

Andrzej Pawluczuk



Kompilator asm dla I8080/I8085

Białystok, lipiec 2022 (rev. 1.01)



Trochę na fali zainteresowania klasycznymi procesorami, trochę z powodu własnej potrzeby, powstało oprogramowanie do tworzenia programów na klasyczne mikroprocesory. Moja historia zainteresowania mikroprocesorami jest prawie tak stara jak same mikroprocesory. W latach 80-tych XX wieku zaczęły przenikać do naszego kraju same układy. Na sławetnym volumenie dawało się nabyć układy szmuglowane z zagranicy. Do budowy komputerów oprócz układów niezbędne jest jeszcze oprogramowanie narzędziowe. Z tym było trochę gorzej, gdyż warto pamiętać, że era kompów PC dopiero wchodziła do naszego kraju i właściwie poza uczelnią nie dawało się korzystać z takiego wsparcia.

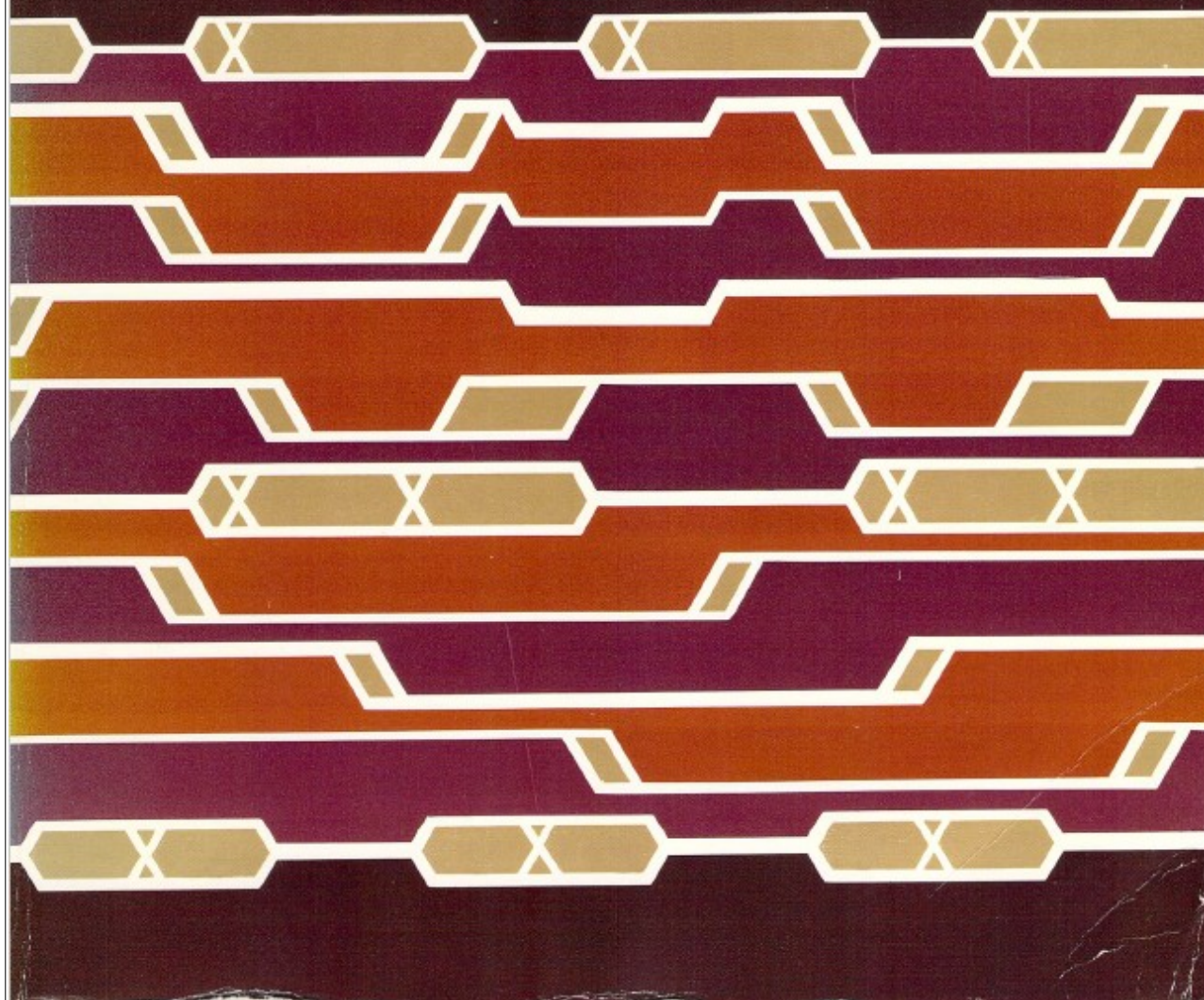
Skoro nie było kompów, to również nie było szmuglu samego oprogramowania. Pierwsze potyczki z prockami bazowały na ręcznym kompilowaniu programów. Ćwiczyło to w jakiś sposób pamięć, bo kody najczęściej stosowanych instrukcji były w małym paluszku. Później zaczęły pojawiać się sprzęty typu SPECTRUM. Było to już jakieś lepsze wsparcie.

W tamtych czasach nie było internetu, więc takie informacje jak lista instrukcji czy kody instrukcji trzeba było jakoś zdobywać. Tak się jakoś fajnie złożyło, że miałem dostęp do takiej książki:

