ASEM-51 Instrukcja obsługi- polskie tłumaczenie na podstawie instrukcji oryginalnej z dnia 31 grudnia 2002 roku (tłumaczenie z dnia 08.05.2007r.)

Od tłumaczącego.

Na kompilator ASEM5113 trafiłem kilka lat temu, lecz dopiero stosunkowo niedawno postanowiłem się z nim zapoznać i wykorzystać. Z wykształcenia jestem elektronikiem, a zawodowo zajmuję się automatyką. Mikrokontrolery rodziny 8051 poznałem sam po zakończeniu nauki. Nie jestem typem osoby, którą można nazwać specjalistą w tej dziedzinie, ale uważam, że zabawa z 51-ką jest dobrym wstępem do zrozumienia programowania mikrokontrolerów oraz ogólnych zasad pisania programów w języku asemblera. Wcześniej pracowałem z kompilatorem dostarczanym z zestawem AVT2250, ale po pewnym czasie przestał mi on wystarczyć. Długo szukałem innego, darmowego lub taniego narzędzia. Gdy myślałem, że nie znajdę, trafiłem na opisywane tu narzędzie. Moim zdaniem jest świetne. Niestety dla wielu hobbystów problemem jest bariera językowa. Postanowiłem więc przetłumaczyć dokumentację na język polski. Nie jest to profesjonalne tłumaczenie. Powstawało przez około pół roku i było pisane "w locie". Nie wszystko dało się przetłumaczyć wprost, więc tam, gdzie były problemy, starałem się korzystając z własnej wiedzy opisać to, co autor miał na myśli swoimi słowami. Starałem się trzymać stosowanego nazewnictwa oraz jak najbliżej zostać przy oryginalnej treści, ale wiem, że wciąż może być wiele błędów, mało jasnych wyrażeń lub moich "wpadek". Proszę więc o wyrozumiałość i ewentualne poinformowanie mnie o zauważonych "bykach".

Może nasuwać się pytanie: po co? Po co to robiłem? Widziałem, jak wielu utalentowanych ludzi rozwijało swe zdolności po wprowadzeniu ich w interesujący ich właśnie temat. Są to osoby zdolne, a nawet utalentowane, ale nie znające języka angielskiego na tyle, by dokumentacja urządzenia bądź programu pozwoliła im na rozwijanie zainteresowań. Często są to ludzie młodzi, którzy znają język, ale nie znają jego elementów technicznych. Postanowiłem im pomóc tym właśnie tłumaczeniem. Jeśli dziś Ty należysz do tego grona ludzi, może tak właśnie rozwiniesz swe hobby? Może za kilka lat, a może nawet miesięcy będziesz zasługiwać na miano speca? Może hobby stanie sie sposobem na życie? Może... Ale nie gdybajmy. Zapraszam do lektury i praktycznych zabaw z mikrokontrolerami z rodziny 8051 oraz kompilatorem ASEM51!

Pomyślnych wiatrów!

Maciej Sochaczewski
maciej.sochaczewski@interia.pl

SPIS TREŚCI

Przedmowa do wersji 1.0
Przedmowa do wersji 1.2
Przedmowa do wersji 1.3

I. Wstęp

II. Pierwsze kroki
- II.1 Implementacja dla systemu DOS i Windows
  - II.1.1 Pliki
  - II.1.2 Instalacja pod MS-DOS lub Windows
  - II.1.3 Operacje przy użyciu konsoli DOS
  - II.1.4 Ustawienia zaawansowane DOS
  - II.1.5 Uruchamianie ASEM-51 w środowisku IDE firmy Borland
  - II.1.6 Uruchamianie ASEM-51 w środowisku 3.1x
  - II.1.7 Uruchamianie ASEM-51 w edytorze BRIEF
  - II.1.8 ASEMX- asembler dla trybu chronionego DOS
  - II.1.9 ASEMW- asembler dla tryb konsoli w Win32
  - II.1.10 Narzędzie HEXBIN
- II.2 Implementacja dla Linuksa
  - II.2.1 Pliki
  - II.2.2 Instalacja w systemie Linux
  - II.2.3 Operacje przy użyciu konsoli systemu Linux
  - II.2.4 Ustawienia zaawansowane systemu Linux
  - II.2.5 Narzędzie HEXBIN
- II.3Program demonstracyjny

III. Język asemblera w kompilatorze ASEM-51
- III.1 Wyrażenia
- III.2 Symbole
- III.3 Stałe
- III.4 Wyrażenia
- III.5 Zestaw instrukcji procesora 8051
- III.6 Pseudo instrukcje
- III.7 Typy segmentów
- III.8 Kontrolki Asemblera
  - III.8.1 Kontrolki podstawowe
  - III.8.2 Kontrolki ogólne
- III.9 Symbole predefiniowane
- III.10 Kompilacja warunkowa
  - III.10.1 Instrukcja IFxx
  - III.10.2 Instrukcje IFxx i ELSEIFxx
- III.11 Makroprocesor
  - III.11.1 Proste wywoływane makra
  - III.11.2 Parametry makr
  - III.11.3 Makra powtarzalne
  - III.11.4 Symbole lokalne
  - III.11.5 Operatory Makr

III.11.6 Przedwczesny koniec ekspansji makra.
III.11.7 Wywołania zagnieżdżonych i powtarzanych makr.
III.11.8 Definicje makr zagnieżdżonych
III.11.9 Reprezentacja w pliku "List File"

IV. Kompatybilność z asemblerem firmy Intel

IV.1 Obostrzenia
IV.2 Wyrażenia
IV.3 Dalsze różnice

V. Format pliku "List File"

VI. Wspierane mikrokontrolery rodziny 8051

Dodatek A: Komunikaty błędów kompilatora ASEM-51
A.1 Błędy kompilacji
A.2 Błędy wykonania

Dodatek B: Komunikaty błędów narzędzia HEXBIN
B.1 Błędy konwersji
B.2 Błędy wykonania

Dodatek C: Predefiniowane symbole

Dodatek D: Zarezerwowane słowa kluczowe

Dodatek E: Specyfikacja formatu Intel- HEX

Dodatek F: Zestaw kodów ASCII

Dodatek G: Literatura

Dodatek H: Znaki firmowe

Dodatek I: Instrukcje mikrokontrolera 8051 w porządku numerycznym

Dodatek J: Instrukcje mikrokontrolera 8051 w porządku leksykalnym

Wstęp do wersji 1.0

Dziś mikrokontrolery są używane w szerokim zakresie aplikacji, od prostych konsumenckich produktów po zespolone komponenty do awioniki. To sprawiło, że nie zdziwiła mnie obecność 80C31 na płytce teletekstu, którą kupiłem jakiś czas temu. Ponieważ urządzenie miało słaby interfejs i wiele błędów programowych, pomyślałem, że potrafiłbym to zrobić lepiej więc zacząłem się rozglądać za kompilatorem asemblera dla 8051. Ale w kontraście z ogromną ilością sprzedanych elementów sprzętowych, liczba ludzi rozwijających narzędzia programowe dla tego mikrokontrolera wydawała się nieporównywalnie mniejsza, podobnie jak liczba narzędzi w sprzedaży.
Było niewiele dobrych, profesjonalnych kompilatorów asemblera za 250$ i więcej- zbyt kosztownych do użytku hobbystycznego. Poza bezużytecznymi wersjami demo nie było okrojonych wersji startowych lub szkolnych. Znalazłem również kilka wersji shareware i wersji do użytku publicznego, ale nawet one były słabe i niezbyt można było im ufać lub nie posiadały pełnej specyfikacji 8051, lub miały swoją fantastyczną składnię, która była w 100% kompatybilna z sobą ale była daleka od standardów Intela. Nie podobały mi się wcale!
Wyglądało na to, że brak użytecznego i takiego, na które można sobie pozwolić oprogramowania narzędziowego dla mikrokontrolerów. A szkoda, bo ich uniwersalność, prosta architektura i niska cena wpływa na szerokie zainteresowanie tymi mikrokontrolerami, szczególnie w nauce i hobby.
I tak zdecydowałem, że napiszę poręczny kompilator asemblera dla 8051 na komputer PC.

I oto on: ASEM-51 v1.0

Mam nadzieję, że pomoże on odkryć wspaniały świat mikrokontrolerów.

Miłej zabawy!

Diesenhofen, 19 lipca 1994
W. W. Heinz

Przedmowa do wersji 1.2

Ponad rok minął od chwili, gdy w październiku 1994 roku światło dzienne ujrzała wersja ASEM-51 V1.1.Pomimo, że nie spędziłem całego tego czasu nad ASEM51, V1.2 pojawia się z kilkoma dodatkami, kilkoma poprawkami i kilkoma funkcjonalnymi lub wewnętrznymi udoskonaleniami!
Najważniejsze elementy nowej wersji to prawie perfekcyjnie przygotowywany plik listy z zależnościami i kilka nowych opcji drukowania, program ładujący dla płytki ewaluacyjnej MCS-51 i wiele nowych plików *.MCU.
Po szczegóły proszę zajrzeć do notki informacyjnej ASEM-51 V1.2.

Ostatnie dwa lata nauczyły mnie, że freeware nie jest jednoznaczne z darmowym ani dla autora ani dla użytkowników.
ASEM-51 nie mógłby powstać z samych kilku wolnych godzin spędzonych wyłącznie na rozwoju oprogramowania. Musiałem również zamówić system PASCALa do rozwijania oprogramowania, wiele literatury oraz płytkę ewaluacyjną z procesorem 80C535.
Dystrybucja wolnego oprogramowania zdaje się większym problemem niż jego tworzenie. Po pierwsze, trzeba kupić modem. Następnie wiele czasu, pieniędzy, sił, problemów i "ciekawych" dyskusji wymaga, by program został zamieszczony (lub nie) na kilku serwerach ftp.
Publikacja oprogramowania na płytach sharewareowych CD-ROMach, to po pierwsze dowiedzieć się, które z nich się do tego nadają. Do tego najlepiej kupić tuzin lub dwa (oraz napęd CD-ROM), i wysłać program do tych, które wydają się najbardziej popularne.
Zainteresowani użytkownicy muszą w końcu zamówić modemy lub napędy CD-ROM i ponosić te same koszty, by otrzymać "freeware" z tych ogólnie dostępnych źródeł.
W końcu może okazać się tańsze, szybsze i bardziej przekonujące po prostu kupienie profesjonalnego programu (jeśli taki istnieje) w sklepie komputerowym na rogu. Ale to nawet nie połowa zabawy!

ASEM-51 V1.1 był zamieszczony i dystrybuowany na ponad 60 serwerach ftp na całym świecie, pobrany do wielu BBS i publikowany na przynajmniej dwóch płytach CD-ROM.
Ale otrzymałem tylko maile od 9 użytkowników, lokalnego karalucha i międzynarodowego potwora. Te dwa listy prosiły mnie o zgodę na sprzedaż ASEM-51 dla (ich) zysków i spełzły na niczym.
Większość maili od użytkowników zaczynała się od "Skopiowałem Pana kompilator z serwera ftp, którego nie pamiętam. Wygląda dobrze na pierwszy rzut oka! Tak po drodze, czy ma Pan kartę katalogową 80Cxyz?" lub coś w tym rodzaju.
Przez ten czas dostałem tylko jedną informację o błędzie. Był zgłoszony telefonicznie, więc nie mogłem go sprawdzić. Dwa poważne błędy zostały poprawione od wersji 1.1, ale znalazłem je sam w listopadzie 1995 roku.
Z pewnością ASEM-51 nie jest oprogramowaniem z głównego strumienia, ale mówiąc szczerze, jestem trochę zawiedziony słabym odzewem użytkowników!

W końcu powinienem podziękować osobom, które pomogły mi wydać ASEM-51: Andreas Kies, który testował wszystkie poprzednie wersje kompilatora. Pomógł mi wydać i dystrybuować pierwsze wersje i zająć się darmowym kontem dla ASEM-51 od samego początku.
Gabriele Novak - sprawdzenie ortografii w całej dokumentacji.
Werner Allinger- testowanie ostatnie wersje beta programu ładującego. I w końcu, ale nie na końcu, chciałbym podziękować zainteresowanym użytkownikom za ich komentarze i sugestie.

Deisenhofen, 22 stycznia 1997 roku
W. W. Heinz

Przedmowa do wersji 1.3

Sześć lat minęło od kiedy pojawiła się wersja 1.2. Jestem prawie pewien, że użytkownicy myśleli, że to była ostatnia wersja! Wygląda jednak na to, że miałem trochę wolnego czasu, by wesprzeć, przejrzeć, zmienić, rozwinąć, itd. ASE<-51 wciąż i wciąż przez wiele wiele lat, ale nigdy, by publikować oficjalne nowe wersje. Ale teraz po pół roku testowania wersji beta, czas nadszedł, by pokazać ASEM-51 Wersja 1.3 (finalna). Najważniejszymi cechami są:

-nowe platformy programowe: Win32 i Linux (386)
-makra
-dramatycznie poprawiona kompilacja warunkowa
-generacja plików w formacie OMF-51 (z informacjami dla debuggerów)
-37 nowych MCU
-dokumentacja w formacie ASCII i w HTML

Po szczegółowe informacje odsyłam do notki informacyjnej ASEM-51 V1.3. Od jesieni 2001 roku, ASEM-51 ma oficjalną stronę w Internecie: http://plit.de/asem-51/

Podziękowania:
Andreas Kies testował wszystkie wersje beta kompilatora. Bez jego wiedzy na temat Linuksa, wersja dla niego byłaby bardzo trudnym zadaniem.
Werner Allinger testował ostatnie wersje beta i przygotował darmową stronę sieciową dla ASEM-51 od samego początku.
Axel Kielhorn napisał plik DS5000.MCU
Anders Sandstroem przysłał mi 87LPC762.MCU i 87LPC768.MCU
Michael R. Przygotował 80C32X2.MCU
Chcę także podziękować wszystkim pozostałym użytkownikom za ich raporty o błędach, wsparcie, komentarze i sugestie.

Bayreuth, 31grudnia 2002 roku
W. W. Heinz

I. Wstęp

ASEM-51 jest dwuprzejściowym kompilatorem makroasemblera dla mikrokontrolerów Intela rodziny MCS-51. Pracuje na komputerze klasy PC w systemie MS-DOS, Windows i Linux. Kompilator w wersji dla DOS (ASEM.EXE) wymaga tylko 256kB wolnej pamięci i systemu MS-DOS w wersji 3.0 lub wyższej. Nowy kompilator pracujący w trybie chronionym (ASEMX.EXE) wymaga procesora 286 lub lepszego, i przynajmniej 512kB wolnej pamięci XMS. Nowa wersja dla konsoli w trybie Win32 (ASEMW.EXE) wymaga procesora 386 lub lepszego oraz systemu Windows 9x, NT, 2000 lub XP. Nowa wersja dla systemu Linux wymaga systemu bazującego na architekturze 386. Nowa dokumentacja HTML wymaga procesora Pentium 90MHz lub lepszego i przeglądarki internetowej.

ASEM-51 jest wzbogaconym podzbiorem standardu Intela, co gwarantuje maksymalną zgodność z istniejącymi kodami źródłowymi dla 8051. ASEM51 może generować dwa rodzaje plików obiektowych: w formacie Intel-HEX, który jest bezpośrednio akceptowany przez większość programatorów EPROM oraz OMF-51,którego wymaga większość symulatorów, emulatorów oraz debugerów (tzw. programów odpluskwiających). ASEM-51 jest możliwy do wykorzystania dla projektów hobbystycznych, edukacyjnych oraz komercyjnych bazujących na małych i średnich mikrokontrolerach. Jednakże ASEM-51 został również zaprojektowany do kompilacji bardzo dużych programów! Jego najważniejsze cechy to:

- mały, kompaktowy, łatwy w użyciu, dobrze udokumentowany i godny zaufania
- łatwy w instalacji, prawie nie wymaga konfiguracji
- możliwość pracy w trybie tekstowym z linii poleceń, w trybie wsadowym i możliwość pracy w sieci
- w pełni kompatybilny z rokiem 2000
- pliki wykonywalne dla systemu DOS (tryb realny i tryb chroniony), Win32 oraz dla Linuksa
- składnia kompatybilna ze standardem Intela
- pięć liczników lokacji, po jednym dla każdej z przestrzeni adresowych rodziny MCS-51
- w plikach źródłowych można używać wyrażeń logicznych i arytmetycznych oraz czasu
- sprawdzanie typu segmentu dla instrukcjach
- automatyczna optymalizacja kodu dla skoków i wywołań
- makra (to naprawdę działa)
- praca z zagnieżdżonymi plikami dołączanymi
- zagnieżdżona kompilacja warunkowa
- generacja formatu OMF-51 (z informacjami dla debugerów)
- plik wynikowy w formacie Intel-HEX
- narzędzie do konwersji plików z hex na binarne
- wbudowane symbole rejestrów SFR dla 8051 (mogą być blokowane)
- bezpośrednie wsparcie ponad siedemdziesięciu procesorów rodziny 8051
- specjalne wsparcie procesorów 83C75x firmy Philips
- wsparcie dla banku rejestrów 8051
- szczegółowy plik wynikowy z tablicą symboli lub referencji
- dalsze dodatkowe opcje drukowania
- dokumentacja w formacie ASCII i HTML
- program wsadowy do testowania w płycie testowej z MCS-51
- wsparcie dla łatwego użycia w popularnym pakiecie IDE firmy Borland
- limitowany przez autora serwis aktualizacyjny

ASEM-51 został stworzony z użyciem:
Borland- Pascalmit Objekten 7.0 (c) Borland International 1992
Delphi 2.0 Client/Server Suite (c) Borland International 1996
FreePascal 1.0.6 (c) Florian Klaempfl 2002

II Pierwsze kroki

Ten rozdział opisuje dystrybucję ASEM-51, jej instalację na wspieranych platformach oraz sposób użycia w dniu codziennym.

II. Implementacja dla DOS i Windows

Wersja 1.2 występowała tylko w implementacji dla trybu realnego MS-DOS. Od tamtej pory dodano obsługę pracy w trybie chronionym DOS oraz w Win32.
W odróżnieniu od nowej implementacji dla systemu Linuks, wszystkie wersje DOS i Windows są identyczne funkcjonalnie i ich podstawowe operacje mogą być opisane wspólnie. Tylko kilka specjalnych cech i funkcji musi zostać opisanych osobno.
Od chwili, gdy powinno być możliwe współdzielenie kodu z wersją dla Linuksa, wszystkie wersje DOS i Windows potrafią czytać pliki ASCII formowane dla DOSa i UNIXa, ale zapisują pliki tylko w natywnym formacie (DOS).

II.1.1Pliki

Twoja dystrybucja ASEM-51 dla DOS/ Windows powinna zawierać następujące grupy plików:

1.)	ASEM_51.DOC	ASEM-51 instrukcja użytkownika w formacie ASCII
	DOCS.HTM	plik indeksowy dokumentacji w formacie HTML
	*.HTM	następne strony dokumentacji HTML
	*.GIF	pliki graficzne GIF związane z dokumentacją HTML
	*.JPG	pliki graficzne JPEG związane z dokumentacją HTML
	ASEM.EXE	kompilator asemblera (dla trybu realnego systemu DOS)
	ASEM.PIF	plik informacji programu dla systemu Windows 3.1x
	ASEM.ICO	plik ikony dla Windows 3.1x programu ASEM
	ASEM2MSG.EXE	program filtru wiadomości dla pakietu IDE firmy Borland (DOS)
	ASEM2MSG.PAS	plik źródłowy w Turbo Pascalu programu ASEM2MSG.EXE
	ASEMX.EXE	kompilator asemblera (dla trybu chronionego systemu DOS)
	ASEMX.PIF	plik informacji programu dla systemu Windows 3.1x
	ASEMX.ICO	plik ikony dla Windows 3.1x programu ASEMX
	DPMI16BI.OVL	16- bitowy serwer DPMI firmy Borland dla ASEMX.EXE
	RTM.EXE	16- bitowy menager wykonania DPMI firmy Borland
	ASEM32.BAT	uruchamia ASEMX z 32- bitowym serwerem DPMI firmy Borland
	ASEMW.EXE	kompilator asemblera dla trybu konsoli Win32
	HEXBIN.EXE	program konwersji plików hex do formatu binarnego (DOS)
	HEXBINW.EXE	program konwersji plików hex do formatu binarnego (Win32)
	DEMO.A51	przykładowy program źródłowy asemblera 8051
	*.MCU	pliki definicyjne procesorów 8051 (by zobaczyć szczegółową listę mikrokontrolerów, zobacz rozdział "VI. Lista wspieranych układów 8051")
2.)	BOOT51.DOC	dokumentacja użytkownika programu BOOT-51 w formacie ASCII
	BOOT51.HTM	plik indeksowy dokumentacji HTML programu BOOT-51
	BOOT51.A51	plik źródłowy programu BOOT-51 (wymaga kompilatora ASEM-51 V1.3)
	CUSTOMIZ.EXE	plik parametryzacji programu BOOT-51
	BOOT.BAT	plik wsadowy programu ładowania aplikacji
	UPLOAD.BAT	plik wywoływany tylko przez BOOT.BAT
	COMPORT.EXE	program nastawiania portu szeregowego komputera PC

	RESET51.EXE	program do resetowania systemu docelowego przez port szeregowy
	SLEEP.EXE	program oczekujący na czas powrotu z trybu reset
	BLINK.A51	prosty program testowy dla BOOT-51
3.)	README.1ST	szybka informacja w formacie ASCII
	LICENSE.DOC	licencja do ASEM-51w formacie ASCII
	RELEASE.130	notka w formacie ASCII o ASEM-51
	SUPPORT.DOC	przewodnika wsparcia w formacie ASCII
	INSTALL.BAT	wykonuje instalację w systemie DOS
	KILLASEM.BAT	usuwa wszystkie pliki pakietu ASEM-51 (DOS)

Pierwsza grupa zawiera wszystkie pliki bezpośrednio związane z kompilatorem.
Druga grupa zawiera wszystkie pliki bezpośrednio powiązane z programem ładującym.
Trzecia grupa zawiera pliki dokumentacji i wsparcia dla całej instalacji.

II.1.2 Instalacja w MS-DOS lub Windows

ASEM-51 nie wymaga zakręconych programów instalacyjnych ani konfiguracyjnych. W najprostszym przypadku możesz skopiować archiwum do katalogu roboczego i cieszyć się dobrodziejstwem prawdziwej kompatybilności plug-and-play!
W innym razie instalacja ASEM-51 w systemie MS-DOS jest bardzo prosta:
-Utwórz nowy pusty katalog początkowy na dysku twardym.
-Rozpakuj archiwum z dystrybucją ASEM-51 do tego katalogu lub skopiuj wszystkie pliki archiwum do niego.
-Ustaw katalog jako domyślny, uruchom dołączony plik wsadowy INSTALL.BAT i postępuj zgodnie z instrukcjami.

Jeśli nie lubisz, gdy coś pracuje automatycznie, lub coś nie jest dość jasne, ASEM-51 może zostać zainstalowany ręcznie według poniższego opisu:

-Stwórz nowy katalog na dysku twardym, np. C:\ASEM51.
-Skopiuj wszystkie pliki archiwum ASEM-51 do utworzonego przed chwilą katalogu.
-Dodaj go do swojej zmiennej środowiskowej PATH w pliku AUTOEXEC.BAT, np.

```
PATH C:\DOS; C:\UTIL; C:\ASEM51
```

-Jeśli to nie zostało dokonane podczas rozpakowywania plików, utwórz podkatalog C:\ASEM51\MCU i skopiuj do niego wszystkie pliki z rozszerzeniem *.MCU.
-Utwórz kolejny podkatalog, np. C:\ASEM51\HTML i skopiuj do niego wszystkie pliki z rozszerzeniem *.HTM, *.GIF i *.JPG. (By odczytać tą instrukcję w formacie HTML, wywołaj swoją przeglądarkę internetową z plikiem C:\ASEM51\HTML\DOCS.HTM!)
-Opcjonalnie zdefiniuj zmienną środowiskową ASEM51INC w AUTOEXEC.BAT, by wyspecyfikować ścieżkę przeszukiwania plików dołączanych, np.

```
SET ASEM51INC=C:\ASEM51\MCU;D:\MICROS\MCS51\INCL
```

-Dla prawidłowej współpracy 16 bitowego serwera DPMI na komputerach z pamięcią RAM przekraczającą 16MB należy się upewnić, że EMM386.EXE (dołączony do DOS 5.0 i późniejszych) jest załadowany i należy zdefiniować DPIMEM w pliku AUTOEXEC.BAT jk poniżej:

```
SET DPIMEM=MAXMEM 16383
```

-A teraz należy zrestartować komputer.

Wsadowy plik instalacyjny INSTALL.BAT powinien pracować poprawnie w środowisku MS-DOS, Windows 3.1x i Windows 9x. Nie został on jeszcze przetestowany na innych wersjach systemu Windows. Szczególnie dla systemów bazujących na silniku NT, czyli Windows NT 4.0, 2000 i XP, wskazana jest instalacja ręczna.

II.1.3 Operacje przy użyciu konsoli DOS

ASEM-51 podobnie, jak najlepsze narzędzia komercyjne, wspiera w pełni pracę w linii komend lub wsadową. Mimo tego może on być integrowany z innymi narzędziami deweloperskimi, jeśli jest to wymagane. Kompilator jest wywoływany przez wpisanie:

ASEM <źródło> [<plik obiektowy> [<plik listingu>]] [<opcje>]

Gdzie:
<źródło> jest plikiem źródłowym napisanym w asemblerze dla mikrokontrolerów 8051,
<plik obiektowy> jest plikiem wynikowym
<plik listingu> jest plikiem z dokładną listą programu.
Parametry <plik obiektowy> i <plik listingu> są opcjonalne. Kiedy je pominiemy, nazwy plików będą takie same, jak nazwa pliku źródłowego, ale z rozszerzeniem HEX (lub OMF) i LST. Wszystkie nazwy plików w parametrach mogą być wpisane bez rozszerzeń. W takim przypadku kompilator doda domyślne rozszerzenia zgodnie z poniższą tabelą:

Plik	rozszerzenie
<źródło>	.A51
<plik obiektowy>	.HEX (z opcją /OMF-51: .OMF)
<plik listingu>	.LST

Jeśli nie chcesz, by plik miał rozszerzenie, zakończ je używając '.'! Zamiast nazw plików możesz wyspecyfikować nazwy urządzeń, by przekierować wyjście na portyI/O. Nazwy urządzeń mogą być zakończone przy użyciu ':'!
Nie jest sprawdzane, czy wyspecyfikowane urządzenie rzeczywiście istnieje w systemie.
Istnieje możliwość wczytania pliku z wejścia (np. z CON:) zamiast z pliku, ale nie jest to wskazane. Od czasu, gdy ASEm-51 jest kompilatorem dwuprzebiegowym, zawsze czyta plik dwukrotnie

ASEM rozpoznaje następujące opcje:

/INCLUDES: ścieżka1 [;ścieżka2[;...; ścieżka n]]
/DEFINE: symbol [:wartość{:typ]]
/OMF-51
/COLUMNS
/QUIET

Gdy opcja /INCLUDE jest użyta, kompilator przeszukuje wyspecyfikowane ścieżki by odczytać dołączane pliki, których nie może znaleźć w katalogu roboczym. Jako ścieżka może wystąpić dowolna ilość katalogów oddzielona przez znak ';'. Katalogi będą przeszukiwane w kolejności od lewej do prawej.
Ścieżka wpisana wraz z parametrem /INCLUDE jest przeszukiwana przed ścieżką zapisaną w opcjonalnej zmiennej środowiskowej ASEM51INC!

