

Zespół Szkół Łączności

**Technikum Łączności
im. Obrońców Poczty Polskiej
ul. Podwale Staromiejskie 51/52**

PRACA DYPLOMOWA

TEMAT:

Emulator Pamięci EPROM

1998/1999/5D/17

Wykonał:

Tomasz Nowakowski

Klasa: 5D

Konsultant:

mgr inż. Weronika Pyda - Ledwon

GDAŃSK 1999

SPIS TREŚCI

WSTĘP	2
CHARAKTERYSTYKA OGÓLNA EMULATORA	3
OPIS KONSTRUKCJI EMULATORA PAMIĘCI EPROM	4
OPIS DZIAŁANIA UKŁADU EMULATORA	4
DOSTĘP PROCESORA DO PAMIĘCI	8
SPOSÓB KOMUNIKACJI Z KOMPUTEREM PC	10
OPIS PROGRAMU '51	12
OPIS PROGRAMU 'EEPROM.EXE'	16
OPIS DEASSEMBLERA '51	16
SPOSÓB KOMUNIKACJI Z EMULATOREM	17
SPECYFIKACJA FORMATU INTEL HEX II	19
INSTRUKCJA OBSŁUGI EMULATORA	20
PRZYGOTOWANIE EMULATORA DO PRACY	20
WYMAGANIA I FUNKCJE PROGRAMU EEPROM	22
INSTALACJA PROGRAMU EEPROM.....	22
CHARAKTERYSTYKA OKIENEK DIALOGOWYCH PROGRAMU EEPROM.....	23
SPIS KŁAWISZY FUNKCYJNYCH.....	34
FORMAT PLIKU DEFINICJI ETYKIET	34
PROBLEMY NAPOTKANE PODCZAS KONSTRUOWANIA EMULATORA	37
BIBLIOGRAFIA	38
RYSUNKI PŁYTEK DRUKOWANYCH.....	39

Wstęp

Emulator pamięci EPROM jest sprzętowym emulatorem sterowanym z komputera PC. Jest on przydatny w procesie uruchamiania urządzeń elektronicznych wyposażonych w pamięć EPROM o pojemności do 64 kB. Emulator EPROM'ów jest urządzeniem przyspieszającym testowanie i badanie poprawności programów lub danych zawartych w pamięciach EPROM i używanych w najrozmaitszych systemach mikroprocesorowych. Emulator EPROM umożliwia szybkie sprawdzenie poprawności informacji zapisanej w pamięci EPROM, a poprzez zewnętrzny sygnał RST może także po każdym zapisie inicjować testowane urządzenie. W ten sposób można szybko i skutecznie wyeliminować błędy w programach lub danych przeznaczonych do zapisu w pamięci EPROM. Emulator został zbudowany z myślą o projektach opartych na serii mikroprocesorów 8051. Program użytkowy obsługujący emulator został napisany w języku C++ i steruje całym procesem symulacji. Program ten ma wbudowany deassembler instrukcji procesora 8051, co znacznie ułatwia analizę programu symulowanego za pomocą emulatora. Niemniej jednak emulator może symulować pamięci EPROM nie tylko używane w systemach '51, ale w dowolnym urządzeniu zawierającym właśnie taką pamięć. Emulator pamięci EPROM ma jeszcze jedną istotną cechę: umożliwia odczytanie pamięci EPROM i zapisanie jej zawartości na dysku w celu późniejszej obróbki danych. Wszystkie funkcje emulatora można wywoływać poprzez system menu.

Charakterystyka ogólna emulatora

Głównym celem pracy było zbudowanie urządzenia, które pozwoliłoby szybko sprawdzić zawartość pamięci EPROM, a później (po zmianie zawartości) zasymulować działanie takiej pamięci w układzie. Urządzenie takie musi być wystarczająco uniwersalne, aby miało szerokie zastosowanie (duża ilość symulowanych układów pamięci), a także musi być proste w obsłudze, aby nie sprawiało dodatkowych problemów. Kolejnym założeniem były minimalne wymagania co do komputera PC oraz niezawodność podczas transmisji danych z komputera do emulatora.

Powyższe kryteria udało mi się spełnić. Emulator umożliwia symulację, jak i odczyt następujących układów 8-bitowych pamięci równoległych:

- 2764 - 8 kB EPROM
- 27128 - 16 kB EPROM
- 27256 - 32 kB EPROM
- 27512 - 64 kB EPROM

Emulator symuluje również wersje CMOS wyżej wymienionych układów. Natomiast po zmianie wtyku emulacyjnego (z 28-pinowego na 24-pinowy) możliwa staje się emulacja pamięci: 2716, 2732 w wersji standardowej i CMOS.

Urządzenie komunikuje się z komputerem PC poprzez port szeregowy RS-232c. Zdecydowałem się na transmisję szeregową z kilku powodów. Pierwszym był fakt, iż port szeregowy umożliwia użycie dłuższego kabla połączeniowego, co w przypadku emulatora może mieć istotny wpływ na jego funkcjonalność. W szczególnych okolicznościach niemożliwe staje się umieszczenie urządzenia testowanego blisko komputera. Drugim powodem była liczba przewodów (grubość kabla) pomiędzy emulatorem a komputerem. Gdyby zastosować port CENTRONICS, to liczba kabli wzrosłaby znacznie, a także utrudniłoby to używanie drukarki (potrzebny byłby specjalny przełącznik drukarkowy). Większa liczba przewodów to grubszy kabel i większe prawdopodobieństwo uszkodzenia (zużycia) kabla, a w konsekwencji utrudnienia poprawnej pracy emulatora. Grubszy kabel jest również mniej poręczny i może sprawiać kłopoty, gdy emulator będzie pracował w skomplikowanym układzie badanym. Transmisja szeregową jest także prostsza do zrealizowania pod względem sprzętowym. Do tego celu wystarczy użyć np. układu 8251. Jest to układ do przeprowadzania synchronicznej lub asynchronicznej transmisji szeregową. Układ ten w całości zajmuje się przebiegiem transmisji (badanie bitów startu, stopu itp.), a dane wejściowe (wyjściowe) podawane są w sposób równoległy. Jednak do budowy emulatora, ze względu na większą elastyczność, zastosowałem mikroprocesor serii '51. Procesor ten ma wbudowany układ transmisji szeregową, który może współpracować z komputerem PC jedynie przy użyciu dodatkowego konwertera napięć TTL / RS232c.

Podczas konstruowania emulatora zwracałem szczególną uwagę na ostateczną wielkość urządzenia w związku z tym zastosowałem procesor firmy Atmel AT89C51. Pozwoliło to zmniejszyć znacząco płytki emulatora, a także znacznie uprościło konstrukcję urządzenia.

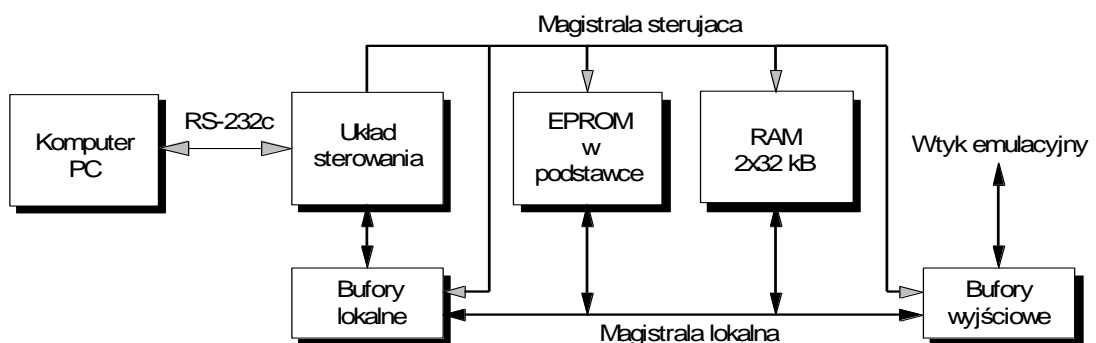
Opis konstrukcji emulatora pamięci EPROM

Opis działania układu emulatora

Emulator pamięci EPROM zbudowany jest w oparciu o procesor AT89C51. Jest to mikrokomputer z serii 8051 wyposażony w 4 kB wewnętrznej pamięci programu wykonanej w technologii Flash EPROM. Poza tą zmianą procesor jest w pełni zgodny ze swoim protoplastą 80C51. Zastosowałem ten właśnie procesor w celu zmniejszenia liczby układów scalonych emulatora. Miało to znaczący wpływ na ostateczną wielkość urządzenia. Prototyp był zbudowany w oparciu o procesor 80C31 pracujący z zewnętrzną pamięcią programu i konsekwencją tego była większa i bardziej skomplikowana płytką.

Procesor AT89C51 ma wbudowany układ transmisji szeregowej, układ przerwań, dwa układy licznikowe oraz 128 bajtów wewnętrznej pamięci RAM. Układ transmisji szeregowej może pracować w trybie synchronicznym lub asynchronicznym przysyłając 8 bitów danych wraz z bitem parzystości. Układ ten taktowany jest sygnałem przepelnienia jednego z liczników (T1). Przy taktowaniu procesora kwarcem o wartości 11,0592 MHz maksymalna prędkość transmisji wynosi 57600 bps (bitów na sekundę). Jest to wystarczająca szybkość przesyłania danych pomiędzy komputerem a emulatorem.

Schemat blokowy emulatora przedstawiony został na rysunku 1. Urządzenie wyposażone jest w 64 kB pamięci RAM zbudowanej na dwóch układach 62256 o pojemnościach 32 kB każdy oraz w podstawkę do odczytu pamięci EPROM o wielkości do 64 kB. Procesor AT89C51 ma 16-bitową magistralę adresową, więc może zaadresować najwyżej 64 kB pamięci. W naszym przypadku maksymalna przestrzeń adresowa (RAM + EPROM) wynosi 128 kB. W związku z tym należało zastosować metodę przełączania banków, aby móc obsłużyć całą pamięć. Jednak zastosowane tu rozwiązanie nie jest typowe dla metody przełączania banków. Aby umożliwić procesorowi dostęp do 128 kB pamięci, zastosowano metodę wybierania typu aktualnie wykorzystywanej pamięci. Procesor najpierw „decyduje” czy chce mieć dostęp do pamięci RAM czy też EPROM i poprzez odpowiednie sygnały sterujące przełącza daną pamięć. Nazwałem to metodą dostępu do pamięci. Tryb emulacji jest także jedną z metod dostępu do pamięci. W tym trybie cała pamięć RAM jest dostępna z zewnątrz (poprzez wtyk emulacyjny), natomiast pamięć EPROM jest nieaktywna. W trybie emulacji procesor nie ma dostępu do pamięci RAM, jak i EPROM.



Rys. 1. Schemat blokowy emulatora pamięci EPROM.

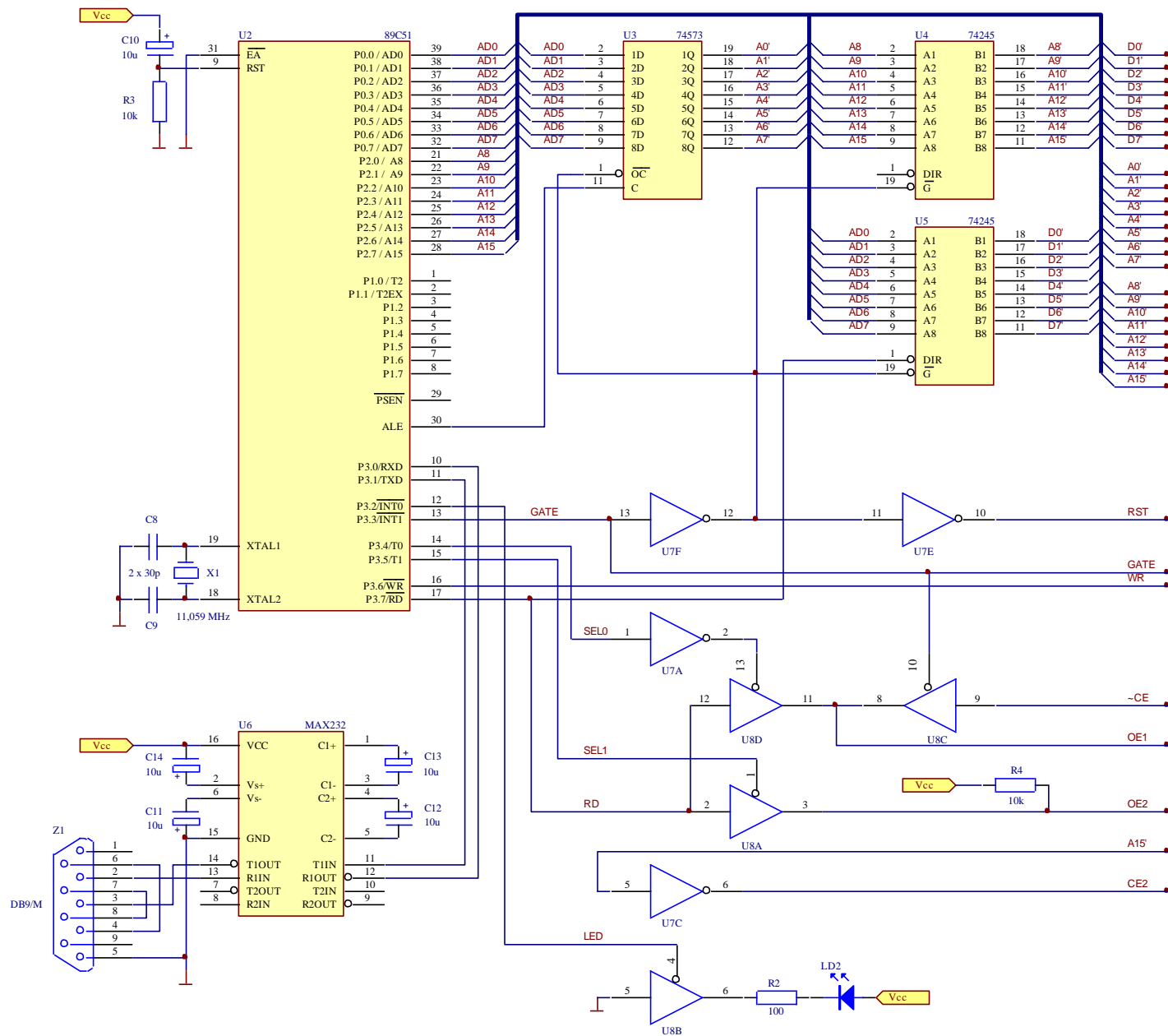
Cały emulator zbudowany jest na dwóch dwustronnych płytkach drukowanych. Pierwsza płytką to płytką sterującą, odpowiedzialna za komunikację z komputerem PC oraz generującą wszystkie sygnały sterujące potrzebne do pracy emulatora. Druga płytką zawiera pamięci oraz bufony wyjściowe. Schemat ideowy płytki sterującej

zamieszczony jest na rysunku 2, natomiast schemat płytki z pamięciami - na rysunku 3. Rysunki płytek zamieszczone są na końcu tekstu.