 OSBORNE/McGraw-Hill

OSBORNE 4 & 8-Bit Microprocessor Handbook

Adam Osborne
Gerry Kane



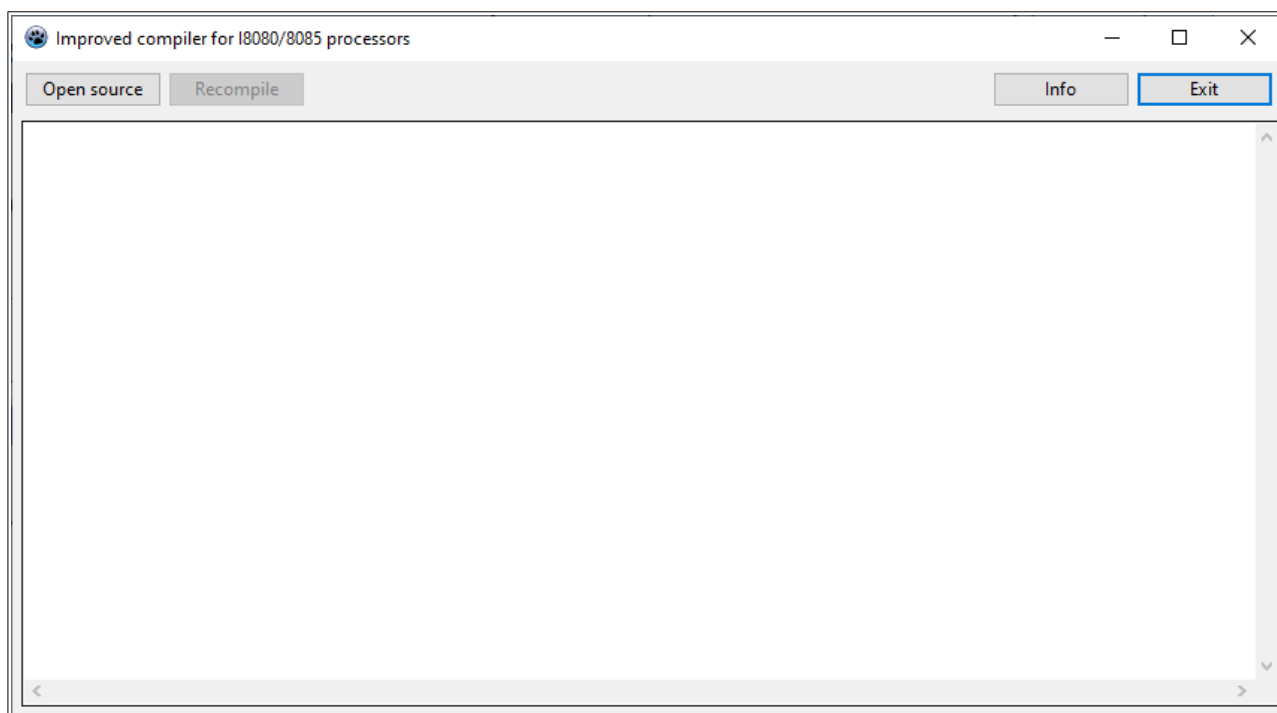
Dzisiaj to ja nawet ściągnąłem (bardziej chyba z sentymentu niż z rzeczywistej potrzeby). Jako wsparcie poszukującym narzędzi do programowania mikroprocesora I8080/I8085 mogę zaproponować własne rozwiązanie. Pierwsze wersje bazowały na DOS'e, więc powstało w dosyć zamierzchłych czasach, by ostatnio ponownie przeżyć swoje 5 minut. Jak zwykle, powstało z potrzeby chwili (tak jak teraz, gdyż od jakiegoś czasu chodzi mi po głowie pewna koncepcja uratowania przed śmiercią na śmietniku kilku procesorów, a przy okazji inni użytkownicy mają wsparcie).

Podstawowa wiedza dotycząca proca I8080 powinna być znana, ale pozwolę sobie napisać trochę na temat tego procka. To może dla niektórych być dosyć nowym doświadczeniem, dla innych spowodować swoistą podróż do tych niezwykłych czasów, gdy te procki były na topie. Postarajmy się pogrzebać we własnej głowie i wydobyć informacje na wierzch.

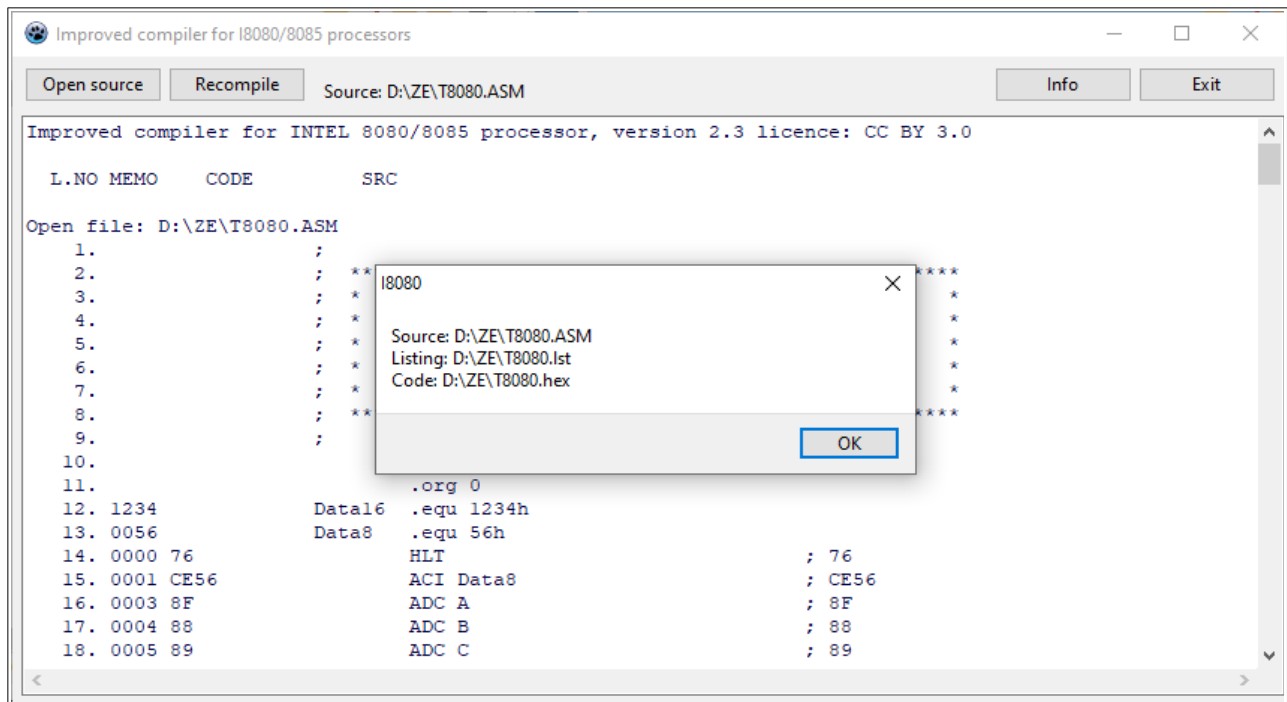
Kwestia oprogramowania narzędziowego dla procka 8085 niewiele się różni w stosunku do 8080, więc można stworzyć takie oprogramowanie tolerujące oba modele mikroprocesorów. Jednak kompilator musi „wiedzieć” dla jakiego procesora generowany jest kod binarny gdyż procesor 8085 jest rozszerzony o kilka instrukcji, które nie występują w 8080.

Kompilator (obecna wersja) jest utworzony w środowisku Lazarus i jest oprogramowaniem nie wymagającym jakiejkolwiek licencji. Może nie jest to jakieś super narzędzie, gdyż nie wykonuje linkowania niezależnie kompilowanych kawałków. Jego użycie sprowadza się do wskazania pliku do kompilacji, który musi zawierać kompletne oprogramowanie w formie źródłowej, chociaż pliki mogą być podzielone na kilka kawałków. Program obsługuje „zakłęcia” *include*, które dołącza do kompletu jakiś kawałek całości, choć skrajnie całość może składać się z jednego kawałka. W dołączanych plikach (przez *include*) mogą być kolejne pliki dołączane tym samym zaklęciem, jednak sytuacja rekurencyjnie zapętłonego *include* jest wykrywana i traktowana jako błąd kompilacji (nie jest wtedy generowany kod dla procka).

