMENT JEST ABSOLUTNY SPRAWDZA
   * ZGODNOSC ADRESU Z ALIGN TYPE, JEZELI BRAK ZGODNOSCI - ADRES SEGMENTU
   * JEST IGNOROWANY I SEGMENT STAJE SIE RELOKOWALNY. *)

  PROCEDURE link;
  VAR
    len, offset: lonum;
    atb, ctb, inp, atc, mat, gap: integer;
    first: boolean;
    curad: adres;
   BEGIN
     curptr:=s0clist;
     lptr:=NIL;
     WHILE curptr^.ptr()NIL DO /* PRZESZUKUJ LISTE I LINKUJ SEGMENTY */
     WITH curptr^ DO
      BEGIN
        error1:=seg;
        error2:=kl;
        cptr:=ptr;
        atb:=typs AND at;
        ctb:=typs AND ct;
        inp:=typs AND ati;

        /* OKRESL ALIGN TYPE LINKOWANEGO SEGMNTU */
        WHILE (cptr^.seg[1]='*') AND (cptr^.ptr()NIL) DO
         BEGIN
           atc:=cptr^.typs AND at;
           inp:=inp OR cptr^.typs AND ati;
           IF atc>atb
            THEN
             BEGIN
               typs:=atc OR ctb;
               atb:=atc
             END;
           cptr:=cptr^.ptr
         END;
        typs:=typs OR inp;
        /* SPRAWDZ ZGODNOSC ALIGN TYPE Z ADRESEM ABSOLUTNYM SEGMENTU */
        IF lptr()NIL
         THEN
```

```
 first:=(lptr^.kl()kl) OR (lptr^.aabs=sm) OR (lptr^.zord=sm)
 ELSE
first:=true;;
IF ((aabs=sm) OR (first AND (aabs=cs))) AND
 (
  ((atb=atpg) AND ((ap.off()0) OR (ap.par AND 17B()0)))
  OR ((atb=atpr) AND (ap.off()0))
  OR ((atb=atwo) AND (ap.off DIV 2 ()0))
  )
 THEN
  BEGIN
    errorcom(49);
    aabs:=no
  END;

 /* OKRESL DLUGOSC SEGMENTU ZLINKOWANEGO */
curad:=ap; /* OKRESL ADRES BIEZACY - ODL OD POCZATKU SEGMENTU */
IF ctb=ctpub
 THEN
 curad.off:=sumator(curad.off, dl, 33);
cptr:=ptr;
WHILE (cptr^.seg[1]='*') AND (cptr^.ptr()NIL) DO
 BEGIN
   IF ctb =ctpub
    THEN
     BEGIN
       alitaddr (curad, cptr^.typs, gap);
       curad.off:=sumator(curad.off, gap, 33);
       cptr^.ap:=curad;
       curad.off:=sumator(curad.off, cptr^.dl, 33)
     END
    ELSE
     BEGIN
       IF ctb=ctsta
        THEN
        dl:=sumator(dl, cptr^.dl, 33)
        ELSE
        IF dl(cptr^.dl
         THEN
         dl:=cptr^.dl;
       cptr^.ap:=curad;
     END;
   cptr:=cptr^.ptr
 END;
IF ctb=ctpub
 THEN
 dl:=curad.off-ap.off;

 /* MODYFIKUJ DLUGOSC SEGMENTU O PODANE SEGSIZE */
len:=dl; /* PIERWOTNA DLUGOSC */
CASE ssc OF
  ' ':
  dl:=ssv;
  '+':
  dl:=sumator(dl, ssv, 22);
  '-':
  IF dl-ssv)dl
   THEN skok1(23)
   ELSE dl:=dl-ssv;
 END;
IF len)dl /* SYGNALIZACJA ZMNIEJSZENIA DLUGOSCI SEGMENTU */
 THEN
 errorcom(26);

 /* KONTROLA SEGMENTU INPAGE */
IF (inp=ati) AND (dl)256)
 THEN
  BEGIN
```

```
            errorcom(28);
            typs:=typs AND atinot
          END;
        curptr:=cptr
      END
  END;
 (*      LINK    *)

 (*+++++++++++++++++POCZATEK PROCEDURY LINKER++++++++++++++++++++++++++++++*)
BEGIN
  a:=false;       (* KONTROLA OBECNOSCI SPECYFIKOWANYCH SEGMENTOW I KLAS
                     ORAZ ILOSCI SEGMENTOW MEMORY    *)
  curptr:=s0clist;
  WHILE curptr^.ptr<>NIL DO
  WITH curptr^ DO
   BEGIN
     error1:=seg;
     error2:=kl;
     IF pusty
      THEN
      IF seg[1]<>' '
       THEN
       skok1(35)
       ELSE
       skok1(34);
     b:=(typs AND ct=ctmem) AND (seg[1]<>'*');
     IF a AND b
      THEN
      errorcom(13);
     a:=a OR b;
     curptr:=ptr
   END;

  a:=false;       (* KONTROLA ROZWIAZANIA EXTERNALI       *)
  FOR c:='?' TO '_' DO
   BEGIN
     glptr:=glob[c];
     WHILE glptr<>NIL DO
     WITH glptr^ DO
      BEGIN
        IF NOT public
         THEN
          BEGIN
            nrg:=1;
            nrs:=0;
            offset:=0;
            a:=true;
          END;
        glptr:=gptr
      END
   END;
  IF a
   THEN
    BEGIN
      errorcom(32);
      new (segments [1, 0]);
      segments [1, 0]^.ap.par:=0;
      segments [1, 0]^.ap.off:=0;
    END;

  link; /* LINKUJ SEGMENTY */

  curptr:=s0clist;   (* UTWORZ TABLICY WSKAZNIKOW DO SEGMENTOW segments *)
  WHILE curptr^.ptr<>NIL DO WITH curptr^ DO
   BEGIN
     segments[nug, nus]:=curptr;
     curptr:=ptr
   END
```

```
END;
(*      LINKER  *)
```

```
(*$E+*)
FUNCTION equ (VAR adres1, adres2: adres): boolean; EXTERNAL;
FUNCTION gt (VAR adres1, adres2: adres): boolean; EXTERNAL;
FUNCTION shiftr (x, c: integer): integer; EXTERNAL;
FUNCTION sumator (sk1, sk2: lonum; nrerror: integer): lonum; EXTERNAL;
PROCEDURE alitaddr (ad: adres; typ: integer; VAR delta:lonum); EXTERNAL;
PROCEDURE skok1 (nrerror: integer); EXTERNAL;
PROCEDURE errorcom (nrerror: integer); EXTERNAL;

        (*************************************)
        (*           EOS                     *)
        (*************************************)

/*
 * PROCEDURA OBLICZA ADRES KONCA SEGMENTU (ADRES OSTATNIEGO BAJTU CIALA
 * SEGMENTU.
 * WE: stad: ADRES POCZATKU SEGMENTU (BASE I OFFSET)
 *     len:  DLUGOSC SEGMENTU
 * WY: spad: ADRES OSTATNIEGO BAJTU SEGMENTU (BASE I OFFSET)
 * GDY OBSZAR SEGMENTU PRZEKRACZA OBSZAR ADRESOWY 8086 WYJSCIE
 * PROCEDURY DO ETYKIETY 1 W MAIN
 */
PROCEDURE eos (stad: adres; len: lonum; VAR spad: adres);
VAR
  ls, lo: integer;
 BEGIN
   IF len=0
    THEN
    spad:=stad
    ELSE
     BEGIN
       len:=len-1;
       ls:=shiftr(len,4);
       lo:=len AND 17B+stad.off;
       spad.par:=stad.par+ls+lo DIV 16;
       IF spad.par<stad.par
        THEN
        skok1(37);
       spad.off:=lo MOD 16
     END
 END;
/*      EOS     */

        (*************************************)
        (*           ADADDR                  *)
        (*************************************)

/* PROCEDURA ZWIEKSZA DANY ADRES O ZADANA WIELKOSC.
 * SPRAWDZA CZY OTRZYMANY ADRES MIESCI SIE W OBSZARZE ADRESOWYM 8086.
 * WE: adr - DANY ADRES
 *     delta - WARTOSC ZWIEKSZENIA
 * WY: adw - OTRZYMANY ADRES
 */
PROCEDURE adaddr (adr: adres; delta: lonum; VAR adw: adres);
 BEGIN
   IF delta<>0
    THEN
    delta:=delta+1;
  eos (adr, delta, adw)
 END;
/* ADADDR */