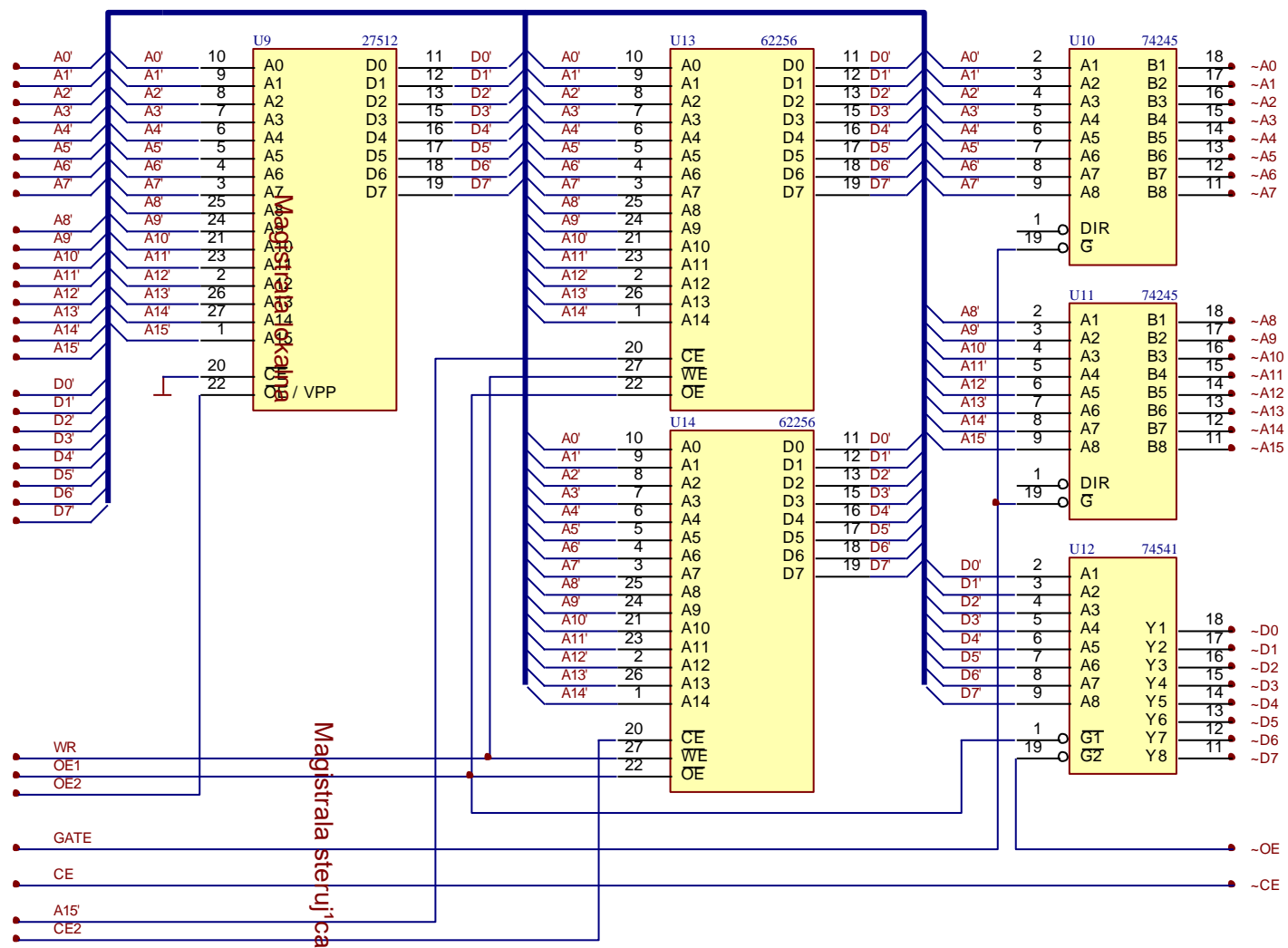
Układ zasilania (rysunek 4) został zbudowany w oparciu o stabilizator 7805 (U1) oraz kondensatory C1 i C2. Jest to trój-wyprowadzeniowy stabilizator napięcia dodatniego o wartości +5 V. Dioda D1 włączona na wejściu stabilizatora zabezpiecza układ U1 przed spalaniem w razie odwrotnej polaryzacji napięcia wejściowego. Przed diodą znajduje się wyłącznik główny emulatora. Dioda LED (LD1) służy do sygnalizacji włączenia zasilania. Układ emulatora powinien być zasilany napięciem stałym od 6 do 15 V. Górna granica wynika z zastosowania bardzo małego radiatora dla układu U1, a zbyt wysokie napięcie zasilające może spowodować przegrzewanie się stabilizatora. Dolna granica napięcia zasilania jest narzucona przez producenta układu 7805.

W celu zminimalizowania poboru prądu z zasilacza cały emulator został zbudowany przy użyciu układów serii HCT. Mimo to udało się uzyskać jeszcze większą oszczędność wykorzystując stan IDLE procesora AT89C51. Procesor w tym stanie nie wykonuje żadnych czynności, a jedynymi czynnymi układami wewnętrznymi są: układ przerwań, liczniki oraz układ transmisji szeregowej.

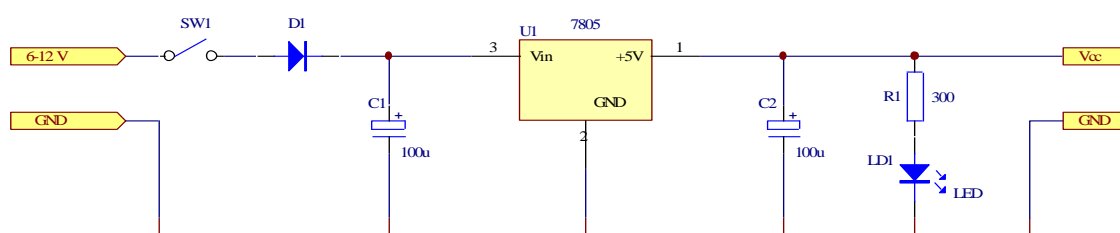
Układ sterowania (rysunek 2) składa się z procesora (U2), układów U7 i U8 generujących sygnały na magistrali sterującej oraz układu MAX232 (U6), służącego do konwersji napięć TTL / RS-232c. Procesor AT89C51 steruje całą pracą emulatora. Zajmuje się on odbieraniem i nadawaniem danych do komputera oraz kontrolą transmisji, a także generowaniem odpowiednich sygnałów sterujących pamięciami. Układy U3, U4 i U5 służą jako bufony magistrali lokalnej i podczas zapisu lub odczytu pamięci przez procesor są aktywne. Układ U3 oprócz buforowania magistrali lokalnej zatrzymuje, podczas dostępu procesora do pamięci, młodszą część adresu. Bufor danych (U5) zmienia kierunek przesyłania informacji za pomocą wejścia DIR. Gdy jest ono w stanie wysokim, bufor przesyła dane z wejść A do B, gdy sygnał DIR jest na poziomie logicznego zera, bufor przesyła dane z wejść B do A. Wejście DIR sterowane jest za pomocą sygnału RD procesora, ma on poziom niski w momencie odczytu, a w stanie nieaktywności pozostaje jedynką logiczną. Układy U10, U11 oraz U12 pracują jako bufony wyjściowe i są aktywne tylko podczas emulacji.



Rys. 2. Schemat ideowy układu sterującego emulatorem.



Rys. 3. Schemat ideowy płytki z pamięciami.



Rys. 4. Schemat układu zasilania.

Przy włączeniu zasilania następuje inicjacja procesora wraz z ustawieniem wszystkich parametrów transmisji szeregowej. Ustawiane są odpowiednie sygnały sterujące i zostaje sprawdzona pamięć RAM emulatora. Procedura sprawdzenia pamięci RAM najpierw wypełnia całą pamięć (64 kB) przypadkową wartością, a następnie odczytuje i porównuje wartość odczytaną z wartością wcześniej zapisywaną. Po sprawdzeniu każda komórka zostaje wyzerowana. Jeśli podczas sprawdzania pamięci okaże się, że jeden z układów pamięci jest uszkodzony, procesor przerywa pracę i sygnalizuje ten fakt poprzez miganie diodą LED (LD2). Jeśli natomiast wszystkie komórki pamięci są sprawne, to emulator przechodzi w stan gotowości. W tym stanie emulator oczekuje na polecenia wysyłane z komputera.

Dostęp procesora do pamięci

Pamięć RAM zbudowana jest na dwóch układach pamięci 62256 po 32 kB każdy (rysunek 3). Wyboru układu dokonuje najstarszy bit magistrali adresowej (A15'). Pamięci 62256 mają tylko 15-bitową magistralę adresową (od A0 do A14) w związku z tym należy wykorzystać sygnał CE (Chip Enable) do przełączania pomiędzy układami. Do układu U13, który pracuje jako pierwsze 32 kB (z przestrzeni 64 kB) sygnał A15' jest podłączony bezpośrednio do wyprowadzenia CE, natomiast do układu U14, który pracuje jako starszy blok 32 kB na wyprowadzenie CE podany jest zanegowany sygnał A15' czyli CE2. Jeśli procesor wystawi na magistrali adresowej adres mniejszy od 8000h to sygnał A15' będzie w stanie zera logicznego więc aktywna stanie się pamięć U13. W przypadku adresów powyżej 8000h sygnał A15' jest w stanie wysokim więc blokuje pamięć U13, a poprzez negację uaktywnia pamięć U14. Zastosowany sposób obsługi pamięci nosi nazwę przełączania banków pamięci. Tak połączone dwa układy pamięci RAM są widziane przez procesor jak zwykła pamięć RAM o pojemności 64 kB. Rozwiązanie to jednak ma pewną wadę - utraciliśmy sygnał CE, który pozwala na odłączenie całkowitej pamięci od systemu. Emulator musi obsłużyć do 128 kB pamięci (RAM + EPROM), aby tego dokonać całą pamięć zorganizowano w dwa bloki po 64 kB każdy. Pierwszym 64 kB blokiem jest pamięć RAM, a drugim blokiem jest pamięć EPROM dołączana poprzez podstawkę zewnętrzną. Rozwiązanie tu zastosowane nazwałem metodą dostępu do pamięci. Metoda ta nie przełącza banków pamięci sygnałami CE, tylko odpowiednio kieruje sygnał odczytu procesora (RD) do poszczególnych układów pamięci. Należy zauważyć, że przestrzeń adresowa pamięci różni się dla odczytu i zapisu. W przypadku zapisu wynosi ona 64 kB, natomiast w przypadku odczytu równa jest 128 kB. Dlatego metoda dostępu do pamięci dotyczy tylko cyklu odczytu pamięci. W cyklu zapisu zapisywana jest zawsze pamięć RAM.

Wyboru pamięci do odczytu dokonuje się za pomocą 3-bitowej magistrali sterującej, która służy również do ustawiania trybu symulacji. Znaczenie poszczególnych linii tej magistrali przedstawiono w tabeli 1. Istnieją trzy tryby dostępu do pamięci:

- tryb dostępu do pamięci RAM
- tryb dostępu do pamięci EPROM
- tryb symulacji (zewnętrzny dostęp do pamięci RAM)

Układ obsługi pamięci został zbudowany przy użyciu trzech buforów trójstanowych (U8A, U8C i U8D) aktywowanych niskim stanem, bramki NOT (U7A) oraz rezystora R4. Układ obsługi pamięci w zależności od trybu dostępu przesyła odpowiednie sygnały sterujące do pamięci.

GATE	SEL0	SEL1	Opis
0	0	1	Tryb symulacji
1	0	0	Tryb dostępu do pamięci ROM
1	1	1	Tryb dostępu do pamięci RAM

Tab. 1. Sygnały sterowania dostępem do pamięci.

W trybie dostępu do pamięci RAM sygnał RD jest przesyłany przez bufor U8D do wyprowadzeń OE (Output Enable) obu układów pamięci RAM. W tym czasie dwa pozostałe bufony sterujące (U8A i U8C) są nieaktywne (ich wyjścia są w stanie wysokiej impedancji).

Tryb symulacji jest bardzo podobny do poprzedniego. Różnica polega na źródle sygnału odczytu. W tym trybie aktywny jest bufor U8C, a sygnał odczytu (\sim CE) przychodzi z zewnątrz, czyli z wtyku emulacyjnego. Od strony systemu badanego sygnał \sim CE jest sygnałem selekcji pamięci w większości przypadków na stałe zwartym do masy, co powoduje ciągły odczyt pamięci RAM emulatora. Ma to istotny wpływ na czas dostępu do pamięci w trybie emulacji, ponieważ pomiędzy układem badanym a pamięcią RAM są jeszcze bufony magistrali zewnętrznej, które wprowadzają pewne opóźnienie (w serii HCT ok. 10 ns). Rzeczywisty sygnał odczytu pamięci z zewnątrz (\sim OE) jest sygnałem otwierającym bufor danych U12, w związku z tym nawet jeśli sygnał \sim CE będzie cały czas miał poziom logicznego „0”, to dane z pamięci emulatora nie będą dostępne na zewnątrz dopóki nie pojawi się niski poziom sygnału otwierającego bufor danych U12. Bufor U12 ma dwa wejścia uaktywniające go w związku z tym drugie wejście podłączone jest do sygnału OE1. Zapobiega to otwieraniu się bufora danych (U12), gdy nie wystąpił sygnał odczytujący pamięć RAM (OE1). Z punktu widzenia systemu badanego nie ma znaczenia co robi sygnał \sim CE, jest istotne tylko to, żeby dane pojawiły się na wyjściu w momencie gdy \sim CE i \sim OE są w stanie niskim.

Ostatnim trybem dostępu jest tryb odczytu pamięci EPROM. Pamięć ta umieszczona w zewnętrznej podstawce jest na stałe wybrana (sygnał CE pamięci ma poziom „0” logicznego), a impulsy odczytu przekazywane są poprzez bufor U8A, który jest aktywny tylko w tym trybie dostępu. Rezystor R4 służy do „podciągnięcia” wyjścia bufora U8A do plusa zasilania, w momencie gdy jest on nieaktywny, aby uniknąć przypadkowych odczytów pamięci EPROM.