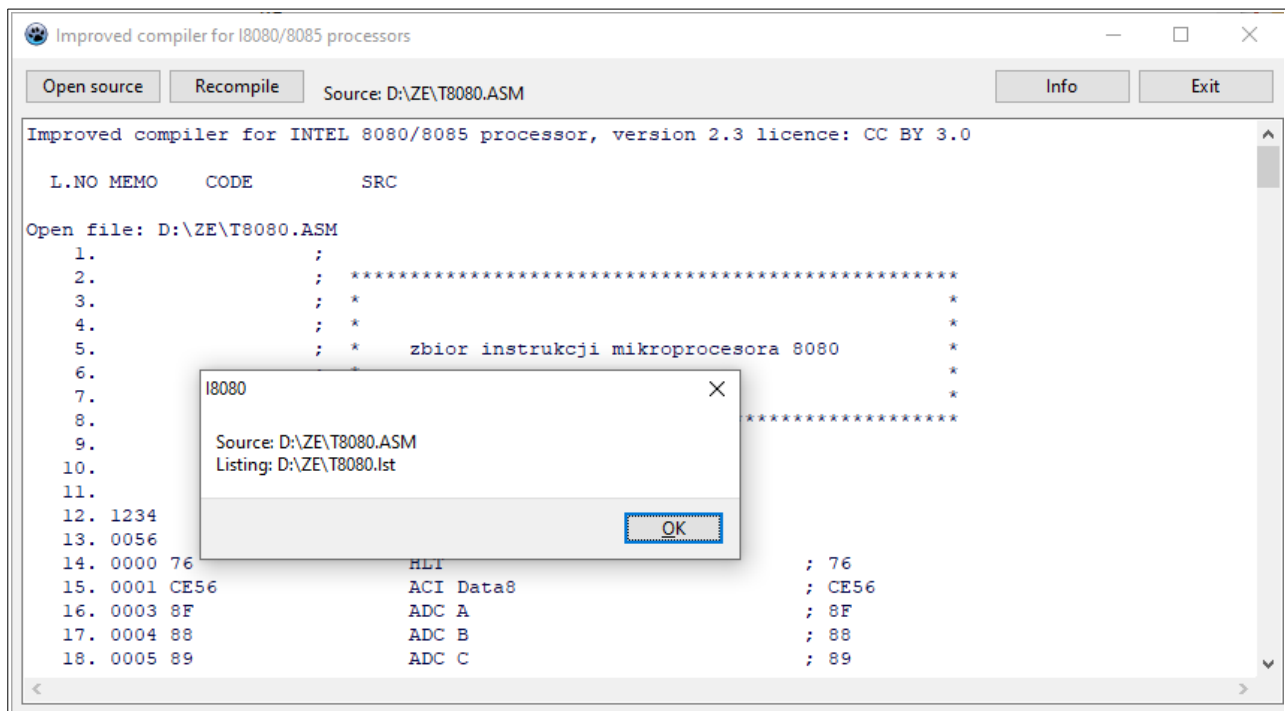
Po uruchomieniu kompilatora:



Najprostsze użycie programu, to jest uruchomić go i tak go pozostawić. Program źródłowy można edytować w dowolnym edytorze (może być NOTATNIK) i po zapisaniu zmian na dysku, myszką uchwycić nazwę pliku, przeciągnąć i upuścić na formę kompilatora. Spowoduje to otwarcie pliku, jego kompilację, w wyniku której powstaje raport z kompilacji i ewentualnie plik z programem dla procka 8080/8085 w formacie Intel-hex. Zawsze powstaje raport z kompilacji oraz jeżeli nie było błędów kompilacji, to powstaje plik w formacie Intel-hex, o czym informuje odpowiedni komunikat.



W przypadku błędnej kompilacji, nie powstaje plik dla procka, o czym informuje program.



Każde nowe upuszczenie nazwy pliku na formę kompilatora uruchamia proces kompilacji. Ten sam efekt można uzyskać po wyklikaniu nazwy pliku po użyciu przycisku „Open source”. Również można użyć przycisku „Recompile”, co oznacza, że kompilator ponownie wczyta aktualną wersję pliku źródłowego (ostatnio kompilowanego) i dokona jego kompilację.

Zakłęcie *include* ma następującą składnię (wszystkie dyrektywy zaczynają się od znaku kropki):

.include 'nazwa pliku'

Przykład kompilacji następującego programu:

```
;/
/ *****
/ *
/ *
/ *      zbior instrukcji mikroprocesora 8080
/ *
/ *
/ *****
/
/ .include 't8080.asm'
/
/
```

Generuje następujący raport:

L.NO	MEMO	CODE	SRC
------	------	------	-----

[illegible]

```

1.      ;
2.      ; *****
3.      ; *                                     *
4.      ; *                                     *
.....
58. 013A E3      XTHL      ; E3
59.      ; RIM      ; 20
60.      ; SIM      ; 30
61.      .end

```

11. ;
12. ;

```
Map information about constants and labels:
Data16          CONST=1234 hex [0001001000110100 bin,4660 dec]
Data8           CONST=0056 hex [0000000001010110 bin,86 dec]
```

Raport z kompilacji zawiera następujące informacje:

Improved compiler for INTEL 8080/8085 processor, version 2.3, li...

L.NO	MEMO	CODE	SRC
1.			;
(. . .)			
12.	1234		Data16 .equ 1234h
13.	0056		Data8 .equ 56h
14.	0000 0004 0008	48656C6C 6F20776F 726C64	Hello .defb 'Hello world'
15.	000B	76	HLT ; 76
16.	000C	CE56	ACI Data8 ; CE56

Dla stałych ich wartość

**Generalnie
adres
w pamięci**

**Dla obszarów
stałych ich wartość**

**Dla instrukcji
ich kody**

Podobnie jak w języku C, można z kodzie umieścić zakłęcia do kompilacji warunkowej. Są to:

- *.if* <warunek>
...
.endif
- *if* <warunek>
...
.else
...
.endif
- *.ifdef* <symbol>
...
.endif
- *ifdef* <symbol>
...
.else
...
.endif
- *.ifndef* <symbol>
...
.endif
- *ifndef* <symbol>
...
.else
...
.endif

Do tego dochodzi zakłęcie do definiowania symboli

.def <symbol> = <wartość logiczna>

W wyrażeniach <warunek> mogą wystąpić stałe *true* oraz *false*, zdefiniowane wcześniej (przez *.def*) symbole. W wyrażeniu mogą wystąpić operatory logiczne *and*, *or*, *xor* oraz *not*.