        (*************************************)
        (*           FINDEND                 *)
        (*************************************)

/*
```

```
 * OKRESL ADRES KONCOWY POZYCJI ULOKOWANEJ
 * WE: cptr = WSK POZYCJI
 * WY: ak   = ADRES KONCOWY POZYCJI
 */
PROCEDURE findend (cptr: posptr; VAR ak: adres);
 BEGIN
   IF cptr^.aabs=res
    THEN
     BEGIN
       ak.par:=cptr^.dl;
       ak.off:=cptr^.ssv
     END
    ELSE
    eos(cptr^.ap, cptr^.dl, ak);
 END;
/* FINDEND */

        (*************************************)
        (*           MOVS                    *)
        (*************************************)

/*
 *       POBIERANIE SEGMENTU (JEDNEJ LUB, GDY LINKOWANY - KILKU POZYCJI)
 *       Z LISTY s0clist DO LISTY ltd Z ZA POZYCJI source ZA POZYCJE dest
 */
PROCEDURE movs (source, dest: posptr);
VAR
  curptr, lastptr: posptr;
 BEGIN
   IF source=NIL
    THEN
    curptr:=s0clist
    ELSE
    curptr:=source^.ptr;    /* curptr = WSK 1-SZEJ POZYCJI SEGMENTU */
   lastptr:=curptr;
   WHILE (lastptr^.ptr^.ptr<>NIL) AND (lastptr^.ptr^.seg[1]='*') DO
   lastptr:=lastptr^.ptr;  /* lastptr = WSK OSTATNIEJ POZYCJI SEGMENTU */
   IF source=NIL
    THEN      /* USUN SEGMENT Z LISTY s0clist         */
    s0clist:=lastptr^.ptr
    ELSE
    source^.ptr:=lastptr^.ptr;
   IF dest=NIL
    THEN
     BEGIN   /* WPROWADZ SEGMENT DO LISTY ltd */
       lastptr^.ptr:=ltd;
       ltd:=curptr
     END
    ELSE
     BEGIN
       lastptr^.ptr:=dest^.ptr;
       dest^.ptr:=curptr
     END
 END;
/*      MOVS    */

        (*************************************)
        (*           MEMTOEND                *)
        (*************************************)

/*
 * PROCEDURA PRZEMIESZCZA POZYCJE SEGMENTU MEMORY NA KONIEC LISTY s0clist
 */
PROCEDURE memtoend;
VAR
  curptr, lastptr, memptr, mlptr: posptr;
  memory: boolean;
 BEGIN
```

```
   curptr:=s0clist;
   memptr:=NIL;
   mlptr:=NIL;
   lastptr:=NIL;
   memory:=false;
   /* SZUKAJ POZYCJI SEGMENTU MEMORY */
   WHILE curptr^.ptr()NIL DO
   WITH curptr^ DO
    BEGIN
      memory:=memory AND (seg[1]='*');
      IF (memptr=NIL) AND (zord=mem)
       THEN
        BEGIN
          memptr:=lastptr;
          memory:=true
        END;
      IF memory
       THEN
       mlptr:=curptr;
      lastptr:=curptr;
      curptr:=ptr
    END;
   /* memptr = WSK POZYCJI PRZED SEGMENTEM MEMORY,
    mlptr = WSK OSTATNIEJ POZYCJI KONTYNUACJI SEGMENTU MEMORY */
   IF (mlptr()NIL) AND (lastptr() mlptr)
    THEN
    /* SEGMENT MEMORY JEST NIE NA KONCU LISTY */
     BEGIN   /* PRZESUN POZYCJE SEGMENTU MEMORY */
       IF memptr=NIL
        THEN
         BEGIN
           curptr:=s0clist;
           s0clist:=mlptr^.ptr
         END
        ELSE
         BEGIN
           curptr:=memptr^.ptr;
           memptr^.ptr:=mlptr^.ptr
         END;
       mlptr^.ptr:=lastptr^.ptr;
       lastptr^.ptr:=curptr
     END
 END;
/*      MEM TO END      */

        (*****************************************)
        (*           LOCRES                      *)
        (*****************************************)


/*
 * PROCEDURA LOKUJE NA LISCIE ltd POZYCJE res OPISUJACE POLA ZAREZERWO-
 * WANE OBSZARU ADRESOWEGO 8086 ORAZ NUMERUJE POZYCJE, DLA KTORYCH JEST
 * OKRESLONY PORZADEK LOKOWANIA (CONTROLEM order) I POZYCJE RELOKOWALNE
 * LOKOWANE W KOLEJNOSCI NAPOTKANIA NA WEJSCIU.
 */
PROCEDURE locres;
VAR
  curptr, lastptr, gtptr, ltptr: posptr;
  nrord: integer;
 BEGIN
   curptr:=s0clist;
   nrord:=0;
   lastptr:=NIL;
   WHILE curptr^.ptr()NIL DO /* SZUKAJ POZYCJI OBSZAROW ZAREZERWOWANYCH */
   WITH curptr^ DO
    BEGIN
      IF (zord=no) AND (aabs=no)
       THEN /* KONIEC POZYCJI UTWORZONYCH NA PODST. LINII WYWOLANIA */
```

```
          EXIT;
        IF aabs=res
         THEN
          BEGIN   /* LOKOWANIE RES        */
            ltptr:=NIL;
            gtptr:=ltd;
            /* ODSZUKANIE MIEJSCA W LISCIE ltd DLA DANEJ POZYCJI RES */
            WHILE gtptr^.ptr<>NIL DO
             BEGIN
               IF gt(gtptr^.ap, ap)
                THEN
                EXIT;
               ltptr:=gtptr;
               gtptr:=gtptr^.ptr
             END;
            /* WSTAW POZYCJE RES Z s0clist DO OKRESLONEGO MIEJSCA W ltd */
            movs (lastptr, ltptr);
            IF lastptr=NIL
             THEN
             curptr:=s0clist
             ELSE
              curptr:=lastptr^.ptr
          END
        ELSE
         BEGIN
           IF (zord<>no) AND (seg[1]<>'*')
            THEN
             BEGIN   /* NUMEROWANIE POZYCJI WG ORDER */
               nrord:=nrord+1;
               nord:=nrord
             END;
           lastptr:=curptr;
           curptr:=ptr
         END
     END;
    curptr:=s0clist;
    WHILE curptr^.ptr<>NIL DO /* NUMERUJ POZOSTALE POZYCJE */
    WITH curptr^ DO
     BEGIN
       IF (zord=no) AND (aabs=no)
        THEN
         BEGIN
           nrord:=nrord+1;
           nord:=nrord
         END;
       curptr:=ptr;
     END
 END;
/*      LOCRES  */

        (*************************************)
        (*           LOCSMABS                *)
        (*************************************)

/*
 *       PROCEDURA LOKUJE SEGMENTY ABSOLUTNE NA LISCIE ltd
 */
PROCEDURE locsmabs;
VAR
  ak, ak1: adres;
  curptr, lastptr, lckptr, llckptr, plptr: posptr;
  got: boolean;
 BEGIN
   curptr:=s0clist;
   lastptr:=NIL;
   WHILE curptr^.ptr<>NIL DO /* SZUKAJ SEGMENTOW ABSOLUTNYCH */
   WITH curptr^ DO
   IF aabs=sm
```