Sposób komunikacji z komputerem PC

Jak już wspomniano procesor AT89C51 jest wyposażony w wewnętrzny układ transmisji szeregowej, który może pracować synchronicznie lub asynchronicznie. Oczywiście do komunikacji z komputerem należy użyć trybu asynchronicznego, ponieważ tylko taki typ pracy przewiduje port szeregowy komputera PC. Natomiast pozostałe parametry portu, tj. ilość bitów danych, stopu, występowanie bitu parzystości i prędkość transmisji, mogą być dowolne. W tym przypadku zdecydowano się na transmisję 8-bitową, bez bitu parzystości z maksymalną prędkością wynoszącą 57600 bodów (bitów / sekundę). Zrezygnowałem z bitu parzystości z dwóch powodów. Pierwszym powodem były przeprowadzone testy transmisji i stwierdziłem, że bit parzystości nie jest wymagany, gdyż błędy komunikacyjne nie wystąpiły. Drugim powodem był zastosowany protokół transmisji, który jest zaopatrzony w sumę kontrolną zabezpieczającą przed ewentualnymi przekłamaniami. Na rysunku 5 widać wygląd pojedynczego pakietu danych.

CODE 3 bajty	DATA 0 - 64 bajtów	CRC 2 bajty
------------------------	------------------------------	-----------------------

Rys. 5. Wygląd pojedynczego pakietu danych.

Poszczególne pola oznaczają:

CODE - kod rozkazu wraz z argumentami

DATA - przesyłane dane,

CRC - suma kontrolna pakietu.

Pole **CODE** zawiera zakodowaną operację, którą ma wykonać emulator. W tym polu znajduje się informacja o długości pola **DATA** oraz dodatkowe dane, np. adres, od którego rozpocznie się zapis. Pole **DATA** ma zmienną długość wynoszącą od 0 do 64 bajtów i zawiera dane przesyłane pomiędzy komputerem a emulatorem. W polu **CRC** znajduje się dwubajtowa suma kontrolna, jest to suma wszystkich bajtów pakietu czyli suma bajtów pola **CODE** i pola **DATA**. Po każdym przesłaniu pakietu rozpoczyna się nowy cykl rozkazowy. Emulator czeka na 3-bajtowy kod instrukcji (patrz tab. 2), a gdy go już odbierze następuje cykl przesyłania danych (jeśli rozkaz tego wymagał). W cyklu przesyłania danych emulator albo wysyła zawartość kolejnych komórek pamięci (RAM lub EPROM) albo odbiera dane do zapisu w pamięci RAM. Po zakończonym cyklu przesyłania danych następuje cykl nadawania sumy kontrolnej. Suma kontrolna jest liczona oddzielnie w komputerze PC i emulatorze, następnie emulator po skończeniu wykonywania rozkazu przesyła sumę kontrolną do komputera, gdzie następuje sprawdzenie poprawności sum kontrolnych. Jeśli suma jest błędna, to powtarza się przesyłanie tego samego pakietu. Po kilku (ustalonych programowo) nieudanych próbach transmisji, program sterujący informuje o kłopotach z transmisją. W tabeli 2 przedstawiono spis wszystkich wykonywanych przez emulator operacji wraz z niezbędnymi argumentami.

CODE			Długość danych	Kierunek danych		Opis
1 bajt	2 bajt	3 bajt		PC	Emulator	
01-40h	<i>AddrH</i>	<i>AddrL</i>	1-64		⇒	Zapis danych do pamięci RAM emulatora zaczynając od adresu: $Addr = AddrH * 256 + AddrL$
41-80h	<i>AddrH</i>	<i>AddrL</i>	1-64		⇐	Odczyt danych z pamięci emulatora zaczynając od adresu: $Addr = AddrH * 256 + AddrL$
A0h	-	-	0		x	Ustawienie trybu SYMULACJA
B0h	-	-	0		x	Ustawienie trybu dostępu do pamięci RAM
BBh	-	-	0		x	Ustawienie trybu dostępu do pamięci ROM
C0h	<i>Mode</i>		0		x	Ustawienie prędkości transmisji na tryb <i>Mode</i> (patrz tab. 2)
CCh	<i>Mode1</i>	<i>Mode2</i>	0		x	Ustawienie prędkości transmisji ręcznie
D0h	-	<i>AddrL</i>	1		⇒	Zapis jednego bajtu do pamięci RAM procesora pod adres <i>AddrL</i>
DDh	-	<i>AddrL</i>	1		⇐	Odczyt jednego bajtu z pamięci RAM procesora spod adresu <i>AddrL</i>
E0h	-	-	0		x	Ustawienie trybu POWER DOWN
F0h	-	-	2		⇐	Wysyła dwubajtowy numer wersji oprogramowania emulatora
F1h	-	-	x		⇐	Wysyła ciąg znaków będących logo emulatora
FFh	-	-	x		⇐	Wysyła listę z podziękowaniami

Tab. 2. Lista rozkazów emulatora pamięci EPROM.

Prędkość transmisji	Tryb
300 bps	0
600 bps	1
1200 bps	2
2400 bps	3
4800 bps	4
9600 bps	5
19200 bps	6
57600 bps	7

Tab. 3. Wykaz trybów prędkości transmisji.

Opis programu '51

Program sterujący pracą mikroprocesora AT89C51 znajduje się w wewnętrznej pamięci Flash EPROM. Został on napisany w taki sposób, aby możliwa była jego łatwa analiza i rozbudowa. W tabeli 4 zamieszczono opis wszystkich używanych komórek wewnętrznej pamięci RAM.

Adres	Opis
00h-07h	Rejestry robocze:
00h	Wskaźnik bufora instrukcji
01h	Licznik odebranych / nadanych bajtów w pakiecie
02h	Licznik bajtów H (starszy bajt)
03h	Licznik bajtów L (młodszy bajt)
04h	Adres docelowy H (starszy bajt)
05h	Adres docelowy L (młodszy bajt)
06h	Adres źródłowy H (starszy bajt)
07h	Adres źródłowy L (młodszy bajt)
08h-0Fh	Rejestry pomocnicze używane tylko podczas testu pamięci:
08h	Wartość zapisywana
09h	Wartość odczytywana
0Ah	Licznik bajtów H (starszy bajt)
0Bh	Licznik bajtów L (młodszy bajt)
0Ch	Adres docelowy H (starszy bajt)
0Dh	Adres docelowy L (młodszy bajt)
0Eh	Adres źródłowy H (starszy bajt)
0Fh	Adres źródłowy L (młodszy bajt)
10h-17h	Zmienne systemowe:
10h	Kod rozkazu
11h	Adres H
12h	Adres L
13h	Rozmiar pakietu danych
14h	Suma kontrolna H (starszy bajt)
15h	Suma kontrolna L (młodszy bajt)
16h	Bajt odebrany przez układ transmisji szeregowej
17h	Bajt do nadania przez układ transmisji szeregowej
18h-1Fh	Stos programowy
20h	Bit'y sterujące:
0	Znacznik identyfikacji kodu instrukcji
1	Znacznik odebrania kodu rozkazu
2	Znacznik odebrania starszego bajtu adresu
3	Znacznik odebrania młodszy bajtu adresu
4	Znacznik końca pakietu
5	Znacznik danych do nadania
6	Znacznik danych do odebrania
7	Znacznik dostępu do pamięci RAM procesora

Tab. 4. Mapa pamięci RAM procesora.

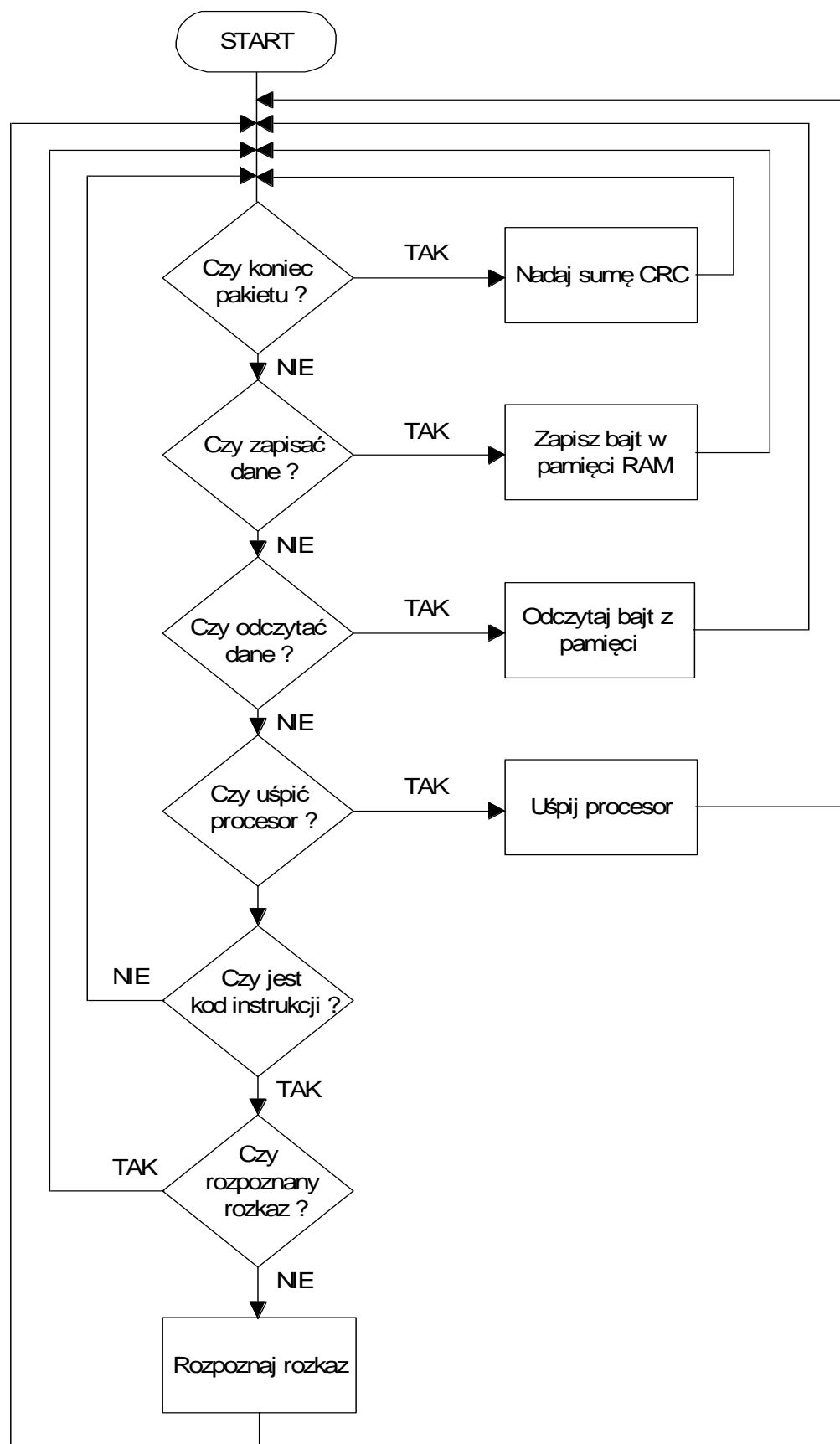
Pętla główna programu składa się z szeregu instrukcji sprawdzających stan odpowiednich znaczników. Na rysunku 6 zamieszczony jest algorytm programu głównego zawartego w pamięci procesora AT89C51. Poniżej przedstawiono fragment listingu programu zawierający główną pętlę programu.

```
-----
MAIN:
    JB 4, SENDCRC           ;czy wysłać CRC
    JB 6, LOADMEM           ;czy zapisać daną w pamięci
    JB 5, SENDMEM           ;czy odczytać pamięć
    JB 3, NOIDLE            ;czy uśpić procesor

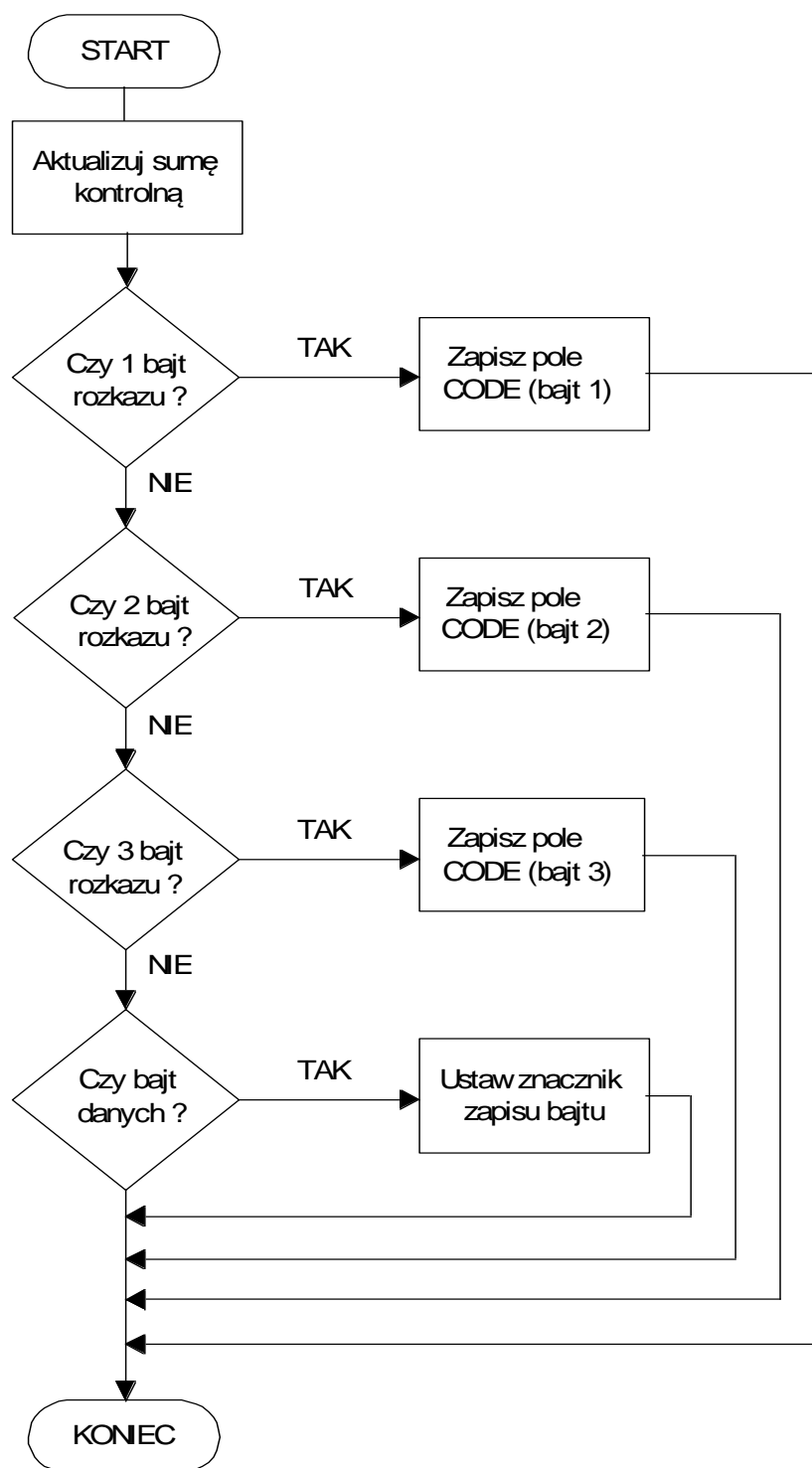
    MOV A, 87H              ;ustawienie procesora
    ORL A, #00000001B       ;w stan
    MOV 87H, A              ;IDLE
NOIDLE:
    JNB 3, MAIN             ;jeśli nie ma rozkazu to skok
    JB 0, MAIN              ;jeśli zdekodowany to też skok
    ACALL DECODE            ;dekoduj rozkaz
CHECK:
    MOV A, R1               ;sprawdzenie czy
    CJNE A, 13H, MAIN       ;nie nadano lub odebrano
    SETB 4                  ;wszystkich bajtów pakietu
    SJMP MAIN
-----
```

Jak widać po wykonaniu sprawdzenia znaczników (bity 4,6,5,3), następuje przejście procesora w stan uśpienia (IDLE). Gdy podczas stanu IDLE lub sprawdzania znaczników zostaną odebrane trzy bajty składające się na rozkaz emulatora, zostaje ustawiony znacznik 3, który zezwala na wywołanie procedury dekodującej rozkaz 'DECODE'. W zależności od rozkazu procedura 'DECODE' ustawia znacznik nadawania lub odbioru danych. Po wykonaniu tej procedury zostaje rozpoczęty cykl przesyłania danych. W przypadku ustawienia jednego ze znaczników (nadawanie lub odbieranie) zostaje wykonana odpowiadająca mu procedura, która kończąc wykonuje skok do etykiety 'CHECK', gdzie zostaje sprawdzona ilość nadanych lub odebranych bajtów pakietu. Gdy liczba bajtów nadanych lub odebranych będzie równa rozmiarowi pakietu zostaje ustawiony znacznik nadawania sumy kontrolnej. Po nadaniu sumy kontrolnej zostaje rozpoczęty nowy cykl oczekiwania na rozkaz.