Przykład użycia (raport z kompilacji):

```
Improved compiler for INTEL 8080/8085 processor, version 2.3, licence: C

L.NO MEMO      CODE      SRC
Open file: D:\lazarus_project\I8080\test5.asm
 1.          ;
 2.          ; Program testowy dla kompilatora
 3.          ;
 4.          .if true
 5.          ;      wariant 1
 6. 0001      .def def1 = true
 7. 0001      .def def2 = true
 8. 0000      .def def3 = false
 9.          ;
10.          .endif
11.          .if false
12.          ;      wariant 2
13.          .
14.          . Tu mozna wpisac wszystko - nie bedzie kompilowa
15.          .
16.          .def def1 = true
17.          .def def2 = true
18.          .def def3 = true
19.          ;
20.          .endif
21.          .ifndef hardcore
22. 0001      .def def4 = true
23. 0001      .def hardcore = true
24.          .else
25.          .def def4 = false
26.          .endif
27.          .org 0h
28. 0000 00      start      nop          ;
29. 0001 3C          inr      a          ;
30. 0002 04          inr      b          ;
31. 0003 0C          inr      c          ;
32. 0004 14          inr      d          ;
33.          .if def1 xor def2 xor def3 xor def4
34.          inx      h          ;
35.          inx      b          ;
36.          inx      d          ;
37.          inx      sp         ;
38.          .if true
39.          add      b          ;
40.          add      c          ;
41.          add      d          ;
42.          add      e          ;
43.          .if false
44.          add      m          ;
45.          .endif
46.          ana      b          ;
47.          ana      c          ;
48.          ani      0abh       ;
49.          ana      m          ;
50.          .else
```

```
51.                                rra                                ;
52.                                rrl                                ;
53.                .endif
54.                                ral                                ;
55.                                rar                                ;
56.                .else
57. 0005 00                        nop                                ;
58. 0006 00                        nop                                ;
59. 0007 00                        nop                                ;
60. 0008 00                        nop                                ;
61. 0009 00                        nop                                ;
62. 000A 00                        nop                                ;
63. 000B 00                        nop                                ;
64. 000C 00                        nop                                ;
65. 000D 00                        nop                                ;
66. 000E 00                        nop                                ;
67. 000F 00                        nop                                ;
68. 0010 00                        nop                                ;
69.                .endif
70. 0011 29                        dad    h                        ;
71. 0012 19                        dad    d                        ;
72. 0013 09                        dad    b                        ;
73. 0014 39                        dad    sp                       ;
74. 0015 C31500    my             jmp    my                       ;
75.                                .end
Close file: D:\lazarus_project\I8080\test5.asm

Compilation :successful

Map information about constants and labels:
my                LABEL:0015 hex
start             LABEL:0000 hex

Code stored in file: D:\lazarus_project\I8080\test5.hex
```

Po kolumnie MEMO można zorientować się, jakie fragmenty programu źródłowego wchodzi do kompilacji, które są pominięte.

W programie źródłowym można zawrzeć informację dotyczącą modelu mikroprocesora, na jaki jest generowany kod programu. Lista instrukcji procesora I8085 jest o dwie instrukcje bogatsza w stosunku do listy instrukcji procesora I8080 (oraz w I8085 istnieją instrukcje nieudokumentowane → sam Intel nie opisał ich a jedynie jeden producent licencyjny). Są to instrukcje RIN oraz SIM. Zakłęcie *.I8080* informuje kompilator, że używany jest procesor 8080. Analogicznie *.I8085* dołącza do dopuszczalnych instrukcji dwie dodatkowe. Zastępczo jest ustalony wariant *.I8080*.

W programie można wprowadzić stałą nadając jej określoną wartość. Postać zapisu jest następująca:

`<stała> .equ <wyrażenie>`

Nie musi to być stała jako taka. W `<wyrażenie>` może wystąpić dowolne wyrażenie z operatorami arytmetycznymi, logicznymi oraz przesunięcia. Wszelkie chwyty są dozwolone a jedynym warunkiem jest to, by wyrażenie przed końcem zbioru było policzalne (elementy występujące w wyrażeniu mogą zostać określone później). W szczególnym przypadku są to etykiety w programie, które mają wartość wynikającą z jej adresu w przestrzeni pamięci. Jako stałe mogą wystąpić liczby binarne (z literką B na końcu), szesnastkowe (z literką H) na końcu, dziesiętne (bez oznaczenia literkowego na końcu), kody znaków jedno (8-bitów) lub dwuznakowych (16-bitów).

W wyrażeniach mogą wystąpić następujące operatory:

- + – operator dodawania arytmetycznego,
- * – operator mnożenia arytmetycznego,
- - – operator odejmowania arytmetycznego lub negacji arytmetycznej
- / operator dzielenia całkowitoliczbowego,
- mod (<dzielnia> , <dzielnik>) – funkcja dająca resztę z dzielenia <dzielnia> przez <dzielnik> ,
- shr – operator przesunięcia w prawo o określoną liczbę bitów w notacji <wartość> shr <liczba bitów> ,
- < – tożsame z shr,
- shl – operator przesunięcia w lewo o określoną liczbę bitów w notacji <wartość> shl <liczba bitów> ,
- > – tożsame z shl,
- or – operator sumy logicznej wykonanej na bitach,
- & – operator iloczynu logicznego wykonanej na bitach,
- xor – operator logiczny exclusive or wykonanej na bitach,
- ^ – tożsame z xor,
- not – operator negacji logicznej wykonanej na bitach,
- ~ – tożsame z not,
- lo (<wyrażenie>) – funkcja dająca młodsze 8 bitów z wyrażenia 16-bitowego,
- hi (<wyrażenie>) – funkcja dająca starsze 8 bitów z wyrażenia 16-bitowego,
- sizeof (<identyfikator typu>) – funkcja dająca wielkość typu wyrażoną w bajtach.

Przykład użycia (abstrakcyjny):

```

;
; Compiler test program
;
        .org 100h
c1      .equ "aA"                ; =6141
c2      .equ c1 > 4              ; =0614
c2a     .equ c1 shr 4            ; =0614
c3      .equ c1 < 8              ; =4100
c3a     .equ c1 shl 8            ; =4100
c4      .equ c1 shr 0            ; =6141
c5      .equ 5Ah                 ; =005A
c6      .equ 10101010b           ; =00AA
c7      .equ 10 + c6             ; =00B4
c8      .equ 'A'                 ; =0041
c9      .equ 'B'                 ; =0042
c10     .equ c8 + c9             ; =0083
c11     .equ c7 * 'A'            ; =2DB4
c12     .equ c5 / ' '            ; =0002
c12m    .equ mod ( c5 , ' ' )    ; =001A
c13     .equ c5 * c8             ; =16DA
c14     .equ 'A' * 'a'           ; =18A1
c15     .equ 'a' - 'A'           ; =0020
c16     .equ c5 * ( c7 + c10 ) + 1000h ; =7D56
c17     .equ lo ( c16 )          ; =0056
c18     .equ hi ( c16 )          ; =007D
c19     .equ c17 & c18           ; =0054
c20     .equ c5 or c2            ; =065E
c21     .equ c10 ^ 0ffh          ; =007C
c22     .equ lo ( ~ c1 )         ; =00BE
c23     .equ ( not c1 )          ; =9EBE
c24     .equ label1 - label3     ;
c25     .equ c26 + c24           ;
c26     .equ (label1+label2)/c5+c7+c8 ; =00FA
label1  .defb '*****'
c27     .equ (label1 + c25 ) / c5 + c7 ;
label2  .defm '#####'
c28     .equ (label1+label2)/c5+c6*c27 ;
        .end

```

Program jest niekompilowalny by pokazać ideę działania kompilatora. Będzie on do końca kombinował, by policzyć wartości wyrażeń. Ze względu na brak etykiety *label3*, nie da się obliczyć wartości wyrażenia identyfikowanego przez *c24*. Brak *c24* powoduje, że wyrażenie na *c25* jest niepoliczalne. To z kolei uniemożliwia obliczenia wartości wyrażeń oznaczonych jako *c27* i w dalszej kolejności *c28*.

Wszystkie stałe mogą wystąpić w operandach instrukcji, jako wartości w stałych obszarach danych oraz w deklaracji zmiennych. W zależności od kontekstu użycia brana jest wartość jako 8-bitowa. By zredukować ją do 8 bitów należy użyć funkcji *lo* lub *hi*.