```
        THEN
         BEGIN
           /* OKRESL ADR KONCA SEGMENTU ABSOLUTNEGO */
           eos(ap, dl, ak);
           /* ODNAJDZ MIEJSCE DLA SEGMENTU ABSOLUTNEGO NA LISCIE ltd */
           lckptr:=ltd;
           llckptr:=NIL;
           got:=false;
           WHILE lckptr^.ptr()NIL DO
            BEGIN /* PRZESZUKUJ LISTE SEGMENTOW ULOKOWANYCH */
              IF NOT got AND gt(lckptr^.ap, ap)
               THEN
                BEGIN /* PLptr = WSK POZYCJI POPRZEDZAJACEJ ODSZUKANE MIJSCE */
                  plptr:=llckptr;
                  got:=true
                END;
              IF gt(lckptr^.ap, ak)
               THEN /* DOSYC MIEJCSA NA SEGMENT */
               EXIT;
              findend (lckptr, ak1);
              IF gt (ak1, ap) OR equ (ak1, ap)
               THEN /* SYGNALIZACJA BLEDU GDY OBSZARY POZYCJI POKRYWAJA SIE */
                BEGIN
                  IF lckptr^.aabs=res
                   THEN
                   skok1(37);
                  error1:=seg;
                  error2:=lckptr^.seg;
                  IF gt(ap, lckptr^.ap)
                   THEN
                   lowad:=ap
                   ELSE
                   lowad:=lckptr^.ap;
                  IF gt(ak, ak1)
                   THEN
                   highad:=ak1
                   ELSE
                   highad:=ak;
                  errorcom(36);
                END;
              llckptr:=lckptr; /* PRZEJSCIE DO NASTEPNEJ POZYCJI ULOKOWANEJ */
              lckptr:=lckptr^.ptr
            END;
           IF NOT got
            THEN
            plptr:=llckptr;
            /* PRZENIES POZYCJE SEGMENTU ABSOLUTNEGO */
           movs(lastptr, plptr);
           IF lastptr=NIL
            THEN
            curptr:=s0clist
            ELSE
            curptr:=lastptr^.ptr
         END
        ELSE
         BEGIN
           lastptr:=curptr;
           curptr:=ptr
         END
 END;
/*LOC SM ABS*/

        (*****************************************)
        (*           LOCCSABS                    *)
        (*****************************************)

/*
 * PROCEDURA LOKUJE KLASY ABSOLUTNE POBIERAJAC JE Z LISTY s0clist I
```

```
 * UMIESZCZAJAC JE NA LISCIE ltd
 */
PROCEDURE loccsabs;
VAR
  curptr, lastptr, lcsptr, lckptr, llckptr: posptr;
  akk, akl: adres;
  typ, gap: integer;

       (*************************************)
       (*           EOC                     *)
       (*************************************)

  /*
   | PROCEDURA OBLICZA ADRESY DLA SEGMENTOW KLASY ABSOLUTNEJ I ADRES
   | KONCOWY KLASY.
   | WE: curptr = WSK DO 1-GO ELEMENTU KLASY
   | WY: ak     = ADRES KONCA KLASY
   |     lcsptr = WSK DO OSTATNIEGO ELEMENTU KLASY
   */
  PROCEDURE eoc (VAR ak: adres);
  VAR
    posad, curad: adres;
    gap: integer;
    csptr: posptr;
   BEGIN
     posad:=curptr^.ap;
     eos(posad, curptr^.dl, curad); /* ADRES KONCA SEGMENTU */
     csptr:=curptr^.ptr;
     lcsptr:=curptr;
     WHILE (csptr^.ptr<>NIL)
     AND (((csptr^.aabs=cs) AND (csptr^.kl=curptr^.kl))
          OR (csptr^.seg[1]='*')) DO
     WITH csptr^ DO
      BEGIN /* LOKUJ KOLEJNE SEGMENTY KLASY */
        error1:=seg;
        error2:=kl;
        IF seg[1]<>'*'
         THEN
          BEGIN /* OBLICZ ADRES POCZATKOWY SEGMENTU */
            adaddr (curad, 1, curad); /* curad=ADR KONCOWY POPRZ. SEGMENTU*/
            alitaddr (curad, typs, gap);
            adaddr (curad, gap, posad); /* posad = ADR POCZ SEGMENTU */
            eos (posad, dl, curad);
          END;
         /* ZAPISZ ADRES POCZATKOWY DO POZYCJI */
        ap.par:=posad.par;
        ap.off:=sumator (posad.off, ap.off, 33);
        lcsptr:=csptr;
        csptr:=ptr
      END
   END;
  /* EOC */

 /**********POCZATEK PROCEDURY LOCCSABS ***********************************/
BEGIN
  curptr:=s0clist;
  lastptr:=NIL;
  WHILE curptr^.ptr<>NIL DO       /* PRZESZUJ LISTE s0clist */
  WITH curptr^ DO
   BEGIN
     IF (aabs=no) AND (zord=no)
      THEN       /* DALEJ NA LISCIE NIE MA KLAS ABSOLUTNYCH */
      EXIT;
     IF aabs=cs
      THEN
       BEGIN   /* LOKUJ KLASE ABSOLUTNA */
         eoc (akk);
         /* curptr = WSK POCZATKU KLASY,
```

```
                lcsptr = WSK OSTATNIEJ POZYCJI KLASY
                ap = ADRES POCZATKU KLASY, akk = ADRES KONCA KLASY */
               /* PRZESZUKUJ LISTE POZYCJI ULOKOWANYCH          */
               lckptr:=ltd;
               llckptr:=NIL;
               WHILE lckptr^.ptr()NIL DO
                BEGIN
                  IF gt(lckptr^.ap, akk)
                   THEN /* ZNALEZIONE MIEJSCE NA KLASE ABSOLUTNA */
                   EXIT;
                   /* KONTROLA NIE NAKLADANIA SIE  KLASY I POZYCJI */
                  findend (lckptr, akl); /* ADRES KONCOWY POZYCJI UL */
                  IF gt(akl, ap) OR equ(akl, ap)
                   THEN /* NAKLADANIE - WYJSCIE Z BLEDEM */
                   IF lckptr^.aabs=res
                    THEN
                    skok1(37)
                    ELSE
                     BEGIN
                       error1:=kl;
                       skok1(53);
                     END;
                    /* PRZEJDZ DO NASTEPNEJ POZYCJI ULOKOWANEJ */
                  llckptr:=lckptr;
                  lckptr:=lckptr^.ptr;
                END;
               /* WPROWADZ KLASE ABSOLUNA NA LISTE POZ. ULOKOWANYCH ltd */
               IF lastptr=NIL /* USUN KLASE Z s0clist */
                THEN
                s0clist:=lcsptr^.ptr
                ELSE
                lastptr^.ptr:=lcsptr^.ptr;
               IF llckptr=NIL /* WPROWADZ KLASE DO ltd */
                THEN
                 BEGIN
                   lcsptr^.ptr:=ltd;
                   ltd:=curptr
                 END
                ELSE
                 BEGIN
                   lcsptr^.ptr:=llckptr^.ptr;
                   llckptr^.ptr:=curptr
                 END;
                /* PRZEJDZ DO NAST POZYCJI NA LISCIE s0clist */
               IF lastptr=NIL
                THEN
                curptr:=s0clist
                ELSE
                curptr:=lastptr^.ptr;
             END
            ELSE
             BEGIN
               lastptr:=curptr;
               curptr:=ptr
             END
        END
    END;
/* LOC CS ABS   */

        (*****************************************)
        (*           LOCREL                      *)
        (*****************************************)

