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

Ośrodek Automatyzacji Procesów Produkcji (OAP)

440 BE10

Główny wykonawca mgr inż. Kazimierz Maliszewski

Wykonawcy

Konsultant

Nr zlecenia 9557 /OAP

Dezasembler instrukcji procesorów Intel 8086/88/186/188.

Zleceniodawca Praca własna OAP.

Prace rozpoczęto dnia Formalnie 1990.06.01 Faktycznie VII.89

Formalnie zakończono dnia 90.11.19 Faktycznie I.91

Z-ca Dyrektora d/s Badawczo-Rozwojowych
dr inż. Jl. Jabłkowski

Kierownik Ośrodka
dr inż. M. Wrzesień

Praca zawiera:

stron 8

rysunków

fotografii

tabel

tablic

załączników 11/9 tasiemka w 30/1/1

Nr rejestr. 6548

Rozdzielnik - ilość egz: 5

Egz. 1 Archiwum – BOINTE

Egz. 2 ~~BOINTE~~

Egz. 3 OAE

Egz. 4 OAP

Egz. 5 OAP

Egz. 6

Analiza deskryptorowa

DEZASEMBLER, MIKROPROCESORY, OPROGRAMOWANIE

Analiza dokumentacyjna

Tytuły poprzednich sprawozdań

681.32:621.377-181.4B Mikroprocesory -
621.3.06 Oprogramowanie

UKD

MERA-PIAP/TW 331/78 5000

SPIS TRESCI

ZAŁĄCZNIKI

1. Zbiór DASM86.TXT - opis dezasemblera 8086 w języku angielskim.
2. Zbiór DASM186.TXT - opis dezasemblera 80186 w języku angielskim.
3. Zbiór DASM86.ASM - program dezasemblera 8086 (tylko w egz. 1 i 5).
4. Zbiór DASM186.ASM - program dezasemblera 80186 (tylko w egz. 1 i 5).
5. Lista instrukcji mikroprocesora Intel 8086/88 utworzona przez DASM86.
6. Lista instrukcji mikroprocesora Intel 80186/188 utworzona przez DASM186.
7. Listing przykładowego programu zawierający instrukcje 80186/188 (fragment).
8. Zawartość pamięci programu z zał. 7 wyprowadzona przez DASM186.
9. Przykładowe użycie DASM86 z maską dopuszczającą listowanie wyłącznie kodów procesorów 80186/188 i 80286.
10. Sledzenie przez DASM186 wykonania fragmentu programu z zał. 7.
11. Sledzenie przez DASM86 wykonania pewnego programu w trybie "Skok" bez wydruku rejestrów.

1. WPROWADZENIE

Dezasembler jest programem, który na podstawie adresu pamięci operacyjnej dekoduje i wyświetla na ekranie lub drukuje instrukcję pamiętaną pod tym adresem.

W ramach niniejszej pracy wykonano dwa programy dezasemblera: dla mikroprocesorów Intel 8086/8088 i dla mikroprocesorów Intel 80186/80188. Są to dwie pary zbiorów systemu IBM-DOS o nazwach odpowiednio:
DASM86.TXT (opis), DASM86.ASM (program), oraz
DASM186.TXT (opis), DASM186.ASM (program).

Niniejsze sprawozdanie dotyczy wersji nr 2 obydwu dezasemblerów. Dokumentacją wykonanej pracy są tabulogramy programów (zbiorów ".ASM"), które podano w załącznikach. Sprawozdanie jest opisem użytkowym, przekazującym głównie treść zbiorów ".TXT". Opis dotyczy obu programów, poza wyraźnie zaznaczonymi różnicami. Nazwa "dezasembler" bez numeru oznacza każdą z wersji.

Dezasembler jest zbiorem procedur napisanych w asemblerze procesora Intel 8086. Spośród nich dwie są przeznaczone do wywołania z zewnątrz dla wykonania dezasemblacji: unass_ (faktyczny dezasembler) i unin_ (inicjator). Występujące w tych i innych nazwach dolne kreski pochodzą z konwencji oznaczania nazw globalnych przez kompilator języka AZTEC C. Obie procedury mogą być wywołane w języku AZTEC C i wtedy należy użyć nazw bez kresek. W przypadku wywolania z języka asemblera trzeba użyć nazw z kreskami.

Dezasembler może pracować w dwóch zasadniczych trybach: statycznym - gdy jest wywołany dla dezasemblacji instrukcji bez związku z jej wykonaniem, i dynamicznym - gdy jest wywołany dla dezasemblacji instrukcji, która ma się właśnie wykonać. W drugim przypadku można uzyskać więcej informacji.

Instrukcja, która ma być dezasemblowana, będzie w niniejszym tekście nazywana bieżącą instrukcją. Instrukcja procesora 8086 lub 80186 składa się z podstawowej czyli głównej instrukcji (która może istnieć samodzielnie) i być może z jednego lub więcej prefiksów, które poprzedzają podstawową instrukcję. Zarówno prefiksy jak i podstawowe instrukcje będą nazywane elementarnymi instrukcjami.

2. WYKORZYSTANIE REJESTROW I PAMIĘCI

Dezasembler został napisany w "małym" modelu pamięci procesora 8086. To znaczy:

* Dezasembler definiuje tylko jeden segment kodu, o nazwie codeseg, i jeden segment danych zmiennych, o nazwie dataseg. Stałe dezasemblera są pamietane w codeseg.
* Wywołania procedur unass_ i unin_ muszą być bliskie (NEAR) w ramach codeseg.
* Rejstr DS przy wejściu do unass_ musi zawierać adres segmentu danych zmiennych programu wywołującego, o nazwie dataseg.

Niniejsza wersja dezasemblera zajmuje około 4kB w obszarze codeseg i około 130B w obszarze dataseg.

Dezasembler używa stosu programu wywołującego, adresowanego bieżącymi wartościami rejestrów SS:SP. Dezasembler wymaga około 30 słów stosu (ta liczba zawiera rozsądny margines).

Dezasembler może być pamiętany w pamięci przeznaczonej tylko do odczytu. Jego zmienne są inicjowane albo przez unin_, albo przy każdym wywołaniu unass_.

Dezasembler nie może być równocześnie używany przez programy współbieżne.

Przy powrocie z unass_ zawartość rejestrów CS,DS,ES,SS,SP i BP pozostaje nie zmieniona. Jest to więcej niż konieczne dla programu wywołującego napisanego w języku AZTEC C. Zawartość pozostałych rejestrów trzeba uważać za zniszczoną.

Przy powrocie z unin_ tylko zawartość AX jest zniszczona.

3. SPRZĘZENIE Z PROGRAMEM WYWOŁUJĄCYM

Zakłada się, że dezasembler będzie włączony do programu uruchomieniowego (debuggera) lub programu monitora mikrokomputera i będzie przez ten program wywoływany. Dezasembler został zaprojektowany dla monitora systemu operacyjnego SIRTOS, który napisano w języku AZTEC C. W dalszym opisie termin "monitor"

oznacza program wywołujący.
Monitor, tak jak dezasembler, musi definiować segmenty dataseg i codeseg, które należy połączyć z segmentami dezasemblera.
Dezasembler używa następujących zewnętrznych (EXTRN) obiektów, które monitor musi zdefiniować jako publiczne (PUBLIC):

1) Dane w segmencie dataseg, adresowanym wartością rejestru DS:

registers (tablica typu słowowego) przy wywołaniu unass_ w trybie dynamicznym musi zawierać następującą sekwencję zawartości rejestrów (14 słów):
AX,BX,CX,DX,DI,SI,BP,SP,SS,ES,DS,CS,IP,flagi.
Musi ona określać stan programu śledzonego tuż przed wykonaniem bieżącej instrukcji (wskazywanej przez CS:IP).

segm_ (zmienna słowowa) przy wywołaniu unass_ w trybie statycznym musi zawierać adres segmentu bieżącej instrukcji.

offs_ (zmienna słowowa) w trybie statycznym musi zawierać offset (adres względny) bieżącej instrukcji względem wartości segm_.

hnlsq_ (sekwencja bajtowa) musi zawierać ciąg znaków, który, jeżeli go wysłać na urządzenie wyjściowe dezasemblacji, powoduje wykonanie przez nie "twardego nowego wiersza". hnlsq_ musi zawierać jeden znak wysuwu wiersza (LF) i musi kończyć się zerem (znakiem null).
hnlsq_ powinna pozycjonować kursor w określonym miejscu ekranu, najlepiej w lewym dolnym rogu, niezależnie od tego, gdzie mógł go umieścić śledzony program (lub inny współbieżny). Następnie obraz na ekranie powinien być podniesiony o jeden wiersz.
Dla monitora ekranowego MERA 7953 N (produkcji MERA ELZAB) sekwencja hnlsq_ może być zdefiniowana w języku asemblera następująco:

```
hnlsq_  DB     32  , 32  , 27 , 89, 55, 32   , 13, 10,  0
;tj. ASCII   odstęp odstęp Esc   Y   7 odstęp CR  LF  null
```

Dwa wiodące odstępy są przeznaczone na "połknięcie" przez inną sekwencję Esc, być może przerwaną przez nasz program. Esc 'Y7 ' pozycjonuje kursor w wierszu 23 (55-32), kolumnie 0 (32-32). CR (powrót karetki), tu zbędny, jest dla końcówki, która nie zrozumiała Esc 'Y7 '. LF podnosi obraz na ekranie.
Jeżeli urządzenie wyjściowe nie pozwala na pozycjonowanie kursora, to hnlsq_ powinna być zredukowana do "zwykłego nowego wiersza", tj. CR,LF,0.
W trybie dynamicznym dezasembler wysyła "twardy nowy wiersz" na początku każdej wyświetlanej instrukcji. Jeżeli śledzony program także wysyła znaki do tej samej końcówki, zostaną one umieszczone w tym pustym wierszu.
Ten prowadzący wiersz jest zliczany jako 1 w ulinct_ (zob. niżej). Jeżeli hnlsq_ zawiera ilość znaków LF różną od 1, użytkownik musi odpowiednio interpretować ulinct_.
W trybie statycznym sekwencja hnlsq_ nie jest używana.

2) Zawarta w codeseg procedura wstring_, która może być wywołana w języku C.

wstring_ (ptr) powinna być bliską (NEAR) procedurą, która pisze łańcuch znaków na wyjściowe urządzenie dezasemblacji.
ptr jest słowowym argumentem na stosie. Jest to offset łańcucha względem bieżącej wartości DS. Przy wejściu do wstring_ unass_ może nadać rejestrowi DS adres segmentu dataseg lub codeseg. Wobec tego procedura wstring_ nie może zawierać dostępów do własnych zmiennych w dataseg przy użyciu wejściowej wartości DS.
Łańcuch może zawierać dowolne znaki i kończy się zerem (znakiem null).
Dezasembler pisze wyłącznie albo hnlsq_, albo pełne wiersze,

zakończone przez LF,CR,0. W niniejszej wersji najdłuższy wyprowadzany wiersz ma 80 znaków, włączając kończące 0. Zgodnie z konwencją języka C, przy powrocie z wstring_ argument musi pozostać na stosie (jego wartość wyjściowa jest nieistotna). Zawrtość rejestrów CS,SS,SP i BP musi być zachowana, pozostałych może być zniszczona.

Dezasembler definiuje następujące dane publiczne (PUBLIC), zawarte w dataseg, które są w istocie jego argumentami wyjściowymi:

ulinct_ (zmienna słowowa) zawiera liczbę wierszy wyprowadzonych podczas ostatniego wywołania unass_.

unbuff_ (tablica bajtowa) zawiera ostatnio zdezasemblowaną podstawową instrukcję (nie prefiks) w znakowej postaci ASCII. unbuff_ ma długość 80 bajtów. Użyteczna informacja kończy się znakiem LF (dziesiętnie 10). Nazwa instrukcji zaczyna się w unbuff_+23.

4. POSTAC WYWOŁANIA

Przykładowa postać wywołania dezasemblera w języku asemblera 8086 jest następująca.

1) INICJACJA

```
CALL unnin_      ;bez argumentów
```

Ten podprogram powinien być wywołany tylko raz, przy inicjacji monitora lub po załadowaniu dezasemblera (jeżeli dezasembler jest nierezydentny). Dezasembler inicjalizuje pewne swoje zmienne.

2) DEZASEMBLACJA

```
PUSH tryb        ;tryb=PRZEBIEG, tryb+1=MASKAI
CALL unass_      ;wejście = słowo na stosie, wyjście = rejestr AX
POP  CX          ;usuń argument
MOV  dlugosci,AX
```

Oczywiście, obecność i postać instrukcji PUSH, POP i MOV jest czysto symboliczna.

Dane wejściowe

Niejawne dane wejściowe powinny być dostarczone bądź przez parę segm_:offs_, bądź przez tablicę _registers_. Sekwencja hnlsq_ również musi być zdefiniowana.
Jawną daną wejściową jest słowowy argument na stosie - TRYB. TRYB sklada się z dwóch 1-bajtowych argumentów binarnych: mniej znaczącego (o niższym adresie) PRZEBIEG i bardziej znaczącego MASKAI. Razem określają one tryb, w jakim dezasembler będzie przetwarzał bieżącą instrukcję. Ich znaczenie jest następujące:

PRZEBIEG na swych bitach 0 i 1 definiuje dwie flagi binarne, które będziemy nazywać DYN (bit 0) i REJ (bit 1).
Bity 2-7 PRZEBIEG-u nie są używane i powinny być wyzerowane przez program wywołujący.

DYN=0 (tryb statyczny, "post-mortem")
unass_ dezasembluje i wyświetla bieżącą instrukcję, jeżeli nie jest zamaskowana (zob. dalej) i zwraca długość instrukcji.
Ten tryb jest przeznaczony do wyświetlania/wydruku pamięci programu.

DYN=1 (tryb dynamiczny)
Jeżeli bieżąca instrukcja nie jest zamaskowana (zob. dalej), unass_

wyświetla bieżącą zawartość rejestrów programu (zakładając, że REJ=1), następnie bieżącą instrukcję i dodatkowo miejsca pamięci, do których instrukcja się odnosi (jeżeli takie są) lub inną informację, np. ostrzeżenie o możliwym błędzie. Wszystkie wyświetlane dane opisują stan programu bezpośrednio przed wykonaniem instrukcji.
Ten tryb jest przeznaczony do śledzenia wykonania programu.

REJ=0 Rejestry programu nie są wyświetlane.

REJ=1 W trybie dynamicznym i jeżeli bieżąca instrukcja nie jest zamaskowana, rejestry programu wzięte z _registers_ są wyświetlane.

Flaga REJ jest nieistotna w trybie statycznym - rejestry nigdy nie są wyświetlane.

MASKAI określa na swych bitach rodzaje instrukcji, które mają być wyświetlane zarówno w trybie statycznym jak i dynamicznym.

MASKAI = 0 oznacza, że mają być wyświetlane wszystkie instrukcje w postaci określonej przez PRZEBIEG.

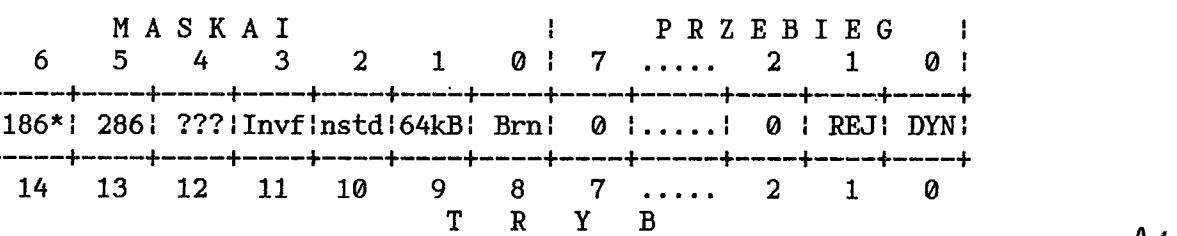
MASKAI # 0 oznacza, że bieżąca instrukcja ma być wyświetlana (wg PRZEBIEG) tylko wtedy, gdy należy do klasy wskazanej przez którykolwiek z bitów maski równy 1.

Klasy instrukcji są przypisane do bitów MASKAI następująco:

Bit	Nazwa	Zawrtość klasy
0	Brn	Instrukcje skokowe (ang. branch) tj. takie, które zmieniają sekwencję wykonania: skoki (bezwarunkowe lub warunkowe), instrukcje typu LOOP, instrukcje wywołania i powrotu z procedury, przerwania i powrotu z przerwania.
1	64kB	Instrukcje, które przecinają lub osiągają granicę bloku 64kB.
2	nstd	Instrukcje niestandardowe
3	Invf	Instrukcje o niedozwolonej postaci (ang. invalid form)
4	???	Nie używane kody instrukcji
5	286	Instrukcje mikroprocesora 80286 (iAPX 286)
6	186	(dezasembler 8086) instrukcje mikroprocesorów 80186/80188
6	LEAV	(dezasembler 80186) instrukcja LEAVE w trybie dynamicznym gdy BP<SP
7	None	Pusta klasa (bez wyświetlania).