Algorytm procedury obsługującej układ transmisji szeregowy procesora AT89C51 znajduje się na rysunku 7. Procedura ta odbiera kolejne bajty rozkazu umieszczając je w odpowiednich komórkach pamięci (patrz tab. 4) oraz oblicza sumę kontrolną po każdym odebraniu lub nadaniu bajtu przez łącze szeregowe. Procedura odbioru znaków poprzez port szeregowy oddziela również trój-bajtowy kod instrukcji emulatora od danych.



Rys. 6. Algorytm programu głównego '51.



Rys. 7. Algorytm procedury obsługi portu szeregowego w procesorze 89C51.

Opis programu 'EEPROM.EXE'

Opis deasemblera '51

Deassembler jest to program, który analizując kod binarny programu przedstawia go za pomocą nazw mnemonicznych asemblera. Innymi słowy jest to program zamieniający informację binarną na postać zrozumiałą dla programisty. Głównym powodem, dla którego zdecydowałem się na utworzenie deasemblera była jego przydatność podczas uruchamiania systemów opartych o procesor 8051.

Procesory z rodziny '51 mają 8-bitowy dekodery rozkazów w związku z tym mogą wykonywać najwyżej 256 różnych operacji. Jest to na pierwszy rzut oka dosyć dużo, ale uwzględniając różne tryby adresowania liczba rozkazów okazuje się nie taka wielka. Przykłady rozkazów procesora 8051 pokazane są w tabeli 5.

Postać mnemoniczna	Kod operacji	1 bajt argumentu	2 bajt argumentu
NOP	00h	-	-
MOV A, #50h	74h	50h	-
ANL 10h, #50h	52h	10h	50h

Tab. 5. Przykłady rozkazów 8051.

Procedura deasemblująca w programie EEPROM analizuje kod operacji pod względem podobieństwa argumentów rozkazu, a nie pod względem typu wykonywanej (przez procesor '51) operacji. Jest to znacznie szybsza metoda dekodowania rozkazów. Polega ona na wyszukiwaniu wszystkich rozkazów, które mają takie same argumenty (np. wszystkie skoki warunkowe krótkie), takie jak:

- JZ rel
- JNZ rel
- JC rel
- JNC rel
- SJMP rel

Mnemoniki rozkazów są pamiętane w tablicy. Gdy deassembler napotka nowy rozkaz do dekodowania najpierw wyświetla jego mnemonik, który jest niezmienny, a dopiero później dekoduje niezbędne argumenty. Jak łatwo zauważyć powyższe rozkazy mają taki sam argument 'rel' czyli adres względny. Procedura deasemblująca sprawdza czy rozkaz należy do podobnej grupy, a następnie po odpowiedniej konwersji wyświetla jego argument(y). Tym sposobem do zdekodowania 256 rozkazów procedura ma tylko kilkanaście różnych algorytmów przedstawienia wyników dekodowania programu (kompletnych instrukcji w asemblerze). Ważną informacją dla procedury dekodującej program '51 jest długość rozkazu. W zależności od tego jaki długi jest rozkaz, procedura deasemblująca pobiera (lub nie) odpowiednią ilość danych. Długości rozkazów oraz ich mnemoniki są zapisane w pliku, który przy uruchamianiu programu jest ładowany do odpowiednich tablic. Tablice te są później wykorzystywane przez procedurę deasemblującą. Kolejne elementy w tablicach zawierających długości rozkazów i ich mnemoniki są ułożone według kodu rozkazu procesora 8051. Dla przykładu rozkaz 'RET' ma kod 22h i jest bezargumentowy tzn. jego długość wynosi 1 bajt. Procedura deasemblująca po napotkaniu w buforze danych kodu rozkazu równego 22h odczytuje jego długość zapisaną pod adresem 22h w tablicy długości oraz jego mnemonik o tym samym adresie z tablicy mnemoników. W podanym przykładzie długość rozkazu odczytana z tablicy będzie równa jednemu bajtowi. W związku z tym za mnemonikiem 'RET' nie będą już umieszczone żadne argumenty. Jest istotne, aby

dekodując kolejny rozkaz nie pomylić go z argumentami. Jak widać w tabeli 5 rozkazy procesorów rodziny '51 mogą mieć od 1 do 3 bajtów długości. Łatwo zauważyć, że najszybciej dekodowane będą rozkazy 1-bajtowe, ponieważ do ich wyświetlenia nie potrzeba pobierać żadnych dodatkowych danych, a także nie trzeba wyświetlać argumentów. Najszybciej dekodowane będą również rozkazy, których kody są sprawdzane wcześniej w procedurze. Jednak opóźnienia spowodowane różną długością rozkazów, a także budową procedury deasemblującej są niezauważalne podczas pracy programu.

Sposób komunikacji z emulatorem

Program EEPROM do komunikacji z emulatorem wykorzystuje port szeregowy komputera PC. Nie stosuję standardowych procedur obsługi portu RS-232c zawartych w BIOS'ie (przerwanie 14h) z istotnej przyczyny: maksymalna prędkość transmisji oferowana przez te procedury wynosi tylko 9600 bps. Do obsługi portu szeregowego napisano specjalny zestaw procedur, które nie ograniczały prędkości transmisji oraz oferowały znacznie większe możliwości. Jedną ze specyficznych cech procedur obsługi układu transmisji szeregowej jest obsługa kolejki FIFO w układzie 16550.

Wiele nowszych komputerów PC jest już wyposażonych w ten właśnie UART (ang. **U**niversal **A**synchronous **R**eceiver and **T**ransmitter) lub w jego odpowiednik. Układ 16550 dzięki swojej 14-bajtowej kolejce FIFO umożliwia bezbłędne przeprowadzanie transmisji przy prędkości nawet 115 kbps. Jego poprzednik 16450 ma również wbudowaną kolejkę FIFO lecz jest ona niesprawna, mimo to układ ten potrafi komunikować się z tą samą prędkością co 16550, ale bardziej obciąża program obsługujący transfer danych.

Odczyt danych z portu szeregowego może być zrealizowany na dwa sposoby. Pierwszy, praktycznie uniemożliwiający przeprowadzenie sprawnej łączności z prędkością większą niż 9600 bps, polega na ciągłym sprawdzaniu czy został odebrany prawidłowy znak i w momencie gdy znak ten został odebrany w całości następuje jego zapis w pamięci. Drugi sposób angażuje system przerwań komputera, ale znacznie obciąża pracę programu głównego. Polega on na tym, że w momencie odebrania całego znaku układ transmisji szeregowej generuje przerwanie i dopiero procedura obsługi przerwania odpowiednio interpretuje odebrany znak. Ten ostatni sposób praktycznie wyklucza efekt gubienia odebranych znaków, który jest nie do uniknięcia w pierwszym przypadku.

Program EEPROM stosuje metodę z przerwaniami, ponieważ w większości przypadków transmisja przebiega z prędkością 57600 bps. Zrealizowano to w bardzo prosty sposób. Procedura, która chce odczytać porcję danych z emulatora najpierw inicjuje port szeregowy, następnie wysyła do emulatora rozkaz odczytu i oczekuje na znacznik końca pakietu danych. Znacznik ten jest ustawiany tylko w procedurze obsługi przerwania zgłaszanego przez układ transmisji szeregowej. Procedura ta odbiera cały pakiet zapisując go w buforze pomocniczym i rozdziela dane od sumy kontrolnej. Po zakończonej transmisji pojedynczego pakietu zostaje ustawiony znacznik, który odbiera procedura odczytująca i następuje sprawdzenie sumy kontrolnej. Gdy wszystko się zgadza następuje wysłanie rozkazu odczytu następnego pakietu danych. Natomiast w przypadku błędu ponawiana jest transmisja tego samego pakietu. Aby zapobiec zawieszeniu komputera w momencie oczekiwania na koniec pakietu jest także odliczany czas oczekiwania, gdy przekroczy on ustawioną programowo wartość, procedura odczytu kończy się generując komunikat błędu.

Zapis danych do emulatora nie wykorzystuje przerw, ponieważ program emulatora nie nadążałby z odbiorem danych wysyłanych z maksymalną prędkością 57600 bps. W związku z tym zapis jest dokonywany tradycyjną metodą kolejno wysyłanych bajtów w pętli programowej. Jedynym ważnym punktem tej metody jest fakt, że procedura zapisująca przed wysłaniem kolejnego bajtu do emulatora musi odczekać pewien odcinek czasu, który potrzebuje emulator na wykonanie wewnętrznych operacji (dekodowanie rozkazu, obliczenie sumy kontrolnej, itp.). Jest to czas ok. 600 μ s. W związku z tym, że program z założenia musi pracować na różnych komputerach (od 286 po Pentium II), to odczekanie tego odcinka czasu wymagałoby napisanie specjalnej procedury, która wykorzystywałaby np. uniwersalny licznik czasowy komputera. Zdecydowałem się jednak na użycie standardowej procedury opóźniającej zawartej w języku C++. Jedyną wadą tego rozwiązania jest niewykorzystanie maksymalnej prędkości przyjmowania danych przez emulator, ponieważ opóźnienie minimalne tej procedury wynosi 1 ms. Jest to jednak pewne zabezpieczenie przed błędnym zapisem danych.

Specyfikacja formatu Intel HEX II

Intel HEX II jest to format zapisu informacji binarnej przeznaczonej dla 8-bitowych procesorów. Składa się on z rekordów i pól. Każdy rekord w pliku zawiera następujące pola:

- **Start** - początek rekordu,
- **Length** - długość rekordu,
- **Address** - adres,
- **Type** - typ rekordu,
- **Data** - dane,
- **Checksum** - suma kontrolna,

Jeden bajt reprezentowany jest za pomocą pary znaków przedstawiającej ten znak w zapisie szesnastkowym (hexadecymalnym). Rekord rozpoczyna się zawsze od pola **Start** czyli od znaku ':' (dwukropek). Pole **Length** składa się z jednego bajtu (dwóch znaków) stanowiących długość pola **Data**. Pole **Address** ma długość dwóch bajtów i jest adresem absolutnym, pod który ma nastąpić zapis danych. W rekordzie zawierającym dane pole **Type** jest zawsze zerem. Ostatni bajt rekordu jest sumą kontrolną długości rekordu, adresu, typu rekordu i pola danych, czyli:

$$\text{Length} + \text{Address} + \text{Type} + \text{Data} + \text{Checksum} = 0$$

Każdy rekord kończy się znakiem nowej linii (CR + LF). Koniec pliku oznaczony jest rekordem kończącym, który zawsze wygląda następująco: ':00000001FF'.

Poniżej przedstawiony jest przykładowy plik w formacie Intel HEX II zawierający jeden rekord danych oraz rekord kończący:

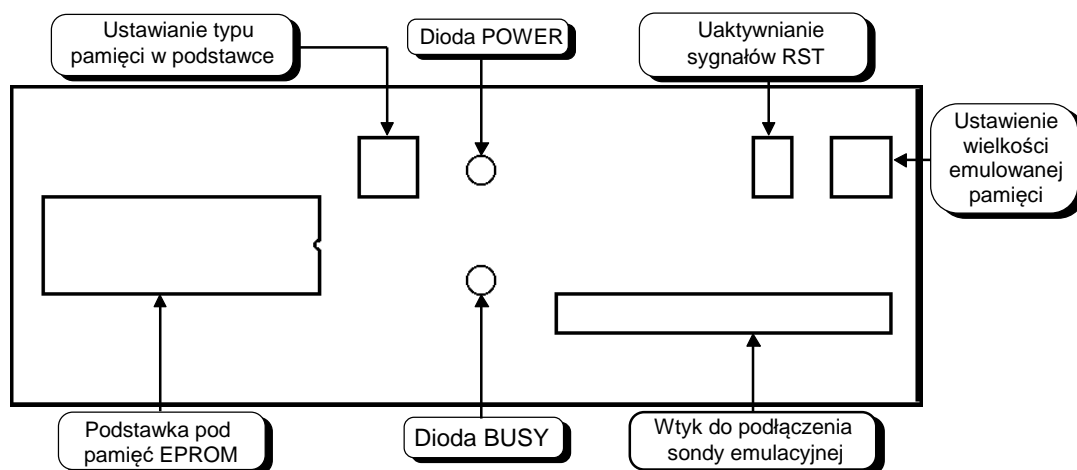
```
:04000B000200743249
:00000001FF
```

Rekord danych zawiera 4 bajty danych o czym świadczy wartość pola **Length**, która wynosi 04h. Adres początkowy **Address** wynosi 000Bh, a kolejne bajty danych mają wartości: 02h, 00h, 74h i 32h. Suma kontrolna wynosi 49h, bo $04h+00h+0Bh+00h+02h+00h+74h+32h+49h = 0$.

