Mirosław Łazoryszczak

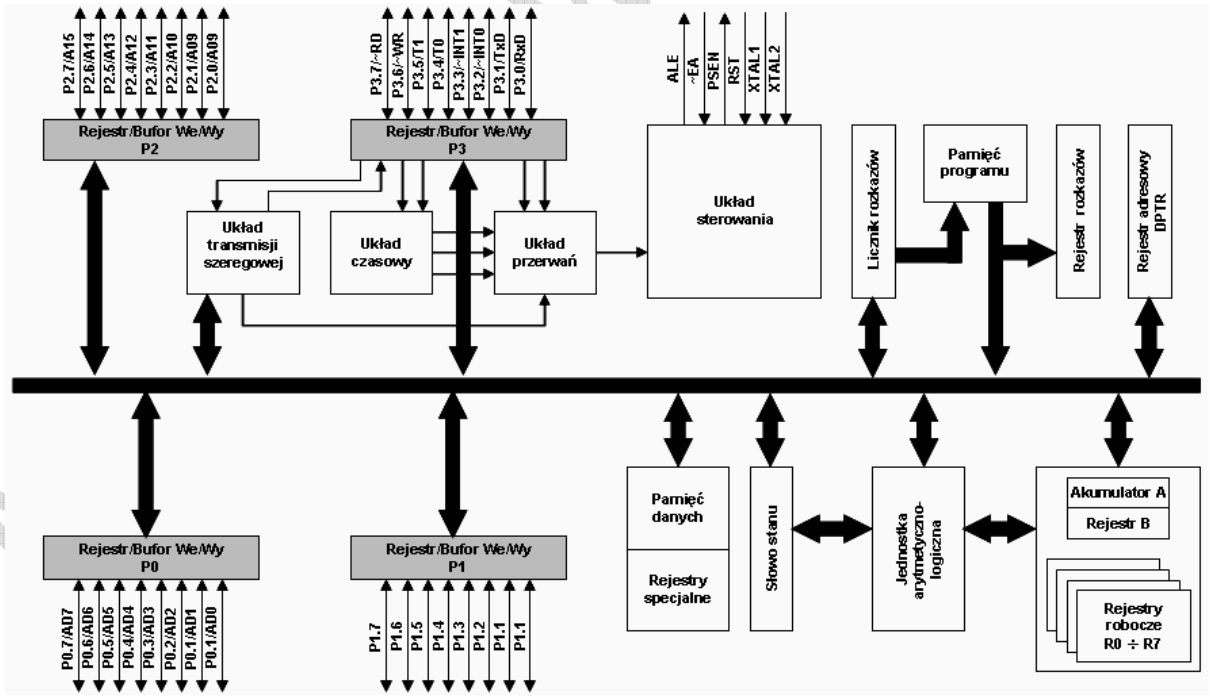
LABORATORIUM nr 1

Temat: Wstęp do mikrokontrolerów rodziny MCS-51

1. ARCHITEKTURA MCS-51

UWAGA:
Niniejszy rozdział stanowi jedynie krótkie wprowadzenie do architektury MCS-51. Kompletny opis wszystkich zagadnień niezbędnych do przeprowadzenia ćwiczeń laboratoryjnych znajduje się w oryginalnej dokumentacji producenta [3]

W ramach rodziny MCS-51 wyróżnia się mikrokontroler 8051 oraz jego rozbudowaną wersję 8052. Mikrokontroler 8052 w stosunku do 8051 różni się rozmiarem pamięci oraz liczbą układów i funkcji dodatkowych.
Mikrokontroler 8051 w podstawowej wersji w odróżnieniu od mikroprocesorów uniwersalnego przeznaczenia poza jednostką arytmetyczno-logiczną (*Arithmetic-Logic Unit*, ALU) charakteryzuje się możliwością interakcji ze światem zewnętrznym za pomocą linii cyfrowych o charakterze wejścia/wyjścia zorganizowanych w tzw. porty. W podstawowej wersji (rys. 1) mikrokontroler zawiera 4 porty (P0, P1, P2, P3). Wymienione wyżej porty mogą pełnić funkcję uniwersalnych wejść/wyjść, lub mogą pełnić dedykowane funkcje z punktu widzenia układów wewnętrznych mikrokontrolera, do których zalicza się np. układ transmisji szeregowej, układ czasowo-licznikowy, układ przerwań. Za pomocą portów możliwa jest także współpraca z zewnętrzną pamięcią programu lub danych. Porty pełnią wówczas funkcję magistrali adresowej i/lub danych (porty P0 i P2).



Rys. 1. Architektura mikrokontrolera 8051.

Mikrokontrolery rodziny MCS-51 są urządzeniami ośmiobitowymi, co oznacza, że zarówno jednostka arytmetyczno-logiczna jak i rejestry przystosowane są do operowania na ośmiu bitach danych. Jedynie rejestr DPTR ma wielkość 16 bitów, jednak jego głównym przeznaczeniem jest adresowanie pamięci zewnętrznej.