/*
 *       LOKOWANIE SEGMENTOW I KLAS RELOKOWALNYCH
 */
PROCEDURE locrel;
VAR
```

```
 curad, endrel, stad: adres;
 lckptr, llckptr, curptr, lastptr: posptr;
 nrlok, gap: integer;
BEGIN
  stad.par:=40B;
  stad.off:=0;
  curad:=stad;
  lckptr:=ltd;
  llckptr:=NIL;
  curptr:=s0clist;
  lastptr:=NIL;
  nrlok:=0;
  WHILE curptr^.ptr<>NIL DO  /* SZUKAJ POZYCJI RELOKOWALNYCH */
  WITH curptr^ DO
   BEGIN
     alitaddr (curad, typs, gap); /* dopasuj adres do A.T. */
     adaddr (curad, gap, curad);
     eos (curad, dl, endrel);
     WHILE lckptr^.ptr<>NIL DO /* ZNAJDZ MIEJSCE DLA SEGMENTU REL */
      BEGIN
        IF lckptr^.seg[1]<>'*'
         THEN /* OPUSZCZANIE SEGMENTOW KONTYNUACJI */
          BEGIN
            IF (nord=nrlok+1) AND gt(lckptr^.ap, endrel)
             THEN /* JEST MIEJSCE DLA SEGM REL PRZED POZ. UL. */
             EXIT;
             /* PRZEJDZ DO NASTEPNEJ POZYCJI ULOKOWANEJ */
            IF lckptr^.zord<>no
             THEN /* KONTROLA ZACHOWANIA ORDER */
              BEGIN
                nrlok:=lckptr^.nord;
                IF nrlok>nord
                 THEN
                  BEGIN
                    error1:=lckptr^.seg;
                    error2:=seg;
                    skok1(39)
                  END
              END;
            findend (lckptr, curad); /* KONIEC NASTEPNEJ POZ. UL. */
            adaddr (curad, 1, curad); /* PIERWSZY WOLNY ADRES */
            alitaddr (curad, typs, gap); /* DOPASUJ ADRES DO A. T. */
            adaddr (curad, gap, curad);
            IF gt (stad, curad)
             THEN
             curad:=stad;
            eos (curad, dl, endrel);
           END;
        llckptr:=lckptr;
        lckptr:=lckptr^.ptr
      END;
     /* LOKUJ SEGMENT RELOKOWALNY */
     movs(lastptr, llckptr);
     ap:=curad;
     llckptr:=curptr;
     lastptr:=ptr; /* ZAPISZ ADRESY DO SEGMENTOW - KONTYNUACJI */
     WHILE (lastptr^.ptr<>NIL) AND (lastptr^.seg[1]='*') DO
      BEGIN
        lastptr^.ap.par:=curad.par;
        lastptr^.ap.off:=sumator (curad.off, lastptr^.ap.off, 40);
        llckptr:=lastptr; /* PRZESUN WSK OSTATNIEJ POZ UL */
        lastptr:=lastptr^.ptr
      END;
     nrlok:=nord;
     adaddr (endrel, 1, curad);
     lastptr:=NIL;
     curptr:=s0clist
   END
```

```
 END;
/* LOCREL */

        (*************************************)
        (*           OFFSET GLOB             *)
        (*************************************)

(*
 * PROCEDURA OBLICZA OFFSETY OBIEKTOW GLOBALNYCH UWZGLENIAJACE
 * OFFSETY POCZATKOWE SEGMENTOW ZAWIERAJACYCH DANE OBIEKTY,
 * OKRESLONE W PROCESIE LOKACJI
 *)
PROCEDURE offsetglob;
VAR
  c: char;
  gp: globalptr;
 BEGIN
   FOR c:='?' TO '_' DO
    BEGIN
      gp:=glob[c];
      WHILE gp<>NIL DO
      WITH gp^ DO
       BEGIN
         offset:=sumator (offset, segments[nrg, nrs]^.ap.off, 40);
         gp:=gptr
       END
    END
 END;
/* OFFSET GLOB */

        (*************************************)
        (*           LOC                     *)
        (*************************************)

/*
 + PROCEDURA OTWIERA LISTE SEGMENTOW I POZYCJI ULOKOWANYCH ltd,
 + LOKUJE SEGMENTY I OBSZARY ZARAZERWOWANE, SPRAWDZA POPRAWNOSC LOKACJI
 + (POKRYWANIE SIE ULOKOWANYCH OBIEKTOW I ZACHOWANIE ZADANEJ KOLEJNOSCI)
 */
PROCEDURE loc;
 BEGIN
   new(ltd);
   memtoend;
   locres;
   locsmabs;
   loccsabs;
   locrel;
   offsetglob;
 END;
/* LOC */
```

```
(**E+*)
FUNCTION inc (VAR ch: char; VAR sum: lonum): integer; EXTERNAL;
FUNCTION sumator (x, y: lonum; nrerror: integer): lonum; EXTERNAL;
PROCEDURE lobin (ld: lonum; VAR lb: tablbin); EXTERNAL;
FUNCTION shiftr (x, c: integer): integer; EXTERNAL;
FUNCTION shiftl (x, c: integer): integer; EXTERNAL;
PROCEDURE binlon (VAR lbin: tablbin; VAR ldl: lonum); EXTERNAL;
PROCEDURE sumbin (l1, l2: tablbin; VAR suma: tablbin); EXTERNAL;
PROCEDURE subbin (l1, l2: tablbin; VAR l: tablbin); EXTERNAL;
PROCEDURE subsbin(l1, l2: tablbin; VAR l: tablbin; VAR ok: boolean); EXTERNAL;
PROCEDURE addrbin (par, off: lonum; VAR lb: tablbin); EXTERNAL;
PROCEDURE longrec (VAR dlr, sum: lonum); EXTERNAL;
PROCEDURE rewrec (VAR sum: lonum); EXTERNAL;
PROCEDURE skok1 (nrerror: integer); EXTERNAL;
PROCEDURE errorcom(i: integer); EXTERNAL;
PROCEDURE outbyte(i: integer); EXTERNAL;

        (*****************************************)
        (*           MOD2                        *)
        (*****************************************)

(*
 + PROCEDURA ODCZYTUJE W 2 PRZEBIEGU MODUL we. PRZETWARZA REKORDY DANYCH A0
 + ZGODNIE Z DANYMI O RELOKOWALNOSCI I UZYSKANYMI W CZASIE LOKACJI ADRESAMI
 + PRODUKUJE MODUL WYJSCIOWY NA PLIKU wy W FORMACIE HEXADECYMALNYM.
 + (HEXADECIMAL OBJECT FILE FORMAT INTEL'A) Z BITEM PARZYSTOSCI.
 *)
PROCEDURE mod2;
VAR
  ch: char;
  il, sum, dlr, stoff: lonum;
  nrseg, curch, lastch: integer;
  data: ARRAY[0..1023] OF char;

        (*****************************************)
        (*           CODEIN                      *)
        (*****************************************)

  (*
   * PROCEDURA ODCZYTU REKORDU DANYCH A0 DO TABLICY data.
   * PROCEDURA KONTROLUJE POPRAWNOSC REKORDU A0
   * (DLUGOSC I ODNIESIENIE DO ISTNIEJACEGO SEGMENTU).
   * WY: data = TABLICA Z DANYMI REKORDU
   *     nrseg = NUMER SEGMENTU ZAWIERAJACEGO DANE
   *     stoff = OFFSET 1-GO BAJTU DANYCH REKORDU
   *)
  PROCEDURE codein;
  VAR
    i,j: integer;
   BEGIN
     longrec(dlr, sum);
     IF dlr>1028
      THEN
      skok1(12);
     il:=dlr-4;
     nrseg:=inc(ch, sum);
     IF nrseg>maxs
      THEN
      skok1(12);
     IF segments[nrmodwe,nrseg]=NIL
      THEN
      skok1(12);
     stoff:=(inc(ch, sum) AND 377B) OR shiftl (inc(ch, sum), 8);
     FOR i:=0 TO dlr-5 DO
      BEGIN
        j:=inc(ch, sum);
        data[i]:=ch
```

```
    END;
   i:=inc(ch, sum);
   IF sum()0
    THEN
    skok1(12)
 END;
(*      CODEIN  *)

      (****************************************)
      (*           REL                        *)
      (****************************************)

(*
 * PROCEDURA ODCZYTUJE REKORD 9C I MODYFIKUJE DANE W TABLICY data
 * NA PODSTAWIE INFORMACJI O RELOKACJI ZAWARTYCH W REKORDZIE 9C.
 * KONTROLA REKORDU we, SUMA KONTROLNA I WSK we TAK JAK DLA rewrec(sum).
 *)
PROCEDURE rel;
VAR
  n, nrch, ccval: lonum;
  sptr: posptr;
  eptr: globalptr;

      (****************************************)
      (*           PRINT ERR EXT              *)
      (****************************************)

  (*
   | PROCEDURA DRUKUJE KOMUNIKAT O BLEDZIE W WYPADKU ODWOLANIA
   | DO NIEROZWIAZANEGO EXTERNALA. ATRYBUTY TAKIEGO EXTERNALA
   | (SEGMENT, OFFSET) SA ROWNE ZERU.
   *)
  PROCEDURE printerrext (eptr: globalptr);
   BEGIN
     dat1:=stoff+n;
     error1:=segments [nrmodwe, nrseg]^.seg;
     error2:=eptr^.etykieta;
     errorcom (31)
   END;
  /* PRINT ERR EXT */