Bity 1 - 6 reprezentują "podejrzane" instrukcje, które mogą być, ale niekoniecznie są błędne. Określono je w UWADZE 2.

Poniższy rysunek przedstawia słowo trybu dezasemblera 8086.



* - LEAV dla dezasemblera 80186.

Dane wyjściowe

Bieżąca instrukcja, jeżeli nie jest zamaskowana, jest wyświetlana lub drukowana na wyjściowym urządzeniu dezasemblacji.

Przy powrocie z unass_ w każdym prawidłowym trybie:

rejestr AX zawiera pełną długość instrukcji (włączając ewentualne prefiksy),
ulinct_ zawiera ilość wierszy wyprowadzonych w czasie tego wywołania,
unbuff_ zawiera podstawową instrukcję w znakowej postaci ASCII (niezależnie od tego, czy instrukcja była wyświetlana, czy nie).

Jeżeli żądany tryb jest niedopuszczalny (nie używane bity w TRYB są różne od zera), unass_ nie robi nic użytecznego i zwraca AX=0. ulinct_ i unbuff_ pozostają nie zmienione.

Wejściowy argument TRYB pozostaje na stosie zgodnie z konwencją wywołań w C.

UWAGA 1 (Wywołanie unass_ w trybie statycznym)

Zakłada się, że program, który wywołuje unass_ z DYN=0, wykorzysta zwracaną w AX długość instrukcji. Jeżeli trzeba zdezasemblować więcej niż jedną instrukcję, program wywołujący powinien dodać długość do bieżącego wskaźnika instrukcji, zawartego w offs_, wskazując w ten sposób na następną instrukcję w pamięci.

W trybach dynamicznych (DYN=1) adresy dezasemblowanych instrukcji są zwykle generowane przez układ przerwań procesora.

UWAGA 2 ("Podejrzane" instrukcje i ich diagnostyka)
---

1) Limit bloku 64kB

Dezasembler pobiera instrukcje z pamięci bajtami. Jeżeli instrukcja jest przecięta przez granicę 64kB segmentu kodu, to reszta jest brana z początku tego samego segmentu. Gdy dezasembler pobiera w trybie dynamicznym dane, to redukuje o 64k (zawija) offsety, które osiągają lub przekraczają 64k.
Słowa danych są brane słowami (nie bajtami) i zależy od procesora (8086/8088, 80186/80188 lub 80286), co się stanie w przypadku dostępu do słowa o offsecie FFFFH. Także różne procesory mogą różnie reagować na instrukcje przecięte przez granicę 64kB.

Jeżeli dezasembler, pracując w dowolnym trybie, stwierdzi, że bieżąca instrukcja narusza granicę segmentu 64kB, to najpierw wyprowadza ostrzegawczy komunikat:
'****** 64kB limit. Uncertain result:' (tj. 'Granica 64kB. Niepewny wynik:'),
a następnie, jeżeli w ogóle jeszcze działa, "podejrzaną" instrukcję.
W trybach bez maskowania komunikat bezpośrednio poprzedza wiersz instrukcji.
W dynamicznych trybach z maskowaniem komunikat poprzedza rejestry (jeżeli mają być drukowane) i wiersz instrukcji.
Postać komunikatu nie zależy od tego, którego segmentu granica jest przecięta, tj. czy jest to segment kodu, danych, stosu czy dodatkowy, czy być może więcej niż jeden.

Granica 64kB jest sygnalizowana przez dezasembler, jeżeli występuje za instrukcją lub w środku wielobajtowej instrukcji lub danej (nie za daną).
W dwóch przypadkach granica bloku 64kB jest sygnalizowana przed daną (zakłada się tryb dynamiczny):

a) jeżeli offset danej obliczony jako suma trzech dodatnich składników, tj. (baza + indeks + przemieszczenie) jest >= 64k,
b) w przypadku instrukcji XLAT, jeżeli dodanie rejestrów BX i AL programu śledzonego daje przeniesienie (gdyż wartość BX, która jest adresem tablicy translacji, jeżeli jest bliska 64k, nie jest uważana za ujemną).

Granica 64kB jest także sygnalizowana dla instrukcji stosowych, a mianowicie

a) jeżeli instrukcja typu PUSH zamierza złożyć na stosie więcej bajtów niż jest miejsca do bazy stosu (tj. więcej niż wynosi bieżąca wartość SP),
b) jeżeli instrukcja typu POP zamierza zdjąć więcej bajtów niż jest aktualnie

na stosie (SP ma wzrosnąć ponad 64k).

Specjalny przypadek ma miejsce, jeżeli program zamierza wykonać instrukcję typu PUSH (PUSH, PUSHF, CALL, INT lub INTO) gdy SP użytkownika ma wartość 0. Standardowy komunikat, podany wyżej, powinien być rozumiany następująco:

a) Jeżeli SP jest rzeczywiście na początku segmentu stosu, ostrzega się użytkownika, że straci cały stos i pierwsza "wepchnięta" wartość pójdzie do słowa o adresie SS:FFFF.
b) Jeżeli SP=0 faktycznie oznacza 10000H (dno 64k-bajtowego segmentu stosu), to instrukcja jest poprawna, ale nasz wydruk zawrtości szczytu stosu wziętej z SS:0 - oczywiście nie i ostrzeżenie powinno być odniesione do niego.

Przypadek granicy 64kB segmentu kodu po elementarnej instrukcji (a więc też po prefiksie) jest sygnalizowany inaczej. Komunikat
'------ 64kB CS limit here ------' (tj. 'Tutaj granica 64kB segmentu kodu') jest wyświetlany za instrukcją i należy go zignorować, jeżeli chodzi o leżący na końcu bloku efektywny skok (w sensie podanym w opisie MASKAI) bez powrotu. Wtedy w przypadku skoku krótkiego trzeba skontrolować adres docelowy, który jest być może znacznie dalej niż w odległości 127B.

2) Instrukcje niestandardowe

Nazwy niektórych instrukcji mogą być listowane z przyrostkiem 'nstd'. Pochodzą one z kodów, które w zasadzie nie powinny być generowane przez kompilator lub asembler, ponieważ są inne kody, które wykonują ich funkcje.
Stwierdzono jednak, że wykonują się one poprawnie na procesorze 8086/8088, tak jak ich standardowe odpowiedniki. Jest to tylko reguła doświadczalna. Na IBM z procesorem 80286 wiele spośród niestandardowych kodów blokuje komputer. Wykonanie niestandardowych instrukcji na procesorze 80186/188 nie było badane.

3) Instrukcje o niedopuszczalnej postaci

Niektóre instrukcje mogą być oznaczone komentarzem 'Invalid form' ('Nieprawidłowa postać'). Jest to poważne ostrzeżenie. Wykonanie takich instrukcji jest niezdefiniowane. Chodzi przede wszystkim o instrukcje, które są poprawne tylko z operandem odnoszącym się do pamięci, a nie do rejestru, np. pośrednie dalekie skoki.

4) Nie używane kody instrukcji

Kody nie używane przez procesory 8086/88, 80186/188 i 80286 są dezasemblowane jako jednobajtowe instrukcje o nazwie ???, bez operanda.
Jeżeli kod w pierwszym bajcie jest prawidłowy, ale jego przedłużenie w drugim bajcie (kod wewnętrzny, KW), ma wartość nie używaną, to nazwa otrzymuje postać ???, a operand ma formę wynikającą z drugiego i być może dalszych bajtów.
W rzeczywistości nawet nie używane instrukcje mogą się w jakiś sposób wykonywać. Dla 80286 okazało się np. że kod F6 z nie używanym KW=1 jest niestandardową postacią instrukcji TEST (F6 z KW=0). Podobnie dla F7, KW=1 i 0. Użytkownik może eksperymentować.

5) Instrukcje mikroprocesora 80286 (iAPX 286)

Klasa obejmuje instrukcje specyficzne dla procesora 80286. Procesor ten wykonuje ponadto wszystkie instrukcje procesorów 80186/80188 i 8086/8088.
Dezasemblacja instrukcji specyficznych dla procesora 80286 ogranicza się do pierwszego bajtu z nazwą instrukcji '286code' i komentarzem 'If so, 2-5 bytes' (tj. 'Jeżeli tak, 2-5 bajtów').
Przy wykonaniu na procesorach 8086/88 lub 80186/188 instrukcje te powinny być traktowane jak nie używane, opisane wyżej.

6) (Dezasembler 8086) Instrukcje procesorów 80186/80188

Klasa obejmuje instrukcje specyficzne dla procesorów 80186/80188, czyli

niedostępne na 8086/88 (ale dostępne na 80286). Są one zwykle dezasemblowane w postaci jednego bajtu z nazwą '186code' i komentarzem 'Possibly >1 byte' (tj. 'Być może >1 bajt'). W pewnych przypadkach po nazwie '186code' występuje prawidłowy operand otrzymany na podstawie dalszych bajtów instrukcji.
Przy wykonaniu na procesorze 8086/88 instrukcje te powinny być traktowane jak nie używane, opisane w p. 4).

6') (Dezasembler 80186) Instrukcja LEAVE w trybie dynamicznym gdy BP<SP

Specyficzna dla procesorów 80186/80188 instrukcja LEAVE normalnie zwalnia stos od bieżącej wartości SP do bieżącej wartości BP (która powinna wskazywać na pozycję położoną głębiej na stosie, tj. o większym lub co najmniej równym adresie), a następnie zdejmuje ze stosu rejestr BP. Jeżeli warunek BP>=SP jest niespełniony w chwili, gdy LEAVE ma się wykonać, dezasembler umieszcza w linii instrukcji następujący komentarz:
'Warning: BP<SP' (tj. 'Ostrzeżenie: BP<SP').

UWAGA 3 (Użycie argumentu MASKAI)

Zalecany sposób użycia argumentu MASKAI jest następujący:

A. MASKAI = 0 ("Post-Mortem" lub "Sledzenie")
Ta maska, a właściwie brak maski, powoduje wyświetlanie wszystkich instrukcji tak w trybie statycznym (PRZEBIEG=0) jak i dynamicznym (PRZEBIEG = 1 lub 3). Jest to zasadniczy sposób wykorzystania dezasemblera.

B. MASKAI = 7FH ("Skok")
Ta maska powoduje wyświetlanie tylko instrukcji w szerokim sensie skokowych i być może "podejrzanych" (te ostatnie będą przypuszczalnie bardzo rzadkie). Ta maska w trybie dynamicznym jest przeznaczona do śledzenia przepływu sterowania w programie.

C. MASKAI = 1 ("Lista skoków")
Ta maska może być użyta w trybie statycznym do wylistowania wszystkich instrukcji skokowych przed startem w trybie "Skok".
Użycie MASKAI=1 w trybie dynamicznym nie jest zalecane, gdyż ewntualna "podejrzana" instrukcja może mieć poważny wpływ na wykonanie programu, włącznie ze zmianą sekwencji wykonania. Jednak zob. p. E dalej.

D. MASKAI = 7EH ("Test")
Ta maska w trybie dynamicznym powoduje ciche, ale kontrolowane wykonanie programu, w którym listuje się tylko "podejrzane" instrukcje.
Sumienny użytkownik może w ten sposób badać nowy lub uszkodzony program. Poszukiwanie "podejrzanych" instrukcji w trybie statycznym może mieć pewien sens po upadku programu - byłaby to prawdziwa analiza "pośmiertna"! Trzeba tylko uważać, żeby nie wziąć danych, jeżeli są przemieszane z instrukcjami, za złe instrukcje (to dotyczy także pełnego wydruku statycznego z p. A, ale wtedy łatwiej wyróżnić dane przez kontekst).
W sumie ta maska nie wydaje się bardzo przydatna, bo po pierwsze - programy utworzone przez standardowe kompilatory nie będą miały wcale "podejrzanych" instrukcji, a po drugie - program może zostać uszkodzony na wiele sposobów bez utworzenia "podejrzanych" instrukcji.

E. Zamaskowanie instrukcji 186, 286, niestandardowych lub nawet nie używanych w trybie "Skok" lub "Test" może być użyteczne, jeżeli wykorzystuje się procesor wyższego rzędu lub szczególny kompilator czy asembler, który generuje niestandardowe, ale poprawne instrukcje.

F. MASKAI = 80H ("Bez wyświetlania")
Dezasembler nie wyświetla żadnych instrukcji, a tylko wypełnia unbuff_ i nadaje wartości wyjściowe AX i ulinct_ (=0).
Użytkownik może działać w dowolny sposób.

UWAGA 4 (Produkty uboczne dezasemblera)

Dezasembler definiuje w codeseg_ dwie bliskie (NEAR) procedury do konwersji binarno - heksadecymalnej, które mogą być wywołane w języku AZTEC C.

uhex2_ (h2,pi) jest podprogramem, który przetwarza mniej znaczący bajt (h2) słowa stosu na dwa znaki szesnastkowe w kodzie ASCII w DS:pi.
pi jest bliskim (NEAR) wskaźnikiem wyniku, tj. offsetem zmiennej w segmencie adresowanym rejestrem DS. DS może zawierać dowolną wartość.
Przy wejściu do uhex2_ zawatrość szczytu stosu musi być następująca:
0..... IP, xxh2, pi.

uhex4_ (h4,pi) jest podprogramem, który przetwarza słowo (h4) ze stosu na 4 znaki szesnastkowe w kodzie ASCII w DS:pi.
pi i DS mają znaczenie jak dla uhex2_.
Przy wejściu do uhex4_ zawatrość szczytu stosu musi być następująca:
0..... IP, h4, pi.

Obie procedury niszczą zawartość rejestrów AL, BX, CX, DX i DI. Inne rejestry pozostają nie zmienione. Argumenty pozostają nie zmienione na stosie.

Obie procedury mogą być także wywołane w asemblerze. Np. uhex2_ może być wywołana następująco:

```
MOV AX, OFFSET pi   ;Adres przeznaczenia
PUSH AX
MOV AL, h2          ;Bajt do konwersji
PUSH AX
CALL uhex2_         ;Wynik w  DS:pi,pi+1. AL,BX,CX,DX,DI zniszczone
ADD SP, 4           ;Usuń argumenty ze stosu
```

*** KONIEC ***

Załącznik nr 1 — Opis dezasemblera 8086
Zbiór DASM86.TXT

Last update: 17.XII.1990

THE 8086 DISASSEMBLER. Version 2

The 8086 Disassembler is a program that, given a memory address, decodes and displays or prints an Intel 8086/8088 instruction stored at that address.
The Disassembler is a set of procedures written in Intel 8086 assembly language, of which two are to be externally called for the purpose of disassembly: unass_, the actual disassembler, and unin_, the initiator. Both procedures can be called in AZTEC C or 8086 assembly language.
The Disassembler can work in two principal modes: static, when it is required to disassemble an instruction of memory contents, and dynamic, when it is called to disassemble an instruction which is about to execute. In the second case more information may be obtained.
The instruction to be disassembled will be called here the current instruction. An 8086 instruction consists of a basic or main instruction (that can exist independently) and possibly of one or more prefixes, that precede the basic instruction. Both prefixes and basic instructions are termed here elementary instructions.

MEMORY AND REGISTER USE

In this version the "small" memory model applies. This means:
* The Disassembler defines only one code segment, named codeseg, and one variable data segment, named dataseg. Disassembler constants are stored within codeseg.
* Calls to unin_ and unass_ must be NEAR calls within codeseg.
* DS register on entry to the unass_ must address calling program's variable data segment, named dataseg.

This version of the Disassembler occupies about 4kB memory in codeseg and about 130B in dataseg.
The Disassembler uses the calling program's stack, addressed with current SS:SP values. It requires about 30 words on stack (this includes a reasonable margin).
The Disassembler may be stored in read-only memory. The Disassembler's variables are initialized either by the unin_ or each time the unass_ is called.
The Disassembler must not be simultaneously used by concurrent programs.
On return from the unass_ the CS,DS,ES,SS,SP and BP register contents are unchanged. It is more than necessary for a caller program assumed in AZTEC C language. Other register contents should be considered destroyed.
On return from the unin_ only AX contents are destroyed.