Kompilator rozpoznaje standardowe typy zmiennych, tj. zmienne o rozmiarze jednego bajta, dwóch bajtów oraz czterech bajtów. Deklaracja zmiennych ma następującą syntaktykę:

`<nazwa zmiennej> .<typ zmiennej> <opcjonalnie wielkość>`

jako typ zmiennej może wystąpić:

- `.byte` – typ zmiennej zajmującej jeden bajt,
- `.word` – typ zmiennej zajmującej dwa bajty,
- `.long` – typ zmiennej zajmującej cztery bajty.

Kompilator toleruje również „archaiczne” zapisy

`<nazwa zmiennej> .defs <wielkość obszaru>`

które rezerwuje dla zmiennej obszar pamięci o podanej wielkości. Zastąpienie zakłęcia `.defs` przez `.byte` daje tożsamy efekt.

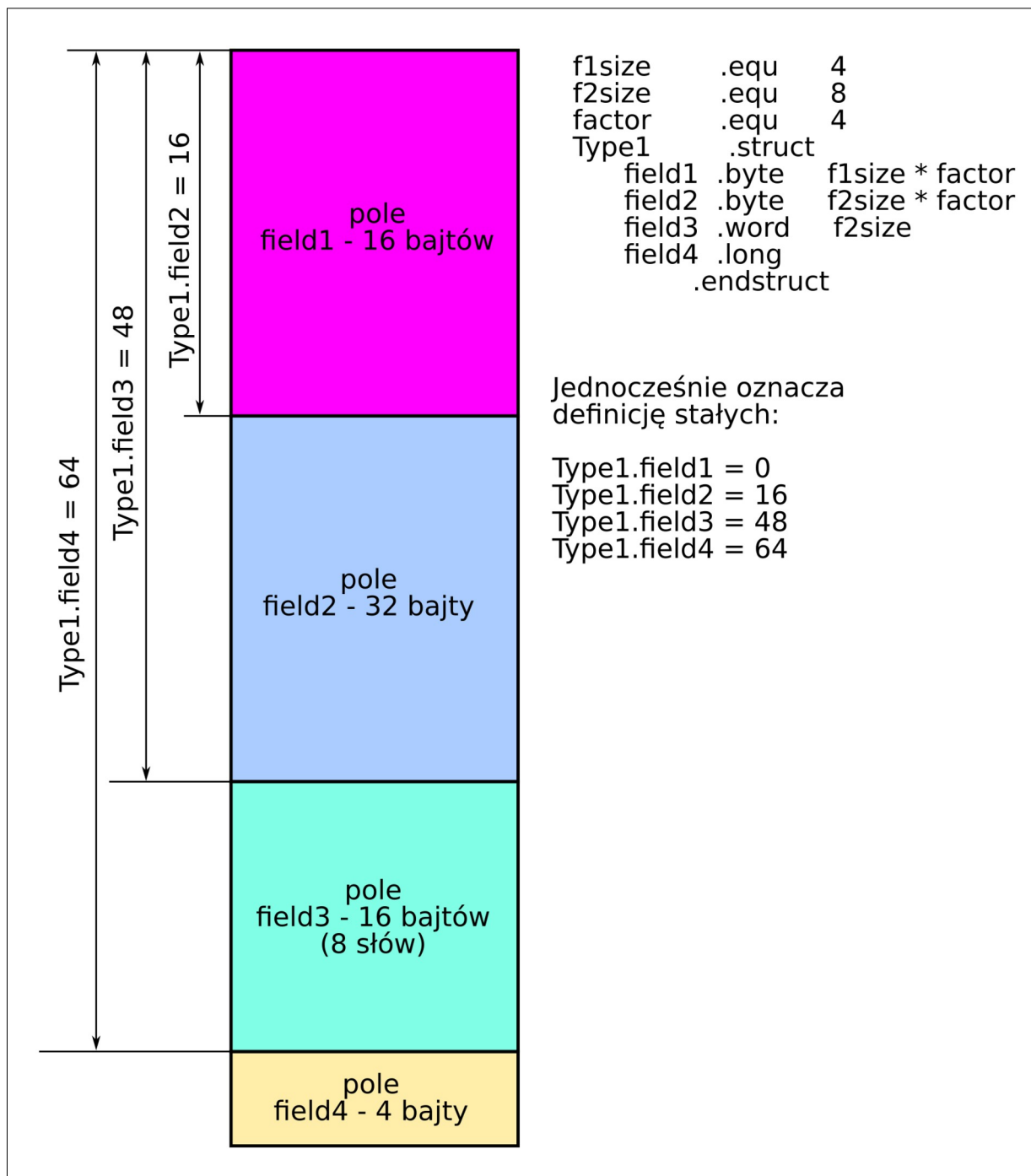
Można zdefiniować własny typ zmiennej jako strukturę (analogia do typu *record* w języku PASCAL lub *struct* w języku C). Postać zapisu jest następująca:

```
<identyfikator typu> .struct
                        <opis pola>
                        <opis pola>
                        <opis pola>
                        <opis pola>
                        .endstruct
```

Przykład:

```
f1size      .equ      4
f2size      .equ      8
factor      .equ      4
Typel       .struct
    field1   .byte     f1size * factor
    field2   .byte     f2size * factor
    field3   .word     f2size
    field4   .long
    .endstruct
```

Odpowiada to utworzeniu obszaru na zmienną o wielkości wynikającej z sumy wszystkich pól. Jednocześnie tworzy stałe określające położenia pól względem początku.