Opcja /DEFINE jest przydatna do wyboru poszczególnych wersji programu przy użyciu linii poleceń, które zostały zaimplementowane z kompilacją warunkową. Pozwala to definiować symbol z wartością i typem segmentu w linii poleceń. Wartość i typ są opcjonalne. Jeśli typ segmentu jest pominięty, jest ustawiony domyślnie na NUMBER (czyli liczba). Wartość symbolu, jeśli została pominięta, przyjmuje wartość 0. Symbol wartości może być dowolną stałą liczbową. Typ symbolu musi być jedną z następujących liter:

C = CODE
D = DATA
I = IDATA

X = XDATA
B = BIT
N = NUMBER (wartość domyślna)

Domyślnie ASEM-51 generuje pliki wynikowe (obiektowe) w formacie Intel-HEX. Gdy użyta zostanie opcja /OMF-51, dodatkowo generowany jest plik OMF.

Opcje mogą być skracane tak długo, jak długo są unikalne.

Przykłady:

0.) ASEM

Gdy jest wywołany bez parametrów, kompilator wyświetla ekran pomocy:

MCS-51 Family Macro Assembler ASEM-51 V1.3

usage: ASEM <source> [<object> [listing>]] [options]

options: /INCLUDES: path1;path2;path3
/DEFINE: symbol [:value[:type]]
/OMF-51
/COLUMNS
/QUIET

1.) ASEM PROGRAM

Dokona kompilacji kodu źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051o nazwie PROGRAM.A51 i wygeneruje plik wyjściowy w formacie Intel-HEX PROGRAM.HEX oraz plik listingu PROGRAM.LST

2.) ASEM TARZAN.ASM JANE JUNGLE.PRN

Powyższe wywołanie dokona kompilacji pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera 8051 o nazwie TARZAN.ASM i wygeneruje plik wyjściowy w formacie Intel-HEX JANE.HEX oraz plik listingu JUNGLE.PRN.

3.) ASEM PROJECT EPROM.

Dokona kompilacji pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera 8051 o nazwie PROJECT.A51 i wygeneruje plik wyjściowy w formacie Intel-HEX EPROM oraz plik listingu PROJECT.LST.

4.) ASEM ROVER /OMF

Dokona kompilacji pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera 8051 o nazwie ROVER.A51 i wygeneruje plik wynikowy w formacie OMF-51 ROVER.OMF i pliku listingu ROVER.LST.

5.) ASEM sample COM2: NUL

Dokona kompilacji pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera 8051 o nazwie SAMPLE.A51, prześle plik wynikowy w formacie Intel-HEX przez interfejs szeregowy COM2 oraz nie wygeneruje pliku listingu.

6.) ASEM APPLICAT /INC:C:\ASEM51\MCU;D:\MICROS\8051\HEADERS

Dokona kompilacji pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera 8051 o nazwie APPLICAT.A51 przeszukując w pierwszej kolejności katalog domyślny w poszukiwaniu plików dodatkowych, a następnie przeszukując C:\ASEM51\MCUi w końcu D:\MICROS\8051\HEADERS.

7.) ASEM UNIVERSL /D:Eva_Board:8000H:C

Dokona kompilacji pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera 8051 o nazwie UNIVERSL.A51 zastępując nazwę Eva_Board w segmencie CODE wartością 8000h.

W przypadku wystąpienie błędów w trakcie kompilacji, są one wyświetlane w konsoli. Może to wyglądać następująco:

MCS-51 Family Macro Assembler ASEM-51 V1.3

APPLICAT.A51(14): must be known on first pass
USERBITS.INC(6): attempt to divide by zero
DEFINES.INC(37): symbol not defined
APPLICAT.A51(20): symbol not defined
APPLICAT.A51(27): no END statement found

5 errors detected

Każdy błąd jest wyświetlony z nazwą pliku źródłowego lud dołączonego, numerem linii, w której znaleziony został błąd oraz komunikatem samego błędu.
Taki format komunikatu ułatwia integrację ASEM-51z innymi istniejącymi narzędziami programistycznymi. Idealne zgranie z Pakietem IDE dla Turbo C++ (i prawdopodobnie innymi) można osiągnąć przez użycie opcji /COLUMNS. Gdy się jej użyje, numery kolumn błędów programu są wyświetlane dodatkowo za numerami linii:

MCS-51 Family Macro Assembler ASEM-51 V1.3

APPLICAT.A51(14,12): must be known on first pass
USERBITS.INC(6,27): attempt to divide by zero
DEFINES.INC(37,18): symbol not defined
APPLICAT.A51(20,18): symbol not defined
APPLICAT.A51(27,1): no END statement found

5 errors detected

Jeśli błędy zostały wykryte w linii makra, nie ma powiązanej lokalizacji w pliku źródłowym. Zamiast tego, błąd jest wyświetlany z nazwą pliku źródłowego lub dołączanego oraz numer linii, z której makro zostało wywołane. (Dla wywoływanych makr jest to linia z wywołaniem i dla bloków zapętlonych jest to linia ENDM.)
By podpowiedzieć użytkownikowi, nazwa makra i numer linii (oraz opcjonalnie kolumny) jest wstawiony przed aktualnym komunikatem o błędzie:

```
        MCS-51 Family Macro Assembler ASEM-51 V1.3

            UARTIO.A51(44,1): RECEIVE(3,22): segment type mismatch
            UARTIO.A51(87,1): REPT(4,19): symbol not defined
            UARTIO.A51(87,1): REPT(8,19): symbol not defined
    UARTIO.A51(87,1): REPT(12,19): symbol not defined

        4 errors detected
```

Numer linii rozszerzenia jest numerem linii zawierającej makro rozszerzające i zaczyna się od 1. Jeśli błąd pojawi się w rozszerzeniu bloku z pętlą, słowo REPT zastąpi nazwę makra.

Opcja /Quiet zawiesza wszystkie komunikaty wyjścia konsoli poza komunikatami błędów.

Gdy ASEM-51 przerywa pracę, zwraca do systemu operacyjnego kod wyjścia:

Sytuacja	kod błędu
brak błędów	0
wykryte błędy programu	1
krytyczne błędy wykonania	2

Uwaga: Ostrzeżenia nie wpływają na kod wyjścia programu!

II.1.4 Ustawienia zaawansowane DOS

By ustawić ścieżkę przeszukiwania dla plików dołączanych, należ ustawić opcjonalną zmienną środowiskową ASEM51INC:

```
SET ASEM51INC=<ścieżka>
```

<ścieżka> może zawierać dowolną ilość katalogów rozdzielonych przez znak ‘;’ . Należy się upewnić, że całe wyrażenie nie zawiera spacji ani znaków tabulacji!
Jeśli ASEM51INC zostanie zdefiniowane, kompilator przeszukuje wyspecyfikowaną <ścieżkę> w poszukiwaniu plików dołączanych do programu, których nie można znaleźć w katalogu roboczym ani w ścieżce wyspecyfikowanej w opcji /INCLUDES.
<ścieżka> będzie przeszukiwana od lewej do prawej.

Przykłady:

1.) SETASEM51INC=C:\ASEM51\MCU;D:\MICROS\MCS51\INCL

Jeśli plik dołączony nie może być znaleziony w katalogu roboczym ani w ścieżce z /INCLUDES (o ile użyto tej opcji), kompilator przeszukuje C:ASEM51\MCU i w końcu D:\MICROS\MCS51\INCL.

2.) SET ASEM51INC=C:\ASEM51\MCU;%PATH%

Jeśli zmienna ASEM51INC jest w powyższy sposób zdefiniowana w pliku AUTOEXEC.BAT po zmiennej PATH, kompilator przeszukuje katalog C:\ASEM51\MCU i wszystkie katalogi zawarte w ścieżce przeszukiwania DOS od lewej do prawej!

Maksymalna długość <ścieżki> jest ograniczona do 255 znaków. Nie można tego ograniczenia wyeliminować przy użyciu komendy SET interpretera COMMAND.COM, ale z użyciem innych interpreterów, jak chociażby 4DOS długość ścieżki może osiągnąć 512 znaków.

Należy zwrócić uwagę na to, że używanie spacji i tabulatorów za nazwami zmiennych w MS-DOS należy dokładnie przemyśleć! Jeśli jedna zmienna zostanie zdefiniowana, jako

SET ASEM51INC =C:\ASEM51\MCU
i SET ASEM51INC=C:\8051\MCU

oznaczać to będzie, że w systemie zostaną zdefiniowane dwie zmienne! Jednakże kompilator rozpozna tylko drugą zmienną. Od czasu, gdy DOS nie odcina spacji i znaków tabulacji z nazw zmiennych, kompilator asemblera nie może tego robić również! Dlatego należy się upewnić, że podczas definiowana zmiennych nie użyto spacji i znaków tabulacji.

II.1.5 Uruchamianie ASEM-51 w środowisku IDE firmy Borland

Użytkownicy pakietu IDE firmy Borland dla Turbo C++ (od wersji 1.0 do 3.0) mogą użyć wspomnianego środowiska IDE do pracy z ASEM-51. (tylko dla wersji DOS!) By to było możliwe, do pakietu ASEm51 dołączono program filtra ASEM2MSG dla wyświetlania informacji o błędach. By zintegrować ASEM-51 z pakietem IDE firmy Borland, należy postępować zgodnie z poniższymi krokami:

-Należy się upewnić, że ASEM-51 został zainstalowany zgodnie z wcześniejszymi wskazówkami lub że ASEM.EXE i ASEM2MSG.EXE są gdzieś w zmiennej środowiskowej PATH.

-Należy uruchomić środowisko IDE firmy Borand kompilatora Turbo C++ dla systemu DOS.

-Dla Turbo C++ 1.0, najpierw kliknij: Options | Full menus | ON

-Kliknąć w basku menu: Options | Transfer

-Gdy okno „Transfer” jest aktywne, należy wcisnąć klawisz Edit.

-Teraz okno „Modify/New Transfer Item” powinno być aktywne. Należy wypełnić następujące pozycje:

Program Title:	ASEM-~51
Program Path:	ASEM
Command Line:	$NOSWAP $SAVECUR $CAP MSG(ASEM2MSG) $EDNAME /C
Translator:	[X]
Hot key:	Shift F8

Następnie należy wcisnąć klawisz New.

-Gdy nastąpi powrót do okna „Transfer”, należy wcisnąć klawisz OK.
-Należy kliknąć na pasku menu: Options | Save | OK.

Teraz powinna być możliwa kompilacja za pomocą ASEM-51 pliku aktywnego okna edytora przy użyciu skrótu klawiaturowego Shift+F8. Komunikaty błędów (jeśli jakieś wystąpią) powinny pojawić się w oknie komunikatów. Można przeglądać komunikaty błędów i przeskakiwać do tekstu źródłowego wciskając <Enter>. Działa to również w przypadku, gdy błąd znajduje się nie w pliku głównym, a w pliku dodanym!

Użytkownicy pakietu Turbo-Pascal 7.0 również mogą zaprząc pakiet IDE do kompilacji. By zintegrować ASEM-51 z Turbo-Pascalem, należy wykonać następujące kroki:

-Należy się upewnić, że ASEM-51 został prawidłowo zainstalowany, zgodnie z poprzednimi informacjami lub, że ASEM.EXE i ASEM2MSG.EXE są dzieś w zmiennej środowiskowej PATH.

-Należy uruchomić środowisko IDE dla Turbo-Pascala 7.0 (lub Borland-Pascala 7.0).
-W pasku menu kliknąć: Options | Tools
-Gdy okno „Tools" jest aktywne, należy wcisnąć klawisz New.
-Powinno być aktywne okno „Modify/New Tool".
Należy wypełnić je w następujący sposób:

Title:	ASEM-~51
Program path:	ASEM
Command line:	$NOSWAP $SAVE CUR $CAP MSG (ASEM2MSG) $EDNAME
Hit keys:	Shift+F8

A następnie należy wcisnąć klawisz OK.

-Po powrocie do okna „Tools", wcisnąć klawisz OK.
-Na pasku menu kliknąć: Options | Enviroment | Preferences
-Gdy okno „Preferences" jest aktywne, należy zablokować pozycję „Close on go to source" . Następnie należy wcisnąć OK.
-Ostatnim ruchem jest kliknięcie w pasku menu: Options | Save

Teraz ASEM-51 może być wywołany przez skrót klawiszowy Shift+F8 i w ten sposób można dokonać kompilacji kodu źródłowego w aktywnym oknie edytora. Informacje o błędach (jeśli jakieś wystąpią) pojawią się w oknie informacyjnym.

Użytkownicy zarówno Turbo C++ jak i Turbo-Pascala powinni wybrać IDE z Turbo C++. W Turbo-Pascalu 7.0 IDE nie obsługuje opcji /COLUMNS (lub /C). Wersje Turbo-Pascala przed wersją 7.0 nie posiadają menu Tools.

Należy pamiętać, że macro transferu $SAVE CUR zapisuje zawartość okna w edytorze do pliku edytowanego (o ile jego zawartość została zmieniana),zanim ASEM.EXE zostanie wywołane! W przypadku, gdy Twój program źródłowy zawiera dalsze pliki źródłowe (które mogą aktualnie być załadowane do innych okien edycyjnych), lepiej wpisać $SAVE ALL. W ten sposób zostaną zapisane do plików wszystkie (modyfikowane) edytowane okna przed wywołaniem ASEM.EXE! Jeśli nie jesteś pewny, wpisz $SAVE PROMPT. Ta opcja poprosi Cię o zapisanie każdego modyfikowanego okna do pliku przed wywołaniem ASEM.EXE. Dalszych informacji na temat makr transferu można znaleźć w pomocy online firmyBorland!

Ważne: ASEM2MSG nie jest kompatybilne z wersjami środowisk IDE dla Win32!

II.1.6 Uruchamianie ASEM-51 w środowisku 3.1x

Oczywiście ASEM i ASEMX działają prawidłowo w oknie dosowym środowiska Windows 3.1x! Ale dla integracji z pulpitem Windows 3.1x dostarczone zostały pliki ASEM.PIF i ASEM.ICO. By dołączyć ASEM-51 do grupy Menadżera Programów należy przeprowadzić następujące kroki:

-Należy się upewnić, że ASEM-51 został prawidłowo zainstalowany dla MS-DOS zgodnie z poprzednimi wskazówkami.
-Uruchom Windows 3.1x i zmaksymalizuj okno Menadżera Plików, jeśli to jest konieczne.
-Podświetl grupę programów, w której ma być umieszczony ASEM-51,np. „Aplikacje".
-Wybierz z menu Menadżera Programów: File | New (Plik | Nowy)
-Kiedy okno „New Program Object" będzie aktywne, wybierz opcję „Program Item" i kliknij OK.
-Teraz powinno być aktywne okno „Program Item Properties". Wypełnij je w następujący sposób:

Description:	ASEM-51	(Opis)
Command Line:	ASEM.PIF	(Komenda)
Working Directory:	(jakikolwiek chcesz)	(katalog roboczy)
Shortcut Key:	(jakikolwiek chcesz)	(klawisz skrótu)
Run Minimized:	[ ]	(uruchom zminimalizowany)

A następnie wciśnij klawisz [Change Icon]. (zmień ikonę)

-Teraz pojawi się komunikat błędu informujący, że nie ma ikon możliwych dla tego typu plików. Wystarczy wcisnąć OK.
-Teraz okno „Change Icon" (zmiana ikony) powinno być wyświetlone. Wypełnij je

File Name: ASEM.ICO

I wciśnij OK. Teraz ikona ASEM-51 powinna zostać wyświetlona w polu ikony. Należy wcisnąć ponownie OK.

-Po powrocie do okna „Program Item Properties" (właściwości programu) wciśnij OK..

(W wersjach narodowych Windows nazwy mogą być nieco inne.)

Teraz ASEM.EXE może być wywołany przez dwukrotne kliknięcie ikony ASEM-51. Po wpisaniu parametrów programu w oknie, które się pojawi, ASEM uruchamia sięw oknie DOS, które pozostanie otwarte po zakończeniu programu, by pozwolić Tobie obejrzeć informacje o błędach.

W zasadzie, instalacja kompilatora pracującego w trybie chronionym ASEMX.EXE może być przeprowadzona w sposób przed chwilą opisany. Jednakże, pole <Description> (opis) powinno być wypełnione jako „ASEM-51 XMS", <Command Line> (komenda) powinna zawierać „ASEMX.PIF", a ikona <File Name> (Nazwa Pliku) powinna zawierać „ASEMX.ICO".

II.1.7 Uruchamianie ASEM-51 w edytorze BRIEF

Użytkownicy BRIEF 3.x mogą zintegrować ASEM-51 ze swoim edytorem poprzez proste zdefiniowanie zmiennej środowiska w swoim pliku AUTOEXEC.BAT

SET BCA51=”ASEM %%s”

W ten sposób wskazuje się komendę dla kompilacji plików z rozszerzeniem *.A51. Po tym ASEM-51 może być wywołane przy użyciu klawiszy Alr-F10.

II.1.8 ASEMX- asembler dla trybu chronionego DOS

Generalnie tryb real-mode kompilatora ASEM.EXE jest wystarczający także dla bardzo dużych programów. Pomimo tego może okazać się, że mamy do dyspozycji zbyt mało pamięci, w przypadku, gdy program zawiera ogromne ilości długich, zdefiniowanych przez użytkownika symboli, lub wiele, lub duże makrodefinicje.
By wypełnić tą lukę, ASEM-51 zawiera nowy kompilator pracujący w tryie chronionym ASEMX.EXE.
ASEMX jest funkcjonalnie identyczny z ASEM, ale może używać rozszerzonej pamięci, by sprostać dużym wymaganiom odnośnie pamięci operacyjnej.
ASEMX współpracuje z 16-bitowym serwerem DPMI DPMI16BI.OVL oraz menadżerem uruchomienia RTM.EXE firmy Borland. Wymaga procesora 286 (lub lepszego) oraz przynajmniej 512 kB wolnej pamięci XMS (zalecane 1MB)!
Gdy wywołany jest ASEMX, DPMI16BI.OVL i RTM.EXE muszą także:

-znajdować się w domyślnym katalogu
-być tam, gdzie jest ASEMX.EXE lub
-muszą być w katalogu, który znajduje się w zmiennej systemowej PATH

Podczas uruchomienia serwer DPMI próbuje zaalokować potrzebną ilość wolnej pamięci XMS do użytku przez ASEMX. Jeśli tego nie chcesz, możesz ograniczyć ilość zaalokowanej pamięci używając zmiennej systemowej DPIMEM:

SET DPIMEM=MAXMEM n

To ograniczy ilość dostępnej pamięci XMS używanej przez DPMI do n kB. Nie należy nigdy ustawiać wartości n większej niż 16383!!!

Generalnie interfejs DPMI jest godny zaufania i normalnie nie wykazuje konfliktów z innymi menadżerami pamięci. ASEMX będzie również pracował z innymi wersjami DPMI16BI.OVL i RTM.EXE dostarczonymi przez firmę Borland (poza TC++ 3.0 i BC++ 3.1).

Występują jednakże problemy w systemach z ponad 16MB pamięci RAM! Bez specyficznej instalacji występuje spora tendencja do awarii programu, zawieszenia systemu lub nawet restartu w przypadku uruchamiania programu DPMI o cechach podobnych do ASEMX.
Dla prawidłowego działania interfejsu DPMI, wymagany jest system MS-DOS 5.0 (lub nowszy) oraz uruchomiony sterownik EMM386.EXE!
Jeśli EMM386.EXE został uruchomiony z parametrami (np. NOEMS), 16-bitowy serwer DPMI Borlanda nie może pracować z pamięcią większą niż 16MB! Jednakże bez parametrów (i=nnnn, x=nnnn są OK.) lub z innym serwerem DPMI można pracować z większą ilością pamięci. W tym przypadku ASEMX może używać do 64MB pamięci rozszerzonej.

W przypadku, gdy ASEMX pracuje w środowisku systemowym z jego własnym serwerem DPMI, np. w oknie trybu DOS, RTM.EXE wykryje ten fakt i użyje aktywnego serwera DPMI zamiast DPMI16BI.OVL. W tej sytuacji zmienna środowiskowa DPIMEM nie ma znaczenia.

By ograniczyć (lub zwiększyć) ilość dostępnej w Windows 3.1x dla DOSowego okna pamięci XMS, należy zmienić plik DOSPRMPT.PIF w katalogu Windows przy pomocy Windowsowego edytora plików PIF.
Po dalsze informacje na temat tego, jak zmienić ilość dostępnej pamięci XMS należy przeczytać mówiącą o tym instrukcję systemu.

Inną interesującą alternatywą dla tego rozwiązania jest 32-bitowy serwer firmy Borland z menadżerem wirtualnej pamięci. Nie może on być dołączony do pakietu ASEM-51 z powodu jego licencji, ale zawarty jest w pakietach Turbo- Assembler 4.0 i 5.0, Borland C++ 4.5 i 5.0x, i może innych firmy Borland. Został on oryginalnie zaprojektowany do narzędzi linii komend firmy Borland, ale działa również z ASEMX. Wymaga procesora typu 386 (lub lepszego) i pozwala zwiększyć wolną przestrzeń pamięci przez jej wymianę z plikiem wymiany, który może zostać utworzony przy pomocy programu MAKESWAP.EXE. Pomimo tego 32-bitowy serwer DPMI DPMI32VM.OVL oraz menadżer 32RTM.EXE są wymagane.
Plik wsadowy ASEM32.BAT dostarczony z archiwum ASEM-51pokazuje, jak uruchomić ASEMX z 64 MB pamięci wirtualnej przy użyciu 32-bitowego serwera DPMI firmy Borand.

II.1.9 ASEMW- asembler dla tryb konsoli w Win32

Kompilatory dla systemu DOS, ASEM oraz ASEMX pracują również w systemach Windows 9x/NT/2000/XP, ale z pewnymi typowymi dla systemu DOS ograniczeniami:
-nazwy plików są ograniczone do formatu 8.3
-łańcuchy określające ścieżki są ograniczone do 64 znaków
-kompilator trybu rzeczywistego (real-mode) ma dostęp do 640kB pamięci RAM
-wsparcie dla systemu DOS jest coraz gorsze w każdej nowej wersji systemu Windows.

By ominąć te niedomagania, archiwum ASEM-51 zawiera nowy kompilator ASEMW.EXE dla trybu konsoli Win32. ASEMW jest funkcjonalnie identyczny z ASEM, ale może sobie radzić z długimi nazwami plików oraz z 32-bitowym zarządzaniem pamięcią, co pozwala na kompilację astronomicznie wielkich programów!

Podpowiedź: Jeśli kochasz długie nazwy plików ze spacjami, musisz zamknąć je w cudzysłów, np.:

ASEMW „Test-Program for my 80C32 Evaluation-board.a51”

II.1.10 Narzędzie HEXBIN

Większość programatorów pamięci EPROM akceptuje format Intel-HEX, który jest plikiem wyjściowym kompilatora ASEM-51. Jednakże dla niektórych programatorów EPROM oraz programatorów specjalnego przeznaczenia może okazać się pożyteczna konwersja pliku HEX na czysty plik binarny. Do takiej konwersji służy dostarczony z pakietem program HEXBIN. Jest on wywoływany w następujący sposób:

HEXBIN <hex> [<bin>] [/OFFSET:o] [/LENGHT:l] [/FILL:f] [/QUIET]

Gdzie <hex> jest plikiem wejściowym w formacie Intel-HEX a <bin> jest plikiem binarnym po konwersji. Parametr <bin> jest opcjonalny. Gdy zostanie pominięty, nazwa pliku wyjściowego to nazwa pliku wejściowego ale z rozszerzeniem BIN. Wszystkie nazwy plików mogą być wpisywane bez rozszerzeń. W takim przypadku program dodaje rozszerzenia domyślne, jak pokazano poniżej:

Plik	rozszerzenie
<hex>	.HEX
<bin>	.BIN

Jeśli plik wyjściowy nie ma mieć rozszerzenia, można je zablokować wpisując wraz z nazwą '.'!
Zamiast nazwy pliku można również wpisać nazwę urządzenia, do którego ma być przekierowany wynik konwersji. Nazwa urządzenia może być zakończona przy użyciu ':'! Program nie sprawdza, czy urządzenie, na które wysyłane są dane rzeczywiście istnieje w systemie.

Plik wyjściowy może być kontrolowany przy pomocy opcji /OFFSET, /FILL i /LENGHT.
Normalnie, pierwszy bajt w pliku binarnym jest pierwszym bajtem rekordu z najniższym adresem pliku HEX. Jeśli występuje potrzeba poprzedzenia danych bajtami pustymi, np. w celu odpowiedniego pozycjonowania pliku w pamięci EPROM, można użyć opcji /OFFSET:

/OFFSET: 1000

Powyższa opcja poprzedzi dane z pliku HEX 4096 bajtami pustymi. Offset zawsze musi być liczbą szesnastkową. Domyślny offset to 0.
Jeśli wstępuje możliwość występowania dziur między rekordami HEX, wartość bajtów wypełniających te dziury może być zdefiniowana przy pomocy opcji /FILL:

/FILL: 0

Taki zapis spowoduje, że wszystkie dziury zostaną wypełnione przy pomocy wartości 0. Tą samą wartością zostaną wypełnione wszystkie bajty poprzedzające kod lub występujące po nim, które zostaną wstawione do pamięci przy użyciu opcji /OFFSET oraz /LENGHT. Wartość opcji /FILL musi zawsze być liczbą szesnastkową. Wartością domyślną opcji jest wartość FFh.
Domyślnie, ostatni bajt pliku binarnego jest ostatnim bajtem o najwyższym adresie pliku HEX. Jeśli plik binarny powinien mieć dokładnie zdefiniowaną długość, wtedy za ostatnim bajtem można dodać odpowiednią ilość bajtów wypełniających (np. by dobrze wypełnić pamięć EPROM) przy użyciu opcji /LENGHT:

/LENGHT: 8000

Taki zapis spowoduje dodanie za ostatnim bajtem pliku HEX tylu bajtów wypełniających, by plik zajmował 32768 bajtów. Wartość parametru /LENGHT powinna być zawsze liczbą szesnastkową.

Jeśli program HEXBIN został wywołany ze wszystkimi opcjami, może na końcu swojej pracy wyświetlić raport konwersji, np.:

Hex File Converter HEXBIN V2.3

offset: 1000H bytes
first adress: 9000H
last adress: A255H
fill peepholes with: 00H
binary image lenght: 8000H bytes

Opcja /QUIET blokuje wyświetlanie informacji w konsoli bez względu na to, czy wyświetlane są komunikaty o błędach.

Opcje mogą być skracane tak długo, jak długo są unikalne!

Przykłady:

0.) HEXBIN
Gdy program jest wywołany bez parametrów, wyświetla ekran pomocy:

Hex File Converter HEXBIN V2.3

usage: HEXBIN <hexfile> [<binary>] [options]

options: /OFFSET: offset
/LENGHT: lenght
/FILL: fillbyte
/QUIET

1.) HEXBIN PROGRAM

Takie wywołanie dokona konwersi pliku w formacie Intel-HEX o nazwie PROGRAM.HEX na plik z zawartością binarną PROGRAM.BIN.