INTERFACE TO THE CALLING PROGRAM

The Disassembler is assumed to be linked to and invoked by a Debugger or a microcomputer's Monitor program. It has been designed for the SIRTOS Monitor, which was written in AZTEC C. In all that follows the term "Monitor" means the calling program.
The Monitor too must define dataseg and codeseg segments, which are to be combined with the Disassembler's.
The Disassembler uses the following external objects, which the Monitor must supply as PUBLICs:

1. Data in the dataseg addressed with DS register value:

registers (WORD array) when unass_ is called in a dynamic mode, must contain this sequence of 14 register contents: AX,BX,CX,DX,DI,SI,BP,SP,SS,ES,DS,CS,IP,flags. This must define the traced program state just before the current instruction (pointed to with CS:IP) is executed.

segm_ (WORD variable) when unass_ is called in static mode, must contain the current instruction's segment address.

offs_ (WORD variable) in static mode must contain the current instruction's offset relatively to segm_ value.

hnlsq_ (BYTE sequence) must contain a sequence of characters that when sent to the disassembly output device makes it do a "hard new line". The hnlsq_ must contain one line feed (LF) character and it must end with a 0 (null byte).
The hnlsq_ should position the cursor in a definite spot of the screen, preferably in the bottom left hand corner, independently of where the traced program (or a concurrent one) might have put it. Following that the screen image should be pushed one line up.
For MERA 7953 N screen display monitor (manufactured by MERA-ELZAB) the hnlsq_ may be defined in assembly language as:

```
hnlsq_  DB    32,    32,   27, 89, 55,  32,   13, 10, 0
```

	I M A S K									R U N			
	7	6	5	4	3	2	1	0	7	2	1	0
	None	186	286	???	Invf	nstd	64kB	Brn	0	0	REGS	DYN
	15	14	13	12	11	10	9	8	7	2	1	0
							M	O	D E			

Procedure output

The current instruction, if it isn't masked out, is displayed on disassembly output device.
On return from the unass_ in any correct mode
AX register specifies the full length of the current instruction (including any prefixes),
ulinct_ contains the number of lines having been displayed by this call, and
unbuff_ contains the basic instruction in readable ASCII form.
The unbuff_ is filled whether the instruction was displayed or not.
If the mode required is inadmissible (not used bits in MODE are not equal to 0) the unass_ does nothing useful and returns AX=0. ulinct_ and unbuff_ are unchanged.
The input MODE argument remains on stack according to language C calling convention.

NOTE 1 (Calling The unass_ In The Static Mode)

A program calling the unass_ with DYN=0 is assumed to make use of the returned instruction length. If multiple instructions are to be disassembled, the caller should add the length to the current instruction pointer, contained in offs_, thus pointing to a following instruction in memory.
In dynamic modes (DYN=1) the addresses of executed instructions are normally generated by processor's interrupt system.

NOTE 2 ("Suspicious" Instructions And Their Diagnostics)

1. 64kB block limit

The Disassembler fetches instructions on byte by byte basis. If an instruction is crossed by 64kB code segment limit, the rest is fetched from the beginning of the same segment. When fetching data in a dynamic mode, the Disassembler reduces by 64k (wraps) offsets that attain or exceed 64k. Data words are fetched in words (not in bytes) and it depends on the processor (8086/8088, 80186/80188 or 80286) what happens if a word is accessed at offset FFFFH. Also, different processors may differently react on instructions crossed by 64kB boundary.
If the Disassembler, working in any mode, finds that the current instruction violates a 64kB segment limit it first displays the warning message:

'****** 64kB limit. Uncertain result:'

and then (if it is still working at all) the offending instruction. In standard uses (without masking) the message immediately precedes the instruction line. In masked dynamic modes the message precedes the registers (if required) and the instruction line.
The form of the message does not depend of which segment boundary is crossed, i.e. if it is a code, data, stack or extra segment, or possibly more than one.

The 64kB boundary may be discovered by the Disassembler after an instruction or in the middle of a multibyte instruction or datum (not after a datum). In two cases the 64k block limit is signalled before a datum (a dynamic mode is assumed):
1) if an offset computed as a sum of three positive components i.e. base + index + displacement is >= 64k, and
2) in the case of XLAT instruction if the addition of user's BX and AL registers yields a carry (as BX value being translate table address, if near to 64k, is not taken for negative).

64kB limit is also signalled for stack operations, namely if a PUSH-type instruction is going to push on the stack more bytes that there is the space left to the stack segment base (i.e. more than is the current SP value), or if a POP-type instruction is about to take more bytes than there are on the stack up to its 64kB limit (SP is going to rise above 64k).
A special case exists if a program is going to execute a PUSH-type (PUSH, PUSHF, CALL, INT or INTO) instruction while user's SP=0. The standard message, quoted above, should be understood as follows:
a. If SP is really at the beginning of the stack segment, the user is warned that he/she is going to loose the whole stack and the first pushed value will go to (SS:FFFE) word.
b. If SP=0 means actually 10000H (the bottom of 64kB long stack segment) the instruction is correct, but our stack top contents display, taken from SS:0 is obviously not, and the warning should be referred to it.

12

The case of 64kB code segment limit after an elementary (prefix or main) instruction is signalled otherwise. The message

Two leading blanks are let to be "swallowed" by another Esc sequence possibly interrupted by our program. Esc 'Y7 ' positions the cursor in line 23 (55-32), column 0 (32-32). CR (carriage return), here superfluous, is for a terminal that didn't understand Esc 'Y7 '. LF pushes the screen picture.
If the output device doesn't allow for cursor positioning, the hnlsq_ should be reduced to a "plain new line", i.e. CR,LF,0.
In a dynamic mode the Disassembler issues a "hard new line" at the beginning of each displayed instruction. If the traced program also sends characters to the same terminal, they will be placed in this empty line.
This leading line is counted as 1 in ulinct_ (see below). If the hnlsq_ contains more or less LF characters than one, the user must accordingly interpret ulinct_.
In a static mode the hnlsq_ is not used.

2. C-callable wstring_ procedure contained in the codeseg.

wstring_ (ptr) shall be a NEAR procedure that writes a string of characters on the disassembly output device.
ptr is a word argument on stack. It is the string's offset relatively to the current DS value. DS register on input to wstring_ may be set by unass_ to either dataseg or codeseg segment address. Therefore the wstring_ procedure must not access its own variables in dataseg using input DS value.
The string may contain any characters and it ends with a 0 (null byte). The Disassembler writes exclusively either hnlsq_ or full lines, terminated with LF,CR,0.
In this version the longest output line has 80 bytes, including ending 0.
According to C language convention, on return from wstring_, the argument must remain on stack (its output value is irrelevant). The CS,SS,SP and BP register contents must be preserved, other may be destroyed.

The Disassembler defines the following PUBLIC data, contained in dataseg, which are actually its output arguments:

ulinct_ (WORD variable) set to the number of lines of display output produced by the last invocation of unass_;

unbuff_ (BYTE array) containing the last disassembled basic instruction (not a prefix) in ASCII character form. unbuff_ is 80 bytes long. Useful information ends with a LF character. Instruction name begins in unbuff_+23.

INVOCATION FORM

A possible form of calling the Disassembler in assembly language is as follows.

(1) INITIATION

```
CALL unin       ;No arguments
```

This should be called only once, during Monitor initiation (or after Disassembler loading if the Disassembler is nonresident). The Disassembler initializes some internal variables.

(2) DISASSEMBLY

```
PUSH mode       ;mode = RUN,   mode+1 = IMASK
CALL unass_     ;input = word arg on stack,  output = AX register
POP  CX         ;discard the argument
MOV  ilength,AX
```

Of course, the presence and form of PUSH, POP and MOV instructions is purely symbolic.

Procedure input

Hidden input data must be supplied in either segm_:offs_ pair or _registers_ array. hnlsq_ must too be defined.
The explicit input is a word argument on the stack. It will be called here MODE. The MODE consists of two 1-byte binary arguments: the less significant (of lower address) RUN, and the more significant IMASK. Together they specify the mode in which the Disassembler will process the current instruction. Their meaning is as follows:

RUN byte on its bits 0 and 1 defines two binary flags which we shall name DYN (bit 0) and REGS (bit 1).
Bits 2-7 of RUN are not used and must be set to 0 by the caller

DYN=0 (Static mode, "Post-Mortem")
The unass_ disassembles and displays the current instruction if it is not masked out (the description of masking follows), and returns instruction's length.
This mode is intended for a program memory display.

DYN=1 (Dynamic mode)
If the current instruction isn't masked out (see below) the unass_ displays the current values of the program's registers (assuming that REGS=1), then the current instruction,

is displayed after the instruction and it should be ignored if the instruction occupying last bytes of a 64kB block is an effective branch without return (for a short jump the user is advised to check the target address which may be much farther than 127 bytes away).

2. Nonstandard instructions

Some instruction names may occur with 'nstd' suffix. These origin from the codes which in principle should not be generated by a compiler or an assembler because there are other codes that perform their functions.
However they have been found to execute correctly on an 8086/8088, like their standard counterparts. It is only an experimental rule. On IBM with 80286 many of the nonstandard codes block the computer. The execution of nonstandard instructions on an 80186/80188 processor has not been checked.

3. Invalid form instructions

Some instructions may be marked with the 'Invalid form' comment. It's a strong warning. Execution of such instructions is undefined. E.g. indirect far jumps are valid only if the operand is a memory address, not a register reference.

4. Not used instruction codes

The codes not used by either 8086, 80186 or 80286 processor are disassembled as 1-byte instructions named ???, with no operand. If the code in first byte is correct, but its following in the 2nd byte (Internal Code, IC) is not used, the name gets the form '???', and the operand decoded from the 2nd and possibly further bytes is shown.
In fact, even not used instructions may execute in some way. For instance I have discovered that the code F6 together with a not used IC=1 is a nonstandard form of TEST instruction (F6 with IC=0). The same with F7, IC=1 vs 0. The user is free to experiment.

5. 80286 (iAPX 286) instructions

These are 80286-specific instructions. The 80286 processor executes additionally all 80186/80188 and 8086/8088 instructions.
The disassembly of 80286-specific instructions is limited to the first byte with instruction name saying '286code' and the comment: 'If so, 2-5 bytes'.
On an Intel 8086, 8088, 80186 or 80188 (iAPX 86, 88, 186 or 188) these codes should be treated as not used (see item 4 above).

6. 80186, 80188 (iAPX 186, 188) instructions

These are 80186/80188 - specific instructions, i.e. not available on 8086/8088 (but available on 80286). The 80186/80188 processor executes additionally all 8086/8088 instructions.

The disassembly of 80186/80188-specific instructions is usually limited to the first byte with instruction name saying '186code' and the comment 'Possibly >1 byte'.
In some cases, instead of the comment, the operand is shown, decoded from the second and possibly further bytes.
On an Intel 8086/8088 (iAPX 86/88) these codes should be treated as not used (see item 4 above).

NOTE 3 (Using The IMASK Argument)

The recommended use of the IMASK argument is the following.

A. IMASK = 0 ("Post-Mortem" or "Trace")
This causes displaying all instructions either in static (RUN=0) or dynamic mode (RUN = 1 or 3).
It is the principal use of the Disassembler.

B. IMASK = 7FH ("Branch")
This causes displaying only branch and possibly "suspicious" instructions (the last being probably extremely rare).
This mask in a dynamic mode (RUN = 1 or 3) is intended for program flow trace.

C. IMASK = 1 ("Branch list")
This in a static mode (RUN=0) may be used to list all branches of a program to have a short list before starting in the "Branch" mode.
Using IMASK=1 in a dynamic mode is not recommended as an inadmissible instruction may have a serious impact on a program execution, including a change of execution sequence. Still see point E below.

D. IMASK = 7EH ("Test")
This in a dynamic mode (RUN = 1 or 3) causes a silent but controlled execution of a program, with only "suspicious" instructions listed.
A conscientious user may in this way check a new or damaged program. Looking for "suspicious" instructions in a static mode (RUN=0) may be of some use after a program failure (this would be a real 'post-mortem' analysis!). The user must only be careful not to take data, if they are intermingled with

and additionally accessed memory locations (if any) or some other information, e.g. a diagnostic warning. All the data displayed describe the program state immediately before the instruction is executed.
This mode is intended for program execution tracing.

REGS=0 Program registers are not displayed.

REGS=1 In the dynamic mode and if the current instruction is not masked out, the program registers taken from _registers_ are displayed.

REGS flag is irrelevant in the static mode - the registers are never displayed.

IMASK byte specifies on its bits the kinds of instructions which are to be displayed whether in static or dynamic mode.

IMASK = 0 means that all instructions are to be displayed in the form defined by the RUN value.
This is a normal static or dynamic mode.

IMASK ≠ 0 means that the current instruction is to be displayed (according to RUN) only if it belongs to a class indicated by any IMASK bit equal to 1.

The instruction classes are assigned to IMASK bits as follows:

Bit	Name	Class contents
0	Brn	Branch instructions i.e. those that change the execution sequence: jump (unconditional or conditional), LOOP-type instructions, call, return, interrupt and return from interrupt instructions.
1	64kB	Instructions that cross or attain 64kB block limit
2	nstd	Nonstandard instructions
3	Invf	Invalid form instructions
4	???	Not used instruction codes
5	286	80286 (iAPX 286) instructions
6	186	80186, 80188 (iAPX 186, 188) instructions
7	None	Empty class (no display).

The bits 1-6 represent "suspicious" instructions which may be, but not necessarily are wrong. They are defined in the NOTE 2.

The figure below depicts the MODE word.

instructions, for wrong instructions (this concerns also the full static display from point A, but then the data are easier discerned by the context).
Altogether, this mask seems to be of little use since, first - programs created by standard compilers will have no "suspicious" instructions at all, and second - a program can be damaged in many ways without creating "suspicious" instructions.

E. Masking out 186, 286, nonstandard or even not used instructions in the "Branch" or "Test" mode may be useful if a higher rank processor or a particular compiler/assembler (generating nonstandard but otherwise correct instructions) is used.

F. IMASK = 80H ("No display")
None instructions are displayed, only the unbuff_ is filled and AX and ulinct_ (=0) are set. The user may act in any way.

NOTE 4 (The Disassembler's By-products)

The Disassembler defines in codeseg two C-callable NEAR procedures for binary to hexadecimal conversion, which may be used by the calling program.

uhex2_ (h2,pi) subroutine converts the less significant byte (h2) of stack word to 2 hexadecimal ASCII characters in DS:pi.
pi is a NEAR pointer to the result i.e. an offset of a variable in a segment addressed with DS. DS can hold any value.
On entry to uhex2_ the stack top contents must be as follows
0.... IP, xxh2, pi.

uhex4_ (h4,pi) subroutine converts a word (h4) from stack to 4 hexadecimal ASCII characters in DS:pi (pi and DS as above).
On entry to uhex4_ the stack top contents must be as follows
0.... IP, h4, pi.

Both procedures destroy AL,BX,CX,DX and DI register contents. Other registers are unchanged. The arguments remain unchanged on stack.

The procedures may be too called in assembly language.
E.g. hex2_ may be called with:

```
MOV AX, OFFSET pi  ;Destination address
PUSH AX
MOV AL, h2         ;Byte to convert
PUSH AX
CALL uhex2_        ;Result in DS:pi,pi+1. AL,BX,CX,DX,DI destr.
ADD SP, 4          ;Discard the arguments
```

*** THE END ***

Załącznik nr 2 — Opis dezasemblera 80186
Zbiór DASM186.TXT

Last update: 17.XII.1990

THE 80186 DISASSEMBLER. Version 2

The 80186 Disassembler is a program that, given a memory address, decodes and displays or prints an Intel 80186/80188 instruction stored at that address.

The Disassembler is a set of procedures written in Intel 8086 assembly language, of which two are to be externally called for the purpose of disassembly: unass_, the actual disassembler, and unin_, the initiator. Both procedures can be called in AZTEC C or 8086 assembly language.

The Disassembler can work in two principal modes: static, when it is required to disassemble an instruction of memory contents, and dynamic, when it is called to disassemble an instruction which is about to execute. In the second case more information may be obtained.

The instruction to be disassembled will be called here the current instruction. An 80186 instruction consists of a basic or main instruction (that can exist independently) and possibly of one or more prefixes, that precede the basic instruction. Both prefixes and basic instructions are termed here elementary instructions.

MEMORY AND REGISTER USE

In this version the "small" memory model applies. This means:

* The Disassembler defines only one code segment, named codeseg, and one variable data segment, named dataseg. Disassembler constants are stored within codeseg.
* Calls to unin_ and unass_ must be NEAR calls within codeseg.
* DS register on entry to the unass_ must address calling program's variable data segment, named dataseg.

This version of the Disassembler occupies about 4kB memory in codeseg and about 130B in dataseg.

The Disassembler uses the calling program's stack, addressed with current SS:SP values. It requires about 30 words on stack (this includes a reasonable margin).

The Disassembler may be stored in read-only memory. The Disassembler's variables are initialized either by the unin_ or each time the unass_ is called.

The Disassembler must not be simultaneously used by concurrent programs.

On return from the unass_ the CS,DS,ES,SS,SP and BP register contents are unchanged. It is more than necessary for a caller program assumed in AZTEC C language. Other register contents should be considered destroyed.