Instrukcja obsługi emulatora

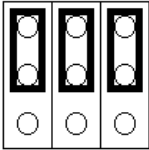
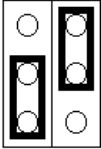
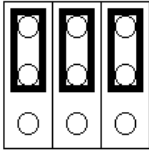
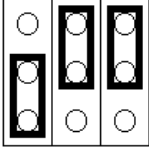
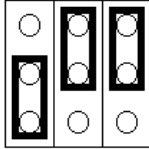
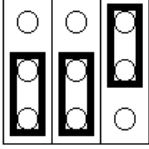


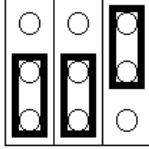
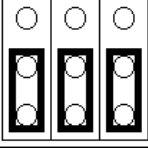
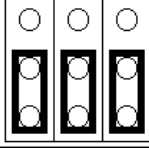
Przygotowanie emulatora do pracy

W celu rozpoczęcia pracy z emulatorem należy najpierw podłączyć go do portu szeregowego komputera PC za pomocą jednego z dołączonych przewodów, w zależności od tego jaki wtyk jest wolny w komputerze. Następnie należy podłączyć zasilacz. Emulator należy włączyć po wcześniejszym uruchomieniu komputera, ponieważ niektóre układy transmisji szeregowej komputera podczas włączania lub restartu wysyłają przypadkowe sygnały mogące zawiesić emulator. Wszelkie zmiany konfiguracji sprzętowej emulatora należy wykonywać przy wyłączonym zasilaniu emulatora, jak i układu badanego, jeśli jest podłączony.



Rys. 8. Widok ścianki przedniej emulatora.

Jeśli chcemy odczytać zawartość pamięci EPROM należy (przy wyłączonym zasilaniu) umieścić układ pamięci w podstawce, a następnie ustawić zworkami jego typ. Wygląd ścianki przedniej emulatora zamieszczony jest na rysunku 8, natomiast opis zwerek konfiguracyjnych znajduje się na rysunku 9. Do podłączenia emulatora z układem badanym służy wtyk emulacyjny. Wtyk należy podłączyć w taki sposób, aby czerwony przewód taśmy wskazywał pin 1 w podstawce pamięci układu testowanego. Numery pinów umieszczone są również na wtyku emulacyjnym. Po umieszczeniu wtyku w podstawce układu badanego należy, podobnie jak w przypadku pamięci EPROM, ustawić typ emulowanego układu za pomocą zwerek. Emulator może również generować sygnał resetujący (RST) układ testowany. Sygnał ten dostępny jest w postaci zwykłej i zanegowanej. Biały przewód oznacza sygnał RST aktywny stanem wysokim, tj. podczas symulacji sygnał ten jest na poziomie zera logicznego, natomiast czarny przewód oznacza sygnał RST aktywny stanem niskim, czyli podczas emulacji jest na poziomie jedynki logicznej. Jeśli będziemy chcieli wykorzystać sygnał RST, to należy podłączyć wejście RESET układu badanego do jednego z przewodów (biały lub czarny), a następnie uaktywnić odpowiedni sygnał za pomocą zwerek.

Konfiguracja typu pamięci w podstawce	Uaktywnianie sygnałów RST	Konfiguracja rozmiaru emulowanej pamięci
2764  8 kB	RST (czarny) RST (biały) 	 8 kB
27128  16 kB		 16 kB
27256  32 kB	 Włączony  Wyłączony	 32 kB
27512  64 kB		 64 kB

Rys. 9. Opis zworek konfiguracyjnych.

Jeśli po włączeniu zasilania emulatora miga czerwona dioda LED oznacza to, że wewnętrzna pamięć RAM jest uszkodzona. W takim przypadku dalsze używanie emulatora staje się niemożliwe. Należy wówczas skontaktować się z autorem pracy.

Po odpowiednim podłączeniu wszystkich przewodów i skonfigurowaniu sprzętu zworkami możemy włączyć emulator. Przy poprawnym skonfigurowaniu portu szeregowego w programie EEPROM.EXE komunikacja między komputerem a emulatorem przebiega bez zakłóceń, a podczas przesyłania danych pomiędzy komputerem a emulatorem świeci się czerwona dioda LED sygnalizując, że emulator jest zajęty. Dokładniejsze informacje na temat programu EEPROM.EXE można znaleźć niżej w tekście.

Wymagania i funkcje programu EEPROM

Program EEPROM służy do obsługi emulatora pamięci EPROM. Jest to program, którego wszystkie funkcje są wybierane za pomocą systemu menu i okienek dialogowych. Program został napisany w języku C++ i pracuje pod kontrolą systemu DOS 6.00 lub wyższym. Program EEPROM działa bez problemów w środowisku Windows 95/98 lecz polskie litery będą widoczne tylko podczas pracy pełnoekranowej. Minimalne wymagania sprzętowe to komputer 286 z wolnymi 360 kB pamięci RAM oraz kartą graficzną VGA. Jednak w celu zapewnienia maksymalnej prędkości przesyłania danych pomiędzy komputerem a emulatorem zalecane jest używanie minimum 386 / 40 MHz.

W niektórych komputerach niemożliwe będzie uzyskanie prędkości powyżej 9600 bps (bitów / sekundę), ponieważ istotną rolę odgrywa tu zainstalowany w komputerze układ transmisji szeregowej. Jeśli układ ten nie będzie układem zgodnym z 16450 lub 16550, to maksymalna prędkość transmisji może nie być zależna od szybkości komputera lecz właśnie od układu UART. Większość nowszych 486 i wszystkie komputery Pentium posiadają jednak układy zgodne z 16550, więc nie będą sprawiały problemów z komunikacją. W celu sprawdzenia jaki typ układu transmisji szeregowej jest zainstalowany w komputerze należy uruchomić program 'DETECT.EXE' zamieszczony na dyskietce. Program ten wyświetla listę wszystkich dostępnych portów szeregowych oraz typ zastosowanego układu transmisji szeregowej.

Instalacja programu EEPROM

Istnieją dwa sposoby instalacji programu EEPROM.

Pierwszy zalecany polega na włożeniu dyskietki instalacyjnej do stacji dysków i ponownym uruchomieniu komputera. Jeśli komputer pracuje pod kontrolą systemu DOS wystarczy nacisnąć kombinację klawiszy <Ctrl>+<Alt>+ lub wcisnąć przycisk RESET na obudowie komputera. Jeśli komputer pracuje pod kontrolą Windows 95/98 to należy w menu **Start** wybrać **Zamknij system**, a następnie **Uruchom ponownie komputer**. Po uruchomieniu programu instalacyjnego z dyskietki zostaną skopiowane wszystkie pliki programu na dysk twardy do katalogu 'C:\EEPROM'. Po zakończeniu instalacji program poprosi o wyjęcie dyskietki ze stacji dysków i ponowne uruchomienie komputera.

Drugim sposobem instalacji jest ręczne rozpakowanie pliku 'DATA.000' do dowolnego katalogu na dysku twardym. Plik ten jest spakowany za pomocą programu ARJ i należy rozpakować go za pomocą polecenia:

ARJ x A:\DATA.000 'nazwa_katalogu',

gdzie 'nazwa_katalogu' oznacza ścieżkę do istniejącego katalogu, w którym chcemy umieścić program EEPROM.

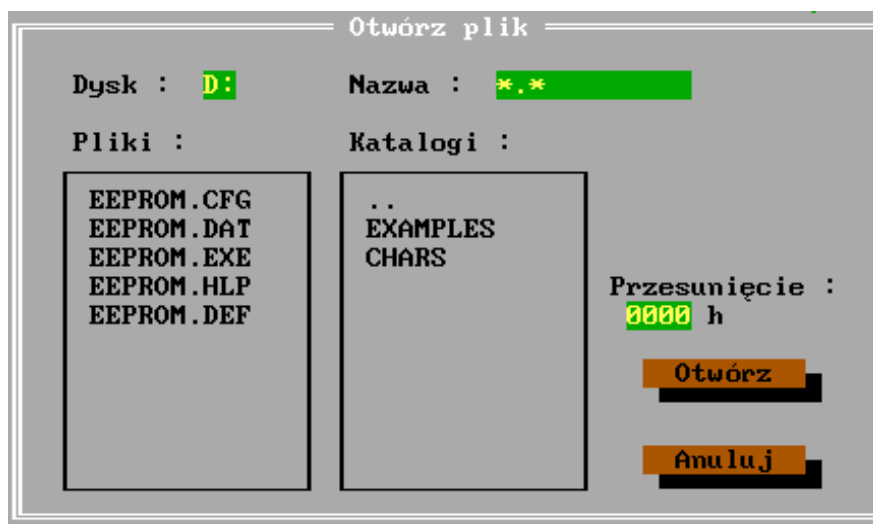
Charakterystyka okienek dialogowych programu EEPROM

W celu uaktywnienia menu programu należy nacisnąć klawisz <F9>. Wyboru menu dokonuje się klawiszami strzałek oraz klawiszem <ENTER>. Po wszystkich elementach okienek dialogowych programu EEPROM poruszamy się za pomocą klawiszy <TAB> (kolejny element okienka) i <Shift> + <TAB> (poprzedni element okienka). Zatwierdzenia wpisanej wartości w polu dokonuje się klawiszem <ENTER>, natomiast klawisz <ESC> przerywa aktualnie wykonywaną operację lub powraca do okna głównego programu. Część funkcji programu dostępna jest za pomocą kombinacji klawiszy, które są umieszczone obok nazwy funkcji w menu.

Menu PLIK

W tym menu dostępne są funkcje dotyczące plików, a także wyjście z programu.

1. Menu **PLIK / Otwórz** służy do otwierania plików ładowanych do bufora lub plików deklaracji etykiet.



Rys. 8. Widok okienka *Otwórz*.

W okienku **Otwórz** znajdują się następujące pola:

- **Nazwa** - służy do wpisywania nazwy pliku, który chcemy otworzyć. Możliwe jest wpisywanie nazw ogólnych (np. *.*). Pliki wówczas wszystkie pasujące do podanej nazwy pliki zostaną wyświetlone w okienku *Pliki*. Pole to służy również do zmiany aktualnie wyświetlanego dysku. W celu zmiany dysku należy wpisać jego nazwę zakończoną dwukropkiem i nacisnąć klawisz Enter (np. 'C: <ENTER>').
- **Dysk** - wyświetla aktualnie przeglądany dysk. Zmiana dysku możliwa jest w polu *Nazwa*.
- **Katalogi** - wyświetla listę katalogów aktualnie przeglądanych dysku.
- **Pliki** - wyświetla listę plików otwartego katalogu.
- **Przesunięcie** - w tym polu możemy wpisać przesunięcie z jakim zostanie załadowany plik. To pole nie ma znaczenia dla plików w formacie Intel HEX II (pliki z rozszerzeniem .HEX). W przypadku wszystkich pozostałych plików liczba szesnastkowa wpisana w tym miejscu będzie traktowana jak początek bufora, czyli od tego miejsca zacznie się ładowanie pliku. Gdy plik podczas ładowania przekroczy adres bufora FFFFh to dalsza część pliku ładowana będzie od adresu 0000h. Maksymalny rozmiar ładowanego pliku nie może jednak przekroczyć 65536 bajtów, dane w pliku znajdujące się powyżej tej granicy będą pominięte.

2. Menu **PLIK / Zachowaj jako** służy do zachowania zawartości bufora na dysku w podanym pliku. Okienko **Zachowaj jako** funkcjonalnie działa tak samo jak okienko **Otwórz**, ale nie posiada pola *Przesunięcie*.



Rys. 9. Widok okienka **Zachowaj jako**.

3. Menu **PLIK / Zachowaj** zachowuje zawartość bufora w pliku, który został ostatnio otwarty. Wykonanie tej operacji nie jest poprzedzone żadnym komunikatem, w związku z tym jeśli podczas pracy programu zmniejszymy rozmiar bufora danych i zapiszemy plik za pomocą opcji **Zachowaj** to możemy utracić dane znajdujące się w niewidocznej części bufora.
4. Menu **PLIK / Koniec** służy do zakończenia pracy programu. Wszelkie nie zapisane informacje zostaną utracone. Po wyjściu z programu nie jest zmieniana zawartość pamięci ani stan pracy emulatora.

Menu EDYCJA

W tym menu dostępne są operacje dotyczące bufora, takie jak: kopiowanie, przesuwanie, wypełnianie itp.

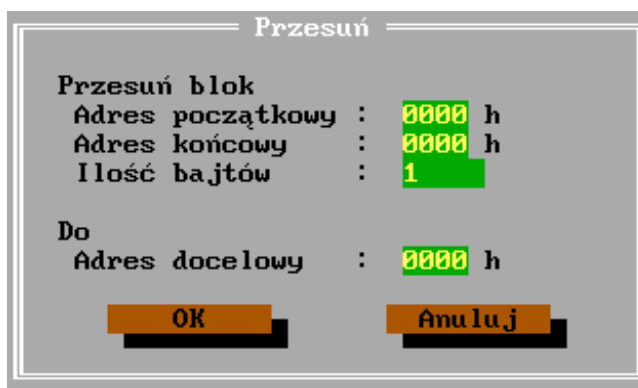
1. Menu **EDYCJA / Kopiuj** służy do kopiowania dowolnego bloku bufora w inne miejsce.



Rys. 10. Widok okienka **Kopiuj**.

Okienko **Kopiuj** zawiera następujące elementy:

- *Adres początkowy* - pole to służy do wpisywania adresu początku bloku, który chcemy skopiować.
 - *Adres końcowy* - pole to służy do wpisywania adresu końca bloku bufora, który chcemy skopiować. Gdy wartość tego pola jest mniejsza od wartości wpisanej do pola *Adres początkowy*, to automatycznie pole to przyjmuje wartość pola *Adresu początkowego*.
 - *Ilość bajtów* - pole to służy do wpisywania wielkości (w bajtach) bloku, który będziemy kopiować. Można to pole pominąć jeśli wpisaliśmy właściwy adres w polu *Adres końcowy*.
 - *Adres docelowy* - w polu tym wpisujemy adres, pod który zostanie skopiowany blok określony wyżej.
2. Menu **EDYCJA / Przesuń** służy do przesuwania dowolnego bloku bufora w inne miejsce.

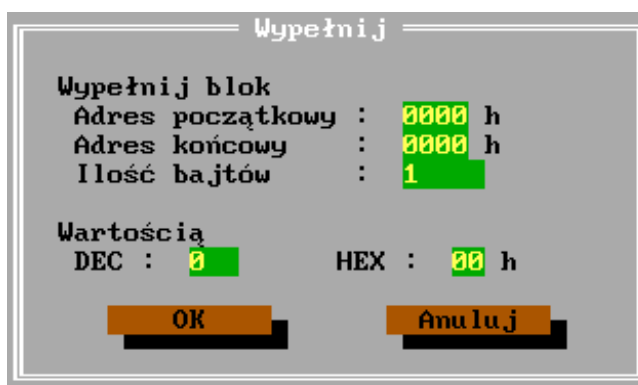


Rys. 11. Widok okienka **Przesuń**.