2.) HEXBIN TARZAN.OBJ JUNGLE/FILL:E5

Takie wywołanie dokona konwersji pliku Intel-HEX o nazwie TARZAN.OBJ do pliku JUNGLE.BIN i wypełnia wszystkie puste bajty w kodzie pliku HEX wartościami binarnymi E5h.

3.) HEXBIN PROJECT EPROM. /off:8000 /lenght:10000 /f:0

Taki zapis dokona konwersji pliku w formacie Intel-HEX o nazwie PROJECT.HEX do pliku z zawartością binarną dla pamięci EPROM, wypełniając wszystkie bajty oraz początkowe 32kB pamięci wartością 0, a także zapełniając całe 64 kB pamięci.

Po zakończeniu program HEXBIN zwraca systemowi kod błędu:

sytuacja	kod błędu
brak błędów	0
błędy konwersji	1
krytyczne błędy wykonania	2

Występuje również wersja programu dla konsoli Win32: EXBINW.EXE!
HEXBINW jest funkcjonalnie identyczna z HEXBIN, ale potrafi działać z plikami o długich nazwach.

II. Implementacja dla Linuksa

Wersja ASEM-51 1.2 była dostępna tylko dla systemu DOS. By pozbyć się DOSowego wyglądu i działania, wiele interfejsów musiało zostać ponownie napisanych, np: linia komend, wykonywanie, konsola wejścia/wyjścia, obsługa plików, środowiska UNIXowego oraz obsługa pamięci. Więcej, zachowanie programów musiało zostać zaadoptowane do zasad UNIXa.
Znaczna część pozostałych akcentów może przypominać system DOS.

Z drugiej strony, programy dla Linuxa potrafią odczytywać pliki ASCII w formatach systemów DOS i UNIX, jednakże zapisywane są one zawsze w formacie systemu UNIX. Wszystkie te różnice sprawiają, że konieczne jest opisanie implementacji dla Linuxa w osobnym rozdziale.

II.2.1Pliki

Dystrybucja dla Linuxa powinna zawierać następujące grupy plików:

1.)	asem_51.doc	instrukcja użytkownika w formacie ASCII
	docs.htm	plik indeksu dokumentacji ASEM-51w formacie HTML
	*.htm	kolejne strony dokumentacji HTML
	*.gif	pliki *.GIF używane przez dokumentację HTML
	*.jpg	pliki *.JPG używane przez dokumentację HTML
	asem	kompilator asemblera (Linux 386)
	asem.1	plik manuala dla kompilatora asem
	hexbin	program konwersji plików HEX do BIN (Linux 386)
	hexbin.1	plik manuala dla hexbin
	demo.a51	program przykładowy w asemblerze dla 8051
	*.mcu	pliki definicji mikrokontrolerów 8051 (dla dokładnej listy MCU należy zajrzeć do rozdziału IV)
2.)	boot51.doc	instrukcja użytkownika programu boot-51 w formacie ASCII
	boot51.htm	plik indeksu dokumentacji HTML programu BOOT-51
	boot51.a51	plik źródłowy w asemblerze dla 8051 programu boot-51 (dla ASEM-51 V1.3 i wyższych)
	customiz	program ustawień użytkownika dla BOOT-51 (Linux 386)
	customiz.1	plik manuala dla customiz
	boot	skrypt powłoki dla aplikacji ładowania programu do pamięci 8051
	boot.1	plik manuala dla boot
	upload	wywoływany tylko przez boot (wersja podstawowa)
	upload.new	nowa wersja programu ładującego (optymalizowana dla stty 2.0 i późniejszych)
	reset51	program do kasowania płyty docelowej przez porty PC
	reset51.1	plik manuala dla reset51
	blink.a51	przykładowy program testowy dla BOOT-51
3.)	README.1ST	skrócona informacja w formacie ASCII
	license.doc	licencja w formacie ASCII dla ASEM-51
	release.130	notatka dla wersji 1.3 ASEM-51
	support.doc	przewodnik wsparcia w formacie ASCII dla ASEM-51
	install.sh	przeprowadza prawidłową instalację ASEM-51 w systemie Linux
	uninst51.sh	usuwa wszystkie pliki pakietu ASEM-51 w systemie Linux

II.2.2Instalacja w systemie Linux

ASEM-51 jest dostępny również dla systemu Linux jako archiwum tar oraz jako pakiet RPM. Jeśli posiadasz pakiet RPM, zaloguj się jako administrator i wpisz

```
rpm –i asem51-1.3-1.i386.rpm
```

Pakiet RPM został przetestowany w systemie SUSE tylko, ale powinien również pracować na innych dystrybucjach Linuksa, które obsługują standard plików FSH (prawidłowo działa również w systemach Mandriva oraz Aurox- pochodnym Fedora Core, dawniej Red Hat- przypis tłumacza).
Jeśli posiadasz archiwum tar, wykonaj następujące kroki:

```
gzip –d asem51-1.3-ELF.tar.gz
tar xvf asem51-1.3-ELF.tar
cd asem51
sh install.sh
```

Jeśli instalujesz ASEM-51 jako root (preferowany sposób), skrypt instalacyjny install.sh zainstaluje cały pakiet w /usr/local/share/asem-51/1.3, i utworzy kilka symbolicznych dowiązań w /usr/local/bin oraz w /usr/local/man/man1.

Jeśli instalujesz ASEM-51 jako inny użytkownik, install.sh próbuje zainstalować oprogramowanie w katalogu domowym pod ~/asem5/1.3 i utworzyć kilka symbolicznych dowiązań w ~/bin oraz w ~/man/man1.

Dalsze szczegóły znajdziesz w informacjach wyświetlanych przez skrypt install.sh w konsoli:

If you haven't installed ASEM-51 as root, it may be necessary to add
~/bin to your PATH, and ~/man to your MANPATH.
/ Jeśli nie zainstalowałeś ASEM-51 jako użytkownik root, konieczne może być dodanie
~/bin do zmiennej środowiskowej PATH oraz ~/man do zmiennej środowiskowej MANPATH/

By ustawić ścieżkę przeszukiwania dla plików dołączanych z definicjami mikrokontrolerów *.mcu, możesz zdefiniować opcjonalną zmienną środowiska o nazwie ASEM51INC.
By to zrobić użytkownicy interpreterów bash, ksh i sh powinni dodać poniższą linię do swoich plików .profile:

```
ASEM51INC=/usr/local/share/asem-51/1.3/mcu
export ASEM51INC
```

Użytkownicy csh, tcsh i zsh powinni dodać poniższą linię do swoich plików .login:

```
setenv ASEM51INC /usr/local/share/asem-51/1.3/mcu
```

Jeśli posiadasz zainstalowany ASEM-51 w swoim katalogu domowym, ASEM51INC powinien wskazywać na ~/asem-51/1.3/mcu.

By odczytać dokumentację HTML, wywołaj swoją przeglądarkę oraz dodaj zakładkę z plikiem indeksu dokumentacji

```
/usr/local/share/asem-51/1.3/html/docs.htm      (dla instalacji jako root)
~/asem-51/1.3/html/docs.htm                     (dla instalacji innego użytkownika)
```

Uwaga! Nie możesz dokonać resetu systemu docelowego z mikrokontrolerem 8051 przy pomocy portu PC, jeśli ASEM-51 nie został zainstalowany jako root!
(Po szczegóły zajrzyj do dokumentacji BOOT-51)

Jeśli ASEM-51 został zainstalowany, a nie odpowiada użytkownikowi, można go łatwo usunąć. Jeśli instalacja miała miejsce przy pomocy pakietu RPM, należy wpisać

rpm –e asem51

Jeśli instalacja miała miejsce z archiwum tar, należy się upewnić się, że usunięcia dokonano jako ten sam użytkownik, który pakiet instalował! Uruchom

uninst51.sh

i gotowe.

II.2.3 Operacje przy użyciu konsoli systemu Linux

Wywołanie kompilatora w systemie Linux jest następujące:

asem [<opcje>] <plik źródłowy> [<plik obiektowy> [<plik listingu>]]

gdzie <źródło> jest plikiem źródłowym w języku asemblera dla mikrokontrolera rodziny 8051, <plik obiektowy> jest plikiem wyjściowym (wynikowym) a <plik listingu> jest plikiem zawierającym dokładną listę programu.
Wszystkie nazwy plików, które są wyraźnie wyspecyfikowane, sa pozostawione bez zmian. Parametry <plik obiektowy> oraz <plik listingu> są opcjonalne. Gdy zostaną pominięte, <plik obiektowy> przyjmie nazwę pliu źródłowego lecz z rozszerzeniem .HEX (lub .OMF), a <plik listingu> przyjmie nazwę pliku źródłowego lecz z rozszerzeniem .LST:

plik	rozszerzenie
<plik obiektowy>	.hex (z opcją -o: .omf)
<plik listingu>	.lst

Zamiast nazw plików można również użyć nazw urządzeń wyjściowych, na które ma być przekierowany pli wynikowy. Nazwy urządzeń powinny się zaczynać od "/dev/". Oczywiście w nazwach urządzeń nie będzie żadnych rozszerzeń. Program nie sprawdza, czy wyspecyfikowane urządzenie rzeczywiście istnieje w systemie ani czy jest odpowiednie dla danego zadania.
Istnieje również możliwość odczytania pliku źródłowego z portu wejściowego, jednakże nie jest to zalecane, gdyż od chwili, gdy ASEM-51 stał się programem dwuprzebiegowym, wejście jest odczytywane dwukrotnie. Maksymalna długość parametrów pliku jest ograniczona do 255 znaków!

Asem rozpoznaje następujące opcje:

krótka opcja		długa opcja	
-i path1:path2:path3	|	--includes=path1:path2:path3	/ustawienie ścieżki przeszukiwania/
-d symbol [:value[:type]]	|	--define=symbol[:value[:type]]	/definiuje symbol, jego wartość i typ/
-o	|	--omf-51	/wyjście to plik dla symulatorów/
-c	|	--columns	/podaje numer kolumny z błędem/
-v	|	--verbose	

Krótkie i długie opcje w tym samym rzędzie są sobie równe. Długie opcje mogą być skracane tak długo, jak długo wydają się unikalne. Ważna jest wielkość liter wszystkich nazw opcji!

Gdy użyta jest opcja –includes, kompilator przeszukuje wpisaną w opcji ścieżkę (ścieżki), gdy pliki dołączone do projektu nie mogą być odnalezione w katalogu głównym. Ścieżka może być dowolną ilością katalogów oddzielonych przez znaki ':'. Katalogi będą przeszukiwane od lewej do prawej.
Ścieżka wyznaczona w opcji –includes jest przeszukiwana przed ścieżką ustawioną jako zmienna środowiska ASEM51INC!
Maksymalna długość ścieżki jest ograniczona do 255 znaków.

Opcja –define jest przydatna by wydzielić poszczególne warianty programu w linii komend, które mają być kompilowane przy użyciu kompilacji warunkowej. Pozwala to zdefiniować symbol z wartością i typem w linii komend. Wartość i typ są opcjonalne. Typ domyślnie jest ustawiany na NUMBER (numer, wartość liczbowa), jeśli nie zostanie wyspecyfikowany. Wartość domyślna symbolu to 0 w przypadku pominięcia definicji. Wartość symbolu może być dowolną stałą liczbową. Typ symbolu musi być jednym z poniższych znaków:

C	=	CODE	
D	=	DATA	
I	=	IDATA	
X	=	XDATA	
B	=	BIT	
N	=	NUMBER	(wartość liczbowa- typ ustawiony domyślnie)

Domyślnie kompilator generuje plik wynikowy w formacie Intel-HEX. Gdy używa się opcji --omf-51, generowany jest plik OMF-51.

Przykłady:

0.) asem

Wywołanie bez parametrów spowoduje, że kompilator wyświetli poniższe informacje:

MCS-51 Family Macro Assembler ASEM-51 V1.3

usage: asem [options] <source> [<object. [<listing>]]

options:	-i	--includes=path1:path2:path3
	-d	--define=symbol [:value[:type]]
	-o	--omf-51
	-c	--columns
	-v	--verbose

1.) asem program.a51

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie program.a51 i wygenerowanie pliku w formacie Intel-HEX program.hex oraz pliku listingu program.lst

2.) asem tarzan.asm jane jungle.prn

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie tarzan.asm i wygeneruje plik w formacie Intel-HEX o nazwie jane oraz plik listingu jungle.prn.

3.) asem project. eprom

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie project oraz wygeneruje plik wyjściowy w formacie Intel-HEX o nazwie eprom.lst.

4.) asem -o rover.a51

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie rover.a51 i wygeneruje plik wynikowy w formacie OMF-51 o nazwie rover.omf oraz plik listingu rover.lst.

5.) asem sample.a51 /dev/ttyS0 /dev/null

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie sample.a51, wysyła plik wyjściowy w formacie Intel-HEX do portu szeregowego /dev/ttyS0 oraz blokuje generowanie pliku listingu przez wysłanei jego do /dev/null.

6.) asem -i /usr/local/include/asem-51:~/8051/inc app.a51

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie app.a51, podczas gdy wszystkie pliki dołączane najpierw będą poszukiwane w bieżącym katalogu, następnie w /usr/local/include/asem-51 i ostatecznie w ~/8051/inc.

7.) asem –define=Eva_Board:8000H:C universal.a51

Powyższe wywołanie spowoduje kompilację pliku źródłowego programu napisanego w asemblerze dla mikrokontrolera rodziny 8051 o nazwie universal.a51. Podczas kompilacji napotkany w kodzie programu ciąg słów EVA_BOARD zostanie zamieniony wartością 8000H.

Gdy błędy w programie zostaną wykryte, odpowiednie komunikaty zostaną wygenerowane. Może to wyglądać w następujący sposób:

applicat.a51 (14): must be known on first pass
userbits.inc (6): attempt to divide by zero
defines.inc (37): symbol not defined
applicat.a51 (20): symbol not defined
applicat.a51 (27): no END statement found

Każdy błąd jest opisywany nazwą pliku źródłowego, w którym wystąpił, numerem lini, gdzie wystąpił oraz komunikatem błędu.
Taki format wyjściowy błędów pozwala powiązać ASEM-51 ze środowiskami IDE innych firm. Doskonałe dopasowanie może zostać osiągnięte przy użyciu opcji –columns. Gdy zostanie ona użyta, numery kolumn z błędami są dodatkowo wyświetlane w komunikacie o błędach:

applicat.a51(14,12): must be known on first pass
userbits.inc(6,27): attempt to divide by zero
defines.inc(37,18): symbol not defined

applicat.a51(20,18): symbol not defined
applicat.a51(27,1): no END statement found

Gdy błędy zostaną wykryte w liniach zawierających makra, nie ma wtedy odpowiadającego im usytuowania w pliku źródłowym. Zamiast tego błąd jest oznaczany przy pomocy nazwy pliku źródłowego oraz numeru lini, w której makro było wywoływane. (Dla makr wywoływanych jest to numer linii z wywołaniem, a dla bloków z2-tlonych jest to linia ENDM.)
By podpowiedzieć użytkownikowi, gdzie jest błąd, nazwa makra, linia (i opcjonalnie kolumna) jest wstawiana przed aktualnym komunikatem błędu:

uartio.a51(44,1): RECEIVE(3,22): segment type mismatch
uartio.a51(87,1): REPT(4,19): symbol not defined
uartio.a51(87,1): REPT(8,19): symbol not defined
uartio.a51(87,1): REPT(12,19): symbol not defined

Numer linii jest numerem linii, w której znajduje się makro, zaczynającym się od 1. Jeśli błąd pojawia się w makrze zawierającym pętle, słowo REPT zastępuje nazwę makra.

Domyślnie ASEM-51jest całkowicie “cichy”,jeśli nie wykryje żadnego błędu. Jeśli opcja --verbose jest wywoływana, dodatkowo produkt, wersja oraz sumaryczna informacja o błędach jest wyświetlana na standardowym wyjściu:

MCS-51 Family Macro Assembler ASEM-51 V1.3

uartio.a51(44,1): RECEIVE(3,22): segment type mismatch
uartio.a51(87,1): REPT(4,19): symbol not defined
uartio.a51(87,1): REPT(8,19): symbol not defined
uartio.a51(87,1): REPT(12,19): symbol not defined

4 errors detected

Po przerwaniu pracy, ASEM-51 zwraca kod wyjścia procesowi wywołującemu:

sytuacja	kod wyjścia
brak błędów	0
błędy programu	1
krytyczne błędy wykonania	2

Uwaga: Ostrzeżenia są również przesyłane do standardowego wyjścia, ale nie mają wpływu na kod wyjścia.

II.2.4Ustawienia zaawansowane systemu Linux

By ustawić ścieżkę przeszukiwania dla plików nagłówkowych, można ustawić opcjonalną zmienną środowiskową ASEM51INC:

1.) Dla interpreterów bash, ksh, sh:

ASEM51INC=<ścieżka>
export ASEM51INC

2.) Dla csh, tcsh, zsh:

setenv ASEM51INC <ścieżka>

<ścieżka> może być dowolną liczbą katalogów rozdzielonych przez znak ':'. Należy się upewnić, że całe wyrażenie nie zawiera żadnych spacji ani tabulatorów! Jeśli ASEM51INCjest zdefiniowana, kompilator przeszukuje ustawioną <ścieżkę> w poszukiwaniu plików, które nie zostały odnalezione w katalogu bieżącym ani w katalogach wyspecyfikowanych opcją –includes. Katalogi <ścieżki> będą przeszukiwane od lewej do prawej.

Przykłady:

1.) bash:

```
ASEM51INC=/usr/local/include/asem-51:~/micros/mcs51/inc
export ASEM51INC
```

Jeśli pliki nagłówkowe nie mogą zostać odnalezione w katalogu bieżącym, ani w ścieżce wpisanej w opcji –includes (jeśli została użyta), kompilator przeszukuje /usr/local/include/asem-51 oraz ~/micros/mcs51/inc.

2.) csh:

```
setenv ASEM51INC /usr/local/include/asem-51
```

Jeśli ASEM51INCjest zdefiniowana w systemie w powyższy sposób, kompilator na końcu poszukiwania sprawdzi katalog /usr/local/include/asem-51 w poszukiwaniu plików.

Maksymalna długość <ścieżki> jest ograniczona do 255 znaków.

II.2.5 Narzędzie HEXBIN

Większość programatorów pamięci EPROM akceptuje pliki w formacie Intel-HEX, które są plikami wynikowymi kompilatora ASEM-51. Jednakże czasem dla programatorów specjalnego przeznaczenia może się okazać użyteczne przekonwertowanie pliku HEX do pliku z zawartością binarną. Do tego rodzaju konwersji służy narzędzie programowe HEXBIN dostarczone z ASEM-51.
Jego wywołanie jest następujące:

hexbin [<opcje>] <plik Intel-HEX> [<plik z zawartością binarną>]

gdzie <plik Intel-HEX> jest plikiem wejściowym w formacie Intel-HEX, a <plik z zawartością binarną> jest plikiem z programem, który został przekonwertowany do wartości binarnych. Wszystkie nazwy plików pozostają bez zmian. Parametr <plik z zawartością binarną> jest parametrem opcjonalnym. Gdy zostanie pominięty, nazwa pliku wynikowego zostaje skopiowana z pliku źródłowego (HEX), lecz z rozszerzeniem ".bin".
Maksymalna długość parametrów pliku jest ograniczona do 255 znaków!

Zamiast nazwy pliku można użyć nazwy urządzenia, do którego nastąpi przekierowanie wyjścia lub wejścia. Nazwy urządzeń zaczynają się od "/dev/". Oczywiście do nazw urządzeń nie będą dodawane żadne rozszerzenia. Nazwa urządzenia lub jego prawidłowość nie są sprawdzane podczas konwersji.

Hexbin rozpoznaje następujące opcje:

krótka opcja		długa opcja	
-o <offset>	|	--offset=<offset>	//przesunięcie
-l <lenght>	|	--lenght=<lenght>	//długość
-f <fillbyte>	|	--fill=<fillbyte>	//wypełnienie
-v	|	--verbose	

Krótkie i długie opcje w tym samym rzędzie są równoważne. Długie opcje mogą być skracane tak długo, jak długo są unikatowe. Wszystkie nazwy opcji są wrażliwe na wielkość pisanych liter!

Plik binarny może być kontrolowany przy pomocy opcji –offset, --fill oraz –leght.
Normalnie pierwszy bajt w pliku binarnym odpowiada pierwszemu bajtowi w pliku HEX na najniższym adresie. Jeśli istnieje potrzeba przesunięcia początku pliku (np. by uzyskać potrzebną alokację pliku w pamięci EPROM), można tego dokonać przy pomocy opcji –offset:

--offset=1000

Powyższy zapis spowoduje wstawienie na początku 4096 bajtów wypełniających przed danymi z pliku HEX. Wartość parametru offset musi być liczbą szesnastkową. Domyślnie offset jest ustawiony na 0.
W przypadku, gdy w pliku występują puste bajty, których program nie wykorzystuje, można zdefiniować wartość, która ma wypełnić puste miejsca przy pomocy opcji –fill, jak poniżej:

--fill=0

Powyższy zapis spowoduje, że wszystkie puste przestrzenie w programie zostaną wypełnione wartością 0, podobnie, jak wszystkie bajty wypełniające wstawione przy pomocy opcji –offset i –lenght. Wartość parametru fill musi zawsze być wartością szesnastkową (8-bitową). Domyślną wartością opcji jest przyjazna dla pamięci wartość FFh.
Domyślnie ostatni bajt w pliku binarnym jest ostatnim bajtem w pliku szesnastkowym. Jeśli plik binarny powinien mieć zdefiniowaną z góry długość, można go zwiększyć wstawiając za ostatnim bajtem odpowiadającym ostatniemu bajtowi pliku HEX bajty wypełniające przy pomocy opcji –lenght:

--lenght=8000

Powyższy zapis spowoduje wstawienie za ostatnim bajtem zostanie wstawione tyle bajtów wypełniających, by długość pliku wyniosła dokładnie 32768 bajtów. Parametr opcji musi być zawsze wartością szesnastkową.

Domyślnie, HEXBIN jest całkowicie “cichy”, jeśli nie wykryje żadnego błędu.
Jeśli użyta zostanie opcja --verbose, dodatkowo informacja o produkcie, jego wersji oraz raport konwersji pliku zostaje wysłany do standardowego wyjścia:

Hex File Converter HEXBIN V2.3

offset:	FF0H bytes
first address:	7FF0H
last address:	8255H
fill peephole with:	A5H
binary image lenght:	2000H bytes

Przykłady:

0.) hexbin

Po wywołaniu bez parametrów, HEXBIN wyświetli ekran pomocy:

Hex File Converter HEXBIN V2.3

usage: hexbin [options] <hexfile> [<binary>]

options: -o --offset=<offset>
-l –lenght=<lenght>
-f –fill=<fillbyte>
-v –verbose

1.) hexbin program.hex

Taki zapis spowoduje konwersję pliku w formacie Intel-HEX o nazwie program.hex na plik z zawartością bitową o nazwie program.bin.

2.) hexbin -f E5 tarzan.obj jungle.bin

Taki zapis spowoduje przekonwertowanie pliku w formacie Intel-HEX o nazwie tarzan.obj na plik o zawartości binarnej jungle.bin i wypełnienie bajtów pustych wartością binarną E5h.

3.) hexbin –off=8000 -l10000 –fill=0 project.hex eprom

Taki zapis spowoduje przekonwertowanie pliku w formacie Intel-HEX o nazwie project.hex na plik binarny dla pamięci eprom, wypełnienie pierwszych 32KB bajtami wypełniającymi, zapełnienie pamięci po ostatnim bajcie z pliku HEX tak, by całość zajmowała 64KB i wypełnienie bajtów nie używanych przez program wartościami 0.

Po zakończeniu działania programu HEXBIN, zwraca on kod wyjścia do procesu wywołującego:

sytuacja	kod wyjścia
brak błędów	0
błędy konwersji	1
krytyczne błędy wykonania	2

II.3Program demonstracyjny

Podczas stawiania pierwszych kroków z nowym kompilatorem, dobrze jest mieć program, który można przy jego pomocy skompilować. Do takich celów służy program demonstracyjny DEMO.A51 dostarczony z pakietem, który może zostać użyty do pierwszego testu kompilatora po instalacji. By tego dokonać powinieneś mieć zainstalowany zgodnie ze wskazówkami z poprzednich rozdziałów kompilator lub mieć wszystkie pliki w katalogu roboczym.
W systemie MS-DOS lub w oknie dosowym systemu Windows, wpisz

ASEM DEMO
HEXBIN DEMO

po znaku zachęty. ASEM i HEXBIN powinny zakończyć pracę bez błędów i na dysku powinny pojawić się następujące pliki:

DEMO.HEX	plik w formacie Intel-HEX
DEMO.LST	plik listingu programu DEMO.A51
DEMO.BIN	plik z zawartością binarną pliku DEMO.HEX

W systemie Linux

```
asem demo.a51
hexbin demo.hex
```

Ponownie asem oraz hexbin powinny zakończyć siebez błędów i powinny pojawić sięna dysku pliki:

demo.hex	plik w formacie Intel-HEX
demo.lst	plik listingu programu demo.a51
demo.bin	plik z zawartością binarną pliku demo.hex

Jeśli cokolwiek pójdzie źle, ASEM-51 nie jest prawidłowo zainstalowany, w pakiecie może brakować jakiegoś pliku lub kompilator nie może znaleźć pliku definicyjnego 8052.mcu!
demo.a51 może posłużyć również jako przykładowy program, który zawiera (prawie) wszystkie instrukcje mikrokontrolera, pseudoinstrukcje, zmienne kontrolne oraz meta instrukcje, które zostały zaimplementowane w ASEM-51. W razie jakichkolwiek wątpliwości związanych z użyciem poszczególnych komend, plik demo.a51 może okazać się pomocną podpowiedzią.
Odmiennie do innych kompilatorów, plik listingu pozwala porównać bez problemu plik źródłowy z wygenerowanym kodem maszynowym.

III Język asemblera w kompilatorze ASEM-51

Użytkownik powinien znać mikrokontrolery rodziny 8051 oraz programowanie w asemblerze tej rodziny. Ta instrukcja nie wyjaśnia architektury mikrokontrolerów rodziny MCS-51 ani nie prowadzi wykładu na temat podstaw programowania w asemblerze. Instrukcja opisuje jedynie ogólną składnię asemblera oraz instrukcje asemblera, które zostały zaimplementowane w kompilatorze ASEM-51.

III.1 Wyrażenia

Plik źródłowy składa się z sekwencji, które posiadają jedną z poniższych form:

[symbol:]	[instrukcja [argument]]	[;komentarz]
symbol	instrukcja argument	[;komentarz]
$zmienna kontrolna [(argument)]		[;komentarz]

Wszystko, co jest wpisane w nawiasach jest opcjonalne.
Maksymalna długość linii kodu źródłowego wynosi 255 znaków.
Wszystko za znakiem ';' do końca linii jest komentarzem. Puste linie są również uważane za komentarz.
Elementy języka asemblera mogą być rozdzielane przez spacje lub tabulatory.
Poza stałymi łańcuchowymi, wszędzie duże i małe znaki są równoważne.