Z układem jednostki arytmetyczno-logicznej ściśle związany jest rejestr nazywany akumulatorem (A lub ACC), oraz rejestr B. Rejestr A służy do przechowywania argumentów oraz wyników operacji arytmetyczno-logicznych. Niezwykle istotną funkcję pełni rejestr PSW (ang. *Program Status Word*), zwany także rejestrem znaczników lub inaczej flag (Tabela 1).
Rejestr PSW zawiera bity, które z jednej strony mają znaczenie informacyjne – uzupełniają wynik ostatniej operacji wykonywanej przez ALU (np. znacznik przeniesienia CY informuje o fakcie wystąpienia przeniesienia w przypadku ostatnio wykonywanej przez ALU operacji), a z drugiej sterujące mogą być ustawiane przez programistę (np. bity RS1 i RS0 – służą do wyboru aktywnego bloku rejestrów roboczych). Rejestry te mają funkcję ogólne – mogą być wykorzystywane do przechowywania wartości zmiennych oraz argumentów i wyników operacji arytmetyczno-logicznych.

Tabela. 1: Rejestr stanu procesora (PSW) i znaczenie flag.

CY	AC	F0	RS1	RS0	OV	-	P

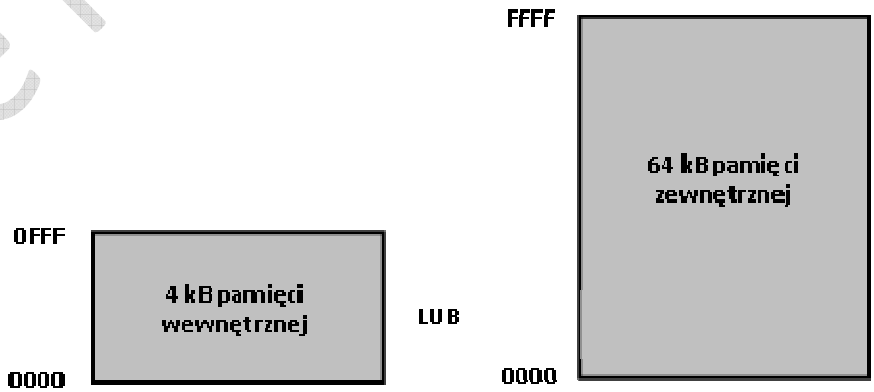
CY	PSW.7	Przeniesienie
AC	PSW.6	Przeniesienie pomocnicze
F0	PSW.5	Znacznik ogólnego przeznaczenia
RS1	PSW.4	Selektor banku rejestrów (bit 1)
RS0	PSW.3	Selektor banku rejestrów (bit 0)
OV	PSW.2	Nadmiar
-	PSW.1	Znacznik definiowalny przez użytkownika
P	PSW.0	Parzystość wyznaczana na podst. liczby „1” w rejestrze A

Mikrokontroler 8051 posiada 4 bloki rejestrów ogólnego przeznaczenia. W każdym bloku znajduje się 8 rejestrów ośmiobitowych. Rejestry te posiadają nazwy R0 ÷ R7. W danym momencie aktywny może być tylko jeden blok rejestrów. Wybór aktywnego bloku rejestrów możliwy jest z pomocą binarnej kombinacji bitów RS1, RS0 w rejestrze PSW (Tabela 2). Blok rejestrów specjalnych znajduje się w wewnętrznej pamięci RAM, która zostanie opisana poniżej.

Tabela. 2: Wybór aktywnego bloku rejestrów za pomocą bitów RS1, RS0 z rejestru PSW.

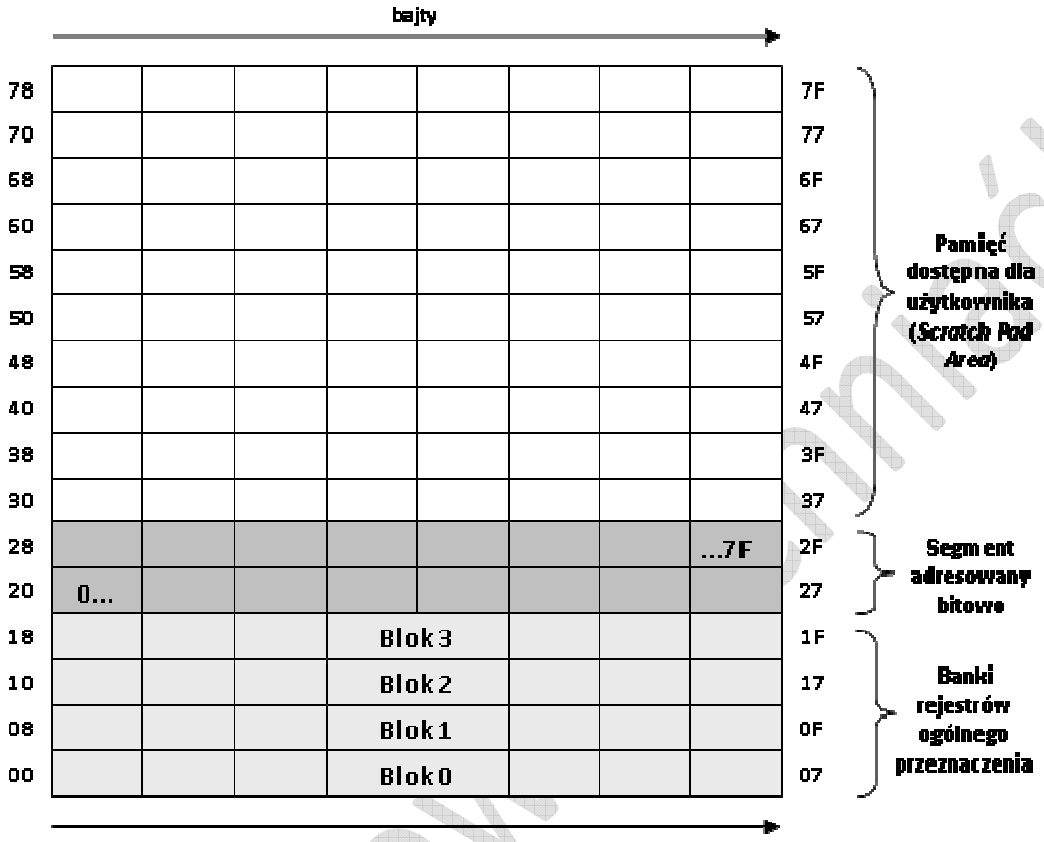
RS1	RS0	Aktywny blok rejestrów
0	0	0
0	1	1
1	0	2
1	1	3