      (****************************************)
      (*           TAKE PTR SEG               *)
      (****************************************)

  (*
   | PROCEDURA ODCZYTUJE ZNAK Z REKORDU 9C, BEDACY NUMEREM
   | SEGMENTU. SPRAWDZA, CZY SEGMENT O TYM NUMERZE ISTNIEJE
   | I POBIERA JEGO POINTER.
   | WY: sptr - POINTER SEGMENTU
   *)
  PROCEDURE takeptrseg;
   BEGIN
     ccval:=inc (ch, sum);
     nrch:=nrch+1;
     IF ccval>maxs
      THEN
      skok1 (12);
     sptr:=segments [nrmodwe, ccval];
     IF sptr=NIL
      THEN
      skok1 (12)
   END;
  /* TAKE PTR SEG */

      (****************************************)
      (*           TAKE PTR EXT               *)
      (****************************************)
```

```
(*
 | PROCEDURA ODCZYTUJE ZNAK Z REKORDU 9C, BEDACY NUMEREM EXTERNALA.
 | SPRAWDZA, CZY EXTERNAL O TYM NUMERZE ISTNIEJE, CZY JEST
 | ROZWIAZANY I POBIERA JEGO POINTER.
 | WY: eptr - POINTER EXTERNALA
 *)
PROCEDURE takeptrext;
 BEGIN
   ccval:=inc (ch, sum);
   nrch:=nrch+1;
   IF ccval>maxe
    THEN
    skok1 (12);
   eptr:=externals [nrmodwe, ccval];
   IF eptr=NIL
    THEN
    skok1 (12);
   IF eptr^.nrs=0
    THEN /* EXTERNAL NIEROZWIAZANY */
    printerrext (eptr)
 END;
/* TAKE PTR EXT */

    (****************************************)
    (*           RELOFF                     *)
    (****************************************)

(*
 | PROCEDURA ODCZYTUJE DANE O OFFSET RELOKOWALNOSCI Z REKORDU 9C
 | I MODYFIKUJE ZGODNIE Z NIMI 2 BAJTY TABLICY data.
 | WE: ccval = 1-SZY BAJT INFORMACJI O RELOKOWALNOSCI
 |     WSK we NA 2-GI BAJT INFORMACJI O RELOKOWALNOSCI
 | WY: WSK we NA BAJT ZA INFORMACJA O RELOKOWALNOSCI
 | PROCEDURA MODYFIKUJE SUME KONTROLNA sum I ILOSC BAJTOW ODCZYTANYCH
 | REKORDU 9C.
 *)
PROCEDURE reloff;
VAR
  relval, ofval: lonum;
 BEGIN
   n:=(ccval-304B)*256+inc(ch, sum);
   IF n>il
    THEN
    skok1(12);
   ccval:=inc(ch, sum);
   IF (ccval>=130B) AND (ccval<=133B)
    THEN          (*      58H - WG ASSUME *)
    WITH assume[ccval-130B] DO
    IF ex
     THEN
     WITH externals [nrmodwe, nex]^ DO
      BEGIN
        IF nrs = 0
         THEN
         printerrext (externals [nrmodwe, nex] );
        ofval:=offset
      END
     ELSE
     ofval:=segments[nrm, nrms]^.ap.off
    ELSE
    IF ccval=122B
     THEN
      BEGIN         (*      52H - WG NR EXTERNALA *)
        takeptrext;
        ofval:=eptr^.offset
      END
     ELSE
```

```
      IF ccval=120B
       THEN
        BEGIN        (*      50H - WG NR SEGMENTU *)
          takeptrseg;
          ofval:=sptr^.ap.off;
        END
       ELSE
       skok1(12);
     nrch:=nrch+4;
     longrec (relval, sum);
     IF relval<>0
      THEN
      relval:=sumator (relval, ofval, 40)
      ELSE
       BEGIN
         relval:=0-((ord(data[n]) AND 377B) OR shiftl (ord(data[n+1]), 8));
         IF relval>ofval
          THEN
          skok1(40);
         relval:=ofval-relval
       END;
     data[n]:=chr( relval AND 377B);
     data[n+1]:=chr( shiftr( relval, 8))
  END;
(*      RELOFF  *)

    (****************************************)
    (*            RELBASE                   *)
    (****************************************)

(*
 | PROCEDURA ODCZYTUJE DANE O BASE RELOKOWALNOSCI Z REKORDU 9C
 | I ZGODNIE Z NIMI MODYFIKUJE 2 BAJTY W TABLICY data.
 | WE, WY - P. PROCEDURA reloff
 *)
PROCEDURE relbase;
VAR
  relval, basval: lonum;
 BEGIN
   n:=(ccval-310B)*256+inc(ch, sum);
   IF n>il
    THEN
    skok1(12);
   ccval:=inc(ch, sum);
   IF (ccval)=134B) AND (ccval(=137B)
    THEN                  (* 5C-5FH - WG ASSUME  *)
    WITH assume [ccval-134B] DO
     BEGIN
       IF nrms=0
        THEN
        printerrext (externals[nrmodwe, nex]);
       sptr:=segments[nrm, nrms]
     END
    ELSE
    IF ccval=124B
     THEN                        (*      54H - WG NR SEGMENTU *)
     takeptrseg
     ELSE
     IF ccval=126B
      THEN                        (*      56H - WG NR EXTERNALA *)
       BEGIN
         takeptrext;
         sptr:=segments[eptr^.nrg, eptr^.nrs]
       END
      ELSE
      skok1(12);
   basval:=sptr^.ap.par;
   relval:=(ord(data[n]) AND 377B) OR shiftl (ord(data[n+1]), 8);
```

```
   relval:=basval+relval;
   data[n]:=chr( relval AND 377B);
   data[n+1]:=chr( shiftr( relval, 8));
   nrch:=nrch+2
 END;
(*      RELBASE         *)

    (****************************************)
    (*           RELDISP                    *)
    (****************************************)

(*
 | PROCEDURA ODCZYTUJE DANE O DISPLACEMENT RELOKOWALNOSCI
 | Z REKORDU 9C I ZGODNIE Z NIMI MODYFIKUJE 2 BAJTY W TABLICY data.
 | WE I WY - P. PROCEDURA reloff.
 *)
PROCEDURE reldisp;
VAR
  ext, dispok: boolean;
  destpar, destoff, srcpar, srcoff, relval, disp: lonum;
  dest, relbin, destm, src, dispbin, stsegbin, curoffbin: tablbin;
 BEGIN
   n:=(ccval-204B)*256+inc (ch, sum); /* TYP REL 84H */
   IF n>il
    THEN
    skok1 (12);
   ccval:=inc (ch, sum);
   /* OBLICZ ADRES CELU destpar, destoff */
   IF (ccval)=210B) AND (ccval(=213B)
    THEN /* 88H - WG ASSUME */
    WITH assume [ccval-210B] DO
     BEGIN
       ext:=ex;
       IF ext
        THEN /* ASSUME SEG EXTERNAL */
        WITH externals [nrmodwe, nex]^ DO
         BEGIN
           IF nrs=0
            THEN
            printerrext (externals [nrmodwe, nex]);
           destpar:=segments [nrg, nrs]^.ap.par;
           destoff:=offset
         END
        ELSE /* ASSUME SEGMENT */
         BEGIN
           destpar:=segments [nrm, nrms]^.ap.par;
           destoff:=segments [nrm, nrms]^.ap.off
         END
     END
    ELSE
    IF ccval=200B
     THEN /* 80H - WG NUMERU SEGMENTU */
      BEGIN
        takeptrseg;
        ext:=false;
        destpar:=sptr^.ap.par;
        destoff:=sptr^.ap.off
      END
     ELSE
     IF ccval=202B
      THEN /* 82H - WG NUMERU EXTERNALA */
       BEGIN
         takeptrext;
         ext:=true;
         WITH eptr^ DO
          BEGIN
            destpar:=segments [nrg, nrs]^.ap.par;
            destoff:=offset
```