On return from the unin_ only AX contents are destroyed.

INTERFACE TO THE CALLING PROGRAM

The Disassembler is assumed to be linked to and invoked by a Debugger or a microcomputer's Monitor program. It has been designed for the SIRTOS Monitor, which was written in AZTEC C. In all that follows the term "Monitor" means the calling program.

The Monitor too must define dataseg and codeseg segments, which are to be combined with the Disassembler's.

The Disassembler uses the following external objects, which the Monitor must supply as PUBLICs:

1. Data in the dataseg addressed with DS register value:

registers (WORD array) when unass_ is called in a dynamic mode, must contain this sequence of 14 register contents: AX,BX,CX,DX,DI,SI,BP,SP,SS,ES,DS,CS,IP,flags. This must define the traced program state just before the current instruction (pointed to with CS:IP) is executed.

segm_ (WORD variable) when unass_ is called in static mode, must contain the current instruction's segment address.

offs_ (WORD variable) in static mode must contain the current instruction's offset relatively to segm_ value.

hnlsq_ (BYTE sequence) must contain a sequence of characters that when sent to the disassembly output device makes it do a "hard new line". The hnlsq_ must contain one line feed (LF) character and it must end with a 0 (null byte).
The hnlsq_ should position the cursor in a definite spot of the screen, preferably in the bottom left hand corner, independently of where the traced program (or a concurrent one) might have put it. Following that the screen image should be pushed one line up.
For MERA 7953 N screen display monitor (manufactured by MERA-ELZAB) the hnlsq_ may be defined in assembly language as:

```
hnlsq_  DB    32,    32,   27, 89, 55,  32,   13, 10, 0
;e.g. ASCII blank, blank, Esc, Y,  7,  blank, CR, LF, null
```

The figure below depicts the MODE word.

				IMASK								RUN			
7	6	5	4	3	2	1	0	7	.....	2	1	0			
None	LEAV	286	???	Invf	instd	64kB	Brn	0	.....	0	REGS	DYN			
15	14	13	12	11	10	9	8	7	.....	2	1	0			
						M	O	D	E						

Procedure output

The current instruction, if it isn't masked out, is displayed on disassembly output device.

On return from the unass_ in any correct mode
AX register specifies the full length of the current instruction (including any prefixes),
ulinct_ contains the number of lines having been displayed by this call, and
unbuff_ contains the basic instruction in readable ASCII form.
The unbuff_ is filled whether the instruction was displayed or not.

If the mode required is inadmissible (not used bits in MODE are not equal to 0) the unass_ does nothing useful and returns AX=0. ulinct_ and unbuff_ are unchanged.

The input MODE argument remains on stack according to language C calling convention.

NOTE 1 (Calling The unass_ In The Static Mode)

A program calling the unass_ with DYN=0 is assumed to make use of the returned instruction length. If multiple instructions are to be disassembled, the caller should add the length to the current instruction pointer, contained in offs_, thus pointing to a following instruction in memory.
In dynamic modes (DYN=1) the addresses of executed instructions are normally generated by processor's interrupt system.

NOTE 2 ("Suspicious" Instructions And Their Diagnostics)

1. 64kB block limit

The Disassembler fetches instructions on byte by byte basis. If an instruction is crossed by 64kB code segment limit, the rest is fetched from the beginning of the same segment. When fetching data in a dynamic mode, the Disassembler reduces by 64k (wraps) offsets that attain or exceed 64k. Data words are fetched in words (not in bytes) and it depends on the processor (8086/8088, 80186/80188 or 80286) what happens if a word is accessed at offset FFFFH. Also, different processors may differently react on instructions crossed by 64kB boundary.

If the Disassembler, working in any mode, finds that the current instruction violates a 64kB segment limit it first displays the warning message:

'****** 64kB limit. Uncertain result:'

and then (if it is still working at all) the offending instruction. In standard uses (without masking) the message immediately precedes the instruction line. In masked dynamic modes the message precedes the registers (if required) and the instruction line.

The form of the message does not depend of which segment boundary is crossed, i.e. if it is a code, data, stack or extra segment, or possibly more than one.

The 64kB boundary may be discovered by the Disassembler after an instruction or in the middle of a multibyte instruction or datum (not after a datum). In two cases the 64k block limit is signalled before a datum (a dynamic mode is assumed):

1) if an offset computed as a sum of three positive components i.e. base + index + displacement is >= 64k, and
2) in the case of XLAT instruction if the addition of user's BX and AL registers yields a carry (as BX value being translate table address, if near to 64k, is not taken for negative).

64kB limit is also signalled for stack operations, namely if a PUSH-type instruction is going to push on the stack more bytes than there is the space left to the stack segment base (i.e. more than is the current SP value), or if a POP-type instruction is about to take more bytes than there are on the stack up to its 64kB limit (SP is going to rise above 64k).

A special case exists if a program is going to execute a PUSH-type (PUSH, PUSHF, CALL, INT or INTO) instruction while user's SP=0. The standard message, quoted above, should be understood as follows:

a. If SP is really at the beginning of the stack segment, the user is warned that he/she is going to loose the whole stack and the first pushed value will go to (SS:FFFE) word.
b. If SP=0 means actually 10000H (the bottom of 64kB long stack segment) the instruction is correct, but our stack top contents display, taken from SS:0 is obviously not, and the warning should be referred to it.

Two leading blanks are let to be "swallowed" by another Esc sequence possibly interrupted by our program. Esc 'Y7 ' positions the cursor in line 23 (55-32), column 0 (32-32). CR (carriage return), here superfluous, is for a terminal that didn't understand Esc 'Y7 '. LF pushes the screen picture.
If the output device doesn't allow for cursor positioning, the hnlsq_ should be reduced to a "plain new line", i.e. CR,LF,0.
In a dynamic mode the Disassembler issues a "hard new line" at the beginning of each displayed instruction. If the traced program also sends characters to the same terminal, they will be placed in this empty line.
This leading line is counted as 1 in ulinct_ (see below). If the hnlsq_ contains more or less LF characters than one, the user must accordingly interpret ulinct_.
In a static mode the hnlsq_ is not used.

2. C-callable wstring_ procedure contained in the codeseg.

wstring_ (ptr) shall be a NEAR procedure that writes a string of characters on the disassembly output device.
ptr is a word argument on stack. It is the string's offset relatively to the current DS value. DS register on input to wstring_ may be set by unass_ to either dataseg or codeseg segment address. Therefore the wstring_ procedure must not access its own variables in dataseg using input DS value.
The string may contain any characters and it ends with a 0 (null byte). The Disassembler writes exclusively either hnlsq_ or full lines, terminated with LF,CR,0. In this version the longest output line has 80 bytes, including ending 0.
According to C language convention, on return from wstring_, the argument must remain on stack (its output value is irrelevant). The CS,SS,SP and BP register contents must be preserved, other may be destroyed.

The Disassembler defines the following PUBLIC data, contained in dataseg, which are actually its output arguments:

ulinct_ (WORD variable) set to the number of lines of display output produced by the last invocation of unass_;

unbuff_ (BYTE array) containing the last disassembled basic instruction (not a prefix) in ASCII character form. unbuff_ is 80 bytes long. Useful information ends with a LF character. Instruction name begins in unbuff_+23.

INVOCATION FORM

A possible form of calling the Disassembler in assembly language is as follows.

(1) INITIATION

```
CALL unin       ;No arguments
```
This should be called only once, during Monitor initiation (or after Disassembler loading if the Disassembler is nonresident). The Disassembler initializes some internal variables.

(2) DISASSEMBLY

```
PUSH mode       ;mode = RUN,   mode+1 = IMASK
CALL unass_     ;input = word arg on stack,  output = AX register
POP  CX         ;discard the argument
MOV  ilength,AX
```
Of course, the presence and form of PUSH, POP and MOV instructions is purely symbolic.

Procedure input

Hidden input data must be supplied in either segm_:offs_ pair or _registers_ array. hnlsq_ must too be defined.
The explicit input is a word argument on the stack. It will be called here MODE. The MODE consists of two 1-byte binary arguments: the less significant (of lower address) RUN, and the more significant IMASK. Together they specify the mode in which the Disassembler will process the current instruction. Their meaning is as follows:

RUN byte on its bits 0 and 1 defines two binary flags which we shall name DYN (bit 0) and REGS (bit 1).
Bits 2-7 of RUN are not used and must be set to 0 by the caller

DYN=0 (Static mode, "Post-Mortem")
The unass_ disassembles and displays the current instruction if it is not masked out (the description of masking follows), and returns instruction's length.
This mode is intended for a program memory display.

DYN=1 (Dynamic mode)
If the current instruction isn't masked out (see below) the unass_ displays the current values of the program's registers (assuming that REGS=1), then the current instruction, and additionally accessed memory locations (if any) or some

The case of 64kB code segment limit after an elementary (prefix or main) instruction is signalled otherwise. The message
'------ 64kB CS limit here ------'
is displayed after the instruction and it should be ignored if the instruction occupying last bytes of a 64kB block is an effective branch without return (for a short jump the user is advised to check the target address which may be much farther than 127 bytes away).

2. Nonstandard instructions

Some instruction names may occur with 'nstd' suffix. These origin from the codes which in principle should not be generated by a compiler or an assembler because there are other codes that perform their functions.
However they have been found to execute correctly on an 8086/8088, like their standard counterparts. It is only an experimental rule. On IBM with 80286 many of the nonstandard codes block the computer. The execution of nonstandard instructions on an 80186/80188 processor has not been checked.

3. Invalid form instructions

Some instructions may be marked with the 'Invalid form' comment. It's a strong warning. Execution of such instructions is undefined. E.g. indirect far jumps are valid only if the operand is a memory address, not a register reference.

4. Not used instruction codes

The codes not used by either 8086, 80186 or 80286 processor are disassembled as 1-byte instructions named ???, with no operand. If the code in first byte is correct, but its following in the 2nd byte (Internal Code, IC) is not used, the name gets the form '???', and the operand decoded from the 2nd and possibly further bytes is shown.
In fact, even not used instructions may execute in some way. For instance I have discovered that the code F6 together with a not used IC=1 is a nonstandard form of TEST instruction (F6 with IC=0). The same with F7, IC=1 vs 0. The user is free to experiment.

5. 80286 (iAPX 286) instructions

These are 80286-specific instructions. The 80286 processor executes additionally all 80186/80188 and 8086/8088 instructions.
The disassembly of 80286-specific instructions is limited to the first byte with instruction name saying '286code' and the comment: 'If so, 2-5 bytes'.
On an Intel 8086, 8088, 80186 or 80188 (iAPX 86, 88, 186 or 188) these codes should be treated as not used (see item 4 above).

6. LEAVE instruction in dynamic mode when BP < SP

80186/80188 processor's LEAVE instruction normally releases the stack from the current SP value to the current BP value (which is expected to point to a position deeper on stack, i.e. with higher or at least equal address), and then pops BP register. If the condition BP >= SP is not met when LEAVE is about to execute, the Disassembler inserts this comment in the instruction line:
'Warning: BP<SP' .

NOTE 3 (Using The IMASK Argument)

The recommended use of the IMASK argument is the following.

A. IMASK = 0 ("Post-Mortem" or "Trace")
This causes displaying all instructions either in static (RUN=0) or dynamic mode (RUN = 1 or 3).
It is the principal use of the Disassembler.

B. IMASK = 7FH ("Branch")
This causes displaying only branch and possibly "suspicious" instructions (the last being probably extremely rare).
This mask in a dynamic mode (RUN = 1 or 3) is intended for program flow trace.

C. IMASK = 1 ("Branch list")
This in a static mode (RUN=0) may be used to list all branches of a program to have a short list before starting in the "Branch" mode.
Using IMASK=1 in a dynamic mode is not recommended as an inadmissible instruction may have a serious impact on a program execution, including a change of execution sequence. Still see point E below.

D. IMASK = 7EH ("Test")
This in a dynamic mode (RUN = 1 or 3) causes a silent but controlled execution of a program, with only "suspicious" instructions listed.
A conscientious user may in this way check a new or damaged program. Looking for "suspicious" instructions in a static mode (RUN=0) may be of some use after a program failure (this would be a real 'post-mortem' analysis!). The user must only be careful not to take data, if they are intermingled with instructions, for wrong instructions (this concerns also the full static display from point A, but then the data are easier

other information, e.g. a diagnostic warning. All the data displayed describe the program state immediately before the instruction is executed.
This mode is intended for program execution tracing.

REGS=0 Program registers are not displayed.

REGS=1 In the dynamic mode and if the current instruction is not masked out, the program registers taken from _registers_ are displayed.

REGS flag is irrelevant in the static mode - the registers are never displayed.

IMASK byte specifies on its bits the kinds of instructions which are to be displayed whether in static or dynamic mode.

IMASK = 0 means that all instructions are to be displayed in the form defined by the RUN value.
This is a normal static or dynamic mode.

IMASK # 0 means that the current instruction is to be displayed (according to RUN) only if it belongs to a class indicated by any IMASK bit equal to 1.

The instruction classes are assigned to IMASK bits as follows:

Bit	Name	Class contents
0	Brn	Branch instructions i.e. those that change the execution sequence: jump (unconditional or conditional), LOOP-type instructions, call, return, interrupt and return from interrupt instructions.
1	64kB	Instructions that cross or attain 64kB block limit
2	nstd	Nonstandard instructions
3	Invf	Invalid form instructions
4	???	Not used instruction codes
5	286	80286 (iAPX 286) instructions
6	LEAV	LEAVE instruction in dynamic mode when BP < SP (here in 8086 version are 80186 instructions).
7	None	Empty class (no display).

The bits 1-6 represent "suspicious" instructions which may be, but not necessarily are wrong. They are defined in the NOTE 2.

discerned by the context).
Altogether, this mask seems to be of little use since, first - programs created by standard compilers will have no "suspicious" instructions at all, and second - a program can be damaged in many ways without creating "suspicious" instructions.

E. Masking out 286, nonstandard or even not used instructions in the "Branch" or "Test" mode may be useful if a higher rank processor or a particular compiler/assembler (generating nonstandard but otherwise correct instructions) is used.

F. IMASK = 80H ("No display")
None instructions are displayed, only the unbuff_ is filled and AX and ulinct_ (=0) are set. The user may act in any way.

NOTE 4 (The Disassembler's By-products)

The Disassembler defines in codeseg two C-callable NEAR procedures for binary to hexadecimal conversion, which may be used by the calling program.

uhex2_ (h2,pi) subroutine converts the less significant byte (h2) of stack word to 2 hexadecimal ASCII characters in DS:pi. pi is a NEAR pointer to the result i.e. an offset of a variable in a segment addressed with DS. DS can hold any value. On entry to uhex2_ the stack top contents must be as follows: 0.... IP, xxh2, pi.

uhex4_ (h4,pi) subroutine converts a word (h4) from stack to 4 hexadecimal ASCII characters in DS:pi (pi and DS as above). On entry to uhex4_ the stack top contents must be as follows: 0.... IP, h4, pi.

Both procedures destroy AL,BX,CX,DX and DI register contents. Other registers are unchanged. The arguments remain unchanged on stack.