Po zdefiniowaniu typu można powołać zmienną nowego typu oraz w operandach instrukcji używać wprowadzonych symboli. Przykładowo:

```

f1size      .equ    4
f2size      .equ    8
factor      .equ    4
Type1       .struct
    field1   .byte   f1size * factor
    field2   .byte   f2size * factor
    field3   .word   f2size
    field4   .long
.endstruct

( . . . )

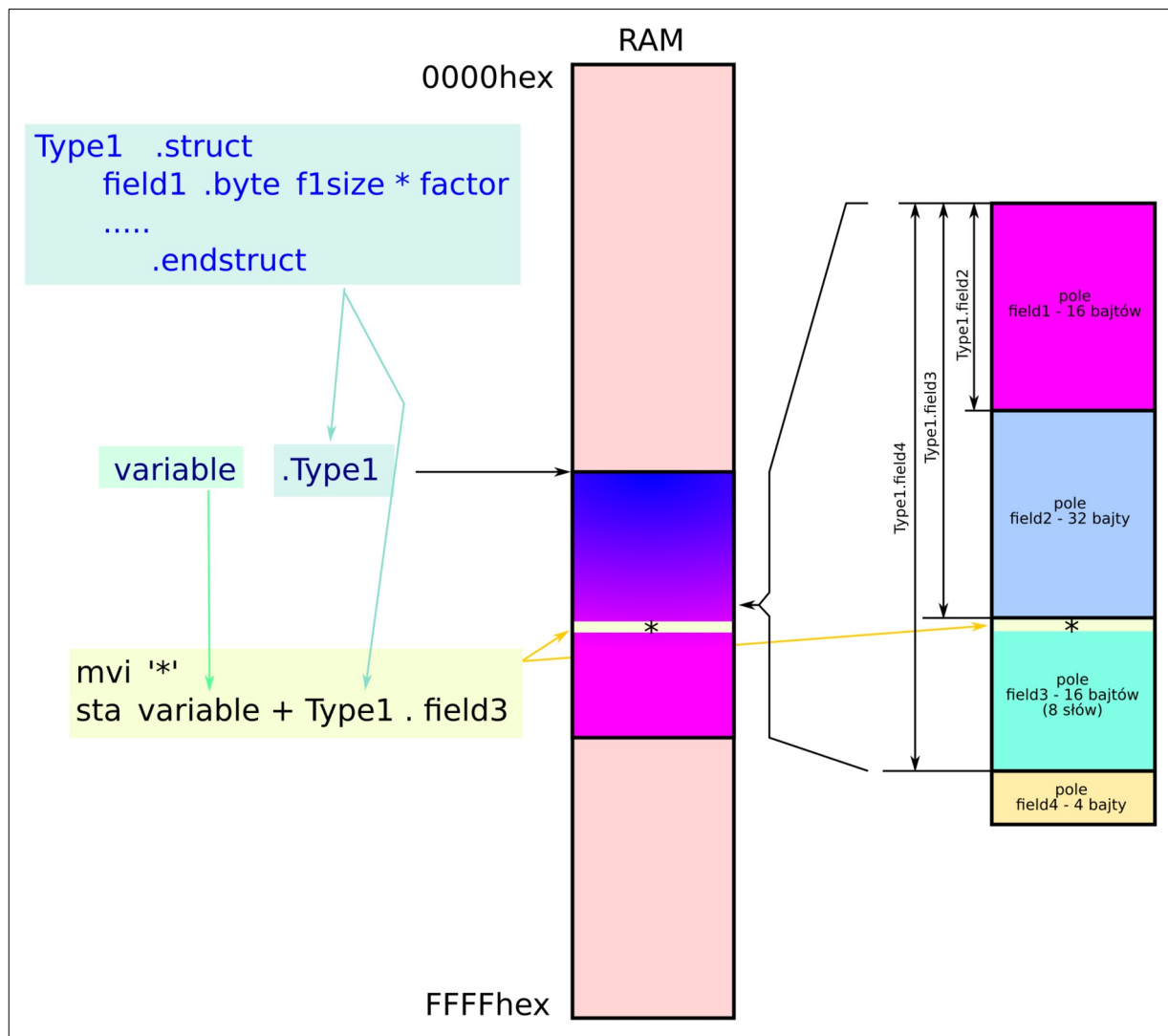
    mvi     '*'
    sta     variable + Type1 . Field3

( . . . )

variable    .Type1

```

daje:



Z typami związana jest funkcja, którą można używać we wszystkich wyrażeniach (przy definiowaniu stałych, tworzeniu obszarów stałych i w operandach instrukcji). Jest to:

- *sizeof* (<typ>) – funkcja zwraca wielkość całego typu wyrażoną w bajtach,
- *sizeof* (<typ>.<pole>) – funkcja zwraca wielkość określonego pola w strukturze określonej przez podaną nazwę typu.

Przykład:

Improved compiler for INTEL 8080/8085 processor, version 2.3, licence: CC

```

L.NO MEMO      CODE      SRC
Open file: D:\micro\asm8080\test7.asm
1.          ;
2.          ; Compiler test program
3.          ;
4.          .org 0h
5. 0004      f1size      .equ      4
6. 0008      f2size      .equ      8
7. 0004      factor      .equ      4
8. 0044      Type1       .struct
9. 0010          field1      .byte f1size * factor
10. 0020          field2      .byte f2size * factor
11. 0010          field3      .word f2size
12. 0004          field4      .long
13.          .endstruct
14. 0004      Type2       .struct
15. 0001          field1      .byte
16. 0001          field2      .byte
17. 0002          field3      .word
18.          .endstruct
19.          ;
20. 0007      Type3       .struct
21. 0004          field1      .long
22. 0002          field2      .word
23. 0001          field3      .byte
24.          .endstruct
25.          ;
26. 0001      expr1       .equ      sizeof ( byte )
27. 0002      expr2       .equ      sizeof ( word )
28. 0004      expr3       .equ      sizeof ( long )
29. 0044      expr4       .equ      sizeof ( Type1 )
30. 0010      expr41      .equ      sizeof ( Type1 . FIELD1 )
31. 0020      expr42      .equ      sizeof ( Type1 . FIELD2 )
32. 0010      expr43      .equ      sizeof ( Type1 . FIELD3 )
33. 0004      expr44      .equ      sizeof ( Type1 . FIELD4 )
34. 0004      expr5       .equ      sizeof ( Type2 )
35. 0001      expr51      .equ      sizeof ( Type2 . field1 )
36. 0001      expr52      .equ      sizeof ( Type2 . field2 )
37. 0002      expr53      .equ      sizeof ( Type2 . Field3 )
38. 0007      expr6       .equ      sizeof ( Type3 )
39. 0004      expr61      .equ      sizeof ( Type3 . field1 )
40. 0002      expr62      .equ      sizeof ( Type3 . field2 )
41. 0001      expr63      .equ      sizeof ( Type3 . field3 )
42. 0047      expr7       .equ      sizeof ( byte ) + sizeof ( word )
+ sizeof ( Type1 )

```

```

43. 000C          expr8          .equ ( sizeof ( byte ) + sizeof
( word ) ) * sizeof ( Type2 )
44.              ;
45.              ;
46.              ;
47. 0000 00          start      nop
48.              mvi          '*'
49. 0001 3200C0      sta          variable1
50. 0004 3253C0      sta          variable5 + Type3 . field3
51. 0007 2149C0      lxi          h , variable4
52. 000A 010100      lxi          b , Type2 . field2
53. 000D 09          dad          b
54. 000E 77          mov          m , a
55. 000F 3C          inr          a
56. 0010 324AC0      sta          variable4 + Type2 . field2
57. 0013 3A05C0      lda          Variable3 + Type1 . field1
58. 0016 3200C0      sta          variable1
59. 0019 C31900      halt        jmp          halt
60.              .org          0c000h
61. C000          variable1      .byte 1
62. C001          variable2      .word 2
63. C005          variable3      .byte sizeof ( Type1 )
64. C049          variable4      .byte sizeof ( Type2 )
65. C04D          variable5      .byte sizeof ( Type3 )
66. C054          variable6      .Type1
67. C098          variable7      .Type2          4
68. C0A8          variable8      .Type3          8
69. C0E0          variable9      .long
70.              .end

```

Close file: D:\micro\asm8080\test7.asm

Compilation :successful

Map information about constants and labels:

```

expr1          CONST=0001 hex [00000000000000001 bin,1 dec]
expr2          CONST=0002 hex [00000000000000010 bin,2 dec]
expr3          CONST=0004 hex [00000000000000100 bin,4 dec]
expr4          CONST=0044 hex [00000000001000100 bin,68 dec]
expr41         CONST=0010 hex [00000000000010000 bin,16 dec]
expr42         CONST=0020 hex [00000000000100000 bin,32 dec]
expr43         CONST=0010 hex [00000000000010000 bin,16 dec]
expr44         CONST=0004 hex [00000000000000100 bin,4 dec]
expr5          CONST=0004 hex [00000000000000100 bin,4 dec]
expr51         CONST=0001 hex [00000000000000001 bin,1 dec]
expr52         CONST=0001 hex [00000000000000001 bin,1 dec]
expr53         CONST=0002 hex [00000000000000010 bin,2 dec]
expr6          CONST=0007 hex [00000000000000111 bin,7 dec]
expr61         CONST=0004 hex [00000000000000100 bin,4 dec]
expr62         CONST=0002 hex [00000000000000010 bin,2 dec]
expr63         CONST=0001 hex [00000000000000001 bin,1 dec]
expr7          CONST=0047 hex [00000000001000111 bin,71 dec]
expr8          CONST=000C hex [00000000000001100 bin,12 dec]
flsize         CONST=0004 hex [00000000000000100 bin,4 dec]
f2size         CONST=0008 hex [00000000000001000 bin,8 dec]
factor         CONST=0004 hex [00000000000000100 bin,4 dec]
halt          LABEL:0019 hex