W mikrokontrolerach rodziny 8051 pamięć programu jest rozdzielona od pamięci danych. W wersji podstawowej mikrokontroler 8051 zwiera 4 kbajty pamięci wewnętrznej. Program może zostać także umieszczony w zewnętrznej pamięci programu o wielkości 64 kbajtów (rys. 2).

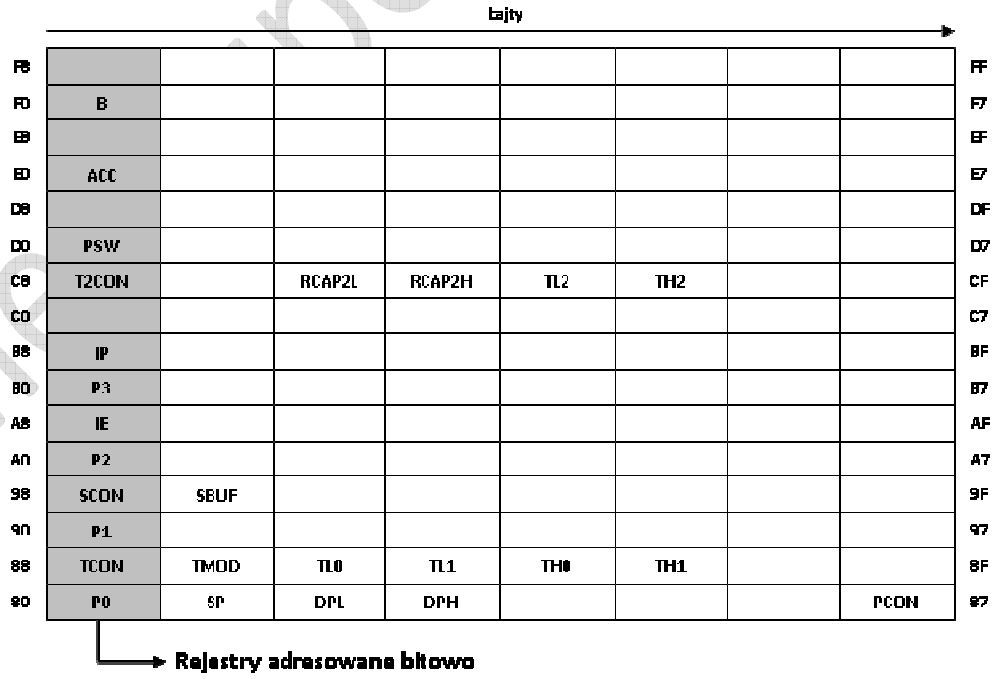


Rys. 2. Organizacja pamięci programu 8051.

Pamięć danych w 8051 ma rozmiar 256 bajtów. Pierwsze 128 bajtów (do adresu 7Fh) może być osiągalne za pomocą adresowania bezpośredniego np. za pomcą instrukcji MOV *data addr* lub pośredniego np. MOV @Ri (rys. 3). Kolejne 128 adresów (w zakresie 80h – FFh) zarezerwowane jest przez blok rejestrów specjalnych (ang. *Special Function Register*, SFR), które mogą być adresowane tylko w sposób bezpośredni (rys. 4).



Rys. 3. Mapa wewnętrznej pamięci danych – obszar pierwszych 128 bajtów.



Rys. 4. Mapa wewnętrznej pamięci danych – obszar powyżej 128 bajtów.

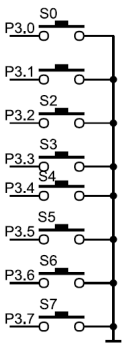
W odróżnieniu od mikroprocesorów uniwersalnych 8051 posiada unikalny tryb adresowania bitowego. Oznacza to, iż wybrane bity mogą być osiągalne w sposób bezpośredni. W mikrokontrolerze 8051 adresowanie bitowe dotyczy wybranych rejestrów specjalnych (rys. 4) oraz wybranego obszaru pamięci o adresach z zakresu 20h ÷ 2Fh.

2. ŚRODOWISKO PRACY

Do wykonania ćwiczeń laboratoryjnych opartych na mikrokontrolerze 8051 wykorzystany zostanie zestaw uruchomieniowy ZL2MCS51 [2] oraz bezpłatne oprogramowanie obejmujące kompilator oraz programator. Poniżej omówione zostaną wybrane aspekty budowy płyty uruchomieniowej niezbędne do realizacji zadań oraz sposób posługiwania się oprogramowaniem.