```
            END
          END
         ELSE
         skok1 (12);
         /* OBLICZ WARTOSC RELOKOWANIA */
       IF ext
        THEN
         BEGIN
           relval:=inc (ch, sum)+inc (ch,sum); /* PRZEWIN 2 ZNAKI */
           relval:=(ord (data [n]) AND 377B) OR shiftl (ord (data [n+1]), 8)
         END
        ELSE
        relval:=(inc (ch, sum) AND 377B) OR shiftl (inc (ch, sum), 8);
       nrch:=nrch+4;

       /* ADRES CELU = ADRES CELU + WARTOSC RELOKOWANIA */
       addrbin (destpar, destoff, dest);
       IF relval AND 100000B<>0
        THEN /* WARTOSC RELOKOWANIA UJEMNA */
         BEGIN
           relval:=0-relval;
           lobin (relval, relbin);
           subbin (dest, relbin, destm)
         END
        ELSE /* WARTOSC RELOKOWANIA DODATNIA */
         BEGIN
           lobin (relval, relbin);
           sumbin (dest, relbin, destm)
         END;
        /* OBLICZ DISPLACEMENT */
       WITH segments [nrmodwe, nrseg]^ DO
       addrbin (ap.par, ap.off, stsegbin);
       lobin (stoff+n+2, curoffbin);
       sumbin (stsegbin, curoffbin, src);
       subsbin (destm, src, dispbin, dispok);
       IF NOT dispok
        THEN /* GDY DISPLACEMENT POZA ZAKRESEM */
        skok1 (51);
       binlon (dispbin, disp);
       data [n]:=chr (disp AND 377B);
       data [n+1]:=chr (shiftr (disp, 8))
     END;
    /* REL DISP */

       (*****************************************)
       (*           ASSSEG                      *)
       (*****************************************)

    (*
     | PROCEDURA ODCZYTUJE DANE O ASSUME WG SEGMWNTU Z DANEGO MODULU
     | I ZAPISUJE JE W TABLICY assume.
     | WE I WY - P. PROCEDURA reloff
     *)
    PROCEDURE assseg;
    VAR
      rejseg: integer;
     BEGIN
       rejseg:=ccval-100B;
       takeptrseg;
       IF rejseg<>inc(ch, sum)
        THEN
        skok1(12);
       IF ccval<>inc(ch, sum)
        THEN
        skok1(12);
       nrch:=nrch+2;
       WITH assume[rejseg] DO
        BEGIN
```

```
        nrm:=nrmodwe;
        nrms:=ccval;
        ex:=false
      END
  END;
 (*      ASSSEG  *)

     (*************************************)
     (*           ASSEXT                  *)
     (*************************************)

 (*
  | PROCEDURA ODCZYTUJE DANE O ASSUME WG SEG EXTERNAL
  | I ZAPISUJE JE W TABLICY assume. WE I WY - P. PROCEDURA reloff.
  *)
 PROCEDURE assext;
 VAR
   rejseg: integer;
  BEGIN
    rejseg:=ccval-110B;
    takeptrext;
    IF rejseg<>inc(ch, sum)-8
     THEN
     skok1(12);
    IF ccval<>inc(ch, sum)
     THEN
     skok1(12);
    nrch:=nrch+2;
    WITH assume[rejseg] DO
     BEGIN
       nrm:=eptr^.nrg;
       nrms:=eptr^.nrs;
       ex:=true;
       nex:=ccval
     END
  END;
 (*      ASSEXT  *)

 (*******************POCZATEK PROCEDURY REL*****************************)
BEGIN
  longrec(dlr, sum);
  nrch:=1;
  ccval:=inc(ch, sum);
  WHILE nrch<dlr DO
   BEGIN
     IF (ccval>=304B) AND (ccval<=307B)
      THEN                  (*      0C4H    *)
      reloff
      ELSE
      IF (ccval>=310B) AND (ccval<=313B)
       THEN                 (*      0C8H    *)
       relbase
       ELSE
       IF (ccval>=100B) AND (ccval<=103B)
        THEN                (*      40H     *)
        assseg
        ELSE
        IF (ccval>=110B) AND (ccval<=113B)
         THEN               (*      48H     *)
         assext
         ELSE
         IF (ccval>=204B) AND (ccval<=207B)
          THEN              (*      84H     *)
          reldisp
          ELSE
          skok1(12);
     ccval:=inc(ch, sum);
     nrch:=nrch+1
```

```
    END;
   IF (nrch()dlr) OR (sum()0)
    THEN
    skok1(12)
 END;
(*      REL     *)

      (*****************************************)
      (*           CODEOUT                     *)
      (*****************************************)

(*
 * PROCEDURA WYPROWADZ DANE Z TABLICY data NA PLIK wy W POSTACI REKORDOW
 * HEXADECYMALNYCH.
 * WE: il = ILOSC BAJTOW W TABLICY DATA
 *     nrseg = NUMER SEGMENTU DO KTOREGO NALEZA DANE
 *     stoff = OFFSET 1-GO BAJTU DANYCH
 *     usba = AKTUALNY UPPER SEGMENT BASE ADDRESS
 *)
PROCEDURE codeout;
VAR
  i,j,k,ar: lonum;
 BEGIN
   ar:=segments[nrmodwe,nrseg]^.ap.par;
   IF usba()ar
    THEN
     BEGIN  (* WYPROWADZ EXTENDED ADDRESS RECORD    *)
       usba:=ar;
       write(wy,chr(72B));     (*      ':'     *)
       chksum:=0;
       outbyte(2);
       outbyte(0);
       outbyte(0);
       outbyte(2);
       outbyte(usba DIV 256);
       outbyte(usba MOD 256);
       outbyte(-chksum);
       writeln(wy);
     END;
   ar:=stoff+segments [nrmodwe, nrseg]^.ap.off;
   i:=0;
    REPEAT          (* WYPROWADZ DATA RECORDS       *)
     IF il-i)maxob
      THEN
      j:=maxob
      ELSE
      j:=il-i;
     IF j)0
      THEN
       BEGIN
         write(wy,chr(72B));
         chksum:=0;
         outbyte(j);
         outbyte(ar DIV 256);
         outbyte(ar MOD 256);
         outbyte(0);
         FOR k:=0 TO j-1 DO
         outbyte(ord(data[i+k]));
         outbyte(-chksum);
         writeln(wy);
         i:=i+j;
         ar:=ar+j
       END
    UNTIL i=il
 END;
(*      CODEOUT         *)

      (*****************************************)
```

```
      (*           DUPOUT                        *)
      (*****************************************)

/*
 * PROCEDURA DEKODUJE REKORD A2 ZAPISANY W TABLICY data I ZMODYFIKOWANY
 * INFORMACJAMI O RELOKOWALNOSCI Z REKORDU 9C.
 * DANE ZADEKLAROWANE PRZY POMOCY dup ZOSTAJA WYPROWADZONE
 * NA PLIK wy W POSTACI REKORDOW HEXADECYMALNYCH.
 * WE: data = TABLICA ZAWIERAJACA ZMODYFIKOWANY REKORD A2
 *     il   = ILOSC BAJTOW W TABLICY data
 *     nrseg= NUMER SEGMENTU DO KTOREGO NALEZA DANE
 *     stoff= OFFSET 1-GO BAJTU DANYCH W REKORDZIE
 *     usba = AKTUALNY UPPER SEGMENT BASE ADDRESS
 *     wy   = PLIK WYJSCIOWY
 * WY: usba = AKTUALNY UPPER SEGMENT BASE ADDRESS
 */
PROCEDURE dupout;
VAR
  i,pm,bo: integer;
  ar: lonum;
  rechex: ARRAY[0..maxob] OF integer;
  count, ctw, nump, ip: ARRAY[0..8] OF integer;

      (*****************************************)
      (*           FIELD                       *)
      (*****************************************)