```



```

start          LABEL:0000 hex
variable1      LABEL:C000 hex
variable2      LABEL:C001 hex
variable3      LABEL:C005 hex
variable4      LABEL:C049 hex
variable5      LABEL:C04D hex
variable6      LABEL:C054 hex type: Type1
variable7      LABEL:C098 hex type: Type2
variable8      LABEL:C0A8 hex type: Type3
variable9      LABEL:C0E0 hex
Type1          STRUCT, size=0044 hex [68 dec], orgin=0000 hex
               field1      size=0010 hex [16 dec], offset=0 dec
               field2      size=0020 hex [32 dec], offset=+16 dec
               field3      size=0010 hex [16 dec], offset=+48 dec
               field4      size=0004 hex [4 dec], offset=+64 dec
Type2          STRUCT, size=0004 hex [4 dec], orgin=0000 hex
               field1      size=0001 hex [1 dec], offset=0 dec
               field2      size=0001 hex [1 dec], offset=+1 dec
               field3      size=0002 hex [2 dec], offset=+2 dec
Type3          STRUCT, size=0007 hex [7 dec], orgin=0000 hex
               field1      size=0004 hex [4 dec], offset=0 dec
               field2      size=0002 hex [2 dec], offset=+4 dec
               field3      size=0001 hex [1 dec], offset=+6 dec

Code stored in file: D:\micro\asm8080\test7.hex

```

Przy definiowaniu zmiennej można za identyfikatorem typu podać liczbę (ogólnie wyrażenie policzalne w chwili użycia). Brak takiej liczby jest tożsame z podaniem jej jako jeden. Taki zabieg oznacza zwielokrotnienie tworzonej zmiennej określoną liczbę razy i można to traktować jako *array of <coś>*.

W powyższym przykładzie wystąpiło:

```

variable3      .byte sizeof ( Type1 )
variable6      .Type1

```

i właściwie jest to tożsame: zajmuje w pamięci identyczny obszar natomiast kompilator nie kontroluje, czy używane zmienne i typy wzajemnie mają ze sobą coś wspólnego. To człowiek panuje nad wszystkim a kompilator ma jedynie pomóc.

Czasami może się tak zdarzyć, że określona etykieta (etykieta w programie, zmienna) musi mieć parzysty adres. Piszac program trudno jest zagwarantować, by taki wymóg został spełniony. By wesprzeć społeczność piszących, kompilator oferuje pewne wsparcie w postaci zaklęcia *.align*.

Postać jest następująca:

.align <liczba>

gdzie <liczba> to: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024

Użycie tej formuły powoduje, że następny zapis będzie miał adres w pamięci dostosowany do oczekiwanych wymogów (podzielny przez podaną jako operand wartość).

Przykład użycia:

```
Improved compiler for INTEL 8080/8085 processor, version 2.3, licence: CC

  L.NO MEMO      CODE      SRC
Open file: D:\micro\asm8080\test6.asm
  1.          ;
  2.          ; Compiler test program
  3.          ;
  4.          .org 0h
  5. 0000 00      start    nop
  6. 0001 CD4000   call    entry      ;
  7. 0004 C34200   jmp     stop       ;
  8.          .align      64
  9. 0040 00      entry    nop        ;
 10. 0041 C9      ret      ;
 11. 0042 C34200   stop     jmp      stop ;
 12.          ;=====;
 13.          .org 8000h      ;
 14. 8000      space    .defs 400h    ;
 15. 8400      screen   .defs 400h    ;
 16. 8800      var0     .byte         ;
 17.          .align      2        ;
 18. 8802      var1     .byte         ;
 19.          .align      2        ;
 20. 8804      var2     .byte         ;
 21.          .align      2        ;
 22. 8806      var3     .byte         ;
 23.          .align      2        ;
 24. 8808      var4     .byte         ;
 25.          .align      2        ;
 26. 880A      var5     .word         ;
 27.          .align      8        ;
 28. 8810      var6     .word         ;
 29.          .align     256        ;
 30. 8900      var7     .byte 128     ;
 31. 8980      var8     .byte         ;
 32.          .end
Close file: D:\micro\asm8080\test6.asm

Compilation :successful
```

Map information about constants and labels:

<i>entry</i>	<i>LABEL:0040 hex</i>
<i>screen</i>	<i>LABEL:8400 hex</i>
<i>space</i>	<i>LABEL:8000 hex</i>
<i>start</i>	<i>LABEL:0000 hex</i>
<i>stop</i>	<i>LABEL:0042 hex</i>
<i>var0</i>	<i>LABEL:8800 hex</i>
<i>var1</i>	<i>LABEL:8802 hex</i>
<i>var2</i>	<i>LABEL:8804 hex</i>
<i>var3</i>	<i>LABEL:8806 hex</i>
<i>var4</i>	<i>LABEL:8808 hex</i>
<i>var5</i>	<i>LABEL:880A hex</i>
<i>var6</i>	<i>LABEL:8810 hex</i>
<i>var7</i>	<i>LABEL:8900 hex</i>
<i>var8</i>	<i>LABEL:8980 hex</i>

Code stored in file: D:\micro\asm8080\test6.hex

Czasami (a właściwie bardzo rzadko) zachodzi potrzeba relokacji kodu, czyli wykonania kodu programu, który jest <tutaj> (bo jego miejscem składowania jest <tutaj>) a powinien znajdować się w <innym miejscu>. Kompilator wspiera takie zagrywki i oferuje zakłęcie *.radix*. Ma ono następujący zapis:

.radix <lokacja w pamięci>

zapis w postaci

.radix \$

oznacza powrót do normalności.

Zastosowanie tego zakłęcia powoduje, że raport kompilacji zawiera kolejną kolumnę LINK, która odpowiada miejscu uruchomienia kodu.