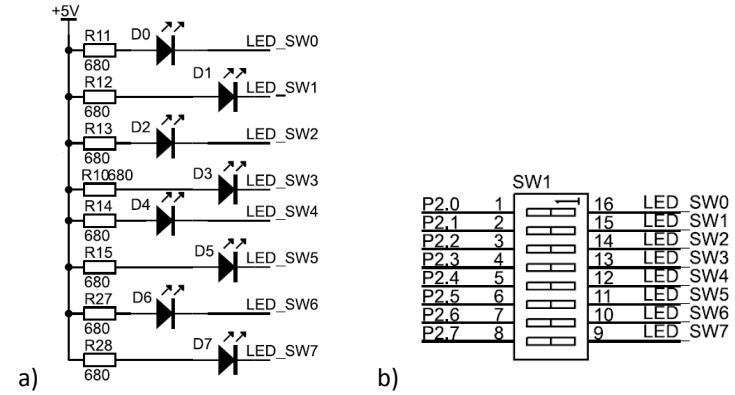
2.1. PŁYTA URUCHOMIENIOWA ZL2MCS51

Testowanie programów napisanych podczas zajęć laboratoryjnych odbywać się będzie przy pomocy zestawu uruchomieniowego ZL2MCS51 wyprodukowanego przez BTC Korporacja. Pełen schemat płyty oraz szczegóły techniczne znajdują się w dokumentacji [2].
W zestawie wykorzystano mikrokontroler Atmel AT89C51RD2. Jego możliwości w zakresie układów dodatkowych są znacznie większe od pierwowzoru Intela, aczkolwiek do wykonania ćwiczeń wystarczy znajomość rdzenia 8051 zgodnego z oryginalną dokumentacją producenta [3]. Dokumentacja do mikrokontrolera Atmel AT89C51RD2 [1] zawiera jedynie opis rozszerzeń i ew. zmian w stosunku do mikrokontrolera 8051.



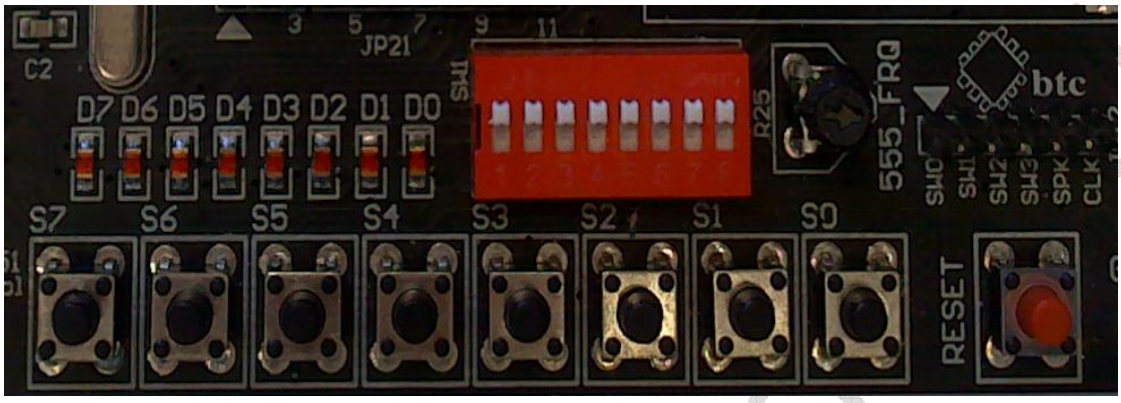
Rys. 5. Sposób podłączenia przycisków do portów mikrokontrolera

Na płycie uruchomieniowej znajduje się 8 przycisków (ang. *push button*) dołączonych bezpośrednio do portu P3 (rys. 5). Z poniższego rysunku wynika także sposób działania przycisków: w stanie wyciśniętym na wejściu portu dzięki wewnętrznemu rezystorowi podciągającemu panuje stan wysoki. Wciśnięcie przycisku oznacza, iż na wejściu portu pojawi się stan niski (zwarcie do masy).