Przykład:

```
HERE:  MOV A,#0FFH          ;definiuje znacznik HERE i zapisuje do A wartość FFH

       YEAR EQU 2002        ;definiuje symbol dla aktualnego roku

$INCLUDE (80C517.MCU)       ; dołącza do pliku plik definicji rejestrów SAB80C517
```

III.2Symbole

Symbole są definiowanymi przez użytkownika nazwami dla adresów, liczb i makr.
Ich maksymalna długość to 31 znaków. Mogą być dłuższe, ale wszystko po 31 znaku jest ignorowane.
Symbole mogą zawierać litery, cyfry oraz znaki '_' i '?'.
Symbol NIE MOŻE się zaczynać od liczby!
Duże i małe litery są rozważane jako te same znaki.
Uwaga: Słowa kluczowe asemblera nie mogą być redefiniowane jako symbole użytkownika!

Przykłady: Czy_to_rzeczywiscie_SYMBOL_? jest symbolem!

III.3Stałe

Stałe numeryczne składają się z sekwencji cyfr, po których znajduje się specyfikator typu. Pierwszy znak musi zawsze być cyfrą dziesiętną. Dopuszczalnymi liczbami i specyfikatorami typów są:

stała	cyfry	specyfikator
binarna	0 ... 1	B
ósemkowa	0 ... 8	Q lub O
dziesiętna	0 ... 9	D lub brak
szesnastkowa	0 ... F	H

Poniższe stałe są równe co do wartości:

1111111B	binarna
177Q	ósemkowa
177o	ósemkowa
127	dziesiętna
127d	dziesiętna
07FH	szesnastkowa

Stałe znakowe mogą być używane, kiedykolwiek wartość numeryczna jest dozwolona.
Stała znakowa składa się z jednej lub dwu znaków drukowanych zamkniętych w pojedynczych lub podwójnych cudzysłowach. Cudzysłów może zostać wyrażony przez dwa następujące po sobie cudzysłowy, np:

'X'	8 bitowa stała:	58H
“a@”	16 bitowa stała:	6140H
' ' ' '	8 bitowa stała:	27H

Po wyrażeniu DB, znaki mogą mieć dowolną długość.
W tym przypadku nazywane są łańcuchami, np:
DB 'To jest tylko tekst!”

III.4Wyrażenia

Wyrażenia arytmetyczne składają się z operandów, operatorów oraz nawiasów.
Operandami mogą być symbole zdefiniowane przez użytkownika, stałe lub specjalne symbole asemblera.
Wszystkie operandy są traktowane jako liczby 16-bitowe bez znaków.
Specjalnymi symbolami asemblera, które mogą być używane jako operandy są:

AR0, ... , AR7	adres bezpośredni rejestrów od R0 do R7
$	licznik lokalizacji aktualnie aktywnego segmentu (adres początku aktualnie wykonywanego wyrażenia)

Poniższe operatory są zaimplementowane w kompilatorze:

Operatory ogólne:	+	identyczne z:	+x = x
	-	dopełnienie do dwóch:	-x = 0-x
	NOT	dopełnienie do jedności:	NOT x = FFFFH-x
	HIGH	bajt wysokiego znaczenia	
	LOW	bajt niskiego znaczenia	

Operatory binarne:	+	dodawanie bez znaku	
	-	odejmowanie bez znaku	
	*	mnożenie bez znaku	
	/	dzielenie bez znaku	
	MOD	reszta bez znaku	
	SHL	przesunięcie bitów w lewo	
	SHR	przesunięcie bitów w prawo	
	AND	funkcja logiczna AND	
	OR	funkcja logiczna OR	
	XOR	funkcja logiczna EX-OR	
	.	operator bitowy używany przy zmiennych adresowanych bitowo	
	EQ lub =	równe	wynikiem są: 0 gdy PRAWDA
	NE lub <>	różne	wynikiem są: FFFFH gdy FAŁSZ
	LT lub <	mniejsze niż	
	LE lub <=	mniejsze lub równe	
	GT lub >	większe niż	
	GE lub >=	większe lub równe	

Operatory, które nie są specjalnymi znakami, a są słowami kluczowymi muszą być odseparowane od ich operandów przynajmniej jedną spacją lub jednym znakiem tabulacji.
Generalnie wyrażenia są rozwiązywane od lewej do prawej zgodnie z pierwszeństwem operatorów, które można zmienić używając nawiasów. Nawiasy mogą być zagnieżdżane do dowolnego poziomu.
Wyrażenia zawsze są rozwijane do szesnastobitowej liczby bez znaku. Przekroczenie zakresu jest ignorowane.
Gdy wynik musi być liczbą ośmiobitową, bajt wysoki musi mieć wartość 00 lub FF.

Kolejność działań:

()		^	najwyższa
+ - NOT HIGH LOW	(ogólnie)	|	
.		|	
* / MOD		|	
SHL SHR		|	
+ -	(binarnie)	|	
EQ = NE <> LT < LE <= GT > GE >=		|	
AND		|	
OR XOR		v	najniższa

Przykład: Wynik wyrażenia P1.((87=3)/10 AND -1 SHR 0DH) wyniesie 91H.

III.5Zestaw instrukcji procesora 8051

Kompilator ASEM-51 zawiera implementację wszystkich instrukcji maszynowych mikrokontrolera 8051, włącznie z ogólnymi instrukcjami skoków oraz wywołań. Kompilator ma zaimplementowane dwie instrukcje

JMP <adres>
CALL <adres>

które nie reprezentują kodu maszynowego: ogólny skok oraz ogólne wywołanie.
Te instrukcje zostaną zamienione na skok lub wywołanie, niekoniecznie najkrótsze, które “dosięgną” wyspecyfikowanego adresu.
JMP może zostać przetłumaczone na SJMP, AJMP lub LJMP, a CALL może zostać zamienione na ACALL lub LCALL. Trzeba pamiętać, że decyzja kompilatora może nie być optymalna. Dla adresów kodu, które znajdują się poza aktualną instrukcją, kompilator zawsze używa LJMP lub LCALL. Jednakże dla wywołań adresów przed aktualną instrukcją, CALL i JMP mogą być narzędziem, które w znacznym stopniu mogą zredukować rozmiar kodu wynikowego bez jakichkolwiek problemów.

Przy użyciu zmiennej $PHILIPS, ASEM-51 może zostać przełączony do skróconej listy instrukcji mikrokontrolerów serii Philips 83C75x. To blokuje instrukcje LJMP, LCALL oraz MOVX, a także pseudoinstrukcje XDATA, XSEG, i skoki wraz z wywołaniami zawsze zostaną skompilowane jako wywołania z adresowaniem absolutnym.

Reszta instrukcji mikrokontrolerów rodziny 8051 została wyszczególniona w załączniku D.
Załączniki I oraz J zawierają tablice instrukcji z ich kodami maszynowymi, mnemonikami, argumentami, długością, zmienianymi flagami oraz czasem wykonywania. Kompleksowy przykładowy program DEMO.A51 zawiera wszystkie instrukcje mikrokontrolera 8051 wypisane z użyciem prawidłowej składni.

Dla szczegółowych informacji na temat architektury oraz instrukcji, zajrzyj do dokumentacji HTML MCS51MAN.HTM dołączonej do pakietu. (Wymaga przeglądarkiinternetowej oraz pełnego dostępu do Internetu!)

III.6Pseudo instrukcje

W poszczególnych rozdziałach wszystkie pseudo instrukcje ASEM-51 są opisane. Symbole leksykalne są napisane małymi literami, a słowa kluczowe asemblera są napisane dużymi literami.

Argumenty instrukcji są reprezentowane przez <arg>, <arg1> lub coś w tym rodzaju. Wyrażenia numeryczne są przedstawione jako <expr>, <expr1> itd. Elementy składni zamknięte w nawiasach są opcjonalne.
Zapis "..." oznacza zawsze "lista z dowolną liczbą elementów".

DB <arg1> [,<arg2> [<arg3>...]] definiuje bajty

Instrukcja DB rezerwuje i inicjuje ilość bajtów z wartościami użytymi jako argumenty. Argumenty mogą być wyrażeniami (które muszą być w formacie 8 bitowym) lub łańcuchami znakowymi dowolnej długości. Użycie DB jest dozwolone tylko w segmencie CODE!

Przykład: DB 19, 'January',98,(3*7+12)/11

DW <expr1> [,<expr2> [,<expr3> ...]] definiuje „słowa" (zmienne typu word)

Instrukcja DW rezerwuje pamięć na zmienne typu word i inicjuje je wartościami zdefiniowanymi w argumentach. Każdy argument może być wyrażeniem i wymaga dwóch bajtów przestrzeni.
Użycie DW jest dozwolone jedynie w segmencie CODE!

Przykład: DW 0,0C800H,1999,4711

DS <expr> definiuje przestrzeń

Wyrażenie rezerwuje daną ilość bajtów bez ich inicjacji w aktualnym segmencie.
Wartość <expr> musi być znana przy pierwszym przejściu!
Użycie DS jest dozwolone w każdym segmencie poza segmentem BIT!

Przykład: DS 200H

DBIT <expr> definiuje bity

Wyrażenie rezerwuje daną ilość bitów bez ich inicjacji. Wartość <expr> musi być znana przy pierwszym przejściu. Użycie DBIT jest dozwolone tylko w segmencie BIT!

Przykład: DBIT 16

NAME <symbol> definiuje nazwę modułu

Definiuje nazwę modułu dla pliku OMF-51. Jeśli nie zdefiniowano nazwy modułu, nazwa modułu jest pobierana z nazwy pliku źródłowego. Kiedy generowany jest plik w formacie Intel-HEX, instrukcja NAME nie jest brana pod uwagę. Nazwa modułu musi być prawidłowym symbolem asemblera. Tylko jedna nazwa instrukcji jest dozwolona w programie. Symbol może być jednak redefiniowany w poszczególnych programach.

Przykład: NAME My_1st_Program

ORG <expr> początek lokacji segmentu

Wyrażenie ustawia licznik lokacji w aktualnym segmencie do wartości <expr>.
Wartość <expr> musi być znana w pierwszym przejściu i musi być większa lub równa wartości początkowej segmentu. Domyślna wartość liczników wszystkich segmentów na początku programu wynosi 0.

Przykład: ORG 08000H

USING <expr> definiuje, który bank rejestrów ma być używany

Wyrażenie ustawia używany bank rejestrów na <expr>, gdzie wartość <expr> może wynosić od 0 do 3. Instrukcja USING ma wpływ na symbole specjalne asemblera AR0, ..., AR7, które reprezentują bezpośredni adres rejestrów R0, ..., R7 w aktualnym banku rejestrów.
Wartość <expr> musi być znana przy pierwszym przejściu! Domyślną wartością dla USING jest 0.

Przykład: USING 1

END koniec programu

To musi być ostatnie wyrażenie w pliku źródłowym. Po wyrażeniu END dozwolone są tylko komentarze i puste linie!

Przykład: END ;koniec programu

<symbol> EQU <expr> definiuje stałą numeryczną
<symbol> EQU <reg> definiuje rejestr
<symbol> SET <expr> definiuje zmienną numeryczną
<symbol> SET <reg> definiuje zmienną w rejestrze

Instrukcja EQU definiuje symbol dla stałej numerycznej lub dla rejestru. Jeśli wyrażenie numeryczne <expr> jest przypisane do symbolu, będzie on typu numerycznego. Jeśli rejestr <reg> jest przypisany do symbolu, będzie on typu rejestrowego. <reg> może być jednym ze specjalnych symboli asemblera: A, R0, R1, R2, R3, R4, R5, R6 lub R7. Symbol raz zdefiniowany przy użyciu EQU nie może być zmieniany!
Instrukcja SET pracuje podobnie do EQU, jednakże symbole zdefiniowane przy użuciu SET mogą być redefiniowane przy użyciu instrukcji SET.
Wartość <expr> i <reg> musi być znana przy pierwszym przejściu!
Symbol zdefiniowany przy użyciu SET nie może być redefiniowany przy użyciu EQU!
Symbol zdefiniowany przy użyciu EQU nie może być redefiniowany przy użyciu SET!
Przy przejściu drugim, referencje do symboli zdefiniowanych przy użyciu SET są zamieniane na ostatnią wartość symbolu, która była definiowana przy przejściu pierwszym.
Symbol rejestru może być używany jako operand w całym programie zamiast adresu rejestru.
Dalsze odwołanie symboli do rejestrów jest zabronione!

Przykłady:
```
MAXMONTH  EQU  12
OCTOBER   EQU  MAXMONTH-2
COUNTREG  EQU  R5

CHAPTER   SET  1
CHAPTER   SET  CHAPTER+1
CHAPTER   SET  A
```

<symbol> CODE <expr> definiuje adres ROM
<symbol> DATA <expr> definiuje adres bezpośredni RAM
<symbol> IDATA <expr> definiuje pośredni adres RAM

<symbol> BIT <expr>	definiuje adres w przestrzeni bitowej
<symbol> XDATA <expr>	definiuje adres w zewnętrznej pamięci RAM

Powyższe instrukcje definiują symboliczne adresy dla pięciu segmentów pamięci (przestrzeni adresowych) mikrokontrolera 8051. Dla DATA, IDATA oraz BIT, wartość <expr> nie może przekroczyć FFH! Wartość parametru <expr> musi być znana przy pierwszym przejściu. Raz zdefiniowany symbol przy pomocy powyższych instrukcji nie może być ponownie definiowany.

Przykłady:			
	EPROM	CODE	08000H
	STACK	DATA	7
	V24BUF	IDATA	080H
	REDLED	BIT	P1.5
	SAMPLER	XDATA	0100H

CSEG [AT <expr>]	przechodzi do segmentu CODE [do adresu]
DSEG [AT <expr>]	przechodzi do segmentu DATA [do adresu]
ISEG [AT <expr>]	przechodzi do segmentu IDATA [do adresu]
BSEG [AT <expr>]	przechodzi do segmentu BIT [do adresu]
XSEG [AT <expr]	przechodzi do segmentu XDATA [do adresu]

Powyższe instrukcje powodują przejście do jednej z pięciu przestrzeni adresowych mikrokontrolera 8051. Jeśli adres bazowy segmentu jest wyspecyfikowany przy użyciu “AT <expr>”, nowy segment o adresie absolutnym jest rozpoczynany i licznik lokacji przestrzeni jest ustawiany do wartości <expr>. Jeśli wyrażenie “AT <expr>” jest pomijane, licznik lokacji zachowuje poprzednią wartość poszczególnych segmentów. Wartość <expr> musi być znana przy pierwszym przejściu kompilatora.
Na początku programu, domyślnym segmentem jest segment CODE, a adresy wszystkich segmentów równe są zero.

Przykłady:		
	DSEG	;przełączenie do segmentu DATA
	CSEG AT 8000h	;nowy segment CODE o adresie 8000h
	XSEG AT 0	;nowy segment XDATA o adresie początkowym 0

III.7Typy segmentów

Każde kompilowane wyrażenie jest przypisane do segmentu zależnego od jego operandów i operatorów. Typ segmentu wskazuje przestrzeń adresową, do której wynik może należeć, jeśli był użyty jako adres.
Występuje sześć możliwych typów segmentów:

CODE
DATA
IDATA
XDATA
BIT
NUMBER (bez typu)

Większość wyników wyrażeń ma tym NUMBER. Oznacza to, że nie posiadają one typu. Jednakże w niektórych przypadkach może być przydatne przypisanie mu typu segmentu.
Poniższe sześć zasad jest stosowanych, gdy oceniany jest typ segmentu:

1. Stałe liczbowe zawsze są bez typu. W konsekwencji, ich segment jest typu NUMBER.
2. Symbole są przypisywane do typu segmentu podczas ich definiowania. Symbole, które są definiowane przy użyciu EQU lub SET nie posiadają typu segmentu.
3. Wyniki ogólnych operacji (+, -, NOT, HIGH, LOW) przyjmą typ segmentu operandów.
4. Wyniki operacji binarnych (poza "+", "-" oraz ".") nie będą posiadały typu segmentu.
5. Jeśli tylko jeden operand w operacji binarnego "+" lub "-"posiada typ segmentu, wynik będzie posiadał ten sam typ segmentu. We wszystkich pozostałych przypadkach, wynik nie będzie posiadał typu segmentu.
6. Wynik operacji bitowej "." będzie zawsze posiadał typ segmentu BIT.

Przykłady:

Poniższe symbole zostały zdefiniowane w programie:

OFFSET	EQU	16
START	CODE	30H
DOIT	CODE	0100H
REDLED	BIT	P1.3
VARIAB4	DATA	20H
PORT	DATA	0C8H
RELAY	EQU	5

1.)	Wyrażenie START+OFFSET+3	będzie miało typ segmentu CODE
2.)	Wyrażenie START+DOIT	nie będzie miało typu segmentu
3.)	Wyrażenie DOIT-REDLED	nie będzie miało typu segmentu
4.)	Wyrażenie 2*VARIAB4	nie będzie miało typu segmentu
5.)	Wyrażenie PORT.RELAY	będzie miało typ segmentu BIT

Typ segmentu jest sprawdzany, gdy wyrażenia pojawiają się jako adresy. Jeśli wyniki wyrażenia nie jest "beztypowy" i nie posiada typu segmentu, instrukcja jest oflagowana komunikatem błędu. Jedynymi wyjątkami są segmenty DATA oraz IDATA, które wychodząc z założenia są kompatybilne w przestrzeni adresowej od 0 do 7FH. Od czasu, kiedy ASEM-51 posiada wsparcie tylko dla segmentów absolutnych, adresy te zawsze wskazują na te same fizyczne miejsca w wewnętrznej pamięci.

Przykład:

```
Line    I       Addr    Code            Source

1:                      N       30              DSEG AT 030H            ;wewnętrzna pamięć RAM
2:                      30      N       01      COUNT: DS 1             ;zmienna licznika
3:
4:                                              CSEG            ;ROM
5:      0000    C2      30      START: CLR COUNT
                                                        ^
                                @@@@@ segment type mismatch @@@@@
```

Instrukcja CLR jest oflagowana komunikatem błędu "segment type mismatch". Jednakże COUNT jest etykietą z segmentem DATA!

III.8Kontrolki Asemblera

ASEM-51 ma zaimplementowane kontrolki, które przyspieszają proces kompilacji i generowania pliku listingu. Występują dwie grupy kontrolek: podstawowe i ogólne.
Podstawowe kontrolki mogą być tylko używane tylko na początku programu i mają wpływ na proces kompilacji. Mogą być poprzedzane przez wyrażenia kontrolne, linie puste i linie komentarzy. Jeśli te same kontrolki podstawowe są użyte kilkukrotnie z różnymi parametrami, ostatnie wystąpienie kontrolki się liczy.
Ogólne kontrolki mogą być używane gdziekolwiek w programie. Przeprowadzają one jedną akcję lub spełniają swoją rolę do czasu, gdy są kasowane lub zmieniane przez wyrażenia na nich operujące.
Kontrolki zawsze zaczynają sie znakiem $, po którym występuje jedna lub kilka kontrolek kompilatora.
Kontrolki kompilatora mogą mieć kilka operandów łańcuchowych, które muszą być otoczone nawiasami.
Numeryczne operandy są wyrażeniami arytmetycznymi, które muszą być znane przy pierwszym przejściu.
Operandy łańcuchowe są łańcuchami znaków, które są zamknięte w nawiasach zamiast cudzysłowów.
Analogicznie do łańcuchów w cudzysłowach, zabronione jest używanie znaków kontrolnych (w tym tabulatorów)! Ogranicznik łańcucha ')' może by reprezentowany przez dwa następujące po sobie znaki ')'.
Jeśli wyrażenie kontrolne zmienia tryb pliku listingu, wyrażenie kontrolne jest zawsze wymieniane w poprzednim trybie listingu.
Poniższa tabela pokazuje wszystkie zaimplementowane kontrolki i ich skróty:

Kontrolka	Typ	Domyślnie	Skrót	Znaczenie
$COND	G	$COND	---	wypisuje pełne wyrażenie Ifxx.. ENDIF
$NOCOND	G		---	nie wyświetla linii nie spełniających warunku
$CONDONLY	G		---	wyświetla tylko kompilowane linie
$DATE(łańcuch)	P	' '	$DA	dołącza łańcuch z datą do nagłówka strony
$DEBUG	P	$NODEBUG	$DB	dołącza informacje debugowania do pliku wyjściowego
$NODEBUG	P		$NODB	nie dołącza informacji debugowania
$EJECT	G		$EJ	rozpoczyna nową stronę w pliku listingu
$ERROR(łańcuch)	G		---	wymusza błąd zdefiniowany przez użytkownika
$WARNING(łańcuch)	G		---	wyświetla komunikat ostrzeżenia w konsoli
$GEN	G	$GEN	$GE	wyświetla listę wywołań makr i linii rozszerzających
$NOGEN	G		$NOGE	wyświetla tylko listę wywołań makr
$GENONLY	G		$GO	wyświetla tylko listę linii rozszerzających
$INCLUDE(plik)	G		$IC	dołącza plik źródłowy
$LIST	G	$LIST	$LI	wyświetla listę następujących po sobie linii źródłowych
$NOLIST	G		$NOLI	nie wyświetla listy linii źródłowych
$MACRO(n)	P	$MACRO(50)	$MR	rezerwuje n% wolnej pamięci na makra
$NOMACRO	P		$NOMR	rezerwuje całą pamięć na tabelę symboli
$MOD51	P	$MOD51	$MO	używa predefiniowanej listy symboli SFR
$NOMOD51	P		$NOMO	blokuje predefiniowaną listę symboli SFR
$NOBUILTIN	P	list SFR	---	nie wyświetla listy predefiniowanych symboli
$NOTABS	P	use tabs	---	nie używa tabulatorów w pliku listingu
$PAGING	P	$PAGING	$PI	zezwala na formatowanie stron pliku listingu
$NOPAGING	P		$NOPI	blokuje formatowanie stron pliku listingu
$PAGELENGHT(n)	P	n=64	$PL	ustawia ilość linii dla strony dla listingu
$PAGEWIDTH(n)	P	n=132	$PW	ustawia ilość kolumn dla pliku listingu
$PHILIPS	P	MCS-51	---	przełącza kompilator na wspieranie rodziny 83C75x
$SAVE	G		$SA	zachowuje aktualny stan $LIST/$GEN/$COND
$RESTORE	G		$RS	przywraca poprzedni stan /$LIST/$GEN/$COND
$SYMBOLS	P	$SYMBOLS	$SB	tworzy tabelę symboli

Kontrolka	Typ	Domyślnie	Skrót	Znaczenie
$NOSYMBOLS	P		$NOSB	nie tworzy tabeli symboli
$TITLE(łańcuch)	P	copyright	$TT	wstawia łańcuch tytułu w nagłówku strony
$XREF	P	$NOXREF	$XR	tworzy odwołania krzyżowe
$NOXREF	P		$NOXR	nie tworzy odwołań krzyżowych

Dalsze rozdziały zawierają szczegółowe wyjaśnienia do zaimplementowanych kontrolek.

III.8.1Kontrolki podstawowe

$DATE(łańcuch) Wstawia łańcuch z datą w nagłówku pliku listingu.
Jeśli wpisane zostanie $DATE(), wstawiona zostanie aktualna data.
Łańcuch daty zostanie skrócony do maksymalnie 11 znaków.
Domyślnie: brak łańcucha daty.
Kontrolka nie ma znaczenia, gdy użyto kontrolki $NOPAGING

$DEBUG Dołącza informacje debugowania w module OMF-51. Gdy generowany jest plik w formacie Intel-HEX, $DEBUG nie ma wpływu na plik generowany.

$NODEBUG Nie dołącza informacji debugowania. (Domyślne ustawienie!)

$MACRO (n) Zachowuje definicje i rozszerzone wywołania makr. (Ustawienie domyślne!)
Opcjonalnie rezerwuje n% wolnej pamięci dla definicji makr.
(0 <= n <= 100)
Domyślna wartość wynosi n=50.
Kontrolka została zaimplementowana jedynie dla celów kompatybilności.
W ASEM-51 nie ma efektu poza takim, iż kasuje kontrolkę $NOMACRO.

$NOMACRO Nie zachowuje makrodefinicji i rozszerzonych wywołań. Zachowuje całą wolną pamięć na tabelę symboli.
Kontrolka została zaimplementowana jedynie dla celów kompatybilności.
W ASEM-51 blokuje jedynie rozszerzanie makr.

$MOD51 Uruchamia wbudowaną tabelę definicji symboli rejestrów SFR i przerwań. (Ustawienie domyślne!)

$NOMOD51 Wyłącza wbudowaną tabelę definicji symboli rejestrów SFR oraz przerwań.
Predefiniowane symbole ??ASEM_51 oraz ??VERSION nie mogą być wyłączane!

$PAGING Uruchamia formatowanie strony pliku listingu. (Ustawienie domyślne!)

$NOPAGING Wyłącza formatowanie strony pliku listingu.

$PAGELENGHT (n) Ustawia długość strony pliku listingu do n linii.
(12 <= n <= 65535)
Domyślna wartość n=64.
Kontrolka nie ma znaczenia, gdy używana jest kontrolka $NOPAGING.

$PAGEWIDT (n) Ustawia szerokość strony pliku listingu do n kolumn.
(72 <= n <= 255). Domyślna wartość n=132.

$PHILIPS	Uruchamia opcję wsparcia rodziny mikrokontrolerów Philips rodziny 83C75x. Blokuje to instrukcje LJMP, LCALL oraz MOVX, a także pseudoinstrukcje XDATA oraz XSEG. Soki i wywołania zawsze będą kompilowane jako adresowanie absolutne.
$SYMBOLS	Generuje tabelę symboli na końcu pliku listingu. (Ustawienie domyślne!) Gdy kontrolka $XREF jest użyta, $SYMBOLS nie ma efektu.
$NOSYMBOLS	Zawiesza generowanie tabeli symboli na końcu pliku listingu. Gdy $XREF jest użyta, $NOSYMBOLS nie przynosi efektu.
$NOBUILTIN	Zawiesza predefiniowane (wbudowane) symbole w tabeli symboli lub listę odwołań krzyżowych dla łatwiejszego przeglądania. Wyświetlane są jedynie symbole zdefiniowane przez użytkownika.
$NOTABS	zamienia wszystkie użyte znaki tabulacji w pliku listingu na spacje.
$XREF	Generuje listing z odwołaniami krzyżowymi zamiast tabeli symboli. Opcja ta wydłuża proces kompilacji i zajmuje około 67% pamięci więcej!
$NOXREF	Generuje tabelę symboli zamiast listingu z odwołaniami krzyżowymi. (Opcja domyślna!)