Okienko **Przesuń** zawiera następujące elementy:

- *Adres początkowy* - pole to służy do wpisywania adresu początku bloku, który chcemy przesunąć.
- *Adres końcowy* - pole to służy do wpisywania adresu końca bloku bufora, który chcemy przesunąć. Gdy wartość tego pola jest mniejsza od wartości wpisanej do pola *Adres początkowy*, to automatycznie pole to przyjmuje wartość pola *Adresu początkowego*.
- *Ilość bajtów* - pole to służy do wpisywania wielkości (w bajtach) bloku, który będziemy przemieszczać. Można to pole pominąć jeśli wpisaliśmy właściwy adres w polu *Adres końcowy*.
- *Adres docelowy* - w polu tym wpisujemy adres, pod który zostanie przeniesiony blok określony wyżej. Blok określony poprzez *Adres początkowy* i *Adres końcowy* po przeniesieniu zostaje wypełniony wartością 00h.

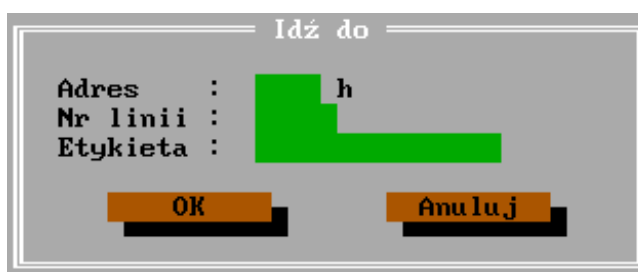
3. Menu **EDYCJA / Wypełnij** służy do wypełniania bufora podaną wartością.



Rys. 12. Widok okienka *Wypełnij*.

Okienko *Wypełnij* składa się z następujących elementów:

- *Adres początkowy* - pole to służy do wpisywania adresu początku bloku, który będziemy wypełniać.
 - *Adres końcowy* - pole to służy do wpisywania adresu końca bloku bufora, który będziemy wypełniać. Gdy wartość tego pola jest mniejsza od wartości wpisanej do pola *Adres początkowy*, to automatycznie pole to przyjmuje wartość pola *Adresu początkowego*.
 - *Ilość bajtów* - pole to służy do wpisywania wielkości (w bajtach) bloku, który będziemy wypełniać. Można to pole pominąć jeśli wpisaliśmy właściwy adres w polu *Adres końcowy*.
 - *DEC* - w polu tym wpisujemy wartość dziesiętną, którą zostanie wypełniony blok bufora określony powyżej.
 - *HEX* - w polu tym wpisujemy wartość szesnastkową, którą zostanie wypełniony blok bufora określony powyżej.
4. Menu **EDYCJA / Wyczyść bufor** służy do wyczyszczenia zawartości całego bufora i wypełnienia go wartością 00h. Czyszczony jest cały bufor niezależnie od jego aktualnego rozmiaru.
5. Menu **EDYCJA / Wyczyść etykiety** służy do skasowania wszystkich załadowanych aktualnie etykiet i przywrócenia etykiet ładowanych domyślnie.
6. Menu **EDYCJA / Idź do** służy do przemieszczania kursora w podane miejsce.



Rys. 13. Widok okienka *Idź do*.

Okienko **Idź do** składa się z następujących elementów:

- *Adres* - pole to służy do wpisywania adresu, pod który chcemy przesunąć kursor.
- *Nr linii* - pole to służy do wpisywania numeru linii, która ma być zaznaczona kursorem. Pole to jest aktywne tylko w momencie gdy aktualnym widokiem jest deassembler.
- *Etykieta* - pole to służy do wpisywania nazwy etykiety adresowej, pod którą chcemy skoczyć. Gdy podana nazwa nie istnieje lub jest błędna zgłaszany jest komunikat błędu w nazwie.

Menu WIDOK

W tym menu możliwa jest zmiana widoku programu. Dostępne są cztery różne widoki, w których każdy w inny sposób przedstawia dane bufora.

1. Menu **WIDOK / DeAssembler** włącza widok deassemblera. W tym trybie nie jest możliwa edycja zawartości bufora. Okno deassemblera podzielone jest na trzy kolumny. W pierwszej kolumnie od lewej umieszczone są numery linii poszczególnych instrukcji programu. Instrukcja znajdująca się pod adresem 0000h jest pierwszą linią programu. W drugiej kolumnie licząc od lewej znajdują się poszczególne rozkazy z argumentami poprzedzone adresem lub etykietą danego rozkazu. W ostatniej (prawej) kolumnie umieszczona jest reprezentacja binarna poszczególnych rozkazów programu.
2. Menu **WIDOK / Edytor HEX 16** włącza widok pierwszego edytora szesnastkowego. Edytor ten mieści 16 kolejnych komórek bufora w linii. Oprócz tego umieszczona jest również kolumna zawierająca znaki w kodzie ASCII. W tym edytorze po naciśnięciu klawisza <ENTER> na wskazanej kursorem komórce bufora możliwa staje się edycja zawartości. Wpisywana liczba jest w kodzie szesnastkowym.
3. Menu **WIDOK / Edytor HEX 8** włącza inną wersję edytora szesnastkowego. Edytor ten mieści 8 bajtów w linii, ale oprócz tego wyświetla również etykiety adresowe komórek oraz ich postać binarną i ASCII. Jak w poprzednim przypadku możliwa jest edycja szesnastkowa komórek bufora.
4. Menu **WIDOK / Mapa bitowa** włącza widok prezentujący dane w formie mapy bitów. Dane są umieszczone w trzech kolumnach w postaci binarnej. Zero logiczne przedstawia nie zamalowana kropka, natomiast jedynkę logiczną oznacza wypełniona kropka. Edycja jest możliwa po naciśnięciu klawisza <ENTER>, a zmiana wartości bitu za pomocą spacji lub strzałek w górę i w dół. Wyboru zmienianego bitu dokonuje się za pomocą strzałek w lewo i w prawo. W celu wyzerowania całej edytowanej komórki należy nacisnąć klawisz , natomiast jeśli podczas edycji będziemy chcieli zrezygnować z wprowadzonych zmian to należy nacisnąć klawisz <ESC>, przywrócona zostanie wówczas wartość komórki jaka była przed edycją.

Menu SYMULACJA

W tym menu dostępne są wszystkie funkcje dotyczące działania emulatora.

1. Menu **SYMULACJA / Zapisz pamięć** służy do zapisywania pamięci emulatora zawartością bufora.



Rys. 14. Widok okienka *Zapisz pamięć*.

Okienko *Zapisz pamięć* składa się z następujących elementów:

- *Adres początkowy* - pole to służy do wpisywania adresu początku bloku, który chcemy przesłać do pamięci emulatora.
 - *Adres końcowy* - pole to służy do wpisywania adresu końca bloku bufora, który będziemy zapisywać w emulatorze. Gdy wartość tego pola jest mniejsza od wartości wpisanej do pola *Adres początkowy*, to automatycznie pole to przyjmuje wartość pola *Adresu początkowego*.
 - *Ilość bajtów* - pole to służy do wpisywania wielkości (w bajtach) bloku, który będziemy przysyłać. Można to pole pominąć jeśli wpisaliśmy właściwy adres w polu *Adresu końcowy*.
 - *Adres docelowy* - w polu tym wpisujemy adres, pod który zostanie zapisany blok określony wyżej. Adres ten będzie fizycznym adresem początku danych widzianych od strony wtyku emulacyjnego.
2. Menu *SYMULACJA / Odczytaj pamięć RAM* służy do odczytywania pamięci RAM emulatora i zapisania jej w buforze.



Rys. 15. Widok okienka *Odczytaj pamięć RAM*.

Okienko **Odczytaj pamięć RAM** składa się z następujących elementów:

- *Adres początkowy* - pole to służy do wpisywania adresu początku bloku, który chcemy odczytać z pamięci RAM emulatora.
 - *Adres końcowy* - pole to służy do wpisywania adresu końca bloku bufora, który będziemy odczytywać z emulatora. Gdy wartość tego pola jest mniejsza od wartości wpisanej do pola *Adres początkowy*, to automatycznie pole to przyjmuje wartość pola *Adresu początkowego*.
 - *Ilość bajtów* - pole to służy do wpisywania wielkości (w bajtach) bloku, który będziemy przysyłać z pamięci RAM emulatora. Można to pole pominąć jeśli wpisaliśmy właściwy adres w polu *Adres końcowy*.
 - *Adres docelowy* - w polu tym wpisujemy adres bufora, pod który zostanie zapisany blok pamięci RAM odczytany z emulatora.
3. Menu **SYMULACJA / Odczytaj pamięć ROM** służy do odczytywania pamięci EPROM emulatora umieszczonej w podstawce i zapisania jej w buforze.



Rys. 16. Widok okienka **Odczytaj pamięć ROM**.

Okienko **Odczytaj pamięć ROM** składa się z następujących elementów:

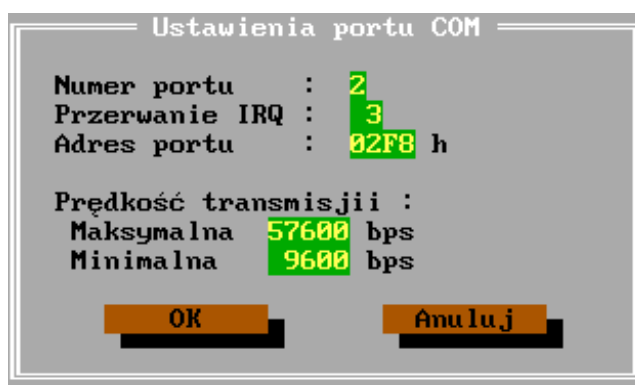
- *Adres początkowy* - pole to służy do wpisywania adresu początku bloku, który chcemy odczytać z pamięci EPROM emulatora.
 - *Adres końcowy* - pole to służy do wpisywania adresu końca bloku bufora, który będziemy odczytywać z emulatora. Gdy wartość tego pola jest mniejsza od wartości wpisanej do pola *Adres początkowy*, to automatycznie pole to przyjmuje wartość pola *Adresu początkowego*.
 - *Ilość bajtów* - pole to służy do wpisywania wielkości (w bajtach) bloku, który będziemy przysyłać z pamięci EPROM emulatora. Można to pole pominąć jeśli wpisaliśmy właściwy adres w polu *Adres końcowy*.
 - *Adres docelowy* - w polu tym wpisujemy adres bufora, pod który zostanie zapisany blok pamięci EPROM odczytany z emulatora.
4. Menu **SYMULACJA / Uruchom symulację** służy do rozpoczęcia emulacji pamięci. Po wykonaniu tego polecenia na wtyku emulacyjnym dostępna jest zawartość pamięci RAM emulatora oraz sygnał RST emulatora przechodzi w stan nieaktywny.
5. Menu **SYMULACJA / Przerwij symulację** służy do zatrzymania emulacji pamięci. Po wykonaniu tego polecenia nie jest możliwe odczytywanie pamięci RAM poprzez wtyk emulacyjny, a także sygnał RST emulatora przechodzi w stan aktywny.

6. Menu **SYMULACJA / Resetuj** służy do wystawienia sygnału RST. Po wykonaniu tego polecenia na złączach RST (czarny i biały przewód zakończony krokodylkiem) pojawia się sygnał resetujący o czasie trwania ok. 500 ms. Podczas gdy aktywny jest sygnał RST na wtyku emulacyjnym pamięć jest niedostępna w związku z tym jeśli sygnał RST nie jest podłączony do badanego układu może to spowodować jego niewłaściwą pracę.

Menu USTAWIENIA

W tym menu możliwa jest zmiana konfiguracji programu.

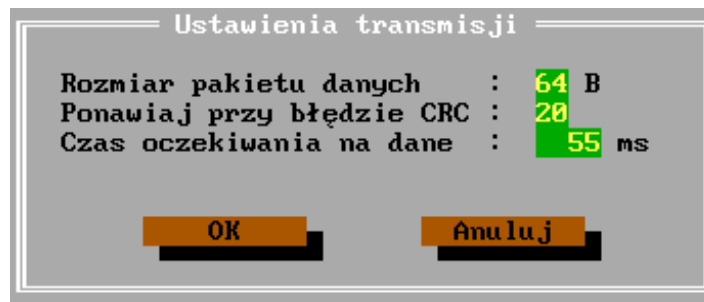
1. Menu **USTAWIENIA / COM** służy do zmiany parametrów portu szeregowego komputera.



Rys. 17. Widok okienka *Ustawienia portu COM*.

Okienko **Ustawienia portu COM** składa się z następujących elementów:

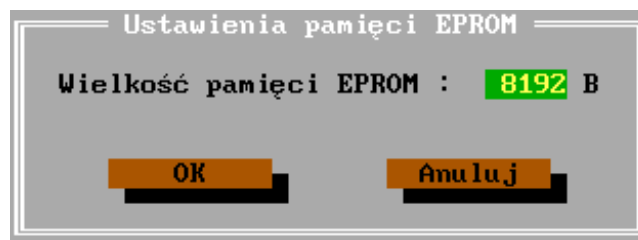
- *Numer portu* - pole to służy do wpisywania numeru portu szeregowego komputera, do którego jest podłączony emulator pamięci EPROM. Przy zmianie wartości tego pola program automatycznie wypełnia pola *Przerwanie* i *Adres portu* w zależności od bieżącej konfiguracji systemu. Oczywiście można powyższe ustawienia zmienić, ale w większości przypadków są one poprawne.
 - *Przerwanie* - pole to służy do wpisywania numeru przerwania obsługiwanego przez port szeregowy komputera.
 - *Adres portu* - w polu tym umieszcza się adres portu COM komputera.
 - *Maksymalna* - prędkość transmisji, w tym miejscu należy wpisać maksymalną prędkość z jaką będzie możliwe przesyłanie danych pomiędzy komputerem a emulatorem. Prędkość ta nie może przekroczyć 57600 bps (bitów / sekundę).
 - *Minimalna* - prędkość transmisji, pole to służy do wpisania minimalnej prędkości transmisji. W przypadku błędów transmisji przy prędkości *Maksymalnej* szybkość przesyłania danych zostanie zredukowana do tej wartości. Wartość tego pola musi być mniejsza od wartości wpisanej w polu *Maksymalna*, w przeciwnym razie pole maksymalna będzie miało taką samą wartość jak pole *Minimalna*.
2. Menu **USTAWIENIA / Transmisja** służy do zmiany parametrów dotyczących protokołu transmisji danych.



Rys. 18. Widok okienka *Ustawienia transmisji*.