The procedures may be too called in assembly language.
E.g. hex2_ may be called with:

```
MOV AX, OFFSET pi  ;Destination address
PUSH AX
MOV AL, h2         ;Byte to convert
PUSH AX
CALL uhex2_        ;Result in DS:pi,pi+1. AL,BX,CX,DX,DI destr.
ADD SP, 4          ;Discard the arguments
```

*** THE END ***

Załącznik nr 5 – Lista instrukcji mp Intel 8086/88 utworzona przez DA-SM86 5
TRYB = 0001H (pseudo-dynamiczny). Pierwsze dwa bajty zmieniają się od 0 do FF

```
 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 0004 0010 24F5 24E8 24E8 5555 0101010101010101
24E8:0012 FFBF9090     ???     [BX+9090]                    DS:9093=4CA1

                    Kody glowne 8086/8088
0000          ADD     [BX+SI],AL                          DS:0004=E8
0101          ADD     [BX+DI],AX                          DS:0005=1224
0202          ADD     AL,[BP+SI]                          SS:0005=EC
0303          ADD     AX,[BP+DI]                          SS:0006=568B
0404          ADD     AL,04h
050590        ADD     AX,9005h
06            PUSH    ES
07            POP     ES                        ST=3F80
0808          OR      [BX+SI],CL                          DS:0004=E8
0909          OR      [BX+DI],CX                          DS:0005=1224
0A0A          OR      CL,[BP+SI]                          SS:0005=EC
0B0B          OR      CX,[BP+DI]                          SS:0006=568B
0C0C          OR      AL,0Ch
0D0D90        OR      AX,900Dh
0E            PUSH    CS                                  If so, 2-5 bytes
0F            286code                                     DS:0004=E8
1010          ADC     [BX+SI],DL                          DS:0005=1224
1111          ADC     [BX+DI],DX                          SS:0005=EC
1212          ADC     DL,[BP+SI]                          SS:0006=568B
1313          ADC     DX,[BP+DI]
1414          ADC     AL,14h
151590        ADC     AX,9015h
16            PUSH    SS
17            POP     SS                        ST=3F80
1818          SBB     [BX+SI],BL                          DS:0004=E8
1919          SBB     [BX+DI],BX                          DS:0005=1224
1A1A          SBB     BL,[BP+SI]                          SS:0005=EC
1B1B          SBB     BX,[BP+DI]                          SS:0006=568B
1C1C          SBB     AL,1Ch
1D1D90        SBB     AX,901Dh
1E            PUSH    DS
1F            POP     DS                        ST=3F80
2020          AND     [BX+SI],AH                          DS:0004=E8
2121          AND     [BX+DI],SP                          DS:0005=1224
2222          AND     AH,[BP+SI]                          SS:0005=EC
2323          AND     SP,[BP+DI]                          SS:0006=568B
2424          AND     AL,24h
252590        AND     AX,9025h
26            ES:     } 1 instrukcja. Tak zawsze przy prefiksach
26            ES:     }
90            NOP     }
27            DAA
2828          SUB     [BX+SI],CH                          DS:0004=E8
2929          SUB     [BX+DI],BP                          DS:0005=1224
2A2A          SUB     CH,[BP+SI]                          SS:0005=EC
2B2B          SUB     BP,[BP+DI]                          SS:0006=568B
2C2C          SUB     AL,2Ch
2D2D90        SUB     AX,902Dh
2E            CS:
2E            CS:
90            NOP
2F            DAS
3030          XOR     [BX+SI],DH                          DS:0004=E8
3131          XOR     [BX+DI],SI                          DS:0005=1224
3232          XOR     DH,[BP+SI]                          SS:0005=EC
3333          XOR     SI,[BP+DI]                          SS:0006=568B
3434          XOR     AL,34h
353590        XOR     AX,9035h
36            SS:
36            SS:
90            NOP
37            AAA
3838          CMP     [BX+SI],BH                          DS:0004=E8
3939          CMP     [BX+DI],DI                          DS:0005=1224
3A3A          CMP     BH,[BP+SI]                          SS:0005=EC
3B3B          CMP     DI,[BP+DI]                          SS:0006=568B
3C3C          CMP     AL,3Ch
3D3D90        CMP     AX,903Dh
3E            DS:
3E            DS:
90            NOP
3F            AAS
40            INC     AX
41            INC     CX
42            INC     DX
43            INC     BX
44            INC     SP
45            INC     BP
46            INC     SI
47            INC     DI
48            DEC     AX
49            DEC     CX
4A            DEC     DX
4B            DEC     BX
4C            DEC     SP
4D            DEC     BP
4E            DEC     SI

C3            RET                               ST=3F80
C4C4          LES     AX,SP                       Invalid form
C5C5          LDS     AX,BP                       Invalid form
C6C690        MOV     DH,90h
C7C79090      MOV     DI,9090h
C8            186code                           Possibly >1 byte
C9            186code                           Possibly >1 byte
CACA90        RETF    90CAh                     ST=3F80,8B00,74D9,CD0A,4221
CB            RETF                              ST=3F80,8B00
CC            INT     3                           0:000C=10C3,212D
CDCD          INT     CDh                         0:0334=0000,0000
CE            INTO                                0:0010=0C67,0070
CF            IRET                              ST=3F80,8B00,74D9
D0D0          RCL     AL,1
D1D1          RCL     CX,1
D2D2          RCL     DL,CL
D3D3          RCL     BX,CL
D4D4          AAM
D5D5          AAD
D6            ???
D7            XLAT                                    DS:0004=E8
D8D8          ESC     03h,AL                    Probably 8087 code
D9D9          ESC     0Bh,CX                    Probably 8087 code
DADA          ESC     13h,DL                    Probably 8087 code
DBDB          ESC     1Bh,BX                    Probably 8087 code
DCDC          ESC     23h,AH                    Probably 8087 code
DDDD          ESC     2Bh,BP                    Probably 8087 code
DEDE          ESC     33h,DH                    Probably 8087 code
DFDF          ESC     3Bh,DI                    Probably 8087 code
E0E0          LOOPNE/Z FFF4                       FALSE
E1E1          LOOPE/Z FFF5                        TRUE
E2E2          LOOP    FFF6                        TRUE
E3E3          JCXZ    FFF7                        FALSE
E4E4          IN      AL,E4h
E5E5          IN      AX,E5h
E6E6          OUT     E6h,AL
E7E7          OUT     E7h,AX
E8E890        CALL    90FD
E9E990        JMP     90FE
EAEA909090    JMP     9090:90EA
EBEB          JMP     SHORT FFFF
EC            IN      AL,DX
ED            IN      AX,DX
EE            OUT     DX,AL
EF            OUT     DX,AX
F0            LOCK
F0            LOCK
90            NOP
F1            ???
F2            REPNE/Z
F2            REPNE/Z
90            NOP
F3            REP/E/Z
F3            REP/E/Z
90            NOP
F4            HLT
F5            CMC
F6F6          DIV     DH
F7F7          DIV     DI
F8            CLC
F9            STC
FA            CLI
FB            STI
FC            CLD
FD            STD
FEFE          ???     DH                          Invalid form
FFFF          ???     DI

                    Grupy
8080909090    ADD     BYTE PTR [BX+SI+9090],90h    DS:9094=4C=L
8089909090    OR      BYTE PTR [BX+DI+9090],90h    DS:9095=18
8092909090    ADC     BYTE PTR [BP+SI+9090],90h    SS:9095=5E=^
809B909090    SBB     BYTE PTR [BP+DI+9090],90h    SS:9096=04
80A4909090    AND     BYTE PTR [SI+9090],90h       DS:9091=5E=^
80AD909090    SUB     BYTE PTR [DI+9090],90h       DS:9092=04
80B6909090    XOR     BYTE PTR [BP+9090],90h       SS:9094=8B
80BF909090    CMP     BYTE PTR [BX+9090],90h       DS:9093=A1
818090909090  ADD     WORD PTR [BX+SI+9090],9090h  DS:9094=184C
818990909090  OR      WORD PTR [BX+DI+9090],9090h  DS:9095=B918
819290909090  ADC     WORD PTR [BP+SI+9090],9090h  SS:9095=045E
819B90909090  SBB     WORD PTR [BP+DI+9090],9090h  SS:9096=8304
81A490909090  AND     WORD PTR [SI+9090],9090h     DS:9091=043E
81AD90909090  SUB     WORD PTR [DI+9090],9090h     DS:9092=A104
81B690909090  XOR     WORD PTR [BP+9090],9090h     SS:9094=5E8B
81BF90909090  CMP     WORD PTR [BX+9090],9090h     DS:9093=4CA1
8280909090    ADD     BYTE PTR [BX+SI+9090],90h    DS:9094=4C=L
8289909090    OR nstd BYTE PTR [BX+DI+9090],90h    DS:9095=18
8292909090    ADC     BYTE PTR [BP+SI+9090],90h    SS:9095=5E=^
829B909090    SBB     BYTE PTR [BP+DI+9090],90h    SS:9096=04
```

51

4F	DEC DI				
50	PUSH AX				
51	PUSH CX				
52	PUSH DX				
53	PUSH BX				
54	PUSH SP				
55	PUSH BP				
56	PUSH SI				
57	PUSH DI				
58	POP AX		ST=3F80		
59	POP CX		ST=3F80		
5A	POP DX		ST=3F80		
5B	POP BX		ST=3F80		
5C	POP SP		ST=3F80		
5D	POP BP		ST=3F80		
5E	POP SI		ST=3F80		
5F	POP DI		ST=3F80		
60	186code			Possibly >1 byte	
61	186code			Possibly >1 byte	
62	186code			Possibly >1 byte	
63	286code			If so, 2-5 bytes	
64	???				
65	???				
66	???				
67	???				
68	186code			Possibly >1 byte	
69	186code			Possibly >1 byte	
6A	186code			Possibly >1 byte	
6B	186code			Possibly >1 byte	
6C	186code			Possibly >1 byte	
6D	186code			Possibly >1 byte	
6E	186code			Possibly >1 byte	
6F	186code			Possibly >1 byte	
7070	JO 0084			FALSE	
7171	JNO 0085			TRUE	
7272	JC/JB 0086			TRUE	
7373	JNC/JAE 0087			FALSE	
7474	JE/JZ 0088			TRUE	
7575	JNE/JNZ 0089			FALSE	
7676	JBE 008A			TRUE	
7777	JA 008B			FALSE	
7878	JS 008C			FALSE	
7979	JNS 008D			TRUE	
7A7A	JPE 008E			TRUE	
7B7B	JPO 008F			FALSE	
7C7C	JL 0090			FALSE	
7D7D	JGE 0091			TRUE	
7E7E	JLE 0092			TRUE	
7F7F	JG 0093			FALSE	
8080909090	ADD BYTE PTR [BX+SI+9090],90h			DS:9094=4C=L	
818190909090	ADD WORD PTR [BX+DI+9090],9090h			DS:9095=8918	
8282909090	ADD BYTE PTR [BP+SI+9090],90h			SS:9095=5E=^	
8383909090	ADD WORD PTR [BP+DI+9090],-70h			SS:9096=8304	
84849090	TEST AL,[SI+9090]			DS:9091=5E=^	
85859090	TEST AX,[DI+9090]			DS:9092=A104	
86869090	XCHG AL,[BP+9090]			SS:9094=8B	
87879090	XCHG AX,[BX+9090]			DS:9093=4CA1	
88889090	MOV [BX+SI+9090],CL			DS:9094=4C=L	
89899090	MOV [BX+DI+9090],CX			DS:9095=8918	
8A8A9090	MOV CL,[BP+SI+9090]			SS:9095=5E=^	
8B8B9090	MOV CX,[BP+DI+9090]			SS:9096=8304	
8C8C9090	MOV [SI+9090],CS			DS:9091=045E	
8D8D9090	LEA CX,[DI+9090]			DS:9092=A104	
8E8E9090	MOV CS,[BP+9090]			SS:9094=5E8B	
8F8F9090	POPnstd [BX+9090]		ST=3F80	DS:9093=4CA1	
90	NOP				
91	XCHG AX,CX				
92	XCHG AX,DX				
93	XCHG AX,BX				
94	XCHG AX,SP				
95	XCHG AX,BP				
96	XCHG AX,SI				
97	XCHG AX,DI				
98	CBW				
99	CWD				
9A9A909090	CALL 9090:909A				
9B	WAIT				
9C	PUSHF				
9D	POPF		ST=3F80		
9E	SAHF				
9F	LAHF				
A0A090	MOV AL,[90A0]			DS:90A0=83	
A1A190	MOV AX,[90A1]			DS:90A1=4C3E	
A2A290	MOV [90A2],AL			DS:90A2=4C=L	
A3A390	MOV [90A3],AX			DS:90A3=0218	
A4	MOVSB		DS:0001=BB	ES:0002=00	
A5	MOVSW		DS:0001=00BB	ES:0002=0000	
A6	CMPSB		DS:0001=BB	ES:0002=00	
A7	CMPSW		DS:0001=00BB	ES:0002=0000	
A8A8	TEST AL,A8h				
A9A990	TEST AX,90A9h				
AA	STOSB			ES:0002=00	
AB	STOSW			ES:0002=0000	
AC	LODSB		DS:0001=BB		
AD	LODSW		DS:0001=00BB		
AE	SCASB			ES:0002=00	