  /*
   | PROCEDURA WYPROWADZA BAJTY WG OPISU POLA W REKORDZIE A2
   | DO BUFORA RECHEX WG WSKAZNIKA bo.
   | GDY BUFOR ZOSTAJE ZAPELNIONY JEST WYPROWADZANY NA PLIK WY
   | W POSTACI REKORDU HEXADECYMALNEGO.
   */
  PROCEDURE field;
  VAR
    j,k,l: integer;
   BEGIN
     l:=ord(data[i]);
     IF i+l>il
      THEN
      skok1(12);
     FOR j:=1 TO l DO
      BEGIN
        rechex[bo]:=ord(data[i+j]);
        bo:=bo+1;
        IF bo=maxob
         THEN
          BEGIN
            write(wy,chr(72B));
            chksum:=0;
            outbyte(bo);
            outbyte(ar DIV 256);
            outbyte(ar MOD 256);
            outbyte(0);
            FOR k:=0 TO bo-1 DO
            outbyte(rechex[k]);
            outbyte(-chksum);
            writeln(wy);
            ar:=ar+bo;
            bo:=0
          END
      END;
     i:=i+l+1;
     pm:=pm-1;
     ctw[pm]:=ctw[pm]+1
   END;
  /*      FIELD   */
```

```
    (*************************************)
    (*           POSD                    *)
    (*************************************)

 /*
  | PROCEDURA ODCZYTUJE ILOSC DUPLIKACJI NOWEGO POZIOMU
  | I ILOSC POZYCJI NA TYM POZIOMIE. SPRAWDZA SPELNIANIE OGRANICZEN
  | NA WIELKOSC TABLICY data I ILOSC BAJTOW REKORDU A2 ZAPISANYCH
  | W TABLICY data ORAZ NA POZIOM ZAGNIEZDZENIA NIE WIEKSZY OD 8.
  */
 PROCEDURE posd;
  BEGIN
    pm:=pm+1;
    IF (pm)8) OR (i)il-4)
     THEN
     skok1(12);
    count[pm]:=(ord(data[i]) AND 377B) OR shift1 (ord(data[i+1]), 8);
    nump[pm]:=(ord(data[i+2]) AND 377B) OR shift1 (ord(data[i+3]), 8);
    i:=i+4;
    ip[pm]:=i;
    ctw[pm]:=0;
  END;
 /*      POSD    */
 /*************POCZATEK PROCEDURY DUPOUT*********************************/
BEGIN
  ar:=segments[nrmodwe,nrseg]^.ap.par;
  IF usba()ar
   THEN
    BEGIN  (* WYPROWADZ EXTENDED ADDRESS RECORD    *)
      usba:=ar;
      write(wy,chr(72B));      (*      ':'     *)
      chksum:=0;
      outbyte(2);
      outbyte(0);
      outbyte(0);
      outbyte(2);
      outbyte(usba DIV 256);
      outbyte(usba MOD 256);
      outbyte(-chksum);
      writeln(wy);
    END;
  ar:=stoff+segments [nrmodwe, nrseg]^.ap.off;
  bo:=0;
  i:=0;
  pm:=0;
  posd;
  WHILE pm()0 DO
  IF nump[pm]=0
   THEN
    BEGIN
      field;
      WHILE (ctw[pm]=nump[pm]) AND (pm)0) AND (count[pm]=1)
      DO
       BEGIN
         pm:=pm-1;
         ctw[pm]:=ctw[pm]+1;
       END;
      IF (ctw[pm]=nump[pm]) AND (pm)0)
       THEN
        BEGIN
          ctw[pm]:=0;
          count[pm]:=count[pm]-1;
          i:=ip[pm];
        END
    END
   ELSE
   posd;
  IF bo()0
```

```
      THEN
       BEGIN
         write(wy,chr(72B));
         chksum:=0;
         outbyte(bo);
         outbyte(ar DIV 256);
         outbyte(ar MOD 256);
         outbyte(0);
         FOR pm:=0 TO bo-1 DO
         outbyte(rechex[pm]);
         outbyte(-chksum);
         writeln(wy)
       END;
     IF il<>i
      THEN
      skok1(12)
  END;
 /*      DUPOUT  */
 (*++++++++++++++++++++++POCZATEK PROCEDURY MOD2 ++++++++++++++++++++++++++*)
BEGIN
  sum:=0;
  curch:=inc(ch, sum);
  IF curch<>200B
   THEN     (* 80H - REKORD NAZWY MODULU    *)
   skok1(12);
  rewrec(sum);
  curch:=inc(ch, sum);
  IF curch<>210B
   THEN     (* 88H - REKORD WERSJI ASM      *)
   skok1(12);
  rewrec(sum);
  curch:=inc(ch, sum);
  IF curch<>226B
   THEN     (* 96H - REKORD NAZW SEGMENTOW I KLAS   *)
   skok1(12);
  rewrec(sum);
  curch:=inc(ch, sum);
  IF curch<>230B
   THEN     (* 98H - REKORD DANYCH SEGMENTU         *)
   skok1(12);              (* MUSI BYC CO NAJMNIEJ 1 REKORD 98H    *)
  WHILE curch=230B DO
   BEGIN       (* REKORDY DANYCH SEGMENTOW     *)
     rewrec(sum);
     curch:=inc(ch, sum)
   END;
  IF curch=216B
   THEN
    BEGIN        (* 8EH - REKORD NIEZNANY        *)
      rewrec(sum);
      curch:=inc(ch, sum);
    END;
  IF curch=214B
   THEN
    BEGIN        (* 8CH - REKORD EXTERNALI       *)
      rewrec(sum);
      curch:=inc(ch, sum);
    END;
  WHILE curch=220B DO
   BEGIN       (* 90H - REKORDY PUBLIC-OW      *)
     rewrec(sum);
     curch:=inc(ch, sum);
   END;
  WHILE (curch=240B) OR (curch=242B) DO
   BEGIN (* 0A0H,0A2H - REKORDY DANYCH I DUP *)
     lastch:=curch;
     codein;
     curch:=inc(ch, sum);
     IF curch=234B
```

```
        THEN
         BEGIN          (* 9CH - REKORD INF O REL  *)
           rel;
           curch:=inc(ch, sum)
         END;
        IF lastch=240B
         THEN
         codeout
         ELSE
         dupout
      END;
     IF curch<>212B
      THEN      (* 8AH - REKORD KONCA MODULU    *)
      skok1(12);
  END;
(*      MOD2    *)
```

```
FUNCTION shiftr(x, c: integer): integer; EXTERNAL;
PROCEDURE errorcom(i: integer); EXTERNAL;
PROCEDURE num4(VAR lst: text; i: lonum); EXTERNAL;
PROCEDURE num5(VAR lst: text; i: lonum; k: integer); EXTERNAL;
(**E+*)

        (*************************************)
        (*           HEADER                  *)
        (*************************************)

(*+ PROCEDURA LISTUJE NAGLOWEK LISTINGU NA PLIK lst. +*)

PROCEDURE header;
VAR
  i,j: integer;
 BEGIN
   writeln(lst);
   writeln(lst,'   PIAP QRL86/SM LINKER-LOCATER');
   writeln(lst);
   writeln(lst,'   INVOCATION COMMAND:');
   FOR i:=0 TO 20 DO /* WYDRUK LINII WYWOLANIA tlin */
    BEGIN
      IF tlin[i]=NIL
       THEN EXIT;
      write(lst,'   ');
      j:=0;
      WHILE (j<=dlw) AND (tlin[i]^[j]<>'%') DO
       BEGIN
         write(lst,tlin[i]^[j]);
         j:=j+1
       END;
      writeln(lst);
    END;
   writeln(lst)
 END;
(*      HEADER  *)

        (*************************************)
        (*           MAP                     *)
        (*************************************)

(*+ PROCEDURA LISTUJE TABLICE SYMBOLI PUBLIC I MAPE PAMIECI. +*)

PROCEDURE map;
VAR
  i,j: integer;
  c: char;
  nl: boolean;
  gl: globalptr;
  curptr: posptr;
  length: lonum;
 BEGIN
   writeln(lst);
   nl:=false;
   FOR c:='?' TO '_' DO /* SPRAWDZ CZY SA GLOBALE */
   nl:=nl OR (glob[c]<>NIL);
   IF nl /* WYDRUK LISTY GLOBALI */
    THEN
     BEGIN
       writeln(lst,'   SYMBOL TABLE'); /* NAGLOWEK LISTY GLOBALI */
       writeln(lst);
       FOR i:=1 TO 2 DO
        BEGIN
          write(lst,'   BASE    OFFSET TYPE SYMBOL');
          write(lst,'                              ');
        END;
       writeln(lst);
```