Okienko *Ustawienia transmisji* składa się z następujących elementów:

- *Rozmiar pakietu danych* - pole to służy do wpisywania rozmiaru pojedynczego pakietu danych. Wartość tego pola to liczba danych przesyłana w pojedynczym pakiecie. Liczba ta nie może przekroczyć 64 bajtów. Generalnie im większa wielkość pakietu tym szybciej przesyłane są dane. Mały rozmiar pakietu przydatny może być w momencie, gdy emulator jest podłączony do komputera za pomocą długiego przewodu lub gdy znajduje się w warunkach przemysłowych, gdzie występują silne zakłócenia zewnętrzne.
 - *Ponawiaj przy błędzie CRC* - w polu tym umieszcza się wartość stanowiącą liczbę powtórzeń transmisji w przypadku napotkania błędu sumy kontrolnej. Do poprawnej pracy wystarczające są trzy powtórzenia, ale zwiększenie ilości powtórzeń do 20 daje pewność transmisji nawet podczas krótkotrwałych przerw w połączeniu pomiędzy komputerem a emulatorem.
 - *Czas oczekiwania na dane* - jest to czas, jaki może maksymalnie upłynąć dopóki program nie stwierdzi braku odpowiedzi ze strony emulatora. W przypadku zawieszenia lub wyłączenia emulatora podczas transmisji danych, po upływie wpisanego tu odcinka czasu zostanie stwierdzony błąd. Wartość wpisywana jest zaokrąglana do całkowitej wielokrotności 55 ms. W większości przypadków wystarczy czas od 55 do 110 ms. Zbyt duża wartość czasu oczekiwania wydłuża czas reakcji komputera na błąd.
3. Menu **USTAWIENIA / EPROM** służy do zmiany wielkości bufora danych. Zmniejszenie bufora nie powoduje utraty jego zawartości.



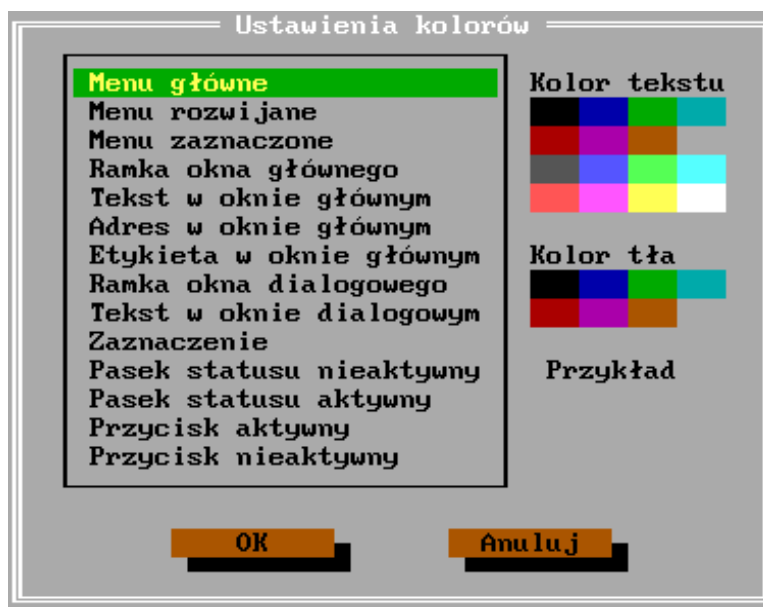
Rys. 19. Widok okienka *Ustawienia pamięci EPROM*.

Okienko *Ustawienia pamięci EPROM* składa się z następujących elementów:

- *Wielkość pamięci EPROM* - pole to służy do wpisywania rozmiaru bufora danych. Zmiana wielkości ma wpływ na wielkość wyświetlanego bufora oraz na wielkość zachowywanych danych w pliku. W przypadku operacji typu **Kopiuj**, **Przesuń**, **Zapisz / Odczytaj pamięć** wartość wpisana w tym polu nie ma znaczenia. Generalnie pole to służy do zmniejszenia widocznego obszaru bufora oraz zmniejszenia ilości danych zapisywanych w pliku. Minimalny rozmiar wynosi 2048 bajtów, natomiast

maksymalny 65536 bajtów. Rozmiar bufora jest powiększany do całkowitej wielokrotności liczby 16.

4. Menu **USTAWIENIA / Kolory** służy do zmiany kolorów programu. Zmiany widoczne są dopiero po ponownym uruchomieniu programu.



Rys. 20. Widok okienka *Ustawienia kolorów*.

Okienko *Ustawienia kolorów* składa się z następujących elementów:

- *Lista* - w tym polu dostępna jest lista elementów programu, których kolor możemy zdefiniować.
 - *Kolor tekstu* - pole to służy do ustawienia koloru tekstu wybranego elementu z *Listy*.
 - *Kolor tła* - w polu tym dokonuje się wyboru tła dla zaznaczonego elementu z *Listy*.
 - *Przykład* - pole to zmienia kolor w zależności od wybieranych wyżej ustawień.
5. Menu **USTAWIENIA / Zachowaj** służy do zachowania bieżących ustawień programu. Zachowywane są wszystkie parametry ustawione w menu **USTAWIENIA** oraz aktualny widok programu.

Menu POMOC

W tym menu dostępna jest pomoc programu, informacje dotyczące pamięci oraz informacje o autorze programu. Do poruszania się po tekście pomocy służą następujące klawisze:

- strzałki w górę i w dół - przesuwanie tekstu o jedną linię w górę lub w dół,
- PgUp - przewinięcie okna o jeden ekran do góry,
- PgDown - przewinięcie okna o jeden ekran w dół,
- Home - przewinięcie okna do początku tekstu,
- End - przewinięcie okna na koniec tekstu,
- ESC - wyjście z pomocy i powrót do okna głównego programu.

1. Menu **POMOC / Instrukcja obsługi** zawiera spis wszystkich klawiszy funkcyjnych programu wraz z ich opisem.
2. Menu **POMOC / Pamięci EPROM** zawiera krótką charakterystykę pamięci, ich podział oraz rozmieszczenie sygnałów najczęściej stosowanych pamięci.

Spis klawiszy funkcyjnych

Klawisz	Opis działania
Klawisze edycyjne	
Strzałki	Przesuwanie kursora w obrębie widoku
Enter	Przejdźcie w tryb edycji (z wyjątkiem deasemblera), w okienkach dialogowych potwierdzenie wprowadzonej zmiany
Esc	Anulowanie wykonywanej operacji
Tab	Przejdźcie do następnego elementu okienka dialogowego
Shift + Tab	Przejdźcie do poprzedniego elementu okienka dialogowego
PgUp	Przewinięcie zawartości okna o jeden ekran do góry
PgDown	Przewinięcie zawartości okna o jeden ekran w dół
Ctrl + PgUp	Przewinięcie zawartości okna o 4 kB (deasemblera o 100 linii) w dół
Ctrl + PgDown	Przewinięcie zawartości okna o 4 kB (deasemblera o 100 linii) w górę
Home	Ustawienie kursora u góry ekranu
End	Ustawienie kursora na dole ekranu
Ctrl + Home	Wyświetlenie zawartości okna od początku pamięci
Ctrl + G	Przemieszcza kursor pod wskazane miejsce
Operacje na buforze	
Ctrl + R	Przesuń
Ctrl + K	Kopiuuj
Ctrl + F	Wypełnij
Ctrl + D	Wyczyść zawartość całego bufora
Zmiana trybu podglądu	
F5	DeAsembler '51
F6	Edytor HEX (16 bajtów w linii)
F7	Edytor HEX (8 bajtów w linii)
F8	Edytor bitowy (mapa bitowa)
Operacje na plikach	
F2	Zachowaj
Shift + F2	Zachowaj jako
F3	Otwórz
Inne operacje	
Ctrl + F10	Uruchom symulację
Ctrl + F11	Zatrzymaj symulację
Ctrl + F12	Resetuj układ badany
F1	Pomoc programu
Shift + F1	O pamięciach EPROM
Alt + X	Wyjście z programu

Format pliku definicji etykiet

W pliku definicji etykiet zawarte są deklaracje etykiet. Etykieta to ciąg znaków, który zostanie użyty przez deassembler w momencie napotkania adresu jaki został przypisany tej etykiecie. Istnieją trzy rodzaje etykiet:

- etykiety adresowe
- etykiety danych
- etykiety bitowe

Etykiety adresowe służą do nazywania adresów w kodzie programu. Są one przydatne w rozbudowanych programach zawierających dużą ilość procedur. Możemy wówczas zdefiniować etykiety, które będą wskazywały na adresy użytych procedur. Etykiety danych natomiast służą do nazywania adresów wewnętrznej pamięci RAM

procesora. Ostatnim typem etykiet są etykiety bitowe, które służą do nazywania bitów przestrzeni adresowalnej bitowo procesora jak i bitów specjalnych (np. RI, TI itp.).

Plik definicji etykiet musi mieć rozszerzenie '.DEF' i jest plikiem tekstowym, w którym format deklaracji etykiety ma postać:

<TYP> <ADRES> <NAZWA>

Pola oddzielone muszą być przynajmniej jedną spacją, a kolejne deklaracje etykiet muszą być w oddzielnych liniach. Pole TYP może przyjmować jedną z trzech postaci: 'C', 'D', 'B', gdzie:

- C - oznacza etykietę adresową
- D - etykietę danych
- B - etykietę bitową

W polu ADRES musi być liczba w kodzie szesnastkowym. W przypadku etykiet adresowych pole to jest liczbą 16-bitową, a w pozostałych typach etykiet pole to jest liczbą 8-bitową. W polu nazwa musi znajdować się ciąg znaków nieprzerwanych spacjami o długości nie większej od 15 znaków dla etykiet adresowych i 10 znaków dla pozostałych typów etykiet. Gdy ciąg będzie dłuższy od podanej granicy pozostałe znaki nie będą brane pod uwagę. Gdy pierwszym lub jedynym znakiem nazwy jest '*' (gwiazdka) to etykieta o podanym adresie zostanie usunięta. W ten sposób można wymusić pokazywanie adresów zamiast etykiet w deasemblerze. W celu zmiany nazwy już zadeklarowanej wcześniej etykiety nie trzeba jej kasować, wystarczy zadeklarować taką etykietę jeszcze raz, ponieważ w deasemblerze brane pod uwagę są zawsze ostatnie załadowane deklaracje.

W celu skasowania wcześniej załadowanych definicji i przywrócenia etykiet domyślnych można posłużyć się opcją *Wyczyść etykiety* z menu **EDYCJA**.

Poniżej zamieszczony jest przykładowy plik z deklaracjami etykiet wszystkich typów:

```
C      0000 RESET
C      0023 *
C      0100 START_JEST_TROCHE_ZA_DLUGI
D      00    LICZBA
B      00    BIT
C      0000 TU_JEST_RESET
```

Jak łatwo zauważyć nie jest wymagane kończenie liczby szesnastkowej literą 'H', ponieważ program zawsze traktuje wpisane liczby jak szesnastkowe. W powyższym przykładzie widać, że etykieta adresowa o adresie 0000h zostaje zadeklarowana dwa razy, w związku z tym użyta zostanie ostatnia jej postać, czyli 'TU_JEST_RESET', a nie 'RESET'. Natomiast etykieta adresowa o adresie 0023h zostanie usunięta i deasembler nie będzie wyświetlał nazwy etykiety tylko jej adres. Kolejna etykieta 0100h jest za długa i deasembler w programie użyje jej 15-znakowego skrótu czyli 'START_JEST_TROC'. Etykieta o nazwie 'BIT' wskazuje na pojedynczy bit przestrzeni adresowalnej bitowo o adresie 00h, natomiast etykieta 'LICZBA' wskazuje komórkę wewnętrznej pamięci RAM procesora o adresie 00h czyli rejestr R0.

Plik zawierający listę deklaracji można załadować tak, jak każdy inny plik poprzez menu **PLIK**. Można również do każdego ładowanego programu stosować inną listę deklaracji, która będzie ładowana automatycznie przy otwieraniu pliku z programem. Aby tego dokonać należy umieścić plik definicji w tym samym katalogu co plik z programem, który będzie korzystał z tych definicji. Plik definicji musi mieć nazwę taką samą, jak nazwa pliku z programem np. jeśli ładowanym plikiem z

programem '51 jest 'PROG01.HEX', to plik z deklaracją etykiet musi mieć nazwę 'PROG01.DEF'.

Część etykiet jest zdefiniowana domyślnie i są to powszechnie stosowane nazwy rejestrów specjalnych procesorów 8051 i 8052. W tabeli 4 przedstawiono listę etykiet, które są ładowane domyślnie.

Nazwa etykiety	Adres	Typ	Nazwa etykiety	Adres	Typ	Nazwa etykiety	Adres	Typ
RESET	0000h	C	IE1	8Bh	B	B.0	F0h	B
INT_EX0	0003h	C	TR0	8Ch	B	B.1	F1h	B
INT_T0	000Bh	C	TF0	8Dh	B	B.2	F2h	B
INT_EX1	0013h	C	TR1	8Eh	B	B.3	F3h	B
INT_T1	001Bh	C	TF1	8Fh	B	B.4	F4h	B
SERIAL	0023H	C	EX0	A8h	B	B.5	F5h	B
INT_T2	002BH	C	ET0	A9h	B	B.6	F6h	B
ACC	E0h	D	EX1	AAh	B	B.7	F7h	B
B	F0h	D	ET1	ABh	B	P0.0	80h	B
PSW	D0h	D	ES	ACH	B	P0.1	81h	B
SP	81h	D	ET2	ADh	B	P0.2	82h	B
DPH	83h	D	EA	AFh	B	P0.3	83h	B
DPL	82h	D	PX0	B8h	B	P0.4	84h	B
P0	80h	D	PT0	B9h	B	P0.5	85h	B
P1	90h	D	PX1	BAh	B	P0.6	86h	B
P2	A0h	D	PT1	BBh	B	P0.7	87h	B
P3	B0h	D	PS	BCh	B	P1.0	90h	B
IP	B8h	D	PT2	BDh	B	P1.1	91h	B
IE	A8h	D	RI	98h	B	P1.2	92h	B
TCON	88h	D	TI	99h	B	P1.3	93h	B
TMOD	89h	D	RB8	9Ah	B	P1.4	94h	B
TH0	8Ch	D	TB8	9Bh	B	P1.5	95h	B
TLO	8Ah	D	REN	9Ch	B	P1.6	96h	B
TH1	8Dh	D	SM2	9Dh	B	P1.7	97h	B
TL1	8Bh	D	SM1	9Eh	B	P2.0	A0h	B
T2CON	C8h	D	SM0	9Fh	B	P2.1	A1h	B
TH2	CDh	D	CP_RL2	C8h	B	P2.2	A2h	B
TL2	CCh	D	C_T2	C9h	B	P2.3	A3h	B
RLDH	CBh	D	TR2	CAh	B	P2.4	A4h	B
RLDL	CAh	D	EXEN2	CBh	B	P2.5	A5h	B
SCON	98h	D	TCLK	CCh	B	P2.6	A6h	B
SBUF	99h	D	RCLK	CDh	B	P2.7	A7h	B
PCON	97h	D	EXF2	CEh	B	P3.0	B0h	B
P	D0h	B	TF2	CFh	B	P3.1	B1h	B
OV	D2h	B	ACC.0	E0h	B	P3.2	B2h	B
RS0	D3h	B	ACC.1	E1h	B	P3.3	B3h	B
RS1	D4h	B	ACC.2	E2h	B	P3.4	B4h	B
F0	D5h	B	ACC.3	E3h	B	P3.5	B5h	B
AC	D6h	B	ACC.4	E4h	B	P3.6	B6h	B
CY	D7h	B	ACC.5	E5h	B	P3.7	B7h	B
IT0	88h	B	ACC.6	E6h	B			
IE0	89h	B	ACC.7	E7h	B			
IT1	8Ah	B						

Tab. 4. Spis etykiet ładowanych domyślnie.