82A4909090	ANDnstd BYTE PTR [SI+90		
82AD909090	SUB BYTE PTR [DI+90		DS:9092=04
82B6909090	XORnstd BYTE PTR [BP+9090],90h		SS:9094=8B
82BF909090	CMP BYTE PTR [BX+9090],90h		DS:9093=A1
8380909090	ADD WORD PTR [BX+SI+9090],-70h		DS:9094=184C
8389909090	OR nstd WORD PTR [BX+DI+9090],-70h		DS:9095=8918
8392909090	ADC WORD PTR [BP+SI+9090],-70h		SS:9095=045E
839B909090	SBB WORD PTR [BP+DI+9090],-70h		SS:9096=8304
83A4909090	ANDnstd WORD PTR [SI+9090],-70h		DS:9091=045E
83AD909090	SUB WORD PTR [DI+9090],-70h		DS:9092=A104
83B6909090	XORnstd WORD PTR [BP+9090],-70h		SS:9094=5E8B
83BF909090	CMP WORD PTR [BX+9090],-70h		DS:9093=4CA1
8C809090	MOV [BX+SI+9090],ES		DS:9094=184C
8C899090	MOV [BX+DI+9090],CS		DS:9095=8918
8C929090	MOV [BP+SI+9090],SS		SS:9095=045E
8C9B9090	MOV [BP+DI+9090],DS		SS:9096=8304
8CA49090	MOVnstd [SI+9090],ES		DS:9091=045E
8CAD9090	MOVnstd [DI+9090],CS		DS:9092=A104
8CB69090	MOVnstd [BP+9090],SS		SS:9094=5E8B
8CBF9090	MOVnstd [BX+9090],DS		DS:9093=4CA1
8E809090	MOV ES,[BX+SI+9090]		DS:9094=184C
8E899090	MOV CS,[BX+DI+9090]		DS:9095=8918
8E929090	MOV SS,[BP+SI+9090]		SS:9095=045E
8E9B9090	MOV DS,[BP+DI+9090]		SS:9096=8304
8EA49090	MOVnstd ES,[SI+9090]		DS:9091=045E
8EAD9090	MOVnstd CS,[DI+9090]		DS:9092=A104
8EB69090	MOVnstd SS,[BP+9090]		SS:9094=5E8B
8EBF9090	MOVnstd DS,[BX+9090]		DS:9093=4CA1
8F809090	POP [BX+SI+9090]	ST=3F80	DS:9094=184C
8F899090	POPnstd [BX+DI+9090]	ST=3F80	DS:9095=8918
8F929090	POPnstd [BP+SI+9090]	ST=3F80	SS:9095=045E
8F9B9090	POPnstd [BP+DI+9090]	ST=3F80	SS:9096=8304
8FA49090	POPnstd [SI+9090]	ST=3F80	DS:9091=045E
8FAD9090	POPnstd [DI+9090]	ST=3F80	DS:9092=A104
8FB69090	POPnstd [BP+9090]	ST=3F80	SS:9094=5E8B
8FBF9090	POPnstd [BX+9090]	ST=3F80	DS:9093=4CA1
C680909090	MOV BYTE PTR [BX+SI+9090],90h		DS:9094=4C=L
C689909090	MOVnstd BYTE PTR [BX+DI+9090],90h		DS:9095=18
C692909090	MOVnstd BYTE PTR [BP+SI+9090],90h		SS:9095=5E=^
C69B909090	MOVnstd BYTE PTR [BP+DI+9090],90h		SS:9096=04
C6A4909090	MOVnstd BYTE PTR [SI+9090],90h		DS:9091=5E=^
C6AD909090	MOVnstd BYTE PTR [DI+9090],90h		DS:9092=04
C6B6909090	MOVnstd BYTE PTR [BP+9090],90h		SS:9094=8B
C6BF909090	MOVnstd BYTE PTR [BX+9090],90h		DS:9093=A1
C78090909090	MOV WORD PTR [BX+SI+9090],9090h		DS:9094=184C
C78990909090	MOVnstd WORD PTR [BX+DI+9090],9090h		DS:9095=8918
C79290909090	MOVnstd WORD PTR [BP+SI+9090],9090h		SS:9095=045E
C79B90909090	MOVnstd WORD PTR [BP+DI+9090],9090h		SS:9096=8304
C7A490909090	MOVnstd WORD PTR [SI+9090],9090h		DS:9091=045E
C7AD90909090	MOVnstd WORD PTR [DI+9090],9090h		DS:9092=A104
C7B690909090	MOVnstd WORD PTR [BP+9090],9090h		SS:9094=5E8B
C7BF90909090	MOVnstd WORD PTR [BX+9090],9090h		DS:9093=4CA1
D0809090	ROL BYTE PTR [BX+SI+9090],1		DS:9094=4C=L
D0899090	ROR BYTE PTR [BX+DI+9090],1		DS:9095=18
D0929090	RCL BYTE PTR [BP+SI+9090],1		SS:9095=5E=^
D09B9090	RCR BYTE PTR [BP+DI+9090],1		SS:9096=04
D0A49090	SHL BYTE PTR [SI+9090],1		DS:9091=5E=^
D0AD9090	SHR BYTE PTR [DI+9090],1		DS:9092=04
D0B69090	186code BYTE PTR [BP+9090],1		SS:9094=8B
D0BF9090	SAR BYTE PTR [BX+9090],1		DS:9093=A1
D1809090	ROL WORD PTR [BX+SI+9090],1		DS:9094=184C
D1899090	ROR WORD PTR [BX+DI+9090],1		DS:9095=8918
D1929090	RCL WORD PTR [BP+SI+9090],1		SS:9095=045E
D19B9090	RCR WORD PTR [BP+DI+9090],1		SS:9096=8304
D1A49090	SHL WORD PTR [SI+9090],1		DS:9091=045E
D1AD9090	SHR WORD PTR [DI+9090],1		DS:9092=A104
D1B69090	186code WORD PTR [BP+9090],1		SS:9094=5E8B
D1BF9090	SAR WORD PTR [BX+9090],1		DS:9093=4CA1
D2809090	ROL BYTE PTR [BX+SI+9090],CL		DS:9094=4C=L
D2899090	ROR BYTE PTR [BX+DI+9090],CL		DS:9095=18
D2929090	RCL BYTE PTR [BP+SI+9090],CL		SS:9095=5E=^
D29B9090	RCR BYTE PTR [BP+DI+9090],CL		SS:9096=04
D2A49090	SHL BYTE PTR [SI+9090],CL		DS:9091=5E=^
D2AD9090	SHR BYTE PTR [DI+9090],CL		DS:9092=04
D2B69090	186code BYTE PTR [BP+9090],CL		SS:9094=8B
D2BF9090	SAR BYTE PTR [BX+9090],CL		DS:9093=A1
D3809090	ROL WORD PTR [BX+SI+9090],CL		DS:9094=184C
D3899090	ROR WORD PTR [BX+DI+9090],CL		DS:9095=8918
D3929090	RCL WORD PTR [BP+SI+9090],CL		SS:9095=045E
D39B9090	RCR WORD PTR [BP+DI+9090],CL		SS:9096=8304
D3A49090	SHL WORD PTR [SI+9090],CL		DS:9091=045E
D3AD9090	SHR WORD PTR [DI+9090],CL		DS:9092=A104
D3B69090	186code WORD PTR [BP+9090],CL		SS:9094=5E8B
D3BF9090	SAR WORD PTR [BX+9090],CL		DS:9093=4CA1
F6809090 90	TEST BYTE PTR [BX+SI+9090],90h		DS:9094=4C=L
F689909090	??? BYTE PTR [BX+DI+9090],90h		DS:9095=18
F6929090	NOT BYTE PTR [BP+SI+9090]		SS:9095=5E=^
F69B9090	NEG BYTE PTR [BP+DI+9090]		SS:9096=04
F6A49090	MUL BYTE PTR [SI+9090]		DS:9091=5E=^
F6AD9090	IMUL BYTE PTR [DI+9090]		DS:9092=04
F6B69090	DIV BYTE PTR [BP+9090]		SS:9094=8B
F6BF9090	IDIV BYTE PTR [BX+9090]		DS:9093=A1
F780909090 90	TEST WORD PTR [BX+SI+9090],9090h		DS:9094=184C
F789909090 90	??? WORD PTR [BX+DI+9090],9090h		DS:9095=8918
F7929090	NOT WORD PTR [BP+SI+9090]		SS:9095=045E
F79B9090	NEG WORD PTR [BP+DI+9090]		SS:9096=8304

```
AF          SCASW                                   ES:0002=0000
B0B0        MOV     AL,B0h
B1B1        MOV     CL,B1h
B2B2        MOV     DL,B2h
B3B3        MOV     BL,B3h
B4B4        MOV     AH,B4h
B5B5        MOV     CH,B5h
B6B6        MOV     DH,B6h
B7B7        MOV     BH,B7h
B8B890      MOV     AX,90B8h
B9B990      MOV     CX,90B9h
BABA90      MOV     DX,90BAh
BBBB90      MOV     BX,90BBh
BCBC90      MOV     SP,90BCh
BDBD90      MOV     BP,90BDh
BEBE90      MOV     SI,90BEh
BFBF90      MOV     DI,90BFh
C0          186code                                 Possibly >1 byte
C1          186code                                 Possibly >1 byte
C2C290      RET     90C2h                   ST=3F80,8B00,74D9,CD0A,4221

F7A49090    MUL     WORD PTR [SI+9090]              DS:9092=A104
F7AD9090    IMUL    WORD PTR [DI+9090]              SS:9094=5E8B
F7B69090    DIV     WORD PTR [BP+9090]              DS:9093=4CA1
F7BF9090    IDIV    WORD PTR [BX+9090]              DS:9094=4C=L
FE809090    INC     BYTE PTR [BX+SI+9090]           DS:9095=18
FE899090    DEC     BYTE PTR [BX+DI+9090]           Invalid form
FE929090    CALL    BYTE PTR [BP+SI+9090]           Invalid form
FE9B9090    CALL FAR BYTE PTR [BP+DI+9090]          Invalid form
FEA49090    JMP     BYTE PTR [SI+9090]              Invalid form
FEAD9090    JMP FAR BYTE PTR [DI+9090]              Invalid form
FEB69090    PUSH    BYTE PTR [BP+9090]              Invalid form
FEBF9090    ???     BYTE PTR [BX+9090]              DS:9094=184C
FF809090    INC     WORD PTR [BX+SI+9090]           DS:9095=8918
FF899090    DEC     WORD PTR [BX+DI+9090]           SS:9095=045E
FF929090    CALL    W.PTR [BP+SI+9090]      SS:9096=8304,2C7F
FF9B9090    CALL FAR [BP+DI+9090]                   DS:9091=045E
FFA49090    JMP     WORD PTR [SI+9090]      DS:9092=A104,184C
FFAD9090    JMP FAR [DI+9090]                       SS:9094=5E8B
FFB69090    PUSH    [BP+9090]                       DS:9093=4CA1
FFBF9090    ???     [BX+9090]
```

Załącznik nr 6 - Lista instrukcji typ Intel 80186/188 utworzona przez DASM186. 6
TRYB=0001H. (pseudo-dysmiczny). Pierwsze 2 bajty zmieniają się od 0 do FF.

```
 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 0004 0080 128C 1144 1144 0000 0000000000000000
115B:1208 C8040000     ENTER   0004h,00h
```

Kody glowne 80186/80188

0000	ADD [BX+SI],AL		DS:0004=00
0101	ADD [BX+DI],AX		DS:0005=0000
0202	ADD AL,[BP+SI]		SS:0005=00
0303	ADD AX,[BP+DI]		SS:0006=000C
0404	ADD AL,04h		
050590	ADD AX,9005h		
06	PUSH ES		
07	POP ES	ST=74F5	
0808	OR [BX+SI],CL		DS:0004=08
0909	OR [BX+DI],CX		DS:0005=0000
0A0A	OR CL,[BP+SI]		SS:0005=00
0B0B	OR CX,[BP+DI]		SS:0006=000C
0C0C	OR AL,0Ch		
0D0D90	OR AX,900Dh		
0E	PUSH CS		
0F	286code		If so, 2-5 bytes
1010	ADC [BX+SI],DL		DS:0004=10
1111	ADC [BX+DI],DX		DS:0005=0000
1212	ADC DL,[BP+SI]		SS:0005=00
1313	ADC DX,[BP+DI]		SS:0006=000C
1414	ADC AL,14h		
151590	ADC AX,9015h		
16	PUSH SS		
17	POP SS	ST=74F5	
1818	SBB [BX+SI],BL		DS:0004=18
1919	SBB [BX+DI],BX		DS:0005=0000
1A1A	SBB BL,[BP+SI]		SS:0005=00
1B1B	SBB BX,[BP+DI]		SS:0006=000C
1C1C	SBB AL,1Ch		
1D1D90	SBB AX,901Dh		
1E	PUSH DS		
1F	POP DS	ST=74F5	
2020	AND [BX+SI],AH		DS:0004=20=
2121	AND [BX+DI],SP		DS:0005=0000
2222	AND AH,[BP+SI]		SS:0005=00
2323	AND SP,[BP+DI]		SS:0006=000C
2424	AND AL,24h		
252590	AND AX,9025h		
26	ES:		
26	ES:	} 1 instrukcja. Tak zawsze dla prefiksów	
90	NOP		
27	DAA		
2828	SUB [BX+SI],CH		DS:0004=28=(
2929	SUB [BX+DI],BP		DS:0005=0000
2A2A	SUB CH,[BP+SI]		SS:0005=00
2B2B	SUB BP,[BP+DI]		SS:0006=000C
2C2C	SUB AL,2Ch		
2D2D90	SUB AX,902Dh		
2E	CS:		
2E	CS:		
90	NOP		
2F	DAS		
3030	XOR [BX+SI],DH		DS:0004=30=0
3131	XOR [BX+DI],SI		DS:0005=0000
3232	XOR DH,[BP+SI]		SS:0005=00
3333	XOR SI,[BP+DI]		SS:0006=000C
3434	XOR AL,34h		
353590	XOR AX,9035h		
36	SS:		
36	SS:		
90	NOP		
37	AAA		
3838	CMP [BX+SI],BH		DS:0004=38=8
3939	CMP [BX+DI],DI		DS:0005=0000
3A3A	CMP BH,[BP+SI]		SS:0005=00
3B3B	CMP DI,[BP+DI]		SS:0006=000C
3C3C	CMP AL,3Ch		
3D3D90	CMP AX,903Dh		
3E	DS:		
3E	DS:		
90	NOP		
3F	AAS		
40	INC AX		
41	INC CX		
42	INC DX		
43	INC BX		
44	INC SP		
45	INC BP		
46	INC SI		
47	INC DI		
48	DEC AX		
49	DEC CX		
4A	DEC DX		
4B	DEC BX		
4C	DEC SP		

CACA90	RETF 90CAh		ST=74F5,E903,FF58,06C7,1274
CB	RETF		ST=74F5,E903
CC	INT 3		0:000C=10C3,0D91
CDCD	INT CDh		0:0334=0000,0000
CE	INTO		0:0010=0C67,0070
CF	IRET		ST=74F5,E903,FF58
D0D0	RCL AL,1		
D1D1	RCL CX,1		
D2D2	RCL DL,CL		
D3D3	RCL BX,CL		
D4D4	AAM		
D5D5	AAD		
D6	???		
D7	XLAT		DS:0004=D7
D8D8	ESC 03h,AL		Probably 8087 code
D9D9	ESC 0Bh,CX		Probably 8087 code
DADA	ESC 13h,DL		Probably 8087 code
DBDB	ESC 1Bh,BX		Probably 8087 code
DCDC	ESC 23h,AH		Probably 8087 code
DDDD	ESC 2Bh,BP		Probably 8087 code
DEDE	ESC 33h,DH		Probably 8087 code
DFDF	ESC 3Bh,DI		Probably 8087 code
E0E0	LOOPNE/Z 00A4		TRUE
E1E1	LOOPE/Z 00A5		FALSE
E2E2	LOOP 00A6		TRUE
E3E3	JCXZ 00A7		FALSE
E4E4	IN AL,E4h		
E5E5	IN AX,E5h		
E6E6	OUT E6h,AL		
E7E7	OUT E7h,AX		
E8E890	CALL 91AD		
E9E990	JMP 91AE		
EAEA909090	JMP 9090:90EA		
EBEB	JMP SHORT 00AF		
EC	IN AL,DX		
ED	IN AX,DX		
EE	OUT DX,AL		
EF	OUT DX,AX		
F0	LOCK		
F0	LOCK		
90	NOP		
F1	???		
F2	REPNE/Z		
F2	REPNE/Z		
90	NOP		
F3	REP/E/Z		
F3	REP/E/Z		
90	NOP		
F4	HLT		
F5	CMC		
F6F6	DIV DH		
F7F7	DIV DI		
F8	CLC		
F9	STC		
FA	CLI		
FB	STI		
FC	CLD		
FD	STD		
FEFE	??? DH		Invalid form
FFFF	??? DI		