Przykłady:

```
$NOMOD51          ;wyłącza definicję symboli SFR mikrokontrolera 8051
$PAGELENGHT(60)   ;ustala długość strony do 60 linii
$PW(80)           ;ustala szerokość strony na 80 znaków
$NOSYMBOLS        ;nie jest konieczna tabela symboli
$NOTABS           ;drukarka nie obsługuje symboli tabulacji
$DATE(2. 8. 95)   ;data ostatniej wersji
$XREF             ;generuje listing z krzyżowymi odwołaniami
$ DEBUG NOPAGING  ;dołącza informacje debugowania w OMF-51
                  ;i blokuje formatowanie strony
```

III.8.2Kontrolki ogólne

$COND	Wyświetla pełną konstrukcję IFxx .. ELSEIFxx .. ELSE .. ENDIF. (Domyśle ustawienie!) Kontrolka jest nadpisywana przez $NOLIST.
$NOCOND	Nie wyświetla linii, które nie spełniają zależności IFxx .. ELSEIFxx .. ELSE .. ENDIF. Opcja nadpisywana jest przez opcję $NOLIST.
$CONDONLY	Wyświetla jedynie linie, które spełniają warunki opisane w IFxx .. ELSEIF .. ELSE .. ENDIF, bez samych wyrażeń IFxx, ELSEIFxx, ELSE, ENDIF. Kontrolka jest nadpisywana przez $NOLIST.
$EJECT	Rozpoczyna nową stronę w pliku listingu. Kontrolka nie przynosi efektu, gdy użyto $NOPAGING.

$ERROR (łańcuch)	Wymusza błąd kompilacji z komunikatem błędu zdefiniowanym przez użytkownika. Opcja ta jest przeznaczona jako wsparcie dla zarządzania konfiguracją i może być użyta skutecznie tylko z kompilacją warunkową.
$WARNING (łańcuch)	Generuje do konsoli komunikat ostrzeżenie i zwiększa licznik ostrzeżeń. Opcja ta również jest przeznaczona do zarządzania konfiguracją.
$GEN	Wypisuje wywołania makr i makra rozszerzone. (Opcja domyślna!) Plik listingu pokazuje w pełni listę zagnieżdżonych wywołań makr. Kontrolka jest nadpisywana przez $NOLIST.
$NOGEN	Wypisuje tylko wywołania makr. Makra rozszerzone nie są wypisywane. Kontrolka jest nadpisywana przez $NOLIST.
$GENONLY	Wypisuje tylko ciało rozszerzonego makra. Wywołania makr i EXITM nie są wypisywane. Kontrolka jest nadpisywana przez $NOLIST.
$INCLUDE (plik)	Dołącza zewnętrzny plik źródłowy do pliku programu zaraz za instrukcją $INCLUDE. Jeśli plik do dołączenia nie został wyspecyfikowany w ścieżce absolutnej i nie może by odnaleziony w domyślnym katalogu, ścieżka wyspecyfikowana z opcją /INCLUDES (Linux: -i) w linii komend (jeśli jest obecna) jest przeszukiwana od lewej do prawej, i jeśli nie może być tam odnaleziona, ścieżka ustalona w zmiennych środowiskowych systemu ASEM51INC (jeśli jest zdefiniowana) jest przeszukiwana od lewej do prawej również. Pliki dołączane mogą być zagnieżdżone na dowolną głębokość.
$LIST	Wypisuje linie kodu źródłowego. (Opcja domyślna!)
$NOLIST	Nie wypisuj linii kodu źródłowego , pod warunkiem, że nie zawierają błędów, do chwili następnego wystąpienia wyrażenia $LIST.
$SAVE	Zachowuje aktualne nastawy $LIST/$GEN/$COND na stosie $SAVE. $SAVE może być zagnieżdżony na dowolną głębokość.
$RESTORE	Przywraca poprzednio zachowane nastawy $LIST/$GEN/$COND.
$TITLE (łańcuch)	Wstawia łańcuch z tytułem w nagłówku pliku listingu. Tytuły mogą być przycinane stosownie do ustalonej (lub domyślnej) szerokości strony. Domyślna wartość: ASEM-51 copyright information. Kontrolka nie ma ma efektu, gdy użyto $NOPAGING.

Przykłady:

```
$NOLIST                 ;wyłącza listing
$INCLUDE (8052.MCU) ;dołącza plik definicji rejestrów dla mikrokontrolera 8052
$LIST                   ;uruchamia listing
$TITLE (Computer-Controlled Combustion Unit for Motorcycles)
$EJ                     ;nowa strona z nowym tytułem
$ERROR (invalid coniguration: buffer size > external RAM size)
$WARNING (int. RAM doesn't meet minimum stack size requirements)
```

```
$SAVEGENONLY CONDONLY ; zapisuje stare nastawy $LIST/$GEN/$COND
                      ;i wypisuje tylko linie, które są rzeczywiście kompilowane.
$RESTORE              ;przywraca poprzedni tryb listingu
```

III.9Symbole predefiniowane

Dla łatwego dostępu do rejestrów SFR i przerwań mikrokontrolerów 8051, ASEM-51 posiada predefiniowane (wbudowane) symbole DATA, BIT i CODE.
Te predefiniowane symbole mogą byc wyłączone przy pomocy kontrolki $NOMOD51.
Dokładniejsze informacje o symbolach i adresach znajdują się w Załączniku C.
W celu identyfikacji kompilatora i jego wersji, następujące symbole są zdefiniowane:

```
??ASEM_51   =   8051H      ASEM-51
??VERSION   =   0130H      version 1.3
```

Te dwa symbole nie mogą być wyłączone!

III.10Kompilacja warunkowa

Kompilacja warunkowa pozwala na kompilację lub ignorowanie części kodu.
Pozwala to na trzymać kod dla kilku wariantów programu w jednym pliku źródłowym,
by ułatwić jego konfigurację i modernizację. Kompilacja warunkowa przydatna jest
do pisania makr z wyobraźnią.
Poniższe czternaście metainstrukcji zostało zaimplementowanych:

```
IF       <expr>          ELSEIF       <expr>
IFN      <expr>          ELSEIFN      <expr>
IFDEF    <symbol>        ELSEIFDEF    <symbol>
IFNDEF   <symbol>        ELSEIFNDEF   <symbol>
IFB      <literal>       ELSEIFB      <literal>
IFNB     <literal>       ELSEIFNB     <literal>
ENDIF                    ELSE
```

Meta instrukcje znajdują się w języku asemblera MCS-51 ale nie są jego częścią! Programiści znający C moga je porównać do komend preprocesora C. W dalszym tekście, IFxx użyty jako zbiorowa nazwa dla instrukcji IF/IFN/IFDEF/IFNDEF/IFB/IFNB. Analogicznie ELSEIFxx jest używany jako zbiorowa nazwa dla instrukcji ELSEIF/ELSEIFN/ELSEIFDEF/ELSEIFNDEF/ELSEIFB/ELSEIFNB (nie zawierających ELSE).

III.10.1Instrukcja IFxx

Prosta instrukcja IFxx ... ENDIF może być użyta do kompilacji zamkniętych wyrażeń, gdy dany warunek jest spełniony:

```
IFxx <warunek>
        <wyrażenie 1>
        <wyrażenie 2>
        .                ;kompilowane, jeśli <warunek> jest PRAWDZIWY
        .
        <wyrażenie n>
ENDIF
```

Wyrażenia od pierwszego do n-tego są kompilowane w przypadku, gdy <warunek> jest spełniony, w przeciwnym razie wyrażenia są ignorowane.

Jeśli możliwe powinno być wybranie dwóch wariantów kodu w różnych warunkach, można tego dokonać używając konstrukcji IFxx .. ELSE .. ENDIF. Jeśli <warunek> w wyrażeniu IFxx jest PRAWDZIWY, wyrażenia od pierwszego do n-tego są kompilowane, a wyrażenia od n+1 do n+msą ignorowane.

```
IFxx <warunek>
        <wyrażenie 1>
        .               ;kompilowane, gdy <warunek> jest PRAWDZIWY
        <wyrażenie n>
ELSE
        <wyrażenie n+1>
        .               ;kompilowane, gdy <warunek> jest NIEPRAWDZIWY
        <wyrażenie n+m>
ENDIF
```

W przypadku, gdy <warunek> nie jest spełniony, jest dokładnie odwrotnie. Oznacza to, że wyrażenia od pierwszego do n-tego są ignorowane a wyrażenia od n+1 do n+m podlegają kompilacji. Działa to również w przypadku, gdy pomiędzy pseudo instrukcjami IFxx lub ELSE nie ma żadnych wyrażeń.

Kiedy więcej niż dwie możliwości musza być rozważane, odpowiednia ilość wyrażeń ELSEIFxx może być zawarta między IFxx i ELSE. W przypadku takiej konstrukcji IFxx .. ELSEIFxx .. ELSE .. ENDIF, tylko wyrażenia w pierwszym spełnionym warunku podlegają kompilacji. Kod w pozostałych warunkach jest ignorowany.
Jeśli żaden warunek nie jest spełniony, tylko wyrażenia znajdujące się po ELSE (jeśli jakieś są) podlegają kompilacji.

```
IFxx <warunek 1>
        .               ;kompilowane, gdy <warunek 1> jest PRAWDZIWY
        .
ELSEIFxx <warunek 2>
        .               ;kompilowane, gdy <warunek 1> jest NIEPRAWDZIWY
        .               ;i <warunek 2> jest PRAWDZIWY
ELSEIFxx <warunek 3>
        .               ;kompilowane, gdy <warunek 1> jest NIEPRAWDZIWY
        .               ;i <warunek 2> jest NIEPRAWDZIWY
        .               ;i <warunek 3> jest PRAWDZIWY
        .
        .
ELSEIFxx <warunek n>
        .               ;kompilowane, gdy <warunek 1> jest PRAWDZIWY
        .               ;<warunek n-1> jest NIEPRAWDZIWY
        .               ;i <warunek n> jest PRAWDZIWY
ELSE
        .               ;kompilowane, gdy warunki od pierwszego
        .               ;do n-tego są NIEPRAWDZIWE
ENDIF
```

Konstrukcja IFxx ... ELSEIFxx ... ELSE ... ENDIF może być zagnieżdżona na dowolną głębokość.

Tryb listingu dla tych konstrukcji może być ustawiony przy pomocy kontrolek $COND, $NOCOND i $CONDONLY.

III.10.2Instrukcje IFxx i ELSEIFxx

Poszczególne instrukcje IFxx działają w następujący sposób:

IF <expr> Warunek IF jest PRAWDZIWY, jeśli wyrażenie <expr> nie jest równe 0.
Wartość <expr> musi być znana podczas pierwszego przejścia!

IFN <expr> Warunek IFN jest PRAWDZIWY, jeśli wyrażenie <expr> jest równe 0.
Wartość <expr> musi być znana podczas pierwszego przejścia!

IFDEF <symbol> Warunek IFDEF jest PRAWDZIWY, gdy <symbol> jest zdefiniowany w programie.
Dalsze operacje na <symbol> są zabronione!

IFNDEF <symbol> Warunek IFNDEF jest PRAWDZIWY, gdy <symbol> nie został zdefiniowany w programie.
Dalsze operacje na <symbol> są zabronione!

IFB <literal> Warunek IFB (jeśli puste) jest PRAWDZIWY, gdy <literal> jest pusty.
<literal> jest łańcuchem zamkniętym w nawiasach kątowych.

IFNB <literal> Warunek IFNB (jeśli nie puste) jest PRAWDZIWY, gdy <literal> nie jest pusty.
<literal> jest łańcuchem zamkniętym w nawiasach kątowych.

Pomimo tego, że wyrażenia IFB oraz IFNB obowiązują również poza makrami,
mogą być stosowane z sukcesem tylko wewnątrz ciał makr.
Zazwyczaj są one używane, by decydować czy argumenty makra pozostawiono puste, czy nie.

Powiązane instrukcje ELSEIFxx działają odpowiednio.

Przykład 1: Konstrukcja IF .. ELSE .. ENDIF

```
TARGET EQU 0          ;configuracja:    1 dla płyty aplikacyjnej
                      ;----------------  0 dla płyty rozwojowej
IF TARGET
        ORG 0         ;adres początku programu dla płyty aplikacyjnej
ELSE
        ORG 8000H     ;adres początku programu dla płyty rozwojowej
ENDIF
```

Aktualnie program jest skonfigurowany dla płyty rozwojowej.

Przykład 2: konstrukcja IFNDEF .. ELSE .. ENDIF

```
;EVA_537 EQU 0              ;symbol niezdefiniowany: płyta aplikacyjna 80C537
                            ;symbol zdefiniowany: płyta rozwojowa 80C537

IFNDEF EVA_537
```

```
        CLOCK EQU 16        ;częstotliwość zegara płyty aplikacyjnej
        CSEG AT 0           ;początek przestrzeni programu dla płyty aplikacyjnej
ELSE
        CLOCK EQU 12        ;częstotliwość zegara dla płyty rozwojowej
        CSEG AT 8000H       ;początek przestrzeni programu dla płyty rozwojowej
ENDIF
```

Aktualnie program jest skonfigurowany na płytę aplikacyjną.

Przykład 3: Konstrukcja IFB .. ELSE .. ENDIF

```
DECIDE MACRO X, Y
        IFB <X&Y>
                NOP
                NOP
        ELSE
                DB '&X,&Y'
        ENDIF
ENDM
```

Jeśli powyższe makro zostanie wywołane, jako

DECIDE Nonsense

parametr X zostanie zastąpiony przez “Nonsense” i parametr Y
przez zerowy łańcuch. Wtedy IFB staje się <Nonsense> i makro zostanie
skompilowane jako:

DB “Nonsense, '

Jeśli makro zostanie wywołane bez argumentów, jako

DECIDE

parametry X i Y zostaną zastąpione łańcuchami zerowymi i IFB przyjmie <>.
W tym przypadku makro zostanie przekompilowane jako:

```
NOP
NOP
```

Makra zostały dokładnie wyjaśnione w rozdziale “III.11 Makroprocesor”.

Przykład 4: Konstrukcja IFNDEF .. ELSEIF .. ELSEIF .. ELSE .. ENDIF

Symbol BAUDRATE służy do definiowania prędkości portu szeregowego UART:

```
IFDEF   BAUDRATE
LJMP    AUTOBAUD            ;automatyczna detekcja prędkości transmisji
ELSEIF BAUDRATE EQ 9600
        MOV TH1,#0FDH       ;9600 bodów
```

```
ELSEIF BAUDRATE EQ 1200
        MOV TH1, #0E8H          ;1200 bodów
ELSE
        $ERROR (baudrate not implemented)
ENDIF
```

Jeśli symbol BAUDRATE nie został zdefiniowany, wykonywany jest skok do etykiety AUTOBAUD.
Jeśli symbol BAUDRATE jest zdefiniowany z jedną z dopuszczonych wartości, 9600 lub 1200, timer 1 jest odpowiednio inicjalizowany.
Jeśli symbol BAUDRATE jest zdefiniowany z inną wartością.
Wygenerowany zostanie komunikat błędu zdefiniowany przez uzytkownika.

III.11Makroprocesor

Makra pozwalają łączyć podstawowe instrukcje asemblera w "super komendy".
W tym celu makra są zdefiniowane jako bloki kodu, które mogą być używane w programie, gdziekolwiek jest to potrzebne.
Jednakże zaawansowane makro używające parametrów, symboli lokalnych i makrooperatorów połączone z kompilacją warunkową, wybiega daleko poza podstawową funkcjonalność!
Używając tylko pięciu słów (MACRO, REPT, ENDM, EXITM, LOCAL) i kilku kontrolek, makroprocesor kompilatora ASEM-51 zapewnia różnorodne, użyteczne funkcje.
Te pięć meta instrukcji nie należy do języka asemblera mikrokontrolerów MCS-51, ale uzupełniają je jako znane już z kompilacji warunkowej. Występują dwa rodzaje makr: makra wywoływane i bloki powtarzalne.

III.11.1Proste wywoływane makra

Makra muszą najpierw zostać zdefiniowane zanim mogą zostać wywołane przez program.
Prosta definicja makra zawiera nazwę makra, która może zostać zdefiniowana przy użyciu słowa kluczowego MACRO, ciało makra i ostatecznie instrukcję ENDM (koniec makra).

```
<nazwa makra>  MACRO
<linia ciała makra nr 1>
<linia ciała makra nr 2>
        .
<linia ciała makra nr m>
ENDM
```

Nazwa makra musi być ważnym, unikalnym symbolem. Nie może być redefiniowana później.
Nazwami nie mogą być słowa kluczowe asemblera ani makroprocesora.
Ciało makra może składać się z dowolnej liczby linii. Ciałem makra może być każdy rodzaj instrukcji asemblera, pseudo instrukcji, kontrolki, meta instrukcji, wywołania makra i nawet dalszych definicji makra.
Ciało makra i cała definicja makra zakańczana jest instrukcją ENDM.
Makra muszą być zdefiniowane zanim zostaną wywołane. Odniesienia do makr umieszczonych za miejscem wywołania są zabronione. Raz zdefiniowane makro może być wywołane przez nazwę w dalszym kodzie programu dowolną ilość razy. Kiedykolwiek makro jest wywoływane, w miejsce wywołania wstawiane jest ciało makra, a następnie kompilowane jako zwykłe linie źródłowe. Ten proces nazywany jest ekspansją makra.

Przykład:

```
MY_FIRST MACRO          ;definicja
mov      A,#42
add      A,R5
ENDM

MY_FIRST                ;wywołanie
```

Po wywołaniu makra MY_FIRST, ciało makra

```
mov      A,#42
add      A,R5
```

zostanie wstawione w miejsce wywołania makra i skompilowane.

III.11.2Parametry makr

Makra wywoływane mogą posiadać parametry, co pozwala na większą elastyczność użycia.
Nazwy formalnych parametrów są specyfikowane w definicji makra za słowem kluczowym MACRO, rozdzielone przecinkami . Wszystkie nazwy parametrów makr muszą być różnymi, obowiązującymi symbolami. Słowa kluczowe nie mogą być nazwami parametrów. Makra mogą mieć dowolną liczbę parametrów tak długo, jak długo mieszczą się w linii. Nazwy parametrów są symbolami lokalnymi, które są znane tylko wewnątrz makra. Poza makrem nie mają one znaczenia!

```
<nazwa makra> MACRO <parametr 1>, <parametr 2>, ... , <parametr n>
<linia ciała makra nr 1>
<linia ciała makra nr 2>
        .
        .
<linia ciała makra nr m>
ENDM
```

Podczas wywołania aktualne parametry mogą być przekazane do makra. Argumenty muszą być rozdzielone przecinkami. Poprawnymi komendami makr są

1. sekwencje arbitralne drukowanych znaków, które nie zawierają spacji, znaków tabulacji, przecinków ani średników.
2. łańcuchy (z pojedynczym lub podwójnym cudzysłowem)
3. pojedyncze drukowane znaki poprzedzone przez '!' jako znak escape
4. sekwencje znaków zamknięte w nawiasach < ... >, które mogą być sekwencjami arbitralnymi prawidłowych argumentów makr (typy 1. - 4.), spacjami, przecinkami i średnikami
5. sekwencje arbitralne prawidłowych argumentów makr (typy 1. - 4.)
6. wyrażenia poprzedzone znakiem '%'

Uwaga: Słowa kluczowe MACRO, EQU, SET, CODE, DATA, IDATA, XDATA, BIT i znak ':' nie mogą być użyte jako pierwszy makro argument, ponieważ one zawsze rozpoczynają definicję symbolu! Z tego powodu muszą być zamknięte w nawiasach < ... >.

Podczas ekspansji makra, aktualne argumenty zamieniają symbole odpowiednich formalnych parametrów, gdziekolwiek są one rozpoznane w ciele makra. Pierwszy argument zamienia symbol pierwszego parametru, drugi argument zamienia symbol drugiego parametru, itd. Nazywa się to zastępstwem.
Bez specjalnego powiązania, kompilator nie rozpozna symbolu parametru, jeśli ten

- jest częścią innego symbolu
- jest zawarty w łańcuchach w cudzysłowach
- pojawia się w komentarzu.

Przykład nr 1:

```
MY_SECOND MACRO CONSTATN, REGISTER
MOV     A,#CONSTANT
ADD     A,REGISTER
ENDM

MY_SECOND 42,R5
```

Po wywołaniu makra MY_SECOND, linie ciała

```
MOV     A,#42
ADD     A,R5
```

są wstawiane do programu i kompilowane.
Parametry CONSTANT oraz REGISTER są zamieniane przez argumenty makra “42” i “R5”.

Liczba argumentów przekazanych do makra może być mniejsza (ale nie większa)
niż liczba formalnych parametrów makra. Jeśli argument zostanie pominięty, właściwy mu parametr jest zamieniany przez łańcuch pusty. Jeśli inne argumenty, niż argumenty końcowe mają być pominięte, mogą być one reprezentowane przez przecinki.

Przykład nr 2:
Makro OPTIONAL posiada osiem parametrów formalnych:

```
OPTIONAL MACRO P1, P2, P3, P4, P5, P6, P7, P8
        .
        .
<ciało makra>
        .
        .
ENDM
```

Jeśli zostanie wywołane w następujący sposób,

```
OPTIONAL 1, 2, , ,5, 6
```

parametry P1, P2, P5 i P6 są zamieniane przez argumenty 1, 2, 5 oraz 6 podczas zastępowania.
Parametry P3, P4, P7 i P8 są zamieniane przez łańcuchy puste.

Do bardziej uniwersalnego projektowania makra, musi istnieć możliwość rozpoznawania pustych argumentów makra i rozgałęziania ich zgodnie z tymi argumentami. Może to być przeprowadzone przy pomocy kompilacji warunkowej, używając meta instrukcji IFB raz IFNB. (zobacz rozdział III.10.2 Instrukcje IFxx i ELSEIFxx).

III.11.3Makra powtarzalne

Makra powtarzalne nie mają nazwy makra i nie mogą być wywoływane wielokrotnie. Są one zawsze rozwinięte po ich definicji. Podczas ekspansji makr, ich linie zawarte w ich ciałach są wykonywane n krotnie (0 <= n <= 65535). Makra powtarzalne rozpoczynają się słowem kluczowym REPT, po którym występuje wyrażenie, które musi być znane podczas pierwszego przejścia. Analogicznie do makr wywoływanych, makra te posiadają ciało makra, które musi być zakończone instrukcją ENDM:

```
REPT <wyrażenie>
<linia ciała makra nr 1>
<linia ciała makra nr 2>
        .
        .
<linia ciała makra nr m>
ENDM
```

Wyrażenie określa, ile razy linie znajdujące się w ciele makra mają być wykonywane. Od czasu, gdy makra powtarzalne zaczynają się od słowa kluczowego REPT, zwie się je czasami „blokami REPT".

Przykład:

```
REPT 5
NOP
ENDM
```

Ten blok REPT zostanie zamieniony na pięć instrukcji NOP znajdujące się zaraz po definicji makra:

```
NOP
NOP
NOP
NOP
NOP
```

III.11.4Symbole lokalne

Symbolami lokalnymi są symbole, które są znane tylko wewnątrz ciała makra, a nie poza ciałem makra. Symbole, które są definiowane dla całego programu, są nazywane symbolami globalnymi dla lepszego ich zrozumienia.
My już jesteśmy zaznajomieni ze specjalnymi rodzajami symboli lokalnych: formalnymi parametrami makr. Pojawiają się one tylko podczas definicji makra. Od chwili, gdy są one zastępowane podczas ekspansji makra, nie mamy z nimi dalszych problemów. Ale co się stanie z symbolami, które są zdefiniowane w ciele makra?

Przykład 1: Poniższe proste makro ma na celu odczytać znak z portu UART 8051 i zapisać go w A:

```
        RECEIVE MACRO
UARTIN: JNB RI, UARTIN
        MOV A, SBUF
```

```
        CLR RI
        ENDM

        To zostanie wykonane tylko raz! Jeśli makro RECEIVE zostanie wywołane
        Kilkakrotnie, etykieta UARTIN będzie definiowana kilkakrotnie.
```

Można to rozwiązać po prostu deklarując lokalny symbol UARTIN.
W tym celu wyrażenie LOCAL zostało przedstawione. Po słowie kluczowym LOCAL, lista lokalnych symboli może zostać wyspecyfikowana, a symbole te należy rozdzielić przecinkami. Symbole te będą obowiązywać tylko wewnątrz makra, które zawiera wyrażenie LOCAL. LOCAL może być wstawione bezpośrednio po słowie MACRO lub REPT i musi poprzedzać pierwszą linię ciała makra. Mogą zawierać dowolną liczbę symboli lokalnych. Ciało makra może być poprzedzone przez lokalne wyrażenia arbitralne.
Lokalne symbole muszą być prawidłowymi symbolami, unikalnymi wewnątrz makra i z nazwami różnymi od formalnych parametrów (jeśli jakieś są). Słowa kluczowe nie mogą być używane jako lokalne nazwy. Jeśli lokalny symbol posiada taką samą nazwę, jak symbol globalny, lokalny zakres ma pierwszeństwo przed zastępstwem.
Kiedy makro jest rozszerzone, lokalne symbole są zawsze zastępowane: formalne parametry są zastępowane przez argumenty makra, i lokalne symbole, które zostały zadeklarowane w wyrażeniu LOCAL, są zastępowane przez unikalne, globalne nazwy, które kompilator generuje podczas każdej ekspansji. Mają one zawsze format ?? xxxx, gdzie xxxx jest unikalnym numerem symbolu.

Przykład 2: Po przeprojektowaniu naszego poprzedniego makra RECEIVE przy użyciu lokalnych symboli, wygląda ono następująco:

```
        RECEIVE MACRO
        LOCAL UARTIN
UARTIN: JNB RI,UARTIN
        MOV A,SBUF
        CLR RI
        ENDM

        Zaawansowane w sposób przedstawiony powyżej makro będzie działać prawidłowo,
        Tak często, jak to wymagane. Gdy wywoływane jest RECEIVE po raz pierwszy,
        Symbol lokalny UARTIN zostanie zastąpiony przez ??0000,

??0000: JNB RI,??0000
        MOV A,SBUF
        CLR RI

        Gdy jest wywołane drugi raz, UARTIN zostanie zastąpione przez ??0001, itd.:

??0001: JNB RI,??0001
        MOV A,SBUF
        CLR RI
```

Jednakże, wskazane jest, by nie definiować globalnych symboli w formacie ??xxxx, by uniknąć konfliktów z Zastępowaniem lokalnych symboli przez makra.

III.11.5Operatory Makr

Istnieje kilka specjalnych znaków kontrolnych, które są bardzo użyteczne w definicjach makr, wywołaniach i wyrażeniach:

;; Komentarz makr:
Normalnie, komentarze w liniach ciała są również zawarte w liniach rozszerzenia. Jeśli komentarz zaczyna się od ‘;;’, nie jest zachowywany podczas definiowania makra. Przez to komentarz nie zajmuje przestrzeni pamięci i pojawia się tylko w pliku listingu w definicji makra, ale nie ma go w rozszerzonych liniach..

! Operator dosłowny:
Jeśli znak escape ‘!’ poprzedza inny drukowalny znak w argumencie makra, kompilator jest zmuszony do traktowania znaku dosłownie. Oznacza to, że zostanie przekazany do makra, nawet jeśli jest to znak kontrolny, podczas gdy sam operator dosłowny jest usuwany.

< > Nawiasy dosłowne:
Jeśli argument makra ma zawierać znaki kontrolne lub znaki separacji, musi on być zamknięty w nawiasy dosłowne < ... >, by przekazać je do makra jako jeden argument łańcuchowy, podczas gdy nawiasy są kasowane. Nawiasy dosłowne mogą być zagnieżdżane na dowolną głębokość.

% Ocenianie:
Jeśli argument makra jest poprzedzony przez operator oceny ‘%’, jest on interpretowany jako równanie, które zostanie wykonane przed przekazaniem go do makra. Aktualny łańcuch argumentu nie będzie wyrażeniem, ale dziesiętną reprezentacją ASCII jego wartości. Wyrażenie musi by znane podczas pierwszego przejścia.