```
      writeln(lst);
      nl:=true;
      FOR c:='?' TO '_' DO /* LISTUJ GLOBALE Z ROZPROSZONEJ TABLICY glob */
       BEGIN
         gl:=glob[c];
         WHILE gl<>NIL DO WITH gl^ DO
          BEGIN
            write(lst,'   ');
            IF nrs=0
             THEN
             num4(lst, 0)
             ELSE
             num4(lst, segments[nrg,nrs]^.ap.par); /* BASE */
            write(lst,'   ');
            num4(lst, offset); /* OFFSET */
            write(lst,'  PUB  '); /* TYP */
            nl:=NOT nl; /* nl = PIERWSZA KOLUMNA W DANYM WIERSZU */
            FOR i:=1 TO dls DO /* NAZWA */
             BEGIN
               IF nl AND (etykieta[i]=' ')
                THEN EXIT; /* KONIEC LINII */
               write (lst,etykieta[i])
             END;
            IF nl
             THEN writeln(lst);
            gl:=gptr
          END
       END;
      IF NOT nl /* KONIEC OSTATNIEJ NIEPELNEJ LINII */
       THEN writeln(lst)
    END;
  writeln(lst);
  writeln(lst,'   MEMORY MAP'); /* NAGLOWEK MAPY PAMIECI */
  writeln(lst);
  write(lst,'   START     STOP    LENGTH');
  writeln(lst,' ALIGN NAME                                    CLASS');
  writeln(lst);
  curptr:=ltd;
  WHILE curptr^.ptr<>NIL DO WITH curptr^ DO /* LISTUJ SEGMENTY */
   BEGIN
     IF seg[1]<>'*'
      THEN
       BEGIN
         write(lst,'   ');
         num5(lst, ap.par, ap.off); /* ADRES POCZATKU SEGMENTU */
         write(lst,'   ');
         IF dl=0 /* OBLICZ ADRES KONCA SEGMENTU */
          THEN
          length:=dl
          ELSE
          length:=dl-1;
         i:=(length AND 17B)+ap.off;
         j:=i DIV 16;
         i:=i MOD 16;
         num5(lst, ap.par+shiftr(length, 4)+j, i);/* LISTUJ ADRES KONCA SEGME
         write(lst,'    ');
         num4(lst, dl); /* DLUGOSC SEGMENTU */
         i:=typs AND at AND atinot; /* ALIGN TYPE SEGMENTU */
         CASE i OF
           atat:   c:='A';
           atwo:   c:='W';
           atpr:   c:='G';
           atpg:   c:='P';
           ELSE    c:='B'
          END;
         write(lst,'   ',c);
         IF typs AND ati=1 /* SEGMENT INPAGE */
          THEN
```

```
                write(lst,'R ')
               ELSE
                write(lst,'   ');
               FOR i:=1 TO dls DO /* NAZWA SEGMENTU */
                BEGIN
                  IF (seg[i]=' ') AND (kl[1]=' ')
                   THEN EXIT; /* KONIEC DRUKU NAZWY SEGMENTU*/
                  write(lst,seg[i])
                END;
               write(lst,'  ');
               FOR i:=1 TO dls DO /* NAZWA KLASY SEGMENTU */
                BEGIN
                  IF kl[i]=' '
                   THEN EXIT; /* KONIEC WIERSZA */
                  write(lst,kl[i])
                END;
               writeln(lst);
             END;
           curptr:=ptr
       END
  END;
(*       MAP      *)
```

```
PROCEDURE errorcom(i: integer); EXTERNAL;
PROCEDURE dyrektywa; EXTERNAL;
PROCEDURE invocationcheck; EXTERNAL;
PROCEDURE linker; EXTERNAL;
PROCEDURE loc; EXTERNAL;
PROCEDURE modul; EXTERNAL;
PROCEDURE mod2; EXTERNAL;
PROCEDURE header; EXTERNAL;
PROCEDURE map; EXTERNAL;
PROCEDURE outbyte(i: integer); EXTERNAL;

        (*************************************)
        (*           SKOK1                   *)
        (*************************************)

(*
 * PROCEDURA WYJSCIA Z BLEDEM Z PODPROGRAMOW. JEST TO PROCEDURA
 * GLOBALNA. DRUKUJE KOMUNIKAT O BLEDZIE O DANYM NUMERZE
 * I WYKONUJE SKOK DO ETYKIETY 1 W PROGRAMIE GLOWNYM ORL
 *)
(**E+*)
PROCEDURE skok1 (nrerror: integer);
 BEGIN
   errorcom (nrerror);
   GOTO 1;
 END;
/* SKOK1 */
(**E-*)

/*@@@@@@@@@@@@@@@@@@@@@@@@@@  PROGRAM MAIN LINKER - LOCATER - OH @@@@@@@@@*/

BEGIN
  listopen:=false;
  wyopen:=false;
  rewrite(ti,'TI:');
  dyrektywa; /* ODCZYT LINII WYWOLANIA */
  IF listing
   THEN /* OTWORZ PLIK LISTINGOWY I DRUKUJ NAGLOWEK */
    BEGIN
      rewrite(lst,pllst,,len);
      IF len=-1
       THEN
        BEGIN
          error1:=pllst;
          skok1 (20)
        END;
      header;
      listopen:=true
    END;
  invocationcheck; /* KONTROLA LINII WYWOLANIA */
  FOR nrmodwe:=1 TO maxm DO       (*      ZERUJ TABLICE SEGMENTOW          *)
  FOR len:=1 TO maxs DO
  segments[nrmodwe,len]:=NIL;
  FOR nrmodwe:=1 TO maxm DO       (*      ZERUJ TABLICE EXTERNALI          *)
  FOR len:=1 TO maxe DO
  externals[nrmodwe,len]:=NIL;
  FOR len:=0 TO 3 DO
   BEGIN        (*      ZERUJ ASSUME            *)
     assume[len].nrm:=maxm;
     assume[len].nrms:=maxs
   END;
  usba:=0;                        (*      ZERUJ USBA              *)
  nrmodwe:=1;

  /* 1 PRZEBIEG ORL - ODCZYT DANYCH O SEGMENTACH I ZMIENNYCH GLOBALNYCH */
  plikptr:=obiekty;
  WHILE plikptr()NIL DO
```

```
 BEGIN
   IF nrmodwe>maxm
    THEN
    skok1 (17);
   plik:=plikptr^.spec;
   reset(we,plik,,len);
   IF len=-1
    THEN
     BEGIN
       error1:=plik;
       skok1 (20)
     END;
   modul; /* ODCZYT DANYCH Z PLIKU WE */
   close(we);
   nrmodwe:=nrmodwe+1;
   plikptr:=plikptr^.next
 END;

linker; /* LINKUJ SEGMENTY Z PLIKOW WE */
loc; /* LOKUJ SEGMENTY W OBSZARZE ADRESOWYM 8086 */
rewrite(wy,plwy,,len); /* OTWORZ PLIK WY */
IF len=-1
 THEN
  BEGIN
    error1:=plwy;
    skok1 (20)
  END;
wyopen:=true;
/* 2 PRZEBIEG ORL - GENERACJA KODU BINARNEGO ABSOLUTNEGO */
nrmodwe:=1;
write(wy,chr(72B));     (* WYPROWADZ EXTENDED ADDRESS RECORD (0) *)
chksum:=0;
outbyte(2);
outbyte(0);
outbyte(0);
outbyte(2);
outbyte(0);
outbyte(0);
outbyte(-chksum);
writeln(wy);
plikptr:=obiekty;
WHILE plikptr<>NIL DO
 BEGIN
   plik:=plikptr^.spec;
   reset(we,plik,,len);
   IF len=-1
    THEN
     BEGIN
       error1:=plik;
       skok1 (20)
     END;
   mod2;
   close(we);
   nrmodwe:=nrmodwe+1;
   plikptr:=plikptr^.next
 END;
write(wy,chr(72B));              (* WYPROWADZ END OF FILE RECORD *)
FOR len:=1 TO 3 DO
outbyte(0);
outbyte(1);
outbyte(377B);
writeln(wy);
IF listopen
 THEN /* DRUK MAPY PAMIECI */
 map;

 /* ZAKONCZENIE PROGAMU */
1:
```

```
  IF listopen
   THEN
   close(lst);
  IF wyopen
   THEN
   close(wy)
END.
```