54

Grupy

8080909090	ADD	BYTE PTR [BX+SI+9090],90h	DS:9094=FE
8089909090	OR	BYTE PTR [BX+DI+9090],90h	DS:9095=06
8092909090	ADC	BYTE PTR [BP+SI+9090],90h	SS:9095=00
809B909090	SBB	BYTE PTR [BP+DI+9090],90h	SS:9096=00
80A4909090	AND	BYTE PTR [SI+9090],90h	DS:9091=8B
80AD909090	SUB	BYTE PTR [DI+9090],90h	DS:9092=27='
80B6909090	XOR	BYTE PTR [BP+9090],90h	SS:9094=00
80BF909090	CMP	BYTE PTR [BX+9090],90h	DS:9093=7C=¦
818090909090	ADD	WORD PTR [BX+SI+9090],9090h	DS:9094=06FE
818990909090	OR	WORD PTR [BX+DI+9090],9090h	DS:9095=9006
819290909090	ADC	WORD PTR [BP+SI+9090],9090h	SS:9095=0000
819B90909090	SBB	WORD PTR [BP+DI+9090],9090h	SS:9096=0000
81A490909090	AND	WORD PTR [SI+9090],9090h	DS:9091=278B
81AD90909090	SUB	WORD PTR [DI+9090],9090h	DS:9092=7C27
81B690909090	XOR	WORD PTR [BP+9090],9090h	SS:9094=0000
81BF90909090	CMP	WORD PTR [BX+9090],9090h	DS:9093=FE7C
8280909090	ADD	BYTE PTR [BX+SI+9090],90h	DS:9094=FE
8289909090	OR nstd	BYTE PTR [BX+DI+9090],90h	DS:9095=06
8292909090	ADC	BYTE PTR [BP+SI+9090],90h	SS:9095=00
829B909090	SBB	BYTE PTR [BP+DI+9090],90h	SS:9096=00
82A4909090	ANDnstd	BYTE PTR [SI+9090],90h	DS:9091=8B
82AD909090	SUB	BYTE PTR [DI+9090],90h	DS:9092=27='
82B6909090	XORnstd	BYTE PTR [BP+9090],90h	SS:9094=00
82BF909090	CMP	BYTE PTR [BX+9090],90h	DS:9093=7C=¦
8380909090	ADD	WORD PTR [BX+SI+9090],-70h	DS:9094=06FE

```
4D              DEC     BP
4E              DEC     SI
4F              DEC     DI
50              PUSH    AX
51              PUSH    CX
52              PUSH    DX
53              PUSH    BX
54              PUSH    SP
55              PUSH    BP
56              PUSH    SI
57              PUSH    DI
58              POP     AX                      ST=74F5
59              POP     CX                      ST=74F5
5A              POP     DX                      ST=74F5
5B              POP     BX                      ST=74F5
5C              POP     SP                      ST=74F5
5D              POP     BP                      ST=74F5
5E              POP     SI                      ST=74F5
5F              POP     DI                      ST=74F5
60              PUSHA
61              POPA                            ST=74F5,E903,FF58,06C7,1274
626290          BOUND   SP,[BP+SI-70]                    SS:FF95=4600,4353
63              286code                                  If so, 2-5 bytes
64              ???
65              ???
66              ???
67              ???
686890          PUSH    9068h
69969909090     IMUL    BP,WORD PTR [BX+DI-70],9090h    DS:FF95=8B5F
6A6A            PUSH    +6Ah
6B6B9090        IMUL    BP,WORD PTR [BP+DI-70],-70h     SS:FF96=5346
6C              INSB
6D              INSW
6E              OUTSB                   DS:0001=00
6F              OUTSW                   DS:0001=6900
7070            JO      0134                                    FALSE
7171            JNO     0135                                    TRUE
7272            JC/JB   0136                                    FALSE
7373            JNC/JAE 0137                                    TRUE
7474            JE/JZ   0138                                    FALSE
7575            JNE/JNZ 0139                                    TRUE
7676            JBE     013A                                    FALSE
7777            JA      013B                                    TRUE
7878            JS      013C                                    FALSE
7979            JNS     013D                                    TRUE
7A7A            JPE     013E                                    FALSE
7B7B            JPO     013F                                    TRUE
7C7C            JL      0140                                    FALSE
7D7D            JGE     0141                                    TRUE
7E7E            JLE     0142                                    FALSE
7F7F            JG      0143                                    TRUE
8080909090      ADD     BYTE PTR [BX+SI+9090],90h       DS:9094=FE
818190909090    ADD     WORD PTR [BX+DI+9090],9090h     DS:9095=9006
8282909090      ADD     BYTE PTR [BP+SI+9090],90h       SS:9095=00
8383909090      ADD     WORD PTR [BP+DI+9090],-70h      SS:9096=0000
84849090        TEST    AL,[SI+9090]                    DS:9091=8B
85859090        TEST    AX,[DI+9090]                    DS:9092=7C27
86869090        XCHG    AL,[BP+9090]                    SS:9094=00
87879090        XCHG    AX,[BX+9090]                    DS:9093=FE7C
88889090        MOV     [BX+SI+9090],CL                 DS:9094=FE
89899090        MOV     [BX+DI+9090],CX                 DS:9095=9006
8A8A9090        MOV     CL,[BP+SI+9090]                 SS:9095=00
8B8B9090        MOV     CX,[BP+DI+9090]                 SS:9096=0000
8C8C9090        MOV     [SI+9090],CS                    DS:9091=278B
8D8D9090        LEA     CX,[DI+9090]                    DS:9092=7C27
8E8E9090        MOV     CS,[BP+9090]                    SS:9094=0000
8F8F9090        POPnstd [BX+9090]               ST=74F5 DS:9093=FE7C
90              NOP
91              XCHG    AX,CX
92              XCHG    AX,DX
93              XCHG    AX,BX
94              XCHG    AX,SP
95              XCHG    AX,BP
96              XCHG    AX,SI
97              XCHG    AX,DI
98              CBW
99              CWD
9A9A909090      CALL    9090:909A
9B              WAIT
9C              PUSHF
9D              POPF                            ST=74F5
9E              SAHF
9F              LAHF
A0A090          MOV     AL,[90A0]                       DS:90A0=EC
A1A190          MOV     AX,[90A1]                       DS:90A1=EC83
A2A290          MOV     [90A2],AL                       DS:90A2=EC
A3A390          MOV     [90A3],AX                       DS:90A3=5704
A4              MOVSB                   DS:0001=00      ES:0002=69=i
A5              MOVSW                   DS:0001=7800    ES:0002=0069
A6              CMPSB                   DS:0001=00      ES:0002=69=i
A7              CMPSW                   DS:0001=7800    ES:0002=0069
A8A8            TEST    AL,A8h
A9A990          TEST    AX,90A9h
AA              STOSB                                   ES:0002=4A=J
AB              STOSW                                   ES:0002=004A
AC              LODSB                   DS:0001=00
```

```
8389909090      OR nstd WORD PTR [BX+DI+9090],-70h       DS:9095=9006
8392909090      ADC     WORD PTR [BP+SI+9090],-70h       SS:9095=0000
839B909090      SBB     WORD PTR [BP+DI+9090],-70h       SS:9096=0000
83A4909090      ANDnstd WORD PTR [SI+9090],-70h          DS:9091=278B
83AD909090      SUB     WORD PTR [DI+9090],-70h          DS:9092=7C27
83B6909090      XORnstd WORD PTR [BP+9090],-70h          SS:9094=0000
83BF909090      CMP     WORD PTR [BX+9090],-70h          DS:9093=FE7C
8C809090        MOV     [BX+SI+9090],ES                  DS:9094=06FE
8C899090        MOV     [BX+DI+9090],CS                  DS:9095=9006
8C929090        MOV     [BP+SI+9090],SS                  SS:9095=0000
8C9B9090        MOV     [BP+DI+9090],DS                  SS:9096=0000
8CA49090        MOVnstd [SI+9090],ES                     DS:9091=278B
8CAD9090        MOVnstd [DI+9090],CS                     DS:9092=7C27
8CB69090        MOVnstd [BP+9090],SS                     SS:9094=0000
8CBF9090        MOVnstd [BX+9090],DS                     DS:9093=FE7C
8E809090        MOV     ES,[BX+SI+9090]                  DS:9094=06FE
8E899090        MOV     CS,[BX+DI+9090]                  DS:9095=9006
8E929090        MOV     SS,[BP+SI+9090]                  SS:9095=0000
8E9B9090        MOV     DS,[BP+DI+9090]                  SS:9096=0000
8EA49090        MOVnstd ES,[SI+9090]                     DS:9091=278B
8EAD9090        MOVnstd CS,[DI+9090]                     DS:9092=7C27
8EB69090        MOVnstd SS,[BP+9090]                     SS:9094=0000
8EBF9090        MOVnstd DS,[BX+9090]                     DS:9093=FE7C
8F809090        POP     [BX+SI+9090]            ST=74F5  DS:9094=06FE
8F899090        POPnstd [BX+DI+9090]            ST=74F5  DS:9095=9006
8F929090        POPnstd [BP+SI+9090]            ST=74F5  SS:9095=0000
8F9B9090        POPnstd [BP+DI+9090]            ST=74F5  SS:9096=0000
8FA49090        POPnstd [SI+9090]               ST=74F5  DS:9091=278B
8FAD9090        POPnstd [DI+9090]               ST=74F5  DS:9092=7C27
8FB69090        POPnstd [BP+9090]               ST=74F5  SS:9094=0000
8FBF9090        POPnstd [BX+9090]               ST=74F5  DS:9093=FE7C
C080909090      ROL     BYTE PTR [BX+SI+9090],90h        DS:9094=FE
C089909090      ROR     BYTE PTR [BX+DI+9090],90h        DS:9095=06
C092909090      RCL     BYTE PTR [BP+SI+9090],90h        SS:9095=00
C09B909090      RCR     BYTE PTR [BP+DI+9090],90h        SS:9096=00
C0A4909090      SHL     BYTE PTR [SI+9090],90h           DS:9091=8B
C0AD909090      SHR     BYTE PTR [DI+9090],90h           DS:9092=27='
C0B6909090      SHLnstd BYTE PTR [BP+9090],90h           SS:9094=00
C0BF909090      SAR     BYTE PTR [BX+9090],90h           DS:9093=7C=¦
C180909090      ROL     WORD PTR [BX+SI+9090],90h        DS:9094=06FE
C189909090      ROR     WORD PTR [BX+DI+9090],90h        DS:9095=9006
C192909090      RCL     WORD PTR [BP+SI+9090],90h        SS:9095=0000
C19B909090      RCR     WORD PTR [BP+DI+9090],90h        SS:9096=0000
C1A4909090      SHL     WORD PTR [SI+9090],90h           DS:9091=278B
C1AD909090      SHR     WORD PTR [DI+9090],90h           DS:9092=7C27
C1B6909090      SHLnstd WORD PTR [BP+9090],90h           SS:9094=0000
C1BF909090      SAR     WORD PTR [BX+9090],90h           DS:9093=FE7C
C680909090      MOV     BYTE PTR [BX+SI+9090],90h        DS:9094=FE
C689909090      MOVnstd BYTE PTR [BX+DI+9090],90h        DS:9095=06
C692909090      MOVnstd BYTE PTR [BP+SI+9090],90h        SS:9095=00
C69B909090      MOVnstd BYTE PTR [BP+DI+9090],90h        SS:9096=00
C6A4909090      MOVnstd BYTE PTR [SI+9090],90h           DS:9091=8B
C6AD909090      MOVnstd BYTE PTR [DI+9090],90h           DS:9092=27='
C6B6909090      MOVnstd BYTE PTR [BP+9090],90h           SS:9094=00
C6BF909090      MOVnstd BYTE PTR [BX+9090],90h           DS:9093=7C=¦
C78090909090    MOV     WORD PTR [BX+SI+9090],9090h      DS:9094=06FE
C78990909090    MOVnstd WORD PTR [BX+DI+9090],9090h      DS:9095=9006
C79290909090    MOVnstd WORD PTR [BP+SI+9090],9090h      SS:9095=0000
C79B90909090    MOVnstd WORD PTR [BP+DI+9090],9090h      SS:9096=0000
C7A490909090    MOVnstd WORD PTR [SI+9090],9090h         DS:9091=278B
C7AD90909090    MOVnstd WORD PTR [DI+9090],9090h         DS:9092=7C27
C7B690909090    MOVnstd WORD PTR [BP+9090],9090h         SS:9094=0000
C7BF90909090    MOVnstd WORD PTR [BX+9090],9090h         DS:9093=FE7C
D0809090        ROL     BYTE PTR [BX+SI+9090],1          DS:9094=FE
D0899090        ROR     BYTE PTR [BX+DI+9090],1          DS:9095=06
D0929090        RCL     BYTE PTR [BP+SI+9090],1          SS:9095=00
D09B9090        RCR     BYTE PTR [BP+DI+9090],1          SS:9096=00
D0A49090        SHL     BYTE PTR [SI+9090],1             DS:9091=8B
D0AD9090        SHR     BYTE PTR [DI+9090],1             DS:9092=27='
D0B69090        SHLnstd BYTE PTR [BP+9090],1             SS:9094=00
D0BF9090        SAR     BYTE PTR [BX+9090],1             DS:9093=7C=¦
D1809090        ROL     WORD PTR [BX+SI+9090],1          DS:9094=06FE
D1899090        ROR     WORD PTR [BX+DI+9090],1          DS:9095=9006
D1929090        RCL     WORD PTR [BP+SI+9090],1          SS:9095=0000
D19B9090        RCR     WORD PTR [BP+DI+9090],1          SS:9096=0000
D1A49090        SHL     WORD PTR [SI+9090],1             DS:9091=278B
D1AD9090        SHR     WORD PTR [DI+9090],1             DS:9092=7C27
D1B69090        SHLnstd WORD PTR [BP+9090],1             SS:9094=0000
D1BF9090        SAR     WORD PTR [BX+9090],1             DS:9093=FE7C
D2809090        ROL     BYTE PTR [BX+SI+9090],CL         DS:9094=FE
D2899090        ROR     BYTE PTR [BX+DI+9090],CL         DS:9095=06
D2929090        RCL     BYTE PTR [BP+SI+9090],CL         SS:9095=00
D29B9090        RCR     BYTE PTR [BP+DI+9090],CL         SS:9096=00
D2A49090        SHL     BYTE PTR [SI+9090],CL            DS:9091=8B
D2AD9090        SHR     BYTE PTR [DI+9090],CL            DS:9092=27='
D2B69090        SHLnstd BYTE PTR [BP+9090],CL            SS:9094=00
D2BF9090        SAR     BYTE PTR [BX+9090],CL            DS:9093=7C=¦
D3809090        ROL     WORD PTR [BX+SI+9090],CL         DS:9094=06FE
D3899090        ROR     WORD PTR [BX+DI+9090],CL         DS:9095=9006
D3929090        RCL     WORD PTR [BP+SI+9090],CL         SS:9095=0000
D39B9090        RCR     WORD PTR [BP+DI+9090],CL         SS:9096=0000
D3A49090        SHL     WORD PTR [SI+9090],CL            DS:9091=278B
D3AD9090        SHR     WORD PTR [DI+9090],CL            DS:9092=7C27
D3B69090        SHLnstd WORD PTR [BP+9090],CL            SS:9094=0000
D3BF9090        SAR     WORD PTR [BX+9090],CL            DS:9093=FE7C
E480909090      TEST    BYTE PTR [BX+SI+9090],90h        DS:9094=FE
```

```
AD          LODSW                                   DS:0001-8988
AE          SCASB                                                 ES:0002=4A=J
AF          SCASW                                                 ES:0002=004A
B0B0        MOV     AL,B0h
B1B1        MOV     CL,B1h
B2B2        MOV     DL,B2h
B3B3        MOV     BL,B3h
B4B4        MOV     AH,B4h
B5B5        MOV     CH,B5h
B6B6        MOV     DH,B6h
B7B7        MOV     BH,B7h
B8B890      MOV     AX,90B8h
B9B990      MOV     CX,90B9h
BABA90      MOV     DX,90BAh
BBBB90      MOV     BX,90BBh
BCBC90      MOV     SP,90BCh
BDBD90      MOV     BP,90BDh
BEBE90      MOV     SI,90BEh
BFBF90      MOV     DI,90BFh
C0C090      ROL     AL,90h
C1C190      ROL     CX,90h
C2C290      RET     90C2h                   ST=74F5,E903,FF58,06C7,1274
C3          RET                             ST=74F5
C4C4        LES     AX,SP                                         Invalid form
C5C5        LDS     AX,BP                                         Invalid form
C6C690      MOV     DH,90h
C7C79090    MOV     DI,9090h
***** 64kB limit. Uncertain result:
C8C89090    ENTER   90C8h,90h
C9          LEAVE                                                 Warning: BP<SP
```

```
F689909090  ???     BYTE PTR [BX+DI+9090],90h        DS:9095=06
 F6929090   NOT     BYTE PTR [BP+SI+9090]            SS:9095=00
F69B9090    NEG     BYTE PTR [BP+DI+9090]            SS:9096=00
 F6A49090   MUL     BYTE PTR [SI+9090]               DS:9091=8B
F6AD9090    IMUL    BYTE PTR [DI+9090]               DS:9092=27='
 F6B69090   DIV     BYTE PTR [BP+9090]               SS:9094=00
F6BF9090    IDIV    BYTE PTR [BX+9090]               DS:9093=7C=|
 F78090909090 TEST  WORD PTR [BX+SI+9090],9090h      DS:9094=06FE
F78990909090 ???    WORD PTR [BX+DI+9090],9090h      DS:9095=9006
 F7929090   NOT     WORD PTR [BP+SI+9090]            SS:9095=0000
F79B9090    NEG     WORD PTR [BP+DI+9090]            SS:9096=0000
 F7A49090   MUL     WORD PTR [SI+9090]               DS:9091=278B
F7AD9090    IMUL    WORD PTR [DI+9090]               DS:9092=7C27
 F7B69090   DIV     WORD PTR [BP+9090]               SS:9094=0000
F7BF9090    IDIV    WORD PTR [BX+9090]               DS:9093=FE7C
 FE809090   INC     BYTE PTR [BX+SI+9090]            DS:9094=FE
FE899090    DEC     BYTE PTR [BX+DI+9090]            DS:9095=06
 FE929090   CALL    BYTE PTR [BP+SI+9090]            Invalid form
FE9B9090    CALL FAR BYTE PTR [BP+DI+9090]           Invalid form
 FEA49090   JMP     BYTE PTR [SI+9090]               Invalid form
FEAD9090    JMP FAR BYTE PTR [DI+9090]               Invalid form
 FEB69090   PUSH    BYTE PTR [BP+9090]               Invalid form
FEBF9090    ???     BYTE PTR [BX+9090]               Invalid form
 FF809090   INC     WORD PTR [BX+SI+9090]            DS:9094=0[illegible]
FF899090    DEC     WORD PTR [BX+DI+9090]            DS:9095=9006
 FF929090   CALL    W.PTR [BP+SI+9090]               SS:9095=0000
FF9B9090    CALL FAR [BP+DI+9090]                SS:9096=0000,0000
 FFA49090   JMP     WORD PTR [SI+9090]               DS:9091=278B
FFAD9090    JMP FAR [DI+9090]                    DS:9092=7C27,06FE
 FFB69090   PUSH    [BP+9090]                        SS:9094=0000
FFBF9090    ???     [BX+9090]                        DS:9093=FE7C
```

```
01C4  BB ---- R
01C7  8E DB

01C9  2B DB
01CB  8E C3
01CD  26: A1 0004
01D1  A3 003B R
01D4  26: A1 0006
01D8  A3 003D R

01DB  26: C7 06 0004 0117 R
01E2  26: 8C 0E 0006
01E7  81 0E 0036 R 0100

01ED  8C 16 0041 R
01F1  89 26 003F R
01F5  8E 16 002C R
01F9  8B 26 002A R

01FD  FF 36 0036 R
0201  FF 36 0032 R
0205  FF 36 0034 R
0209  FF 36 0028 R
020D  FF 36 0030 R
0211  EB 93

0213  C8 0004 00
0217  60
0218  BE 0063 R
021B  62 36 005D R
021F  68 1234
0222  6A 56
0224  6A F9
0226  6A FF
0228  83 C4 08

022B  69 06 006D R 2222
0231  6B 06 006D R 44
0236  BB 0002
0239  69 DB 0400
023D  69 DB 0400
0241  BB 0003
0244  6B DB 05
0247  6B DB 05
024A  6B 85 006D R FF
024F  6B 04 FD


0252  1E
0253  07
0254  BF 006F R
0257  C7 05 6666
025B  C7 45 02 6666
0260  B9 0003
0263  BA 0100
0266  F3/ 6C
0268  BF 0075 R
026B  C7 05 7777
026F  6D

0270  C1 E3 02
0273  C1 06 006D R 03
0278  C1 4D FE 10
027C  61
027D  C9
027E  EB 93

0280  8D 36 006D R
0284  BA ABCD
0287  26: 6F
0289  6E
028A  EB 87
```

```
;-----------------------------------------------
;Dyrektywa TRACE NTR  (NTR=ilosc instr do sledz)
;Stos i DS moj. Musza byc nadane: 1) _registers_
;2) NTR (domyslnie 1), 3) MODTR (poczatkowo 3).
DYRT: MOV BX, dataseg   ;Na wszelki wypadek
    MOV DS, BX
;Chowaj wektor przerwania 1 Debuggera w DEBS1:
    SUB BX,BX
    MOV ES,BX
    MOV AX, ES:[4]
    MOV DEBS1, AX
    MOV AX, ES:[6]
    MOV DEBS1+2, AX
;Podlacz moja OP1 do przerwania 1:
    MOV ES:[4], OFFSET OP1
    MOV ES:[6], CS
    OR  RFL, 0100H  ;Ustaw TF=1 we flagach uzytk
;Przejdz na stos uzytkownika:
    MOV MONSS, SS   ;Chowaj stos moj (zbedne?)
    MOV MONSP, SP
    MOV SS, RSS     ;Blokada przerw na 1 instr
    MOV SP, RSP
;Przygotuj stos jak w obsludze przerwania 1:
    PUSH RFL
    PUSH RCS
    PUSH RIP
    PUSH RBP
    PUSH RDS
  JMP WYKAI ;Wyk akt, a po prz listuj nast instr
;---------------- Instrukcje 186 --------------
I186: ENTER 4,0
    PUSHA
    MOV SI,OFFSET TABL+2
    BOUND SI,DWORD PTR TABL-4
    PUSH 1234H
    PUSH 56H
    PUSH -7
    PUSH 0FFFFH
    ADD SP,8
;-----------------------
    IMUL AX,DANA,2222H
    IMUL AX,DANA,44H
    MOV  BX, 2
    IMUL BX,BX,400H
    IMUL BX,400H
    MOV  BX, 3
    IMUL BX,BX,5
    IMUL BX,5
    IMUL AX,DANA [DI],0FFFFH
    IMUL AX,WORD PTR [SI],-3
;---------------
  ASSUME ES: dataseg
    PUSH DS
    POP ES
    MOV DI, OFFSET ZPORTUB
    MOV WORD PTR [DI], 6666H
    MOV WORD PTR [DI+2], 6666H
    MOV CX, 3
    MOV DX, 100H
 REP INS ZPORTUB, DX
    MOV DI, OFFSET ZPORTUS
    MOV WORD PTR [DI], 7777H
    INS ZPORTUS, DX
;---------------------
    SHL BX,2
    ROL DANA,3
    ROR WORD PTR [DI-2],16
    POPA
    LEAVE
    JMP I186
;----------------
    LEA SI, DANA
    MOV DX, 0ABCDH
    OUTS DX, ES:DANA
    OUTSB
    JMP I186
```