& Zastępowanie:
Znak ‘&’ oddziela nazwy parametrów (symboli lokalnych) od otaczającego je tekstu. Na zewnątrz łańcuchów w cudzysłowach oraz komentarzy, służy on tylko jako ogólny znak rozdzielający. Stosuje się to zawsze, gdy symbol lokalny bezpośrednio poprzedza lub występuje po kolejny łańcuch alfanumeryczny. Wewnątrz łańcucha w cudzysłowach oraz komentarzy symbol lokalny musi poprzedzać znak ‘&’, jeśli ma on zostać tam zastąpiony. Podczas ekspansji każdego makra, kompilator usuwa dokładnie jeden znak ‘&’ z każdej sekwencji znaków ‘&’. Pozwala to na przykład na definiowanie zagnieżdżonych makr wewnątrz ciała makra, które również używa operatora zastępowania ‘&’: zapisuje się po prostu ‘&&’!

Przykład 1:

Komentarz powinien być widoczny tylko w definicji makra LICENSE:

```
LICENSE MACRO
DB 'Copyright'    ;;legalny towar
ENDM
```

Po wywołaniu, ciało makra po ekspansji wygląda w następujący sposób w pliku listingu:

```
DB 'Copyright'
```

Przykład 2: SPECIAL !;

Przekazuje średnik do makra SPECIAL jako bezpośredni argument.
Można to wykonać również używając

SPECIAL <;>

Przykład 3: Makro CONST definiuje 16- bitową stałą w pamięci ROM:

```
CONST MACRO NUMB
DW NUMB
ENDM
```

Jeśli je wywołamy w sposób przedstawiony poniżej,

CONST 0815H+4711-42

Parametr NUMB zostanie zamieniony w następujący sposób:

DW 0815H+4711-42

Jeśli ten sam argument zostanie poprzedzony przez %,

CONST %0815H+4711-42

Wynikiem zastąpienia będzie:

DW 6738

Przykład 4: Podczas zastępowania, oba argumenty makra CONCAT powinny stworzyć nazwę symbolu:

```
CONCAT MACRO NAM, NUM
       MOV R3,#0
NAM&NUM: DJNZ R3,NAM&NUM
       ENDM
```

Gdy CONCAT wywołamy tak

CONCAT LABEL, 08

Parametry NAM i NUM są zastępowane podczas ekspansji makra w następujący sposób:

```
       MOV R3,#0
LABEL08: DJNZ R3, LABEL08
```

III.11.6 Przedwczesny koniec ekspansji makra.

Czasem przydatne jest, by ekspansja makra mogła zostać przerwana zanim osiągnięty zostanie koniec ciała makra. Można to wymusić przy pomocy instrukcji EXITM (exit macro- czyli wyjście z makra).

Jednakże ma to sens tylko w połączeniu z kompilacją warunkową.

Przykład:
```
FLEXIBLE MACRO QANTITY
        DB 'Text'
IF QUANTITY LE 255
EXITM
ENDIF
DW QUANTITY
ENDM
```

Makro FLEXIBLE zawsze musi wstawić łańcuch 'Text' do przestrzeni CODE. Potem powinno wstawić 16-bitową stałą, ale tylko wtedy, gdy wartość numeryczna parametru QUANTITY jest większa niż 255.
W przeciwnym razie ekspansja makra powinna zostać przerwana wyrażeniem EXITM. Jeśli makro wywołamy w następujący sposób,

```
FLEXIBLE 42
```

Zostanie ono zamienione na

```
DB 'Text'
```

w trybie listingu $GENONLY/$CONDONLY.
Jednakże jeśli zostanie wywołane tak,

```
FLEXIBLE 4711
```

zostanie zamienione na:

```
DB 'Text'
DW 4711
```

Gdy ekspansja makra jest przerywana przy użyciu EXITM, wszystkie konstrukcje IFxx, które zostały otwarte wewnątrz ciała makra do tej pory, są zamykane.
Oczywiście ciała makr mogą także zawierać wyrażenia kontrolne. Jeśli plik źródłowy jest dołączony w ciele makra przy użyciu kontrolki $INCLUDE i plik ten lub zagnieżdżony plik zawiera instrukcję EXITM, wszystkie pliki dołączone do poprzedniego poziomu makra są zamykane na tym poziomie, i ekspansja makra na tym poziomie jest natychmiast przerywana.

III.11.7Wywołania zagnieżdżonych i powtarzanych makr.

Ciała makr mogą również zawierać wywołania makr, i ciała tych wywoływanych makr mogą zawierać wywołania makr, itd.
Jeśli wywołanie makra jest widziane poza ekspansją makra, kompilator natychmiast rozpoczyna pracę z ekspansją tego makra. W tym celu linie ciała po ekspansji są wstawiane w miejsce wywołania makra w ciele makra zawierającego to makro, i tak dzieje się tak długo, aż makro całkowicie podlegnie ekspansji. Wtedy ekspansja makra wywołującego kontynuowana jest z ciałem makra zagnieżdżonego, które jest wewnątrz tego makra.

Przykład 1:

```
INSIDE MACRO
SUBB A,R3
ENDM

OUTSIDE MACRO
MOV A,#42
INSIDE
MOV R7,A
ENDM
```

Wewnątrz ciała makra OUTSIDE, wywoływane jest makro INSIDE. Jeśli OUTSIDE jest wywołane (i tryb listingu jest ustawiony na $GENONLY), otrzymujemy coś takiego:

```
Line  I   Addr  Code   Source
15+   1   0000  74 2A  MOV A,#42
17+   2   0002  9B     SUBB A,R3
18+   1   0003  FF     MOV R7,A
```

Od czasu, gdy wywołania makr mogą być zagnieżdżane do dowolnej głębokości (do czasu, gdy dysponujemy wolną pamięć), poziom ekspansji makra jest pokazany w kolumnie I pliku listingu.
Od czasu, gdy makra i pliki dołączane mogą być zagnieżdżane w sekwencjach arbitralnych na dowolnej głębokości, poziom zagnieżdżania jest liczony przez wszystkie poziomy makr i plików. Dla lepszej przejrzystości, znak znajdujący się za globalnym numerem linii to ‘:’ dla plików dowiązanych oraz ‘+’ dla makr.

Jeśli makra wywołują siebie same, zwie się je makrami rekurencyjnymi. W takim przypadku muszą występować kryteria zatrzymania, by uniknąć ciągłego wywoływania makra przez siebie do czasu, gdy kompilator zgłosi brak pamięci! Tu wyjściem jest kolejny raz kompilacja warunkowa:

Przykład 2: Makro COUNTDOWN definiuje stałą 16-bitową od 1 do n w pamięci ROM. n może być przekazane do makra jako parametr:

```
COUNTDOWN MACRO DEPTH
IF DEPTH GT 0
DW DEPTH
COUNTDOWN %DEPTH-1
ENDIF
ENDM
```

Jeśli COUNTDOWN jest wywoływany tak

```
COUNTDOWN 7
```

wynik jest następujący (w trybie listingu $GENONLY/$CONDONLY):

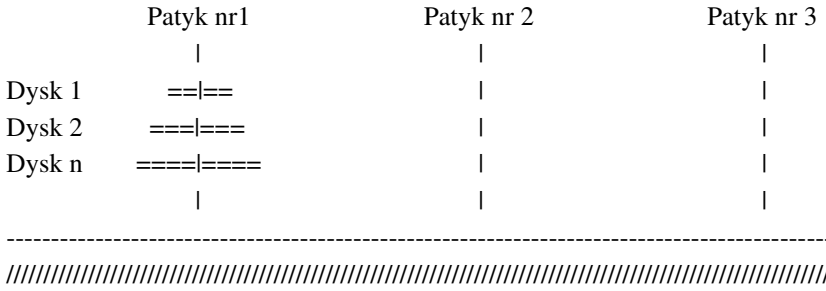
```
Line  I   Addr  Code   Source
16+   1   0000  00 07  DW 7
19+   2   0002  00 06  DW 6
22+   3   0004  00 05  DW 5
```

```
25+     4       0006    00 04           DW 4
28+     5       0008    00 03           DW 3
31+     6       000A    00 02           DW 2
34+     7       000C    00 01           DW 1
```

Po ciemnych wiekach , gdy osiadł kurz i słońce przebiło się przez mrok, nauka komputerowa odkryła metodę programowania rekurencyjnego. Nie było wątpliwości, że to wyło ROZWIĄZANIE!
Wtedy naukowcy komputerowi zaczęli wyjaśniać to studentom. Ale wydawało się, że studenci nie rozumieli tego. Zawsze wyjaśniali, że obliczanie rekurencyjne n! jest głupim przykładem. Wszyscy naukowcy poczuli, że wciąż im czegoś brakowało. Po kolejnych 10 latach trudnych poszukiwań, znaleźli również PROBLEM:

Przykład 3: Wieże Hanoi

Są trzy pionowe kijki na stole. Na kijku 1 jest n dysków o różnej średnicy z otworem w środku, najmniejszy dysk na górze, największy na dole.



Problem polega na przeniesieniu wieży dysków z kijka nr 1 na kijek nr 2 przy pomocy najmniejszej liczby ruchów. Tylko najwyżej położony dysk może być ruszony za jednym razem i żaden dysk nie może leżeć na mniejszym dysku. Kijek trzeci może zostać użyty jako początkowy. To jest ROZWIĄZANIE z użyciem makr w ASEM-51:

```
;Wieże Hanoi

$GENONLY CONDONLY

DISCS EQU 3             ;ilość dysków

HANOI MACRO n, SOURCE, DESTINATION, SCRATCH
        IF n > 0
          HANOI %(n-1), SOURCE, SCRATCH, DESTINATOR
          ;rusz najwyżej położony dysk z kijka &SOURCE na kijek &DESTINATION
          HANOI %(n-1), SCRATCH, DESTINATION, SOURCE
        ENDIF
 ENDM

HANOI DISCS, 1, 2, 3

END
```

Makro rekurencyjne HANOI generuje instrukcję dla PROBLEMU, gdzie instrukcje pojawiają się jako linie komentarza w liniach pliku listingu. Symbol DISCS musi być ustawiony do żądanej liczby dysków. Jeśli HANOI jest wywołane następująco:

HANOI 3, 1, 2, 3

następująca “instrukcja” jest generowana:

```
27+     3       ;       rusz najwyżej położony dysk z kijka 1 na kijek 2
35+     2       ;       rusz najwyżej położony dysk z kijka 1 na kijek 3
44+     3       ;       rusz najwyżej położony dysk z kijka 2 na kijek 3
53+     1       ;       rusz najwyżej położony dysk z kijka 1 na kijek 2
64+     3       ;       rusz najwyżej położony dysk z kijka 3 na kijek 1
72+     2       ;       rusz najwyżej położony dysk z kijka 3 na kijek 2
81+     3       ;       rusz najwyżej położony dysk z kijka 1 na kijek 2
```

Kontrolki GENONLY i CONDONLY zapewniają, że tabela nie będzie zawierać całego wywołania makra i konstrukcji IF.

Ćwiczenie 1:

Zmodyfikuj makro HANOI tak, by generowało tabelę ruchów w pamięci ROM, co mogło by być używane jako wejście dla ramienia robota kontrolowanego przez 8051, które rzeczywiście gra w grę z 3 rzeczywistymi kijkami i realnymi n dyskami.

Ćwiczenie 2:

Udowodnij, że minimalna ilość ruchów to $2^n - 1$. ;-)

III.11.8Definicje makr zagnieżdżonych

Ciała makra mogą także zawierać dalsze makrodefinicje. Jednakże takie zagnieżdżone makrodefinicje nie są prawidłowe dopóki makro, które je zawiera nie ulegnie ekspansji. Oznacza to, że makro musi zostać wywołane zanim możliwe będzie wywołanie zagnieżdżonego makra.

Przykład 1: Makro, które może zostać użyte do definiowania makr z nazwami arbitralnymi, może wyglądać w następujący sposób:

```
DEFINE MACRO MACNAME
MACNAME MACRO
DB 'I am the macro &MACNAME.'
ENDM
ENDM
```

By nie przeładować przykładu przez „jak to się robi”, :-)
makro zagnieżdżone przedstawia się tylko delikatnie
z pomocą odpowiedniego znaku w pamięci ROM. Wywołanie

```
DEFINE Obiwan
```

zdefiniuje makro

```
Obiwan MACRO
DB 'I am the macro Obiwan.'
ENDM
```

a wywołanie

```
DEFINE Skywalker
```

zdefiniuje makro:

```
Skywalker MACRO
DB 'I am the macro Skywalker.'
ENDM
```

Przykład 2: Poniższe makro ma wstawiać w od zmienną ilość instrukcji NOP. W tym celu makro z zagnieżdżonymi blokami REPT wydaje się najlepszym rozwiązaniem:

```
REPEAT MACRO NOPS
REPT NOPS
NOP
ENDM
ENDM
```

Wywołanie

```
REPEAT 4
```

w rezultacie stworzy coś takiego:

```
Line    I       Addr    Code            Source
9+      1               N       0004    REPT 4
10+     1                               NOP
11+     1                               ENDM
12+     2       0000    00              NOP
13+     2       0001    00              NOP
14+     2       0002    00              NOP
15+     2       0003    00              NOP
```

III.11.9Reprezentacja w pliku "List File"

Czasami ekspansja makra ma tendencję do generowania o wiele więcej linii listingu niż kodu. Listować czy nie listować- oto jest pytanie! Wymagania by osiągnąć lepszy pogląd na całość lub mieć więcej detali mogą być różne w różnych fazach tworzenia programu lub w innych sekcjach programu. By zawsze mieć najlepsze rezultaty, kilka generalnych kontrolek zostało wprowadzonych, które oddziałują na reprezentację ekspansji makra oraz konstrukcji IFxx w pliku listingu (zobacz rozdział „III.8 Kontrolki Asemblera”):

Kontrolka	Typ	Domyślna	Skrót	Znaczenie
$GEN	G	$GEN	$GE	listowanie wywołań makr i linii ekspansji
$NOGEN	G		$NOGE	listowanie tylko wywołań makr
$GENONLY	G		$GO	listowanie tylko linii ekspansji
$COND	G	$COND	---	listowanie pełnej konstrukcji IFxx.. ENDIF
$NOCOND	G		---	nie listuj linii dla nieprawdziwych warunków
$CONDONLY	G		---	listuj tylko linie kompilowane
$SAVE	G		$SA	zachowuje aktualny stan $LIST/$GEN/$COND
$RESTORE	G		$RS	przywraca poprzedni stan $LIST/$GEN/$COND

IV Kompatybilność z asemblerem firmy Intel

Wraz z kompilatorem ASM51 intel zdefiniował i zaimplementował język asemblera dla mikrokontrolerów rodziny MCS-51, który zawsze był jedynym prawdziwym standardem w świecie 8051.
Niestety Intel ogłosił “koniec życia” ASM51 (wersja finalna 2.3) wszystkich pozostałych narzędzi programistycznych do końca 1993 roku.
Asembler ASEM-51 jest podzbiorem standardu Intel, który gwarantuje maksymalną kompatybilność z istniejącymi plikami źródłowymi 8051. Posiada on implementację wszystkich instrukcji mnemonicznych 8051 podobnie, jak bogaty i użyteczny zestaw pseudo instrukcji i kontrolek kompilatora Intel.

IV.1 Obostrzenia

Od czasu, gdy ASEM-51 generuje pliki w formacie Intel-HEX (lub absolutnym OMF-51) zamiast relokowanych modułów, cały kod dla 8051 musi znajdować się w jednym pojedynczym pliku. W konsekwencji wszystkie pseudo instrukcje, które współdziałają z relokowanymi segmentami, zewnętrznymi lub publicznymi symbolami, nie zostały zaimplementowane:

PUBLIC
EXTRN
SEGMENT
RSEG

Intelowski styl makr nie jest wspierany! (Więc znak '%' może być używany w komentarzach.)
Do teraz tylko następujące kontrolki asemblera i ich skróty zostały zaimplementowane:

	kontrolki główne	skrót.	kontrolki ogólne	skrót
	$DATE (<string>)	$DA	$EJECT	$EJ
	$DEBUG	$DB	$GEN	$GE
	$NODEBUG	$NODB	$NOGEN	$NOGE
	$MACRO (<PERCENT>)	$MR	$GENONLY	$GO
Kontlol-	$NOMACRO	$NOMR	$INCLUDE (<file>)	$IC
ki	$MOD51	$MO	$LIST	$LI
	$NOMOD51	$NOMO	$NOLIST	$NOLI
Intel	$PAGING	$PI	$SAVE	$SA
	$NOPAGING	$NOPI	$RESTORE	$RS
	$SYMBOLS	$SB	$TITLE	$TT
	$NOSYMBOLS	$NOSB		
	$PAGELENGHT (<lines>)	$PL		

	kontrolki główne	skrót.	kontrolki ogólne	skrót
	$PAGEWIDTH (<columns>) $PW			
	$NOBUILTIN	-----	$COND	-----
Kontrol-	$NOTABS	-----	$NOCOND	-----
ki	$PHILIPS	-----	$CONDONLY	-----
ASEM-			$ERROR (<string>)	-----
51			$WARNING (<string>)	-----

IV.2 Wyrażenia

Kontrolki asemblera nie muszą się zaczynać w kolumnie nr 1, ale mogą być poprzedzone przez dowolną ilość spacji lub znaków tabulacji. Kontrolki główne mogą być również poprzedzone przez linie komentarza i wyrażenia $INCLUDE, dołączone pliki include zawierają tylko kolejne wyrażenia kontrolne oraz komentarze. Plik źródłowy może zawierać puste linie i linie komentarza za wyrażeniem END. Łańcuchy znaków mogą być również zawarte w podwójnych cudzysłowach.
Symbol DATA dla rejestru funkcji specjalnych PCON jest predefiniowany.
Operator bitu '.' jest legalny we wszystkich wyrażeniach, nie tylko w tych, które pasują do segmentu typu BIT.

ASEM-51 przedstawia zestaw meta instrukcji, które uzupełniają instrukcje języka MCS-51 firmy Intel, alenie są jego cześcią.
Meta instrukcje IFxx, ELSEIFxx, ELSE oraz ENDIF pozwalają na kompilację warunkową, do czasu, gdy wystąpią meta instrukcje MACRO, REPT, ENDM, EXITM oraz LOCAL (i niektóre znaki kontrolne) z potężnego języka makr.
Po szczegółowe informacje dotyczące meta instrukcji zobacz do rozdziału “III.10 Kompilacja warunkowa” oraz “III.11 Makroprocesor”.

IV.3 Dalsze różnice

By uczynić semantykę unikalną, szczególnie w odniesieniu do stosowania operatorów ogólnych w wyrażeniach jest inna. Dalej, wyrażenia z operacjami bitowymi “.” przechodzą jako wynik do typu BIT, a nie do NUMBER.
Typ segmentów, które są definiowane przy użyciu EQU lub SET zawsze rozwijane są do NUMBER.
W przeciwnym razie może być trudne, w niektórych wypadkach, wymuszenie definicji symbolu bez typu. Jest to opisane w szczegółach w rozdziałach “III.4 “ oraz w “III.7”.
Poza instrukcją DB, łańcuch stałej zerowej długości ' ' jest niedozwolony.
Kontrolka $NOMOD51 blokuje także predefiniowane adresy w przestrzeni CODE.
Symbole specjalne AR0...AR7 są predefiniowane dla banku 0 zanim pierwszy raz pojawi się wyrażenie USING.

V Format pliku "List File"

Plik listy (listingu) kompilatora ASEM-51 został zaprojektowany tak, by dostarczyć użytkownikowi tak wiele informacji na temat generowanego kodu, jak to możliwe.
Poza listą kodu źródłowego, występuje pięć podstawowych struktur w listingu:

- nagłówek strony
- nagłówek pliku
- nagłówki linii
- diagnostyka błędu

- tabela symboli lub powiązań krzyżowych

Normalnie, każda strona listingu zaczyna się od nagłówka strony w formacie:

ASEM-51 V1.3 Copyright (c) 2002 by W.W. Heinz PAGE 1

Identyfikuje on kompilator, zawiera informację o prawach autorskich i pokazuje informację o aktualnym numerze strony w prawym górnym marginesie. Po nagłówku strony, linie źródłowe są wypisywane w formacie listy. Gdy osiągnięta zostaje maksymalna ilość linii na stronę, następny nagłówek strony jest wstawiany po znaku nowej strony.
Jeśli drukarka nie obsługuje znaków nowej strony, nagłówek może zostać zablokowany przy pomocy kontrolki $NOPAGING. Ilość linii na stronę może zostać dostosowana do formatu papieru przy pomocy kontrolki $PAGELENGHT. Szerokość nagłówka (i wszyskich innych linii) może być ustawiona przy pomocy kontrolki $PAGEWIDTH.

Nagłówek pliku pojawia się tylko na pierwszej stronie. Identyfikuje on kompilator, wypisuje listę wszystkich plików wejściowych i wyjściowych i oznacza kolumny dla nagłówków linii. Typowy nagłówek pliku wygląda w następujący sposób:

```
MCS-51 Family Macro Assembler A S E M – 5 1  V 1.3
=========================================

Source File:    demo.a51
Object File:    demo.hex
List File:      demo.lst


Line  I    Addr  Code                    Source
```

Bezpośrednio po nagłówku pliku następuje lista linii kodu źródłowego.
Każda linia kodu źródłowego jest poprzedzona nagłówkiem linii. Nagłówek linii składa się z czterech kolumn: numeru linii, pliku dołączanego lub poziomu makra, adresu linii i generowanego kodu.
Domyślnie nagłówki linii zawierają znaki tabulacji, by zaoszczędzić przestrzeń dyskową.
Jeśli drukarka lub przeglądarka plików nie obsługuje znaków tabulacji, można je zastąpić spacjami przy użyciu kontrolki $NOTABS.

Kolumna „Line” zawiera globalny numer linii. Nie jest konieczny lokalny numer linii w poszczególnych plikach źródłowych, ale globalny numer, który jest zliczany przez główny plik źródłowy, pliki dołączane i wszystkie linie ekspansji makr.
Od czasu, gdy pliki dołączane i makra mogą być zagnieżdżane arbitralnie, numer linii globalnej jest przerywany przez znak ‘:’ dla głównego pliku źródłowego, wszystkich poziomów plików dołączanych i przez znak ‘+’ dla poziomów ekspansji makr.

Kolumna „I” zawiera flagi z poziomem zagnieżdżenia plików dołączanych lub makr. W głównym pliku źródłowym, kolumna ta jest pusta. Pierwszy plik dołączony otrzymuje poziom 1. Jeśli w tym pliku dołączanym znajduje się kolejny plik dołączany, otrzymuje on poziom 2, itd. Zasada jest identyczna dla zagnieżdżania makr. Jeśli makro jest wywoływane z głównego pliku źródłowego, otrzymuje ono poziom 1. Jeśli makro wywołuje kolejne makro, wywoływane makro otrzymuje poziom 2, itd.
Pliki dołączane i makra mogą być zagnieżdżane w dowolnej sekwencji i do dowolnego poziomu!

W kolumnie „Addr” pokazany jest adres wylistowanej linii w aktualnie aktywnym segmencie pamięci (dla przestrzeni adresowej 8051). Wszystkie adresy są reprezentowane jako liczby szesnastkowe. Adresy w segmencie CODE i XDATA są czterocyfrowe. Adresy we wszystkich pozostałych segmentach są dwucyfrowe. Dla linii, które nie mogą być przypisane do poszczególnych segmentów, pole „Addr” jest puste.

Kolumna „Code” może zawierać do czterech bajtów generowanego kodu, który jest wystarczający dla wszystkich instrukcji mikrokontrolerów rodziny 8051. Kod jest listowany jako równoważny bajty szesnastkowe zaczynając od lewej strony kolumny „Code”.
Jednakże kod generowany dla instrukcji DB i DW może być dłuższy niż cztery bajty. W takich przypadkach linia źródłowa jest poprzedzana przez dodatkowe nagłówki linii do czasu, gdy cały kod linii jest wypisany.
Kolumna „Code” nie zawsze zawiera kod, który zajmuje przestrzeń w segmencie CODE mikrokontrolera 8051.
Inaczej niż w większości innych kompilatorów, lista ASEM-51 zawiera wynik równania wszystkich wyrażeń, które mogą się znajdować w pseudo instrukcjach lub kontrolkach. Wartości te są wypisywane jako reprezentacja szesnastkowa na prawej stronie kolumny „Code”. Typ segmentu tych wyrażeń posiada jeden znak jako flagę po lewej stronie kolumny „Code”:

C	CODE
D	DATA
I	IDATA
X	XDATA
B	BIT
N	liczba bez typu
R	rejestr

Kolumna „Source” zawiera w końcu oryginalny kod źródłowy linii.
Typowy kod źródłowy wygląda w następujący sposób:

```
Line  I  Addr  Code          Source

  1:                       ;Przykładowa lista programu File ;Demo
  2:                       ;--------------------------------
  3:                        $NOMOD51              ;bez 8051 SFR
  4:        N    004F       $PAGEWIDTH (79)       ;79 kolumn/linię
  5:                        $NOTABS               ;zamień tabulatory
  6:        N    90         P1    DATA 090H  ;adres portu 1
  7:        B    93         INPUT BIT  P1.3  ;wejście impulsowe    8:
  9:        N    8000       ORG  08000H     ;ustawienie licznika lokacji
 10:   8000  80 20               SJMP START      ;skok do adresu startu
 11:
 12:   8002  01 07           DB  1,7         ;definicja bajtów
 13:   8004  00 02 00 0C  DW  2,12,9      ;definicja słów
       8008  00 09
 14:   800A  63 6F 66 66  DB  'coffeeright (c) 2002',0    ;łańcuch
       800E  65 65 72 69
       8012  67 68 74 20
       8016  28 63 29 20
       801A  32 30 30 32
       801E  00
 15:   801F  N      0003     DS  3           ;definiuj przestrzeń
 16:
 17:   8022  75 30 00       START:  MOV COUNT,#0     ;zerowanie licznika
```

```
18:   8025  30 93 FD      LLEVEL: JNB INPUT,LLEVEL ;czekaj na wysoki
19:   8028  20 93 FD      HLEVEL: JB  INPUT,HLEVEL ;czekaj na niski
20:   802B  05 30          INC COUNT       ;licz impuls
21:   802D  80 F6          JMP LLEVEL      ;następny impuls
22:
23:       N       30       DSEG AT 030H    ;wewnętrzna RAM
24:    30  N       01       COUNT:  DS 1          ;zmienna licznika
25:
26:                       END
```

Jeśli w linii źródłowej wykryty zostanie błąd, jego pozycja jest oznaczana flagą ^ tak dokładnie, jak to tylko możliwe i wstawiana jest informacja o błędzie. Wygląda to tak, jak poniżej:

```
17:   8022  75 30 00      START:  MOV COUNT,#0    ;zerowanie licznika
18:   8025  30 93 FD      LLEVEL: JNB INPUT,LLEVEL ;czekaj na wysoki
19:   8028  20 93 00      HLEVEL: JB  INPUT,HLEUEL ;czekaj na niski
                                  ^
          @@@@@ symbol not defined @@@@@

20:   802B  05 30               INC COUNT       ;licz impuls
21:   802D  80 F6               JMP LLEVEL      ;następny impuls
```

Diagnostyka błędów na końcu programu wyświetla użyte banki rejestrów i sumaryczną ilość błędów wykrytych podczas kompilacji:

```
register banks used:  0, 1, 3

187 errors detected
```

Bank rejestrów wyliczony jest jako “used”, gdy program przełączył się do tego banku przy pomocy instrukcji USING lub gdy jeden ze symboli specjalnych AR0... AR7 został użyty, gdy bank był aktywny. Komunikat

```
register banks used: - - -
```

oznacza, że żaden z banków nie został użyty bezpośrednio i kod programu może, ale nie musi być niezależny od banków rejestrów.