Problemy napotkane podczas konstruowania emulatora

Pierwszym problemem było napisanie niezawodnych procedur obsługujących port szeregowy komputera. Ponieważ ciężko było dostać odpowiednio bogatą w informacje na temat portu szeregowego literaturę, większość procedur była modyfikowana w trakcie testowania programu.

Drugim istotnym problemem była konstrukcja emulatora. Trudność polegała na zbudowaniu w domowych warunkach dość skomplikowanego małego urządzenia. Problem udało się wyeliminować poprzez zastosowanie mikrokontrolera zawierającego wewnętrzną programowalną pamięć programu.

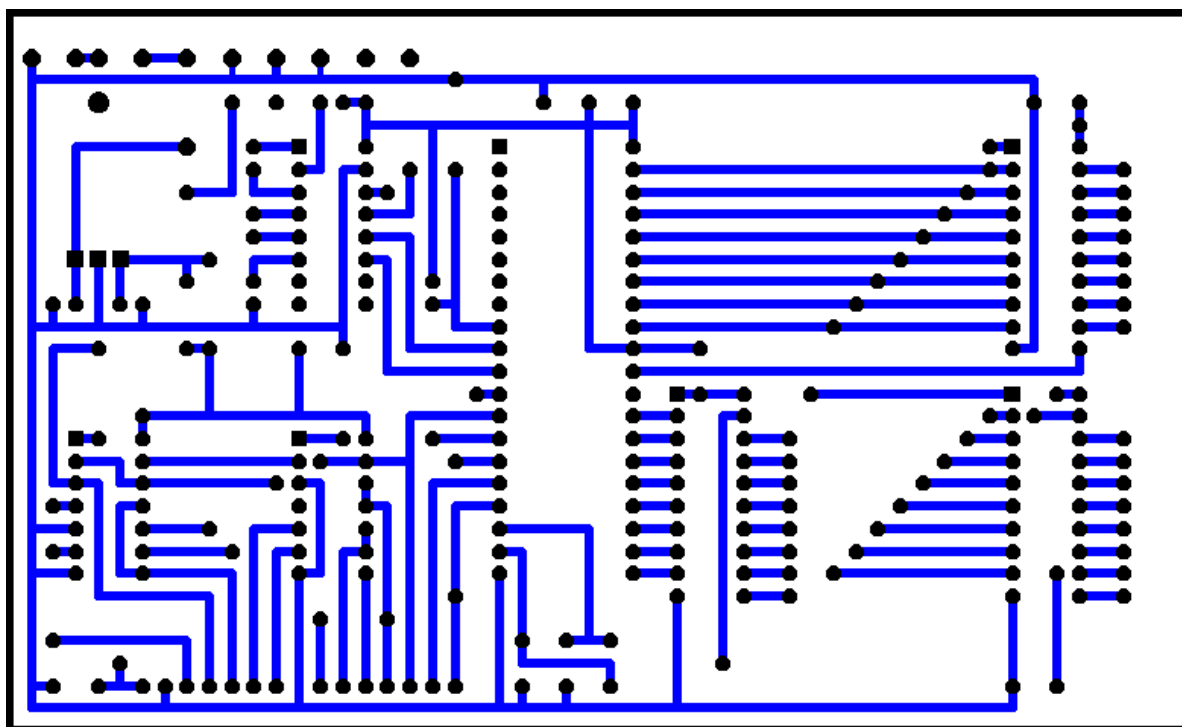
Kolejnym problemem napotkanym podczas wykonywania pracy było zdobycie odpowiedniego wtyku emulacyjnego. Jest on dostępny tylko w jednym sklepie w trójmieście.

Inną trudnością okazał się sam język C++. W pewnym momencie podczas kompilacji programu okazało się, że jest on za duży. Należało wówczas zwrócić szczególną uwagę na optymalne wykorzystanie pamięci i optymalizację dużych procedur. Po zastosowaniu dynamicznej alokacji pamięci dla zmiennych oraz po zoptymalizowaniu procedur pod względem wielkości problem został rozwiązany.

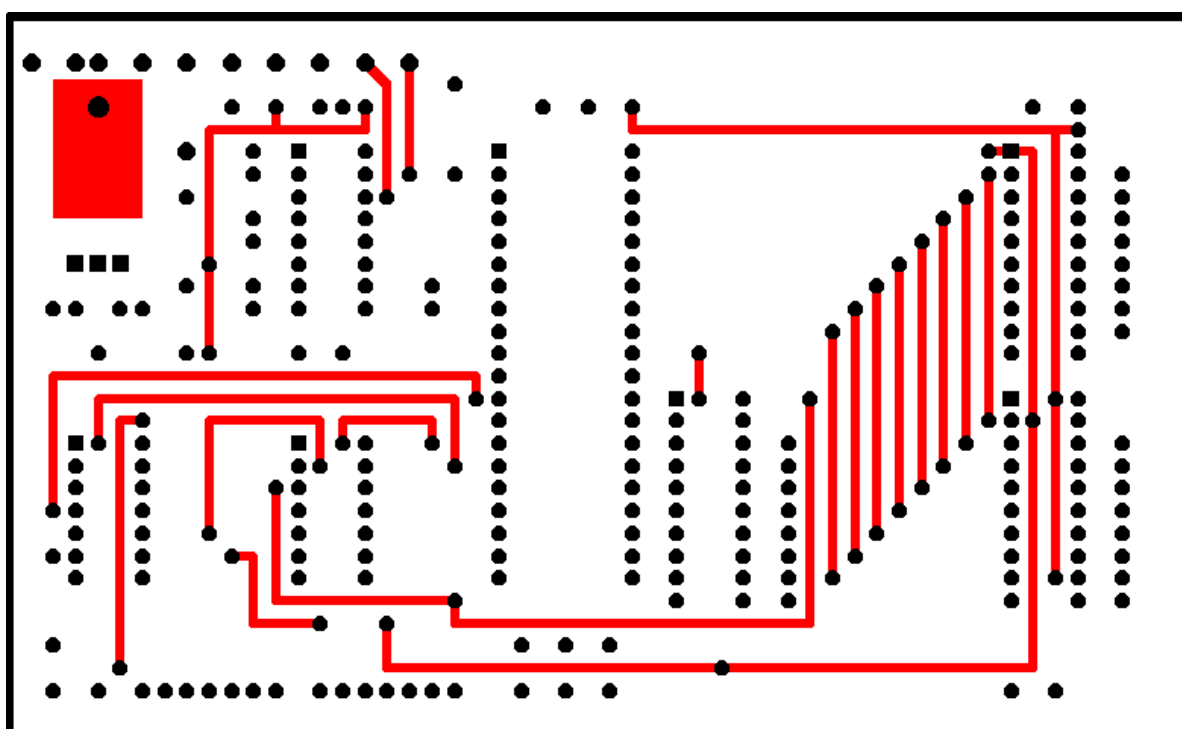
Bibliografia

1. *Cyfrowe układy scalone, Katalog podręczny*, Kraków 1993.
2. Dudek A., *Jak pisać wirusy*, Warszawa 1994, wyd. 2.
3. *Elektronika dla wszystkich 1997-98*.
4. *Elektronika Praktyczna 1996-98*.
5. *Hardware Reference Manual*, Atmel AT89C51.
6. Majczak A., C++. *Od DOS do Win 95 nie tylko dla orłów*, Warszawa 1997.
7. *Mikroprocesory firmy Intel*, red. Cezary Stępień, Warszawa 1992, wyd.1.
8. Money S. A., *Mikroprocesory - poradnik*, tłum. L.Bułhak, L. Śliwa, Warszawa 1996, wyd. 1.
9. Rydzewski A., *Mikrokomputery jednoukładowe rodziny MCS-51*, Warszawa 1997, wyd. 2.

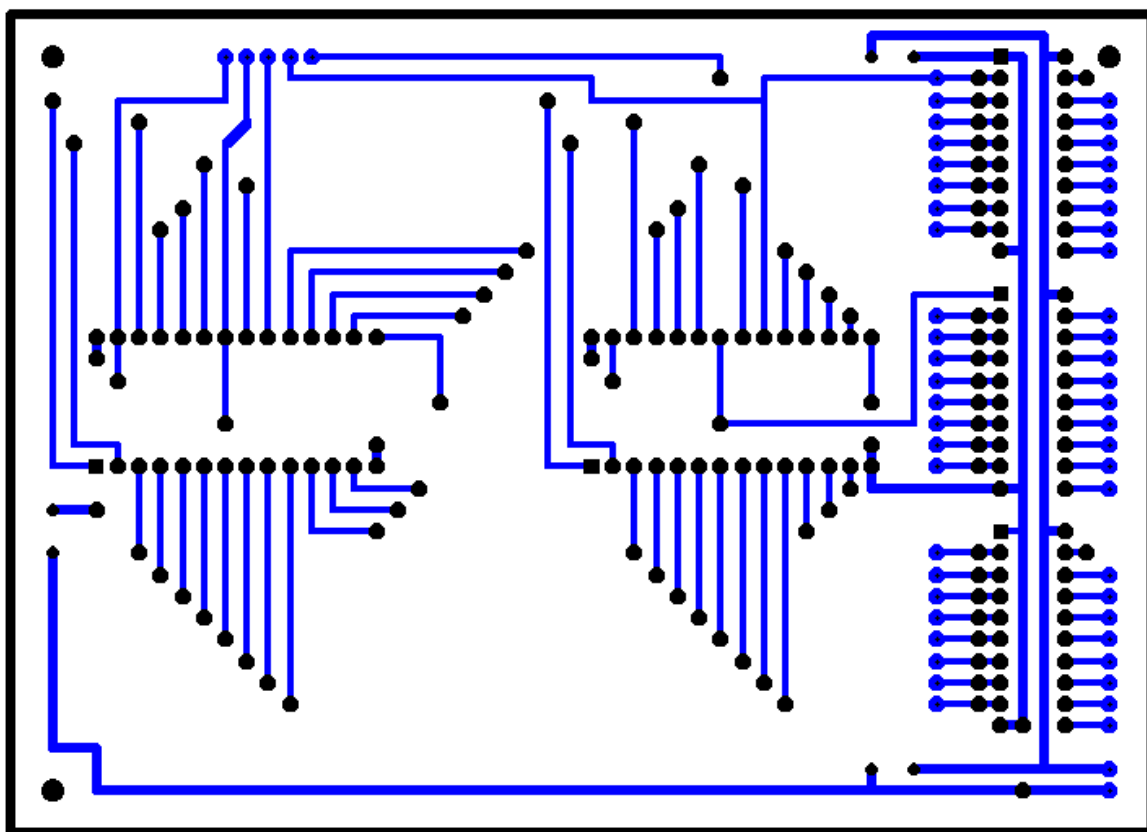
Rysunki płytek drukowanych



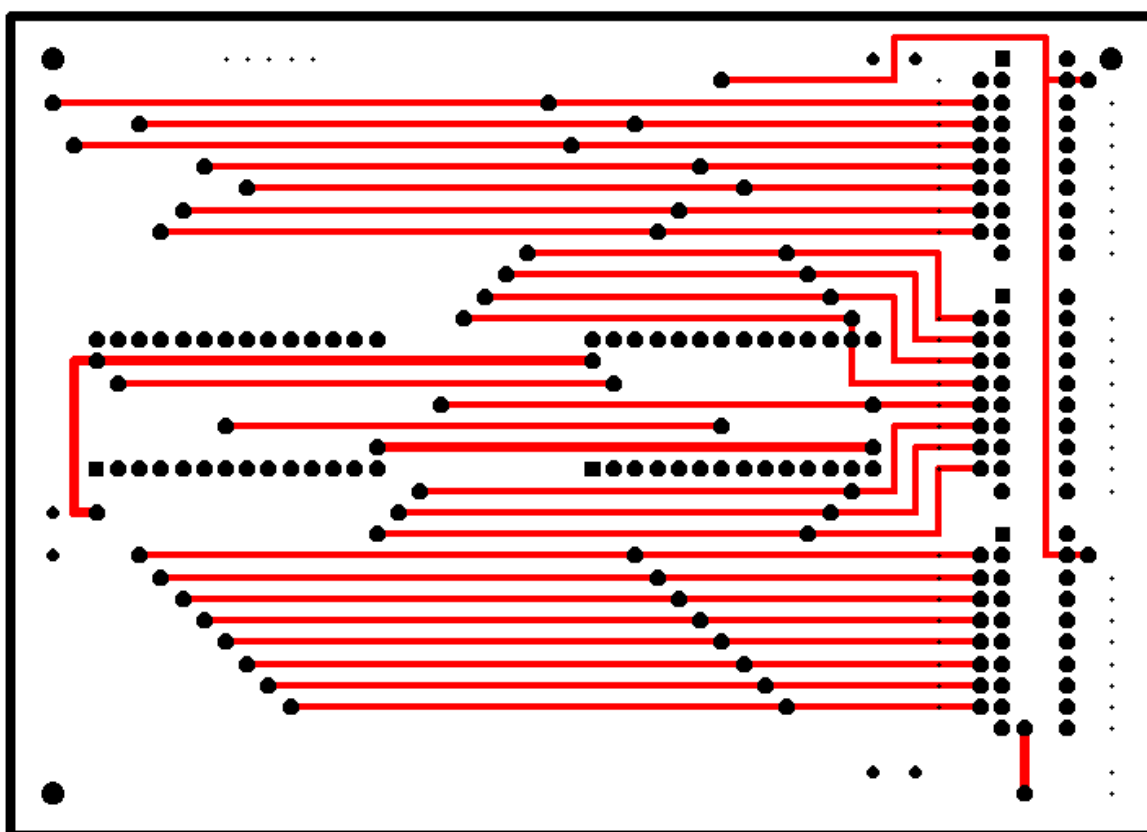
Rys. 10. Widok płytki głównej od strony lutowania.



Rys. 11. Widok płytki głównej od strony elementów.



Rys. 12. Widok płytki z pamięciami od strony lutowania.



Rys. 13. Widok płytki z pamięciami od strony elementów.