Rys. 6. Podłączenie diod LED do portów mikrokontrolera: a) diody, b) przełącznik

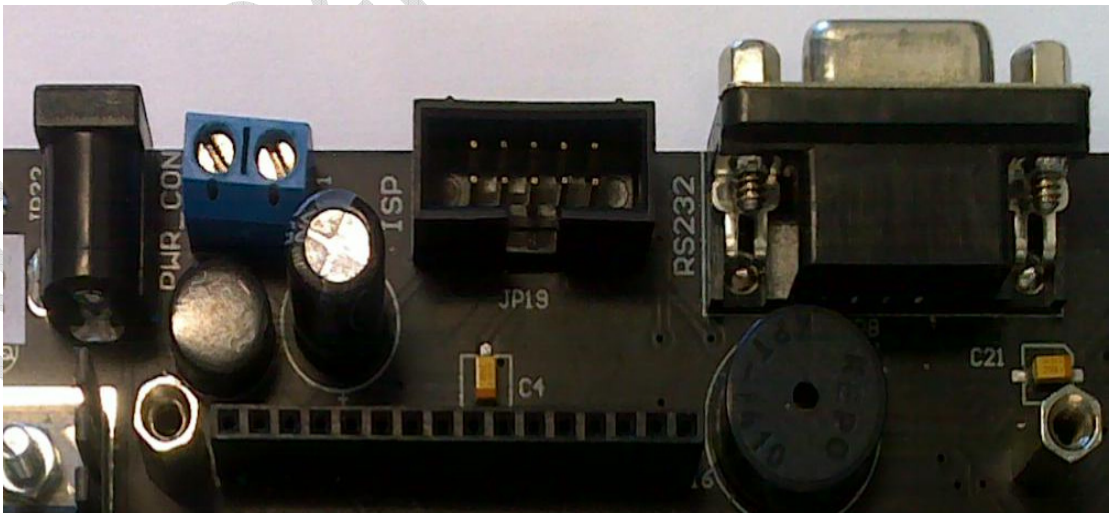
Kolejnym elementem przydatnym w wykonaniu pierwszego zadania są diody LED. Sposób podłączenia diod do portów mikrokontrolera przedstawiono na rys. 6. Anody diod dołączone są za pośrednictwem rezystorów 680 Ω do napięcia zasilania +5 V, natomiast katody doprowadzone są do przełącznika (ang. *switch*), za pomocą którego mogą zostać dołączone fizycznie do portu P2 mikrokontrolera (położenie ON) lub też mogą zostać od niego odłączone.
Obok rzędu ośmiu przycisków znajduje się także przycisk RESET, służący do restartu mikrokontrolera np. po zakończeniu operacji programowania mikrokontrolera. Na rys. 7 przedstawiono fragment płyty uruchomieniowej zawierające opisane wyżej elementy.



Rys. 7. Fragment płyty uruchomieniowe ZL2MCS51 z widocznymi przyciskami, diodami LED, przełącznikiem, oraz przyciskiem RESET.

Na rys. 8 przedstawiono fragment płyty uruchomieniowej zawierającej gniazdo zasilania. Układ zawiera stabilizator scalony 7805, może być więc zasilany napięciem 9 ÷ 12 V za pomocą przewodów wskazanych przez prowadzącego zajęcia. Źródłem napięcia zewnętrznego może być np. zasilacz laboratoryjny bądź popularny, niestabilizowany zasilacz sieciowy.
Przed podłączeniem zestawu uruchomieniowego do zasilania należy połączyć płytkę z komputerem za pomocą odpowiedniego kabla RS 232.

UWAGA:
Podłączenie płyty uruchomieniowej do zasilania może nastąpić po uprzednim sprawdzeniu przez prowadzącego poprawności połączeń oraz ustawień zasilacza laboratoryjnego.



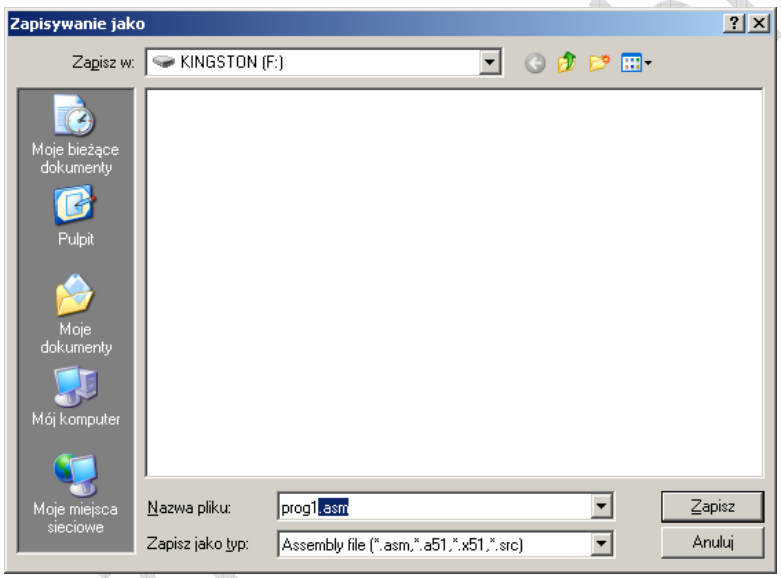
Rys. 8. Fragment płyty uruchomieniowej z gniazdem zasilania oraz złączem RS232

2.2. ŚRODOWISKO PROGRAMOWE

Podstawowym środowiskiem pracy używanym w laboratorium Systemów Wbudowanych jest MIDE-51. Jest to zintegrowane środowisko zawierające asembler (ASEM-51), kompilator języka C (SDCC) oraz symulator/emulator 8051.
Środowisko MIDE nie umożliwia tworzenia projektów wieloplikowych, nie stanowi to jednak przeszkody z punktu widzenia złożoności ćwiczeń realizowanych w trakcie laboratoriów.

UWAGA:
Przed przystąpieniem do pracy z komputerem należy bezwzględnie wykonać procedurę podłączenia do dysku sieciowego użytkownika zgodnie z zewnętrzną instrukcją, dostępną na stanowisku laboratoryjnym.

Po uruchomieniu programu **MIDE51.exe** (Pulpit lub Menu Start) należy utworzyć nowy plik. Następnie należy zapisać plik na dysku wskazując folder domowy użytkownika, wybierając typ pliku (.asm lub .c) podając nazwę pliku (ze względów praktycznych zaleca się ograniczenie nazwy do ośmiu znaków) oraz ręcznie wpisując rozszerzenie nazwy .ASM lub .C, pomimo wyboru typu pliku – rys. 2.