Przykład:

```
Improved compiler for INTEL 8080/8085 processor, version 2.3, licence: CC

  L.NO MEMO LINK      CODE          SRC
Open file: D:\micro\asm8080\test4.asm
  1.                                ;
  2.                                ; Compiler test program
  3.                                ;
  4.                                .org  0h
  5. 0001                      AnyConst1 .equ  1
  6. 0002                      AnyConst2 .equ  2
  7. 0003                      AnyConst3 .equ  3
  8. 0004                      AnyConst4 .equ  4
  9. 0005                      AnyConst5 .equ  5
 10. 0006                      AnyConst6 .equ  6
 11. 0007                      AnyConst7 .equ  7
 12. 0000 0000 00              start    nop
 13.                                ;
 14. 0001 0001 310000          lxi      sp , 0                      ;
 15. 0004 0004 211600          lxi      h , relsrc                  ;
 16. 0007 0007 110080          lxi      d , space                  ;
 17. 000A 000A 013500          lxi      b , rel_stop - rel_start;
 18. 000D 000D CD5500          call     copy                      ;
 19. 0010 0010 CD2280          call     entry                    ;
 20. 0013 0013 C34B00          jmp      continue                ;
 21.
 22. 0016 0016                  relsrc
 23.                                .radix      8000h
 24. 0016 8000                  rel_start
 25. 0016 8000 6D657373          message      .defm 'message to screen'
      001A      61676520
      001E      746F2073
      0022      63726565
      0026      6E
 26. 0027 8011 00              endmess      .defb 0
 27. 0028 8012 00001600          table      .defw start , relsrc ,
rel_start , rel_stop
      002C      00803580
```

```

28. 0030 801A 00801180      .defw message , endmess , entry
    0034      2280
29. 0036 8020 3380          .defw label
30. 0038 8022 00            entry    nop                ;
31. 0039 8023 210080        lxi     h , message      ;
32. 003C 8026 110084        lxi     d , screen        ;
33. 003F 8029 011100        lxi     b , endmess - message ;
34. 0042 802C CD5500        call    copy
35. 0045 802F CD3380        call    label                ;
36. 0048 8032 C9            ret                    ;
37. 0049 8033 00            label    nop                ;
38. 004A 8034 C9            ret                    ;
39. 004B 8035              rel_stop  ret                    ;
40.                          .radix      $                ;
41. 004B 004B 00            continue nop                ;
42. 004C 004C CD5300        call    cont1                ;
43. 004F 004F 216000        lxi     h , stop            ;
44. 0052 0052 E9            pchl                    ;
45. 0053 0053 00            cont1    nop                ;
46. 0054 0054 C9            ret                    ;
47. 0055 0055 7E            copy     mov     a , m          ;
48. 0056 0056 23            inx     h                    ;
49. 0057 0057 12            stax    d                    ;
50. 0058 0058 13            inx     d                    ;
51. 0059 0059 0B            dcx     b                    ;
52. 005A 005A 78            mov     a , b                ;
53. 005B 005B B1            ora     c                    ;
54. 005C 005C C8            rz                    ;
55. 005D 005D C35500        jmp     copy                ;
56.
57. 0060 0060 76            stop     hlt                    ;
58. 0061 0061 C36000        jmp     stop                ;
59.
60.                          ;
61.                          .org     8000h
62. 8000 8000              space     .defs 400h
63. 8400 8400              screen    .defs 400h
64. 8800 8800              var0      .byte 10h
65. 8810 8810              var1      .byte 10h
66. 8820 8820              var2      .byte 10h
67. 8830 8830              var3      .byte 10h
68. 8840 8840              var4      .byte 10h
69. 8850 8850              var5      .word 10h
70. 8870 8870              var6      .word 10h
71. 8890 8890              var7      .byte
72.                          .end

```

Close file: D:\micro\asm8080\test4.asm

Compilation :successful

Map information about constants and labels:

```

AnyConst1      CONST=0001 hex [0000000000000001 bin,1 dec]
AnyConst2      CONST=0002 hex [0000000000000010 bin,2 dec]
AnyConst3      CONST=0003 hex [0000000000000011 bin,3 dec]
AnyConst4      CONST=0004 hex [0000000000000100 bin,4 dec]
AnyConst5      CONST=0005 hex [0000000000000101 bin,5 dec]
AnyConst6      CONST=0006 hex [0000000000000110 bin,6 dec]
AnyConst7      CONST=0007 hex [0000000000000111 bin,7 dec]
cont1          LABEL:0053 hex

```

<i>continue</i>	<i>LABEL:004B hex</i>
<i>copy</i>	<i>LABEL:0055 hex</i>
<i>endmess</i>	<i>LABEL:8011 hex</i>
<i>entry</i>	<i>LABEL:8022 hex</i>
<i>label</i>	<i>LABEL:8033 hex</i>
<i>message</i>	<i>LABEL:8000 hex</i>
<i>relsrc</i>	<i>LABEL:0016 hex</i>
<i>rel_start</i>	<i>LABEL:8000 hex</i>
<i>rel_stop</i>	<i>LABEL:8035 hex</i>
<i>screen</i>	<i>LABEL:8400 hex</i>
<i>space</i>	<i>LABEL:8000 hex</i>
<i>start</i>	<i>LABEL:0000 hex</i>
<i>stop</i>	<i>LABEL:0060 hex</i>
<i>table</i>	<i>LABEL:8012 hex</i>
<i>var0</i>	<i>LABEL:8800 hex</i>
<i>var1</i>	<i>LABEL:8810 hex</i>
<i>var2</i>	<i>LABEL:8820 hex</i>
<i>var3</i>	<i>LABEL:8830 hex</i>
<i>var4</i>	<i>LABEL:8840 hex</i>
<i>var5</i>	<i>LABEL:8850 hex</i>
<i>var6</i>	<i>LABEL:8870 hex</i>
<i>var7</i>	<i>LABEL:8890 hex</i>

Code stored in file: D:\micro\asm8080\test4.hex

Raport z kompilacji zawiera:

Improved compiler for INTEL 8080/8085 processor, version

L.NO	MEMO	LINK	CODE	SRC
Open file: D:\micro\asm8080\test4.asm				
(. . .)				
12	0000	0000	00	start nop
13				;
14	0001	0001	310000	lxi sp , 0 ;
15	0004	0004	211600	lxi h , relsrc ;
16	0007	0007	110080	lxi d , space ;
17	000A	000A	013500	lxi b , rel_stop - rel_start;
18	000D	000D	CD5500	call copy ;
19	0010	0010	C12280	call entry ;
20	0013	0013	C34B00	jmp continue ;
21				;-----
22	0016	0016		relsrc
23				.radix 8000h
24	0016	8000		rel_start
25	0016	8000	6D657373	message .defm 'message to screen'
	001A		61676520	
	001E		746F2013	
	0022		63726565	
	0026		6E	
26	0027	8011	00	endmess .defb 0
27	0028	8012	00001600	table .defw start , relsrc , rel_start , rel_stop
	002C		00803580	
28	0030	801A	00801180	.defw message , endmess , entry
	0034		2280	
29	0036	8020	3380	.defw label
30	0038	8022	00	entry nop ;
31	0039	8023	210080	lxi h , message ;
32	003C	8026	110084	lxi d , screen ;
33	003F	8029	011100	lxi b , endmess - message ;

W tym obszarze zawartość kolumn jest identyczna

W tym obszarze zawartość kolumn jest różna

Adresacja pamięci (gdzie jest umieszczony kod)

Adresacja wynikająca z miejsca uruchomienia kodu

Mechanizm ten może okazać się przydatny przy bootowaniu systemu, gdzie kod programu musi w swoje miejsce załadować kod z jakiegoś nośnika. Skoro jest ładowany w swoje miejsce, to kod ładujący musi się przepisać w inne i z tamtego miejsca wykonać przewidziane operacje.