Załącznik nr 8 – Zawartość fragmentu programu z zał. 7 wyprowadzona przez DASM 1.86

```
3C26:01C4 BB163C        MOV     BX,3C16h                        ..<
3C26:01C7 8EDB          MOV     DS,BX                           ..
3C26:01C9 2BDB          SUB     BX,BX                           +.
3C26:01CB 8EC3          MOV     ES,BX                           ..
3C26:01CD 26            ES:                                     &
3C26:01CE A10400        MOV     AX,[0004]                       ...
3C26:01D1 A33B00        MOV     [003B],AX                       .;.
3C26:01D4 26            ES:                                     &
3C26:01D5 A10600        MOV     AX,[0006]                       ...
3C26:01D8 A33D00        MOV     [003D],AX                       .=.
3C26:01DB 26            ES:                                     &
3C26:01DC C70604001701  MOV     WORD PTR [0004],0117h           ......
3C26:01E2 26            ES:                                     &
3C26:01E3 8C0E0600      MOV     [0006],CS                       ....
3C26:01E7 810E36000001  OR      WORD PTR [0036],0100h           ..6...
3C26:01ED 8C164100      MOV     [0041],SS                       ..A.
3C26:01F1 89263F00      MOV     [003F],SP                       .&?.
3C26:01F5 8E162C00      MOV     SS,[002C]                       ..,.
3C26:01F9 8B262A00      MOV     SP,[002A]                       .&*.
3C26:01FD FF363600      PUSH    [0036]                          .66.
3C26:0201 FF363200      PUSH    [0032]                          .62.
3C26:0205 FF363400      PUSH    [0034]                          .64.
3C26:0209 FF362800      PUSH    [0028]                          .6(.
3C26:020D FF363000      PUSH    [0030]                          .60.
3C26:0211 EB93          JMP     SHORT 01A6                      ..
3C26:0213 C8040000      ENTER   0004h,00h                       ....
3C26:0217 60            PUSHA                                   `
3C26:0218 BE6300        MOV     SI,0063h                        .c.
3C26:021B 62365D00      BOUND   SI,[005D]                       b6].
3C26:021F 683412        PUSH    1234h                           h4.
3C26:0222 6A56          PUSH    +56h                            jV
3C26:0224 6AF9          PUSH    -07h                            j.
3C26:0226 6AFF          PUSH    -01h                            j.
3C26:0228 83C408        ADD     SP,+08h                         ...
3C26:022B 69066D002222  IMUL    AX,WORD PTR [006D],2222h        i.m.""
3C26:0231 6B066D0044    IMUL    AX,WORD PTR [006D],+44h         k.m.D
3C26:0236 BB0200        MOV     BX,0002h                        ...
3C26:0239 69DB0004      IMUL    BX,BX,0400h                     i...
3C26:023D 69DB0004      IMUL    BX,BX,0400h                     i...
3C26:0241 BB0300        MOV     BX,0003h                        ...
3C26:0244 6BDB05        IMUL    BX,BX,+05h                      k..
3C26:0247 6BDB05        IMUL    BX,BX,+05h                      k..
3C26:024A 6B856D00FF    IMUL    AX,WORD PTR [DI+006D],-01h      k.m..
3C26:024F 6B04FD        IMUL    AX,WORD PTR [SI],-03h           k..
3C26:0252 1E            PUSH    DS                              .
3C26:0253 07            POP     ES                              .
3C26:0254 BF6F00        MOV     DI,006Fh                        .o.
3C26:0257 C7056666      MOV     WORD PTR [DI],6666h             ..ff
3C26:025B C745026666    MOV     WORD PTR [DI+02],6666h          .E.ff
3C26:0260 B90300        MOV     CX,0003h                        ...
3C26:0263 BA0001        MOV     DX,0100h                        ...
3C26:0266 F3            REP/E/Z                                 .
3C26:0267 6C            INSB                                    l
3C26:0268 BF7500        MOV     DI,0075h                        .u.
3C26:026B C7057777      MOV     WORD PTR [DI],7777h             ..ww
3C26:026F 6D            INSW                                    m
3C26:0270 C1E302        SHL     BX,02h                          ...
3C26:0273 C1066D0003    ROL     WORD PTR [006D],03h             ..m..
3C26:0278 C14DFE10      ROR     WORD PTR [DI-02],10h            .M..
3C26:027C 61            POPA                                    a
3C26:027D C9            LEAVE                                   .
3C26:027E EB93          JMP     SHORT 0213                      ..
3C26:0280 8D366D00      LEA     SI,[006D]                       .6m.
3C26:0284 BACDAB        MOV     DX,ABCDh                        ...
3C26:0287 26            ES:                                     &
3C26:0288 6F            OUTSW                                   o
3C26:0289 6E            OUTSB                                   n
3C26:028A EB87          JMP     SHORT 0213                      ..
```

Załącznik nr 9 – Przykład użycia DASM186 dla wylistowania z listy 2 zał. 5 wszystkich instrukcji up 80186/188 i 80286. TRYB = 6000 H.


```
-g 33

AX=0000  BX=0000  CX=1A57  DX=0000  SP=0080  BP=0000  SI=25AC  DI=25A4
DS=25AC  ES=2594  SS=25A4  CS=25B9  IP=0033   NV UP EI PL ZR AC PE NC
25B9:0033 E80300        CALL    0039
-e cs:7e 0 60
-e cs:9a 0 60
-g =6c 36

25AC:0012 0F           286code                                  If so, 2-5 bytes
25AC:0012 60           186code                                  Possibly >1 byte
25AC:0012 61           186code                                  Possibly >1 byte
25AC:0012 62           186code                                  Possibly >1 byte
25AC:0012 63           286code                                  If so, 2-5 bytes
25AC:0012 68           186code                                  Possibly >1 byte
25AC:0012 69           186code                                  Possibly >1 byte
25AC:0012 6A           186code                                  Possibly >1 byte
25AC:0012 6B           186code                                  Possibly >1 byte
25AC:0012 6C           186code                                  Possibly >1 byte
25AC:0012 6D           186code                                  Possibly >1 byte
25AC:0012 6E           186code                                  Possibly >1 byte
25AC:0012 6F           186code                                  Possibly >1 byte
25AC:0012 C0           186code                                  Possibly >1 byte
25AC:0012 C1           186code                                  Possibly >1 byte
25AC:0012 C8           186code                                  Possibly >1 byte
25AC:0012 C9           186code                                  Possibly >1 byte
25AC:0012 D0B69090     186code BYTE PTR [BP+9090],1                  ....
25AC:0012 D1B69090     186code WORD PTR [BP+9090],1                  ....
25AC:0012 D2B69090     186code BYTE PTR [BP+9090],CL                 ....
25AC:0012 D3B69090     186code WORD PTR [BP+9090],CL                 ....

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 0004 0010 25B9 25AC 25AC 5555 0101010101010101
25AC:0012 FFBF9090     ???     [BX+9090]                          DS:9093=A241

AX=0004  BX=0004  CX=3F3F  DX=0003  SP=0080  BP=0000  SI=0000  DI=00C7
DS=25AC  ES=2594  SS=25A4  CS=25B9  IP=0036   NV UP EI PL ZR NA PE NC
25B9:0036 90            NOP
-q
```

Załącznik nr 10 – śledzenie przez TASM86 wykonania fragmentu programu z zał. 7.

```
 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 0004 0080 25AC 25B4 25B4 0000 0000000000000000
25C4:0213 C8040000     ENTER   0004h,00h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 007E 007A 25AC 25B4 25B4 0102 0000000100000010
25C4:0217 60           PUSHA                          ST=0213,25C4,0004

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 007E 006A 25AC 25B4 25B4 0102 0000000100000010
25C4:0218 BE6300       MOV     SI,0063h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0102 0000000100000010
25C4:021B 62365D00     BOUND   SI,[005D]                    DS:005D=0061,006D

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0102 0000000100000010
25C4:021F 683412       PUSH    1234h                  ST=0002,0001,007E,007A,0003

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 0068 25AC 25B4 25B4 0102 0000000100000010
25C4:0222 6A56         PUSH    +56h                   ST=1234,0002,0001,007E,007A

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 0066 25AC 25B4 25B4 0102 0000000100000010
25C4:0224 6AF9         PUSH    -07h                   ST=0056,1234,0002,0001,007E

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 0064 25AC 25B4 25B4 0102 0000000100000010
25C4:0226 6AFF         PUSH    -01h                   ST=FFF9,0056,1234,0002,0001

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 0062 25AC 25B4 25B4 0102 0000000100000010
25C4:0228 83C408       ADD     SP,+08h                ST=FFFF,FFF9,0056,1234,0002

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0106 0000000100000110
25C4:022B 69066D002222 IMUL    AX,WORD PTR [006D],2222h          DS:006D=0003

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
6666 0003 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0156 0000000101010110
25C4:0231 6B066D0044   IMUL    AX,WORD PTR [006D],+44h           DS:006D=0003

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 0003 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0156 0000000101010110
25C4:0236 BB0200       MOV     BX,0002h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 0002 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0156 0000000101010110
25C4:0239 69DB0004     IMUL    BX,BX,0400h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 0800 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0156 0000000101010110
25C4:023D 69DB0004     IMUL    BX,BX,0400h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 0000 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0913 0000100100010011
25C4:0241 BB0300       MOV     BX,0003h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 0003 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0913 0000100100010011
25C4:0244 6BDB05       IMUL    BX,BX,+05h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 000F 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0156 0000000101010110
25C4:0247 6BDB05       IMUL    BX,BX,+05h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
00CC 004B 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0156 0000000101010110
25C4:024A 6B856D00FF   IMUL    AX,WORD PTR [DI+006D],-01h        DS:006F=6666

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
999A 004B 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:024F 6B04FD       IMUL    AX,WORD PTR [SI],-03h          DS:0063=0006

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0252 1E           PUSH    DS                     ST=0002,0001,007E,007A,0003

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0005 4567 0002 0063 007E 0068 25AC 25B4 25B4 0196 0000000110010110
25C4:0253 07           POP     ES                     ST=25B4,0002,0001,007E,007A

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0005 4567 0002 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0254 BF6F00       MOV     DI,006Fh

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0005 4567 006F 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0257 C7056666     MOV     WORD PTR [DI],6666h            DS:006F=6666
```

```
 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0005 4567 006F 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:025B C745026666   MOV     WORD PTR [DI+02],6666h            DS:0071=6666

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0005 4567 006F 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0260 B90300        MOV     CX,0003h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0003 4567 006F 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0263 BA0001        MOV     DX,0100h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0003 0100 006F 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0266 F3            REP/E/Z
25C4:0267 6C            INSB

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0002 0100 0070 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0266 F3            REP/E/Z
25C4:0267 6C            INSB

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0001 0100 0071 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0266 F3            REP/E/Z
25C4:0267 6C            INSB

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0000 0100 0072 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0266 F3            REP/E/Z
25C4:0267 6C            INSB

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0000 0100 0072 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0268 BF7500        MOV     DI,0075h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0000 0100 0075 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:026B C7057777      MOV     WORD PTR [DI],7777h               DS:0075=7777

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0000 0100 0075 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:026F 6D            INSW

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 004B 0000 0100 0077 0063 007E 006A 25AC 25B4 25B4 0196 0000000110010110
25C4:0270 C1E302        SHL     BX,02h

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 012C 0000 0100 0077 0063 007E 006A 25AC 25B4 25B4 0102 0000000100000010
25C4:0273 C1066D0003    ROL     WORD PTR [006D],03h               DS:006D=0003

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 012C 0000 0100 0077 0063 007E 006A 25AC 25B4 25B4 0102 0000000100000010
25C4:0278 C14DFE10      ROR     WORD PTR [DI-02],10h              DS:0075=FF02

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
FFEE 012C 0000 0100 0077 0063 007E 006A 25AC 25B4 25B4 0103 0000000100000011
25C4:027C 61            POPA                              ST=0002,0001,007E,007A,0003

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 007E 007A 25AC 25B4 25B4 0103 0000000100000011
25C4:027D C9            LEAVE                             ST=0213,25C4,0004

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 0004 0080 25AC 25B4 25B4 0103 0000000100000011
25C4:027E EB93          JMP     SHORT 0213

 AX   BX   CX   DX   DI   SI   BP   SP   SS   ES   DS   FL =....ODITSZ.A.P.C
0001 0003 0005 4567 0002 0001 0004 0080 25AC 25B4 25B4 0103 0000000100000011
25C4:0213 C8040000      ENTER   0004h,00h
```

Załącznik nr 11 — Śledzenie przez DA86 wykonania pewnego programu w trybie „śledź" bez wyświetlania rejestrów.

MONITOR OPERATORSKI systemu R-T S I R T O S MERA-PIAP Warszawa

```
SRTS>mb
0500:360B B80400        MOV     AX,0004h

0500:360F E880CF        CALL    0592                    ST=0004
Esc chr: ^], Port: 1, Speed: 4800, Parity: None, Echo: Rem, Type ^]? for Help

0500:05A7 E8C504        CALL    0A6F                    ST=0004,1328,3612,0004

0500:0AA7 7D08          JGE     0AB1                    TRUE

0500:0AB5 7506          JNE/JNZ 0ABD                    TRUE

0500:0AC8 E88901        CALL    0C54                    ST=149F,3F59,1320,05AA,0004

0500:0C76 7517          JNE/JNZ 0C8F                    TRUE

0500:0C8F FFE0          JMP     AX

0500:0AD1 C3            RET                             ST=05AA,0004,1328,3612,0004

0500:05B2 740F          JE/JZ   05C3                    TRUE

0500:05C6 E863FF        CALL    052C                    ST=0004,1328,3612,0004

0500:0536 7C07          JL      053F                    FALSE

0500:053D 7529          JNE/JNZ 0568                    FALSE

0500:0544 7515          JNE/JNZ 055B                    FALSE
Esc chr: ^], Port: 1, Speed: 4800, Parity: None, Echo: Rem, Type ^]? for Help

0500:054B 740E          JE/JZ   055B                    TRUE

0500:0560 7406          JE/JZ   0568                    TRUE

0500:0568 E8B307        CALL    0D1E                    ST=1320,05C9,0004,1328,3612

0500:0D23 7501          JNE/JNZ 0D26                    TRUE

0500:0D26 C3            RET                             ST=056B,1320,05C9,0004,1328

0500:056B EBEA          JMP     SHORT 0557

0500:055A C3            RET                             ST=05C9,0004,1328,3612,0004

0500:05CF C3            RET                             ST=3612,0004

0500:3619 C3            RET                             ST=023F

0500:023F E8B80A        CALL    0CFA

0500:0D06 7508          JNE/JNZ 0D10                    FALSE

SRTS>
Esc chr: ^], Port: 1, Speed: 4800, Parity: None, Echo: Rem, Type ^]? for Help
```