Po listowaniu kodu źródłowego i diagnostyce błędów, zaczyna się tabela symboli lub lista odwołań krzyżowych. Domyślnie generowana jest tabela symboli. Tabela symboli zawiera wszystkie symbole zawarte w programie według kolejności alfabetycznej wraz z ich nazwami, typami segmentów, wartościami szesnastkowymi oraz pierwszymi liniami definicji.
Predefiniowane symbole są wypisane bez numeru linii definicji.
Tabela symboli może zostać zawieszona przy pomocy kontrolki $NOSYMBOLS.
Typowa tabela symboli wygląda w sposób następujący:

L I S T O F S Y M B O L S
==========

SYMBOL	TYPE	VALUE	LINE
AKKUM	REGISTER	A	38
COUNT	DATA	30	47
HLEVEL	CODE	802E	35
INPUT	BIT	93	12
LLEVEL	CODE	802B	34
MY_PROGRAM	MODULE		14
P1	DATA	90	
QUANT	NUMBER	0013	22
RECEIVE	MACRO		5
SP	DATA	81	
STACK	IDATA	80	17
START	CODE	8022	31
VOLTDC	XDATA	D785	50

Jeśli użyta zostanie kontrolka $XREF, generowana jest lista odwołań krzyżowych zamiast tabeli symboli. Lista odwołań krzyżowych dla symboli z poprzedniej tabeli wygląda następująco:

C R O S S - R E F E R E N C E - L I S T I N G
==========

SYMBOL	TYPE	VALUE	DEFINED	REFERENCED	
AKKUM	REGISTER	A	38	42	43
COUNT	DATA	30	47	32	40
				43	44
HLEVEL	CODE	802E	35	35	
INPUT	BIT	93	12	34	35
LLEVEL	CODE	802B	34	34	41
MY_PROGRAM	MODULE		14		
P1	DATA	90	12		
QUANT	NUMBER	0007	22	44	
	NUMBER	0013	37		
RECEIVE	MACRO		5		
SP	DATA	81	31		
STACK	IDATA	80		17	31
START	CODE	8022	31	24	
TRASH	undef.	----	42		
VOLTDC	XDATA	D785	50	33	

Zawiera ona wszystkie symbole programu w kolejności alfabetycznej z nazwami symboli, wszystkimi definicjami, włącznie z liniami definicji, typami segmentów i wartościami numerycznymi. Dalej, wszystkie referencje symboliczne są wypisane również. Kolumna SYMBOL zawiera nazwę symbolu, gdy kolumny TYPE,

VALUE i DEFINED mogą zawierać typy segmentów, wartości numeryczne, linie definicji jednego, więcej lub żadnej definicji.
Symbole rejestrów mają symbole “REGISTER”, nazwy modułów mają typ “MODULE”, nazwy makr mają typ “MACRO” i symbole, które zostały odniesione do innych ale nie zdefiniowane, mają flagi “undef.” w kolumnie TYPE.
Rozpoczynając od kolumny REFERENCED do prawego marginesu, jest kilka kolumn (zależnie od szerokości strony), zawierających wszystkie numery linii odwołań (jeśli jakieś są).
Listing odwołań krzyżowych nie różni się tam, gdzie definicje lub odwołania do szczególnych symboli są legalne lub nie. W tym celu odwołuje się do komunikatów błędów w listingu źródła.

VI. Wspierane mikrokontrolery rodziny 8051

Dzisiaj spora ilość układów pochodzenia 8051 jest dostępna i liczba ta wzrasta z miesiąca na miesiąc! Wszystkie one używają tego samego zestawu instrukcji rdzenia MCS-51, ale są one wyposażone w inne bloki funkcjonalne, by pokryć szeroki zakres zapotrzebowań. Różnica w języku programowania polega głównie na różnicach w bloku rejestrów specjalnego przeznaczenia i adresach przerwań. Dobrą praktyką jest używanie zawsze tych samych nazw rejestrów SFR, które zdefiniował producent mikrokontrolera. W tym celu dołączono do kompilatora pliki definicji procesorów *.MCU. Są to pliki dołączane z definicjami rejestrów różnych układów 8051. Jednakże predefiniowane symbole w ASEM-51 muszą zostać wyłączone, by umożliwić dołączenie pliku z definicjami rejestrów SFR innych producentów w sposób pokazany poniżej:

```
$NOMOD51
$INCLUDE (80C515.MCU)
```

Taki zapis spowoduje wyłączenie predefiniowanych symboli rejestrów 8051 i użycie definicji rejestrów mikrokontrolera 80C515 lub 80C535.
Użytkownik może w łatwy sposób zaadoptować ASEM-51 do zupełnie nowego układu rodziny 8051! Wszystko, co musi w tym celu zrobić, to napisać odpowiadający mu plik z definicjami SFR stworzony na podstawie dokumentacji producenta układu.
Nazwa każdego pliku z definicjami nawiązuje do wersji ROM każdego układu. Oczywiście jest ona użyteczna w wersjach z pamięcią EPROM, EEPROM, flash oraz bez wbudowanej pamięci programu (jeśli taka jest) każdego układu. Plik 8051.MCU zawiera dokładnie predefiniowane symbole ASEM-51, ponieważ jego wewnętrzna tabela symboli została wygenerowana właśnie z tego pliku!
By przełączyć ASEM-51 do obsługi układów o zredukowanej liczbie instrukcji, np. Rodziny Philips 83C75x, należy użyć kontrolki $PHILIPS.

Aktualnie następujące definicje procesorów są dostarczane z ASEM-51:

Nazwa	Producent	Wersje
8051.MCU	Intel	8051, 8031, 8751BH
	(and others)	8051AH, 8031AH, 8751H, 8051AHP, 8751H-8
		80C51BH, 80C31BH, 87C51, 80C51BHP
	Atmel	89C51, 89LV51, 87LV51, 80F51, 87F51
8052.MCU	Intel	8052AH, 8032AH, 8752BH
	SIEMENS	80513, 8352-5
80C52.MCU	Intel	80C52, 80C32, 87C52,
		80C54, 87C54, 80C58, 87C58
83C51FX.MCU	Intel	83C51FA, 80C51FA, 87C51FA
		83C51FB, 87C51FB, 83C51FC, 87C51FC

83C51R.MCU	Intel	83C51RA, 80C51RA, 87C51RA, 83C51RB, 87C51RB, 83C51RC, 87C51RC
83C51KB.MCU	Intel	83C51KB
83C51GB.MCU	Intel	83C51GB, 80C51GB, 87C51GB
83C151.MCU	Intel	83C151SB, 87C151SB, 80C151SB 83C151SA, 87C151SA
83C152.MCU	Intel	80C152JA, 83C152JA, 80C152JB 80C152JC, 83C152JC, 80C152JD
83C452.MCU	Intel	83C452, 80C452
8044.MCU	Intel	8044AH, 8344AH, 8744AH
83931HA.MCU	Intel	83931HA, 80931HA
83931AA.MCU	Intel	83931AA, 80931AA
80512.MCU	SIEMENS	80512, 80532
80515.MCU	SIEMENS	80515, 80535, 80515K, 83515-4
80C515.MCU	SIEMENS	80C515, 80C535, 83C515H
83C515A.MCU	SIEMENS	83C515A-5, 80C515A
80C517.MCU	SIEMENS	80C517, 80C537
C501.MCU	Infineon	C501-1R, C501-L
C502.MCU	Infineon	C502-2R, C502-L
C503.MCU	Infineon	C503-1R, C503-L
C504.MCU	Infineon	C504-2R, C504-L
C505.MCU	Infineon	C505-2R, C505-L
C505C.MCU	Infineon	C505C-2R, C505C-L
C505A.MCU	Infineon	C505A-4E, C505A-L
C505CA.MCU	Infineon	C505CA-4E, C505CA-L
C505L.MCU	Infineon	C505L
C508.MCU	Infineon	C508-4R, C508-4E
C509.MCU	Infineon	C509-L
C511.MCU	Infineon	C511, C511A
C513.MCU	Infineon	C513, C513A, C513A-H
C513AO.MCU	Infineon	C513AO
C515.MCU	Infineon	C515-L, C515-1R
C515A.MCU	Infineon	C515A-L, C515A-4R
C515C.MCU	Infineon	C515C-8R
C517A.MCU	Infineon	C517A-L, C517A-4R, 83C517A-5, 80C517A
C540U.MCU	Infineon	C540U
C541U.MCU	Infineon	C541U
C868.MCU	Infineon	C868-1R, C868-1S
83C451.MCU	Philips	83C451, 80C451, 87C451
83C528.MCU	Philips	83C528, 80C528, 87C528, 83C524, 87C524 83CE528, 80CE528, 89CE528
83C550.MCU	Philips	83C550, 80C550, 87C550
83C552.MCU	Philips	83C552, 80C552, 87C552
83C562.MCU	Philips	83C562, 80C562
83C652.MCU	Philips	83C652, 80C652, 87C652 83C654, 87C654, 83CE654, 80CE654
83C750.MCU	Philips	83C750, 87C750
83C751.MCU	Philips	83C751, 87C751
83C752.MCU	Philips	83C752, 87C752
83C754.MCU	Philips	83C754, 87C754

83C851.MCU	Philips	83C851, 80C851
83C852.MCU	Philips	83C852
87LPC762.MCU	Philips	87LPC762
87LPC768.MCU	Philips	87LPC768
80C32X2.MCU	Philips	80C31X2, 80C32X2, 80C51X2, 80C52X2, 80C54X2, 80C58X2, 87C51X2, 87C52X2, 87C54X2, 87C58X2, 89C51X2, 89C52X2, 89C54X2, 89C58X2
80C521.MCU	AMD	80C521, 80C541, 87C521, 87C541, 80C321
80C324.MCU	AMD	80C324
83C154.MCU	OKI	83C154, 80C154, 85C154VS
83C154S.MCU	OKI	83C154S, 80C154S, 85C154HVS
80C310.MCU	DALLAS	80C310
80C320.MCU	DALLAS	80C320, 87C320, 80C323, 87C323
80C390.MCU	DALLAS	80C390
87C520.MCU	DALLAS	87C520, 83C520
87C530.MCU	DALLAS	87C530, 83C530
87C550.MCU	DALLAS	87C550
89C420.MCU	DALLAS	89C420
DS5000.MCU	DALLAS	5000FP, 5000, 5000T, 2250, 2250T
DS5001.MCU	DALLAS	5001FP, 5002FP, 5002FPM, 2251T, 2252T
MAX7651.MCU	Maxim	MAX7651, MAX7652
COM20051.MCU	SMC	COM20051
89C52.MCU	Atmel	89C52, 89C55, 89LV52, 89LV55, 87LV52, 80F52, 87F52
87F51RC.MCU	Atmel	87F51RC, 87F55, 87LV55
89C1051.MCU	Atmel	89C1051
89C2051.MCU	Atmel	89C2051, 89C4051, 89C1051U
89S8252.MCU	Atmel	89S8252, 89LS8252
89S51.MCU	Atmel	89S51
89S52.MCU	Atmel	89S52, 89LS52
89S53.MCU	Atmel	89S53, 89LS53
89S4D12.MCU	Atmel	89S4D12
73M2910.MCU	TDK	73M2910, 73M2910A
AN2131.MCU	Cypress	AN2121, AN2122, AN2125, AN2126, AN2131, AN2135, AN2136

Dodatek A:

Komunikaty błędów kompilatora ASEM-51

A.1 Błędy kompilacji:

Błędy kompilacji powiązane są z konsystencją programu w asemblerze w jego składni i semantyce. Jeśli jeden z tych błędów zostanie wykryty, jest on flagowany w pliku listingu i kompilator kontynuuje pracę. Gdy kompilacja dobiega końca, ASEM kończy działanie zwracając kod wyjścia 1:

Komunikat Błędu	Znaczenie
address below segment base	Wartość licznika alokacji aktualnego segmentu wskazuje poniżej adresu bazowego aktualnego segmentu.
address out of range	Adres skoku lub wywołania instrukcji nie może zostać osiągnięty przy pomocy wybranego trybu adresowania.
already a macro parameter	W makro definicji symbol lokalny jest równy z wcześniej zdefiniowaną nazwą parametru.
argument exceeds end of line	Makro argument zawiera więcej nawiasów otwierających, niż zamykających.
attempt to divide by zero	Podczas rozwiązywania wyrażenia, kompilator próbuje wykonać dzielenie przez zero.
binary operator expected	W tym miejscu wyrażenia, tylko operator binarny jest dozwolony.
comma expected	Na oznaczonej pozycji powinien być znak ';'.
commands after END statement	Za wyrażeniem END znajdują się dalsze instrukcje asemblera.
constant out of range	Wartość liczbowa jest większa niż 65535.
duplicate local symbol	W makro definicji symbol lokalny został zdefiniowany kilka razy lub jest taki sam, jak poprzednio zdefiniowany parametr.
duplicate parameter name	Nie wszystkie nazwy parametrów makra są różne.
ENDIF statement expected	Występują konstrukcje IFxx, które nie zostały zakończone przy pomocy instrukcji ENDIF.
ENDM statement expected	Występują makro definicje, które nie zostały zakończone instrukcją ENDM.
expression out of range	Wynik wyrażenia jest zbyt duży lub zbyt mały dla tego celu.
file name expected	W tym miejscu powinna znajdować się prawidłowa nazwa pliku.

forward reference to macro	Makro zostało wywołane zanim zostało zdefiniowane.
forward reference to register	Symbol rejestru został użyty, zanim został zdefiniowany przy pomocy EQU lub SET.
illegal character	Wyrażenie zawiera znaki, na które język asemblera MCS-51 nie zezwala.
illegal constant	Wystąpił błąd składni w stałej numerycznej.
illegal control statement	Wyrażenie zaczyna się przy użyciu nieznanego słowa kluczowego ze znakiem $.
illegal operand	W tym miejscu wyrażenia oczekiwany jest prawidłowy dla tej funkcji operand.
illegal statement syntax	Wyrażenie zawiera element składni, który nie jest dozwolony w tym kontekście.
invalid base address	Adres segmentu DATA, który nie jest adresowany bitowo został użyty polewej stronie operatora '.'.
invalid bit number	Liczba większa niż 7 została użyta po prawej stronie operatora '.'.
invalid instruction	Instrukcja została poprzednio zablokowana przy użyciu kontrolki $PHILIPS.
macro type operand	Symbol makra został użyty jako operand w wyrażeniu numerycznym.
maximum line length exceeded	Podczas ekspansji makra, zamiana parametrów i/lub lokalnych symboli osiąga długość linii większą, niż dopuszczalne 255 znaków.
misplaced LOCAL instruction	W makro definicji instrukcja LOCAL poprzedzona jest przez linie ciała makra.
misplaced macro instruction	Makroinstrukcja została użyta poza definicją makra lub błędnie wstawiona w strukturze programu.
misplaced macro operator	Makro operator (<, >, !, %, &) został użyty na błędnej pozycji.
module name already defined	W programie jest więcej niż jedno wyrażenie NAME.

must be known on first pass	Wynik wyrażenia musi zostać wyznaczony podczas pierwszego przebiegu kompilatora.
must be preceded by $SAVE	Kontrolka $RESTORE pojawia się bez uprzedniego użycia kontrolki $SAVE.
must be preceded by IFxx	Meta instrukcje ELSEIFxx, ELSE lub ENDIF pojawiają się bez poprzedzenia ich meta instrukcją IFxx.
no END statement found	Program kończy się bez użycia wyrażenia END.
not allowed in BIT segment	Instrukcja nie jest dozwolona w segmencie BIT.
only allowed in BIT segment	Instrukcja jest dozwolona tylko w segmencie BIT.
only allowed in CODE segment	Instrukcja jest dozwolona tylko w segmencie CODE.
operand expected	Instrukcja kończy się zanim jest składniowo kompletna.
phase error	Symbol jest rozwijany do innych wartości przy przejściu 1 przejściu 2, lub makro zostało zdefiniowane inaczej przy przejściu 1 i przejściu 2. Uwaga: jest to powazny wewnętrzny błąd kompilatora ----- i powinien zostać niezwłocznie zgłoszony autorowi!
preceded by non-control lines	Kontrolka ogólna pojawia się po wyrażeniu , które nie jest kontrolką asemblera.
register type operand	Symbol rejestru jest uzyty jako operand w wyrażeniu numerycznym.
segment limit exceeded	Licznik lokacji przekroczył granice aktualnego segmentu.
segment type mismatch	Typ segmentu operandu nie pasuje do typu instrukcji.
string exceeds end of line	Łańcuch znakowy nie został prawidłowo zakończony przy użyciu cudzysłowu.
symbol already defined	Próba redefiniowania symbolu, który

	został już zdefiniowany.
symbol name expected	Powinna w tym miejscu znajdować się prawidłowa nazwa symbolu.
symbol not defined	Odwołanie do symbolu, który nie został zdefiniowany (nie istnieje).
too many closing parentheses	Wyrażenie zawiera więcej nawiasów zamykających niż nawiasów otwierających.
too many opening parentheses	Wyrażenie zawiera więcej nawiasów otwierających niż nawiasów zamykających.
too many operands	Instrukcja zawiera więcej operandów, niż jest wymagane.
unary operator expected	W tej pozycji wyrażenia tylko ogólne operatory są dozwolone.
user-defined error	Wymuszony został komunikat błędu zdefiniowany przez użytkownika przy użyciu kontrolki $ERROR.

A.2 Błędy wykonania

Błędy wykonania są błędami operacyjnymi lub błędami wejścia/wyjścia.
Jeśli jeden z tych błędów zostanie wykryty, jest on oznaczany flagą w konsoli
i ASEM kończy się kodem wyjścia 2:

komunikat błędu	znaczenie
access denied	Brak przywilejów do wykonania funkcji.
ambiguous option name	Zbyt mało wpisanych znaków.
argument missing	Opcja wymaga argumentu.
device or resource busy	Nie można zapisać do zajętego urządzenia (Linuks).
disk full	Brak wolnej przestrzeni dyskowej.
disk write-protected	Nie można zapisać na dysku chronionym przed zapisem.
drive not ready	Brak napędu lub nie został on zamontowany.
duplicate file name	Nie można nadpisać pliku wejściowego lub wyjściowego.
fatal I/O error	Ogólny (nieznany) błąd wejścia/wyjścia dysku lub urządzenia.
file not found	Plik źródłowy lub dołączany nie został znaleziony (DOS/Windows).
illegal option syntax	Opcja wpisana nieprawidowo.
invalid argument	Opcja ma niedozwolony argument.
no input file	Nie ma nazwy pliku w komendzie.
no such file or directory	Plik źródłowy lub dołączany nie został znaleziony. (Linuks)
not a directory	Ścieżka zawiera nazwę, która nie jest katalogiem. (Linuks)
out of memory	Przepełnienie stosu!
path not found	Dysk lub katalog nie został znaleziony. (DOS/Windows)
symbol is predefined	Opcja /DEFINE zawiera predefiniowany symbol.

too many open files	Nie mozna otworzyć większej ilości plików.
too many parameters	Wpisano więcej niż trzy nazwy plików.
unknown option	Opcja nie została zaimplementowana.

Tylko linie komendy DOS zostały wspomniane powyżej dla uproszczenia.
W systemie Linuks, muszą one zostać zamienione przez odpowiadające im opcje Linuksa.

Dodatek B:

Komunikaty błędów narzędzia HEXBIN

B.1 Błędy konwersji

Błędy konwersji odnoszą się do konsystencji pliku Intel-HEX i opcji programu. Jeśli jeden z tych błędów zostanie wykryty, jest on oznaczany flagą i HEXBIN kończy swą pracę z kodem błędu 1:

komunikat błędu	znaczenie
checksum error	Suma kontrolna jest nieprawidłowa.
data after EOF record	Rekordy typu 0 za rekordami typu 1.
file length out of range	Opcja /LENGTH generuje zbyt duży plik.
fill-byte out of range	Opcja /FILL definiuje bajt o wartości więszej niż 255.
hex file format error	Plik nie jest w formacie Intel-HEX.
illegal hex digit	Znak nie jest prawidłową cyfrą szesnastkową.
illegal record type	Typ rekordu nie jest ani typu 0 ani typu 1.
invalid record length	Długość rekordu nie pasuje do rekordu.
multiple EOF records	Więcej niż jeden rekord typu 1.
no data records found	Plik nie zawiera rekordów typu 0.
no EOF record found	Plik kończy się bez rekordu typu 1.
offset out of range	Opcja /OFFSET generuje zbyt duży plik.
record exceeds FFFFH	Przestrzeń adresowa znajduje się poza rekordem.
record exceeds file length	Opcja /LENGTH generuje zbyt mały plik.

Tylko linie komendy DOS zostały wspomniane powyżej dla uproszczenia.
W systemie Linuks, muszą one zostać zamienione przez odpowiadające im opcje Linuksa.

B.2 Błędy wykonania

Błędy wykonania są błędami operacyjnymi lub błędami wejścia/wyjścia.
Jeśli jeden z tych błędów zostanie wykryty, jest on oznaczany flagą i HEXBIN jest generuje kod wyjścia 2:

komunikat błędu	znaczenie
access denied	Brak przywilejów do wykonania tej operacji.
ambiguous option name	Zbyt mało znaków zostało wpisanych.
argument missing	Opcja wymaga argumentu.
device or resource busy	Brak możliwości zapisu do zajętego urządzenia (Linuks).
disk full	Brak wolnej przestrzeni dyskowej.
disk write-protected	Brak możliwości zapisu do dysku chronionego przed zapisem.
drive not ready	Brak napędu lub nie zostałon zamontowany.

duplicate file name	Brak możliwości nadpisania pliku wejściowego lub wyjściowego.
fatal I/O error	Ogólny (nieznany) błąd wejścia/wyjścia napędu lub urządzenia.
file not found	Plik Intel-HEX nie został znaleziony. (DOS/Windows)
illegal option syntax	Opcja została wpisana nieprawidłowo.
invalid argument	Option has an illegal argument.
no input file	Brak nazwy pliku w linii komend.
no such file or directory	Plik Intel-HEX nie został znaleziony. (Linuks)
not a directory	Ścieżka zawiera nazwę, która nie jest katalogiem. (Linuks)
path not found	Dysk lub katalog nie został znaleziony. (DOS/Windows)
too many open files	Nie można otworzyć więcej plików.
too many parameters	Więcej niż dwie nazwy plików zostały wpisane.
unknown option	Niekompletna opcja.

Dodatek C:

Predefiniowane symbole

Adresy przestrzeni DATA:

P0	080H	P1	090H
SP	081H	SCON	098H
DPL	082H	SBUF	099H
DPH	083H	P2	0A0H
PCON	087H	IE	0A8H
TCON	088H	P3	0B0H
TMOD	089H	IP	0B8H
TL0	08AH	PSW	0D0H
TL1	08BH	ACC	0E0H
TH0	08CH	B	0F0H
TH1	08DH		

Adresy przestrzeni BIT:

IT0	088H	EA	0AFH
IE0	089H	RXD	0B0H
IT1	08AH	TXD	0B1H
IE1	08BH	INT0	0B2H
TR0	08CH	INT1	0B3H
TF0	08DH	T0	0B4H
TR1	08EH	T1	0B5H
TF1	08FH	WR	0B6H
RI	098H	RD	0B7H
TI	099H	PX0	0B8H
RB8	09AH	PT0	0B9H
TB8	09BH	PX1	0BAH
REN	09CH	PT1	0BBH
SM2	09DH	PS	0BCH
SM1	09EH	P	0D0H

SM0	09FH	OV	0D2H
EX0	0A8H	RS0	0D3H
ET0	0A9H	RS1	0D4H
EX1	0AAH	F0	0D5H
ET1	0ABH	AC	0D6H
ES	0ACH	CY	0D7H

Adresy przestrzeni CODE:

RESET	0000H	EXTI1	0013H
EXTI0	0003H	TIMER1	001BH
TIMER0	000BH	SINT	0023H

Predefiniowane wartości kompilatora:

??ASEM_51	8051H	??VERSION	0130H

Dodatek D:

Zarezerwowane słowa kluczowe

Specjalne Symbole Asemblera:

$	licznik lokacji
A	akumulator
AB	para rejestrów A/B
AR0,AR1,AR2,AR3,AR4,AR5,AR6,AR7	rejestry adresowane bezpośrednio
C	plaga przeniesienia
DPTR	wskaźnik danych
PC	licznik programu
R0, R1, R2, R3, R4, R5, R6, R7	rejestry

Instrukcje Mnemoniczne

ACALL	DA	JNB	MUL	RR
ADD	DEC	JNC	NOP	RRC
ADDC	DIV	JNZ	ORL	SETB
AJMP	DJNZ	JZ	POP	SJMP
ANL	INC	LCALL	PUSH	SUBB
CALL	JB	LJMP	RET	SWAP
CJNE	JBC	MOV	RETI	XCH
CLR	JC	MOVC	RL	XCHD
CPL	JMP	MOVX	RLC	XRL

Pseudo Instrukcje

AT	DATA	DSEG	IDATA	SET
BIT	DB	DW	ISEG	USING
BSEG	DBIT	END	NAME	XDATA
CODE	DS	EQU	ORG	XSEG
CSEG				

Operatory

AND	GT	LOW	NE	SHL
EQ	HIGH	LT	NOT	SHR
GE	LE	MOD	OR	XOR

Kontrolki Asemblera

$COND	$GO	$NODEBUG	$NOSYMBOLS	$RS
$CONDONLY	$IC	$NOGE	$NOTABS	$SA
$DA	$INCLUDE	$NOGEN	$NOXR	$SAVE
$DATE	$LI	$NOLI	$NOXREF	$SB
$DB	$LIST	$NOLIST	$PAGELENGTH	$SYMBOLS
$DEBUG	$MACRO	$NOMACRO	$PAGEWIDTH	$TITLE
$EJ	$MO	$NOMO	$PAGING	$TT
$EJECT	$MOD51	$NOMOD51	$PHILIPS	$WARNING
$ERROR	$MR	$NOMR	$PI	$XR
$GE	$NOBUILTIN	$NOPAGING	$PL	$XREF
$GEN	$NOCOND	$NOPI	$PW	
$GENONLY	$NODB	$NOSB	$RESTORE	

Meta Instrukcje

ELSE	ELSEIFN	ENDM	IFDEF	LOCAL
ELSEIF	ELSEIFNB	EXITM	IFN	MACRO
ELSEIFB	ELSEIFNDEF	IF	IFNB	REPT
ELSEIFDEF	ENDIF	IFB	IFNDEF	

Dodatek E:

Specyfikacja formatu Intel- HEX

Ten format pliku jest wspierany przez wiele kompilatorów, narzędzi i większość programatorów EPROM. Plik Intel-HEX jest plikiem tekstowym w 7 bitowym ASCII. Zawiera sekwencję rekordów danych i rekord końcowy. Każdy rekord jest linią tekstu, która zaczyna się dwukropkiem i kończy znakami CR i LF. Rekordy danych zawierają do 16 bajtów danych, 16 bitowy adres do ładowania danych, bajt typu rekordu i 8-bitowa suma kontrolna. Wszystkie liczby są reprezentowane przez duże znaki szesnastkowe ASCII.