Rys. 9. Okno zapisu pliku programu. Zaznaczono ręcznie wprowadzone rozszerzenie nazwy .ASM

Najprostszy program w asemblerze zawierający pustą, nieskończoną pętlę główną przedstawia poniższy wydruk.

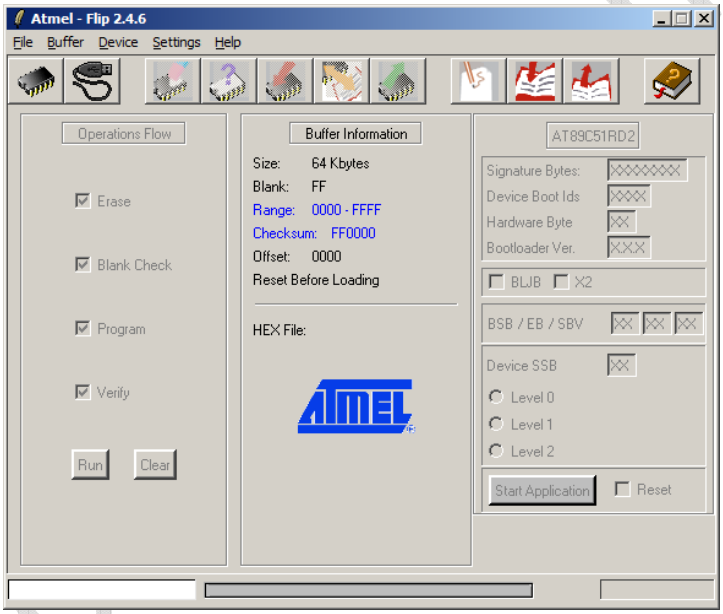
```
01    ORG 0
02    JMP 100
03
04    ORG 100
05    start:
06    JMP start
07    END
```

Wydruk 1: Elementarny program na mikrokontroler 8051 zawierający nieskończoną, pustą pętlę główna

W liniach nr 01 oraz 04 znajduje się pseudoinstrukcja `ORG` wraz z adresem (w omawianym przypadku 0 oraz 100 podane dziesiętnie), której zadaniem jest zapewnienie, że kod znajdujący się po niej zostanie umieszczony w pamięci programu począwszy od adresu odpowiednio 0 oraz 100. W podanym przykładzie pierwsze wystąpienie pseudoinstrukcji `ORG 0` sprawia, iż pierwszą instrukcją programu będzie instrukcja skoku bezwarunkowego `JMP 100` (skok do adresu 100 w pamięci programu). Kolejne wystąpienie pseudoinstrukcji `ORG` oznacza, że kod następujący po niej umieszczony zostanie począwszy od adresu 100. Taka konstrukcja

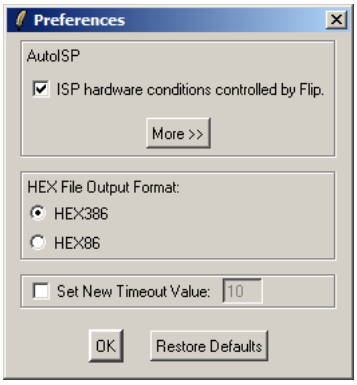
programu ma na celu ominięcie pewnego obszaru pamięci programu, w którym zgodnie z dokumentacją projektantów mikrokontrolera mogą znajdować się adresy o zarezerwowanej funkcji (np. początki procedur obsługi przerwań). Ostatnią pseudoinstrukcją jest `END`, która kończy program i jest wymagana przez asembler ASEM-51 wchodzący w skład środowiska MIDE.
Po zapisaniu programu można wykonać jego kompilację (opcja Build | Build z menu programu lub klawisz F9). Efektem bezbłędnej kompilacji jest wygenerowanie pliku o takiej samej nazwie z rozszerzeniem .HEX. Plik HEX jest plikiem wynikowym, który może posłużyć następnie do zaprogramowania mikrokontrolera lub do przeprowadzenia symulacji programu. Środowisko MIDE posiada zintegrowany symulator TS Controls Emulator 8051, który można uruchomić wraz kompilacją programu źródłowego (opcja Build | Build and Sim z menu lub kombinacja klawiszy Shift+Ctl+F9)
Skompilowany program do pliku .HEX może posłużyć do zaprogramowania mikrokontrolera. Zestaw ZL2MCS51 wyposażony w mikrokontroler AT89C51RD2 może być programowany z pomocą łącza RS-232. Wspomniany mikrokontroler zawiera tzw. *bootloader*, który umożliwia programowanie w systemie bez żadnych dodatkowych układów. Do zaprogramowania mikrokontrolera służy program Flip firmy Atmel (wersja 2.4.6). Na poniższych rysunkach przedstawiona została sekwencja kroków niezbędna do umieszczenia pliku wynikowego w pamięci mikrokontrolera.

Z powodu ograniczeń konta „student” zaleca się skopiowanie folderu zawierającego program Flip do katalogu użytkownika na dysku sieciowym. Widok głównego okna programu przedstawiono na rys. 10.



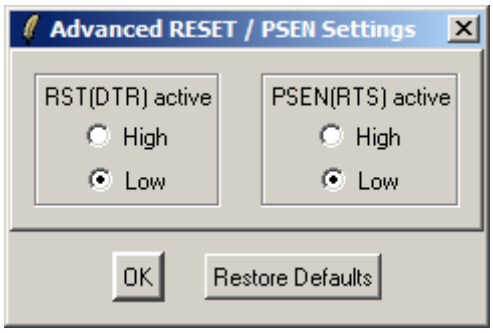
Rys. 10. Okno główne programu Flip służącego do programowania mikrokontrolerów rodziny Atmel

Po uruchomieniu programu konieczne jest jego jednorazowa konfiguracja. W tym celu należy wybrać z menu programu opcję **Settings | Preferences**. Następnie w oknie **Prefereces** należy zaznaczyć opcję „*ISP hardware conditions cotrolled by Flip*” (rys. 11) i wcisnąć przycisk **More>>**.



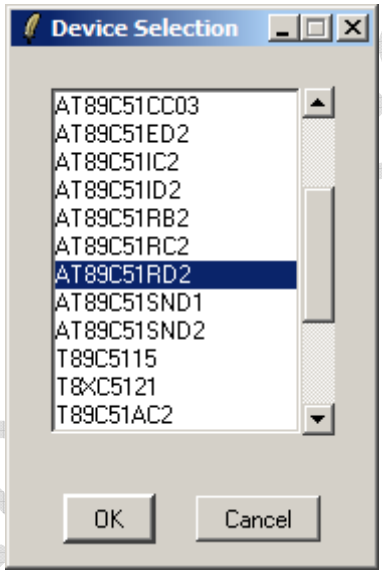
Rys. 11. Okno właściwości ustawień programu Flip.

W kolejnym okienku należy ustawić opcje zgodnie z rys. 12.



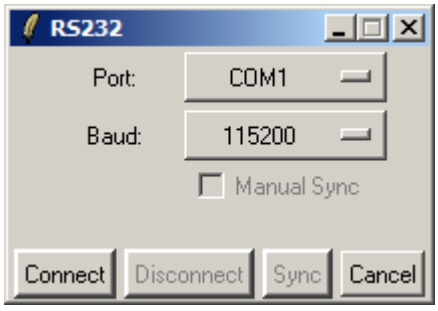
Rys. 12. Okno ustawień zaawansowanych

Po uruchomieniu i skonfigurowaniu programu należy wybrać urządzenie (mikrokontroler), który będzie programowany. W zestawie ZL2MCS51 jest to mikrokontroler AT89C51RD2. Należy wybrać właściwe urządzenie w okienku, które powinno pojawić się po uruchomieniu opcji **Device | Select...** (rys. 13).



Rys. 13. Okno wyboru rodzaju mikrokontrolera

Następnie należy nawiązać połączenie z wybranym mikrokontrolerem za pomocą interfejsu RS-232 poprzez wybranie opcji **Settings | Communication | RS232** (rys. 14) i wciśnięciu przycisku **Connect**.



Rys. 14. Okno połączenia z mikrokontrolerem

Kolejny krok polega na wczytaniu pliku wynikowego z rozszerzeniem .HEX do bufora programu FLIP za pomocą opcji **File| Load HEX File** i zaprogramowaniu mikrokontrolera za pomocą opcji **Device | Program**.

Jeśli proces programowania zakończy się powodzeniem, należy wymusić tryb uruchamiania programu w mikrokontrolerze i tym samym zakończyć sesję *bootloadera* przez wciśnięcie przycisku RESET na płycie uruchomieniowej.

UWAGA:
Ponowne zaprogramowanie mikrokontrolera wymaga powtórzenia opisanych wyżej czynności począwszy od nawiązania połączenia pomiędzy mikrokontrolerem a komputerem PC

3. ZADANIA

Pierwszy program polega na elementarnej obsłudze klawiatury (przycisków S0 ÷ S7). Najprostszy program przeznaczony dla dowolnego mikrokontrolera powinien jednocześnie pełnić funkcję podstawowego systemu operacyjnego. W omawianym przypadku program powinien zawierać nieskończoną pętlę główną w ramach której będzie następował cykliczny odczyt stanu klawisza bądź wszystkich klawiszy jednoczenie. Następnie na podstawie stanu klawiszy ustawione zostaną odpowiednie porty (port), do których dołączone są diody LED.
Na wydruku 2 i 3 znajdują się przykładowe programy sterujące diodami LED za pomocą klawiszy.

```
01    ORG 0
02    JMP 100
03    ORG 100
04    start:
05    MOV P2,P3
06    JMP start
07    END
```

Wydruk 2: Program nr 1 obsługujący diody i przyciski

```
01    ORG 0
02    JMP 100
03    ORG 100
04    start:
05    MOV C, P3.0
06    MOV P2.0, C
07    JMP start
07    END
```

Wydruk 3: Program nr 2 obsługujący diody i przyciski

3.1. ZADANIE NR 1

1. Zaimplementuj programy przedstawione na wydruku 2 i 3.
2. Zaobserwuj różnice w działaniu obu programów i porównaj kod odpowiadający bezpośrednio za obsługę diod i przycisków i omów te różnice.