Rekord danych:

```
Bajt  1          dwukropek (:)
      2 i 3      ilość binarnych bajtów dla tego rekordu
```

```
        4 i 5       adres docelowy dla tego rekordu, starszy bajt
        6 i 7       adres docelowy dla tego rekordu, młodszy bajt
        8 i 9       typ rekordu: 00 (rekord danych)
        10 do  x    bajty danych, dwa znaki każdy
        x+1 do  x+2 suma kontrolna (dwa znaki)
        x+3 do  x+4 CR i LF
```

Typowy rekord danych wygląda następująco

:10E0000002E003E4F588758910F58DF58BD28E302A

Końcowy rekord jest ostatnią linią pliku.
Zasadniczo jego struktura jest podobna do rekordu danych, ale numer bajtów danych jest 00, typ rekorsu to 01 i adres docelowy to 0000.

Rekord końcowy:

```
Bajt  1                 dwukropek (:)
      2 i 3             00   (ilość bajtów danych)
      4 i 5             00   (adres docelowy, starszy bajt)
      6 i 7             00   (adres docelowy, młodszy bajt)
      8 i 9             typ rekordu: 01  (rekord końcowy)
      10 i 11           suma kontrolna (dwa znaki)
      12 i 13           CR i LF
```

Typowy rekord końcowy wygląda w następujący sposób:

:00000001FF

Suma kontrolna to dopełnienie do dwóch ośmiobitowej sumy, bez przeniesienia, licznika bajtu, dwóch bajtów adresów docelowych, bajtu typu rekordu i wszystkich bajtów danych.

Dodatek F:

Zestaw kodów ASCII

hex	00	10	20	30	40	50	60	70
0	NUL	DLE		0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Dodatek G:

Literatura

		Numer zamówienia
Intel:	MCS(R) 51 Microcontroller Family User's Manual	
	MCS-51 Macro Assembler User's Guide	
	8-Bit Embedded Controllers 1990	
	8XC51RA/RB/RC CHMOS Single-Chip 8-Bit Microcontroller	272659-002
	8XC51RA/RB/RC Hardware Description, February 1995	272668-001
	83C51KB High Performance Keyboard Microcontroller	272800-001
	83C51KB Hardware Description	272801-001
	MCS 51 Microcontroller Family User's Manual, Feb 1994	272383-002
	8XC151SA and 8XC151SB Hardware Description, June 1996	272832-001
	Embedded Microcontrollers, 1997	270646-009
	8x931AA, 8x931HA USB Peripheral Controller User's Manual	Sept. 97
SIEMENS:	SAB 8O512/8O532 User's Manual	B2-B3808-X-X-7600
	SAB 8O515/8O535 User's Manual	B2-B3976-X-X-7600
	SAB 8OC515/8OC535 Data Sheet	
	SAB 8OC515A/83C515A-5 Addendum	B158-H6613-X-X-7600
	SAB 8OC515A/83C515A-5 Data Sheet	B158-H6605-X-X-7600
	SAB 8OC517/8OC537 User's Manual	B258-B6075-X-X-7600
	SAB 8OC517A/83C517A-5 Addendum	B158-H6612-X-X-7600
	SAB 8OC517A/83C517A-5 Data Sheet	B158-H6581-X-X-7600
	SIEMENS Microcontrollers Data Catalog	B158-H6569-X-X-7600
	SAB 8O513/8352-5 Data Sheet	B158-B6245-X-X-7600
	SAB-C5O1 User's Manual	B158-H6723-G1-X-7600
	SAB-C5O2 User's Manual	B158-H6722-G1-X-7600
	SAB-C5O3 User's Manual	B158-H665O-G1-X-7600
	Application Notes and User Manuals, CD-ROM	B193-H69OO-X-X-7400
	C5O4 8-Bit CMOS Microcontroller User's Manual	B158-H6958-X-X-7600
	C5O9-L 8-Bit CMOS Microcontroller User's Manual	B158-H6973-X-X-7600
	C515C 8-Bit CMOS Microcontroller User's Manual	B158-H6981-X-X-7600

Infineon: C515 8-Bit CMOS Microcontroller User's Manual
04.98
C515A 8-Bit CMOS Microcontroller User's Manual
08.97
C517A 8-Bit CMOS Microcontroller User's Manual
01.99
C540U/C541U 8-Bit CMOS Microcontroller Data Sheet
10.97
C513AO 8-bit CMOS Microcontroller User's Manual
05.99
C505/C505C 8-Bit CMOS Microcontroller User's Manual
08.97
C505A/C505CA 8-Bit CMOS Microcontroller, Addendum
09.97
C505L 8-bit CMOS Microcontroller, Data Sheet
06.99
C508 8-Bit CMOS Microcontroller User's Manual May
2001
C868 8-Bit CMOS Microcontroller User's Manual, V 0.5.1 Jan
2002

Philips: PCB83C552 User's Manual
PCB83C552, PCB80C552 Development Data
8051-Based 8-Bit Microcontrollers - Data Handbook 1994
83C754/87C754 Preliminary Specification, 1998 Apr 23
87LPC762 Data Sheet, 2001 Oct 26
87LPC768 Data Sheet, 2002 Mar 12
P80C3xX2, P80C5xX2, P87C5xX2 Data Sheet, 2002 Jun 12
P89C51X2/52X2/54X2/58X2 Data Sheet, 2002 Jun 05

AMD: Eight-Bit 80C51 Embedded Processors - Data Book 1990

OKI: MSM80C154, MSM83C154 User's Manual
Microcontroller Data Book, 5th Edition 1990

TDK: TSC 73M2910/2910A Microcontroller, 10/22/96 - rev.

DALLAS: High-Speed Micro User's Guide, V1.3 January 1994, 011994
Secure Microcontroller User's Guide, 062001
DS80C310 High-Speed Micro, 090198
DS80C320/DS80C323 High-Speed/Low-Power Micro, 070196
DS80C390 Dual CAN High-Speed Microprocessor, 090799
DS87C520/DS83C520 EPROM/ROM High-Speed Micro, 110195
DS87C530/DS83C530 EPROM/ROM Micro with Real-Time Clock, 112299
DS87C550 EPROM High-Speed Micro with A/D and PWM, 091698
DS89C420 Ultra High-Speed Microcontroller User's Guide, 020602

Maxim: MAX7651/MAX7652 Programmer's Reference Manual
MAX7651/MAX7652 Data Sheet, 19-2119; Rev 0; 8/01

Atmel: AT89C51, 8-Bit Microcontroller with 4 Kbytes Flash
0265E
AT89C52, 8-Bit Microcontroller with 8 Kbytes Flash
0313E
AT89C55, 8 bit Microcontroller with 20K bytes Flash
AT89LV51, 8-Bit Microcontroller with 4 Kbytes Flash
0303C

AT89LV52, 8-Bit Microcontroller with 8 Kbytes Flash
AT89C1051, 8-Bit Microcontroller with 1 Kbyte Flash
0366C
AT89C2051, 8-Bit Microcontroller with 2 Kbytes Flash
0368C
AT89C4051, 8-Bit Microcontroller with 4K Bytes Flash
Preliminary
AT89C1051U, 8-Bit Microcontroller with 1K Bytes Flash
Preliminary
AT89S8252, 8 Bit Microcontroller with 8K bytes Flash
Preliminary
AT89LS8252, 8-Bit Microcontroller with 8K Bytes Flash 0850B-B-
12/97
AT89S51, 8-bit Microcontroller with 4K Bytes ISP Flash 2487A-
10/01
AT89S52, 8-bit Microcontroller with 8K Bytes ISP Flash 1919A-
07/01
AT89LS52, 8-bit LV Microcontroller with 8K Bytes ISP Flash 2601A-
12/01
AT89S53, 8-Bit Microcontroller with 12K Bytes Flash
Preliminary
AT89LS53, 8-Bit Microcontroller with 12K Bytes Flash 0851B-B-
12/97
AT89LV55, 8-Bit Microcontroller with 20K bytes Flash 0811A-A-
7/97
AT80F51, 8-Bit Microcontroller with 4K Bytes QuickFlash 0979A-A-
12/97
AT87F51, 8-Bit Microcontroller with 4K Bytes QuickFlash 1012A-
02/98
AT80F52, 8-Bit Microcontroller with 8K Bytes QuickFlash 0980A-A-
12/97
AT87F52, 8-Bit Microcontroller with 8K Bytes QuickFlash 1011A-
02/98
AT89S4D12, 8-Bit Microcontroller with 132K Bytes Flash 0921A-A-
12/97
AT87F51RC, 8-Bit Microcontroller with 32K Bytes QuickFlash 1106B-
12/98
AT87F55, 8-Bit Microcontroller with 20K Bytes QuickFlash 1147A-
05/99
AT87LV51, 8-bit Microcontroller with 4K Bytes QuickFlash 1602A-
04/00
AT87LV52, 8-Bit Microcontroller with 8K Bytes QuickFlash 1437A-
07/99
AT87LV55, 8-bit Microcontroller with 20K Bytes QuickFlash 1609A-
04/00

Cypress: The EZ-USB Integrated Circuit, Technical Reference Manual Version
1.9

Literatura niemiecka:

Andreas Roth: Das Microcontroller Kochbuch MCS51,
6. Aufl. 2002, mitp-Verlag, ISBN 3-8266-0722-8

Dodatek H:

Znaki firmowe

Dodatek I:

Instrukcje mikrokontrolera 8051 w porządku numerycznym

Objaśnienia:
- direct = 8-bitowy DATA adres w pamięci wewnętrznej
- const8 = 8-bitowa stała w pamięci CODE
- const16 = 16-bitowa stała w pamięci CODE
- addr16 = 16-bitowy długi adres w przestrzeni CODE
- addr11 = 11-bitowy absolutny adres w przestrzeni CODE
- rel = 8-bitowy adres względny ze znakiem w CODE
- bit = 8-bitowy adres bitu w przestrzeni BIT pamięci wewnętrznej

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
00	NOP		1		1
01	AJMP	addr11	2		2
02	LJMP	addr16	3		2
03	RR	A	1		1
04	INC	A	1	P	1
05	INC	direct	2		1
06	INC	@R0	1		1
07	INC	@R1	1		1
08	INC	R0	1		1
09	INC	R1	1		1
0A	INC	R2	1		1
0B	INC	R3	1		1
0C	INC	R4	1		1
0D	INC	R5	1		1
0E	INC	R6	1		1
0F	INC	R7	1		1
10	JBC	bit, rel	3		2
11	ACALL	addr11	2		2
12	LCALL	addr16	3		2

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
13	RRC	A	1	CY P	1
14	DEC	A	1	P	1
15	DEC	direct	2		1
16	DEC	@R0	1		1
17	DEC	@R1	1		1
18	DEC	R0	1		1
19	DEC	R1	1		1
1A	DEC	R2	1		1
1B	DEC	R3	1		1
1C	DEC	R4	1		1
1D	DEC	R5	1		1
1E	DEC	R6	1		1
1F	DEC	R7	1		1
20	JB	bit, rel	3		2
21	AJMP	addr11	2		2
22	RET		1		2
23	RL	A	1		1
24	ADD	A, #const8	2	CY AC OV P	1
25	ADD	A, direct	2	CY AC OV P	1
26	ADD	A, @R0	1	CY AC OV P	1
27	ADD	A, @R1	1	CY AC OV P	1
28	ADD	A, R0	1	CY AC OV P	1
29	ADD	A, R1	1	CY AC OV P	1
2A	ADD	A, R2	1	CY AC OV P	1
2B	ADD	A, R3	1	CY AC OV P	1
2C	ADD	A, R4	1	CY AC OV P	1
2D	ADD	A, R5	1	CY AC OV P	1
2E	ADD	A, R6	1	CY AC OV P	1
2F	ADD	A, R7	1	CY AC OV P	1
30	JNB	bit, rel	3		2
31	ACALL	addr11	2		2
32	RETI		1		2
33	RLC	A	1	CY P	1
34	ADDC	A, #const8	2	CY AC OV P	1
35	ADDC	A, direct	2	CY AC OV P	1
36	ADDC	A, @R0	1	CY AC OV P	1
37	ADDC	A, @R1	1	CY AC OV P	1
38	ADDC	A, R0	1	CY AC OV P	1
39	ADDC	A, R1	1	CY AC OV P	1
3A	ADDC	A, R2	1	CY AC OV P	1
3B	ADDC	A, R3	1	CY AC OV P	1
3C	ADDC	A, R4	1	CY AC OV P	1
3D	ADDC	A, R5	1	CY AC OV P	1
3E	ADDC	A, R6	1	CY AC OV P	1
3F	ADDC	A, R7	1	CY AC OV P	1
40	JC	rel	2		2
41	AJMP	addr11	2		2
42	ORL	direct, A	2		1
43	ORL	direct, #const8	3		2
44	ORL	A, #const8	2	P	1
45	ORL	A, direct	2	P	1
46	ORL	A, @R0	1	P	1
47	ORL	A, @R1	1	P	1
48	ORL	A, R0	1	P	1
49	ORL	A, R1	1	P	1
4A	ORL	A, R2	1	P	1

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
4B	ORL	A, R3	1	P	1
4C	ORL	A, R4	1	P	1
4D	ORL	A, R5	1	P	1
4E	ORL	A, R6	1	P	1
4F	ORL	A, R7	1	P	1
50	JNC	rel	2		2
51	ACALL	addr11	2		2
52	ANL	direct, A	2		1
53	ANL	direct, #const8	3		2
54	ANL	A, #const8	2	P	1
55	ANL	A, direct	2	P	1
56	ANL	A, @R0	1	P	1
57	ANL	A, @R1	1	P	1
58	ANL	A, R0	1	P	1
59	ANL	A, R1	1	P	1
5A	ANL	A, R2	1	P	1
5B	ANL	A, R3	1	P	1
5C	ANL	A, R4	1	P	1
5D	ANL	A, R5	1	P	1
5E	ANL	A, R6	1	P	1
5F	ANL	A, R7	1	P	1
60	JZ	rel	2		2
61	AJMP	addr11	2		2
62	XRL	direct, A	2		1
63	XRL	direct, #const8	3		2
64	XRL	A, #const8	2	P	1
65	XRL	A, direct	2	P	1
66	XRL	A, @R0	1	P	1
67	XRL	A, @R1	1	P	1
68	XRL	A, R0	1	P	1
69	XRL	A, R1	1	P	1
6A	XRL	A, R2	1	P	1
6B	XRL	A, R3	1	P	1
6C	XRL	A, R4	1	P	1
6D	XRL	A, R5	1	P	1
6E	XRL	A, R6	1	P	1
6F	XRL	A, R7	1	P	1
70	JNZ	rel	2		2
71	ACALL	addr11	2		2
72	ORL	C, bit	2	CY	2
73	JMP	@A+DPTR	1		2
74	MOV	A, #const8	2	P	1
75	MOV	direct, #const8	3		2
76	MOV	@R0, #const8	2		1
77	MOV	@R1, #const8	2		1
78	MOV	R0, #const8	2		1
79	MOV	R1, #const8	2		1
7A	MOV	R2, #const8	2		1
7B	MOV	R3, #const8	2		1
7C	MOV	R4, #const8	2		1
7D	MOV	R5, #const8	2		1
7E	MOV	R6, #const8	2		1
7F	MOV	R7, #const8	2		1
80	SJMP	rel	2		2
81	AJMP	addr11	2		2
82	ANL	C, bit	2	CY	2

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
83	MOVC	A, @A+PC	1	P	2
84	DIV	AB	1	CY OV P	4
85	MOV	direct, direct	3		2
86	MOV	direct, @R0	2		2
87	MOV	direct, @R1	2		2
88	MOV	direct, R0	2		2
89	MOV	direct, R1	2		2
8A	MOV	direct, R2	2		2
8B	MOV	direct, R3	2		2
8C	MOV	direct, R4	2		2
8D	MOV	direct, R5	2		2
8E	MOV	direct, R6	2		2
8F	MOV	direct, R7	2		2
90	MOV	DPTR, #const16	3		2
91	ACALL	addr11	2		2
92	MOV	bit, C	2		2
93	MOVC	A, @A+DPTR	1	P	2
94	SUBB	A, #const8	2	CY AC OV P	1
95	SUBB	A, direct	2	CY AC OV P	1
96	SUBB	A, @R0	1	CY AC OV P	1
97	SUBB	A, @R1	1	CY AC OV P	1
98	SUBB	A, R0	1	CY AC OV P	1
99	SUBB	A, R1	1	CY AC OV P	1
9A	SUBB	A, R2	1	CY AC OV P	1
9B	SUBB	A, R3	1	CY AC OV P	1
9C	SUBB	A, R4	1	CY AC OV P	1
9D	SUBB	A, R5	1	CY AC OV P	1
9E	SUBB	A, R6	1	CY AC OV P	1
9F	SUBB	A, R7	1	CY AC OV P	1
A0	ORL	C, /bit	2	CY	2
A1	AJMP	addr11	2		2
A2	MOV	C, bit	2	CY	1
A3	INC	DPTR	1		2
A4	MUL	AB	1	CY OV P	4
A5	illegal	opcode			
A6	MOV	@R0, direct	2		2
A7	MOV	@R1, direct	2		2
A8	MOV	R0, direct	2		2
A9	MOV	R1, direct	2		2
AA	MOV	R2, direct	2		2
AB	MOV	R3, direct	2		2
AC	MOV	R4, direct	2		2
AD	MOV	R5, direct	2		2
AE	MOV	R6, direct	2		2
AF	MOV	R7, direct	2		2
B0	ANL	C, /bit	2	CY	2
B1	ACALL	addr11	2		2
B2	CPL	bit	2		1
B3	CPL	C	1	CY	1
B4	CJNE	A, #const8, rel	3	CY	2
B5	CJNE	A, direct, rel	3	CY	2
B6	CJNE	@R0, #const8, rel	3	CY	2
B7	CJNE	@R1, #const8, rel	3	CY	2
B8	CJNE	R0, #const8, rel	3	CY	2
B9	CJNE	R1, #const8, rel	3	CY	2
BA	CJNE	R2, #const8, rel	3	CY	2

Kod	Mnemonik	Operandy	Bajty	Flagi		Cykle
BB	CJNE	R3, #const8, rel	3	CY		2
BC	CJNE	R4, #const8, rel	3	CY		2
BD	CJNE	R5, #const8, rel	3	CY		2
BE	CJNE	R6, #const8, rel	3	CY		2
BF	CJNE	R7, #const8, rel	3	CY		2
C0	PUSH	direct	2			2
C1	AJMP	addr11	2			2
C2	CLR	bit	2			1
C3	CLR	C	1	CY		1
C4	SWAP	A	1			1
C5	XCH	A, direct	2		P	1
C6	XCH	A, @R0	1		P	1
C7	XCH	A, @R1	1		P	1
C8	XCH	A, R0	1		P	1
C9	XCH	A, R1	1		P	1
CA	XCH	A, R2	1		P	1
CB	XCH	A, R3	1		P	1
CC	XCH	A, R4	1		P	1
CD	XCH	A, R5	1		P	1
CE	XCH	A, R6	1		P	1
CF	XCH	A, R7	1		P	1
D0	POP	direct	2			2
D1	ACALL	addr11	2			2
D2	SETB	bit	2			1
D3	SETB	C	1	CY		1
D4	DA	A	1	CY	P	1
D5	DJNZ	direct, rel	3			2
D6	XCHD	A, @R0	1		P	1
D7	XCHD	A, @R1	1		P	1
D8	DJNZ	R0, rel	2			2
D9	DJNZ	R1, rel	2			2
DA	DJNZ	R2, rel	2			2
DB	DJNZ	R3, rel	2			2
DC	DJNZ	R4, rel	2			2
DD	DJNZ	R5, rel	2			2
DE	DJNZ	R6, rel	2			2
DF	DJNZ	R7, rel	2			2
E0	MOVX	A, @DPTR	1		P	2
E1	AJMP	addr11	2			2
E2	MOVX	A, @R0	1		P	2
E3	MOVX	A, @R1	1		P	2
E4	CLR	A	1		P	1
E5	MOV	A, direct	2		P	1
E6	MOV	A, @R0	1		P	1
E7	MOV	A, @R1	1		P	1
E8	MOV	A, R0	1		P	1
E9	MOV	A, R1	1		P	1
EA	MOV	A, R2	1		P	1
EB	MOV	A, R3	1		P	1
EC	MOV	A, R4	1		P	1
ED	MOV	A, R5	1		P	1
EE	MOV	A, R6	1		P	1
EF	MOV	A, R7	1		P	1
F0	MOVX	@DPTR, A	1			2
F1	ACALL	addr11	2			2
F2	MOVX	@R0, A	1			2

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
F3	MOVX	@R1, A	1		2
F4	CPL	A	1	P	1
F5	MOV	direct, A	2		1
F6	MOV	@R0, A	1		1
F7	MOV	@R1, A	1		1
F8	MOV	R0, A	1		1
F9	MOV	R1, A	1		1
FA	MOV	R2, A	1		1
FB	MOV	R3, A	1		1
FC	MOV	R4, A	1		1
FD	MOV	R5, A	1		1
FE	MOV	R6, A	1		1
FF	MOV	R7, A	1		1

Dodatek J:

Instrukcje mikrokontrolera 8051 w porządku leksykalnym

Objaśnienia:
direct = 8-bitowy DATA adres w pamięci wewnętrznej
const8 = 8-bitowa stała w pamięci CODE
const16 = 16-bitowa stała w pamięci CODE
addr16 = 16-bitowy długi adres w przestrzeni CODE
addr11 = 11-bitowy absolutny adres w przestrzeni CODE
rel = 8-bitowy adres względny ze znakiem w CODE
bit = 8-bitowy adres bitu w przestrzeni BIT pamięci wewnętrznej

i = numer rejestru 0 lub 1
n = numer rejestru od 0 do 7
a = 32 * m
m = 3 najbardziej znaczące bity adresu absolutnego

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
11+a	ACALL	addr11	2		2
24	ADD	A, #const8	2	CY AC OV P	1
26+i	ADD	A, @Ri	1	CY AC OV P	1
25	ADD	A, direct	2	CY AC OV P	1
28+n	ADD	A, Rn	1	CY AC OV P	1
34	ADDC	A, #const8	2	CY AC OV P	1
36+i	ADDC	A, @Ri	1	CY AC OV P	1
35	ADDC	A, direct	2	CY AC OV P	1
38+n	ADDC	A, Rn	1	CY AC OV P	1
01+a	AJMP	addr11	2		2
54	ANL	A, #const8	2	P	1
56+i	ANL	A, @Ri	1	P	1
55	ANL	A, direct	2	P	1
58+n	ANL	A, Rn	1	P	1
B0	ANL	C, /bit	2	CY	2
82	ANL	C, bit	2	CY	2
53	ANL	direct, #const8	3		2
52	ANL	direct, A	2		1

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
B6+i	CJNE	@Ri, #const8, rel	3	CY	2
B4	CJNE	A, #const8, rel	3	CY	2
B5	CJNE	A, direct, rel	3	CY	2
B8+n	CJNE	Rn, #const8, rel	3	CY	2
E4	CLR	A	1	P	1
C2	CLR	bit	2		1
C3	CLR	C	1	CY	1
F4	CPL	A	1	P	1
B2	CPL	bit	2		1
B3	CPL	C	1	CY	1
D4	DA	A	1	CY P	1
16+i	DEC	@Ri	1		1
14	DEC	A	1	P	1
15	DEC	direct	2		1
18+n	DEC	Rn	1		1
84	DIV	AB	1	CY OV P	4
D5	DJNZ	direct, rel	3		2
D8+n	DJNZ	Rn, rel	2		2
06+i	INC	@Ri	1		1
04	INC	A	1	P	1
05	INC	direct	2		1
A3	INC	DPTR	1		2
08+n	INC	Rn	1		1
20	JB	bit, rel	3		2
10	JBC	bit, rel	3		2
40	JC	rel	2		2
73	JMP	@A+DPTR	1		2
30	JNB	bit, rel	3		2
50	JNC	rel	2		2
70	JNZ	rel	2		2
60	JZ	rel	2		2
12	LCALL	addr16	3		2
02	LJMP	addr16	3		2
76+i	MOV	@Ri, #const8	2		1
F6+i	MOV	@Ri, A	1		1
A6+i	MOV	@Ri, direct	2		2
74	MOV	A, #const8	2	P	1
E6+i	MOV	A, @Ri	1	P	1
E5	MOV	A, direct	2	P	1
E8+n	MOV	A, Rn	1	P	1
92	MOV	bit, C	2		2
A2	MOV	C, bit	2	CY	1
75	MOV	direct, #const8	3		2
86+i	MOV	direct, @Ri	2		2
F5	MOV	direct, A	2		1
85	MOV	direct, direct	3		2
88+n	MOV	direct, Rn	2		2
90	MOV	DPTR, #const16	3		2
78+n	MOV	Rn, #const8	2		1
F8+n	MOV	Rn, A	1		1
A8+n	MOV	Rn, direct	2		2
93	MOVC	A, @A+DPTR	1	P	2
83	MOVC	A, @A+PC	1	P	2
F0	MOVX	@DPTR, A	1		2
F2+i	MOVX	@Ri, A	1		2
E0	MOVX	A, @DPTR	1	P	2

Kod	Mnemonik	Operandy	Bajty	Flagi	Cykle
E2+i	MOVX	A, @Ri	1	P	2
A4	MUL	AB	1	CY OV P	4
00	NOP		1		1
44	ORL	A, #const8	2	P	1
46+i	ORL	A, @Ri	1	P	1
45	ORL	A, direct	2	P	1
48+n	ORL	A, Rn	1	P	1
A0	ORL	C, /bit	2	CY	2
72	ORL	C, bit	2	CY	2
43	ORL	direct, #const8	3		2
42	ORL	direct, A	2		1
D0	POP	direct	2		2
C0	PUSH	direct	2		2
22	RET		1		2
32	RETI		1		2
23	RL	A	1		1
33	RLC	A	1	CY P	1
03	RR	A	1		1
13	RRC	A	1	CY P	1
D2	SETB	bit	2		1
D3	SETB	C	1	CY	1
80	SJMP	rel	2		2
94	SUBB	A, #const8	2	CY AC OV P	1
96+i	SUBB	A, @Ri	1	CY AC OV P	1
95	SUBB	A, direct	2	CY AC OV P	1
98+n	SUBB	A, Rn	1	CY AC OV P	1
C4	SWAP	A	1		1
C6+i	XCH	A, @Ri	1	P	1
C5	XCH	A, direct	2	P	1
C8+n	XCH	A, Rn	1	P	1
D6+i	XCHD	A, @Ri	1	P	1
64	XRL	A, #const8	2	P	1
66+i	XRL	A, @Ri	1	P	1
65	XRL	A, direct	2	P	1
68+n	XRL	A, Rn	1	P	1
63	XRL	direct, #const8	3		2
62	XRL	direct, A	2		1