3.2. ZADANIE NR 2

1. Korzystając z dokumentacji do zestawu uruchomieniowego [2] oraz z obserwacji samej płytki z mikrokontrolerem wyznacz częstotliwość taktowania oraz czas trwania pojedynczego okresu sygnału zegarowego taktującego mikrokontroler.
2. Korzystając z dokumentacji [3] określ, ile taktów zegara przypada na jeden cykl rozkazowy mikrokontrolera 8051.
3. Korzystając z powyższych informacji napisz samodzielnie program, którego efektem będzie uzyskanie efektu migania wybraną diodą LED z częstotliwością najbardziej zbliżoną do 1 Hz (dioda przez 0,5 s powinna się świecić, przez kolejne 0,5 s powinna być zgaszona).
4. Zaproponuj algorytm realizacji zadania.

5. Wykonaj dokładne obliczenia czasu trwania poszczególnych fragmentów programu odpowiadających za realizację faz świecenia i wygaszenia diody w celu wykazania, że okres pulsowania diody wynosi możliwie dokładnie 1 s.

3.3. ZADANIE NR 3

1. Napisz program realizujący dowolny schemat migania diodami umieszczonymi na płytce uruchomieniowej

4. LITERATURA

[1] Atmel: Dokumentacja techniczna mikrokontrolera AT89C51RD2, http://www.atmel.com/dyn/resources/prod_documents/doc4235.pdf.

[2] BTC Korporacja: Dokumentacja techniczna zestawu uruchomieniowego ZL2MCS51, http://www.btc.pl/pdf/zl2mcs51.pdf (dostęp: luty 2011).

[3] Intel, MCS51 Microcontroller Family User's Manual, 1994 (dokumentacja dostępna pod różnymi adresami w sieci Internet).

[4] Majewski J.: Programowanie mikrokontrolerów 8051 w języku C. Pierwsze kroki, BTC, Warszawa, 2005.

[5] Stępień C. (red.): Mikroprocesory firmy Intel, PWN, Warszawa 1992.

DODATEK – LISTA ROZKAZÓW MIKROKONTROLERA RODZINY MCS-51

Tabela. 3: Instrukcje, które mają wpływ na rejestr znaczników [3]

Instrukcja	Flaga		
	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

Tabela. 4: Lista rozkazów MCS-51 wg kategorii [3]

Mnemonic		Description	Byte	Osc.
Arithmetic Operations				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Adddirect byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM toAccumulator	1	12
ADD	A,#date	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,dirsct	Add direct byte to Accumulator with Carry	2	12
ADDC	A.@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#date	Add immediate data to ACC with Carry	2	12
SUBB	A,Rn	Subtract Register from ACC with borrow	1	12
SUBB	A,direct	Subtract direct byte from ACC with borrow	2	12
SUBB	A.@Ri	Subtract indirect RAM from ACC with borrow	1	12

Mnemonic		Description	Byte	Osc.
SUBB	A.#date	Subtract immediate data from ACC with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Regieter	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
Logical Operations				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#date	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct, #data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#date	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct data	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct, #data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
Data Transfer				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#date	Move immediate date to Accumulator	2	12
MOV	Rn.A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#date	Move immediate date to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct, #date	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate date to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to ACC	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to ACC	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to ACC	1	24
MOVX	A,@DPTR	Move External RAM (16 bit addr) to ACC	1	24
MOVX	@Ri,A	Move ACC to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move ACC to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12

Mnemonic		Description	Byte	Osc.
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with ACC	1	12
Boolean Variable Manipulation				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to Carry	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to carry	2	24
ORL	C,/bit	OR complement of direct bit to carry	2	24
MOV	C,bit	Move direct bit to carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct bit is set	3	24
JNB	bit,rel	Jump if direct bit is n not set	3	24
JBC	bit,rel	Jump if direct bit is set & clear bit	3	24
Program Branching				
ACALL	addr11	Absolute Subroutine call	2	24
LCALL	addr16	Long Subroutine call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rei	Compare direct byte to ACC and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to ACC and Jump if Not Equal	3	24
CJNE	Rn,#data,rei	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri, #data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rei	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

Legenda:

Rn – rejestr R7 – R0 z aktywnego banku rejestrów

direct – 8 bitowy adres w pamięci wewnętrznej. Może oznaczać lokalizację wewnętrznej pamięci RAM o adresach 0-127 lub blok rejestrów specjalnych (SFR), np. porty We/Wy, rejestry sterujące, PSW itp. o adresach 128-255

@Ri – 8 bitowa lokalizacja w wewnętrznej pamięci danych adresowana pośrednio za pomocą rejestrów R1 lub Ro

#data – 8 bitowa stała umieszczona w kodzie rozkazu

#data 16 – 16 bitowa stała umieszczona w kodzie rozkazu

addr 16 – 16 bitowy adres przeznaczenia. Wykorzystywany przez instrukcje LCALL oraz LJMP. Skok może być zrealizowany w ramach 64k bajtów przestrzeni adresowej

addr 11 – 11 bitowy adres przeznaczenia. Wykorzystywany przez instrukcje ACALL oraz AJMP. Skok może być zrealizowany wewnątrz tej samej strony (2 kbajty) w pamięci programu co pierwszy bajt instrukcji.

rel – 8 bitowe przesunięcie (offset) w kodzie U2 (*signed*). Wykorzystywane przez instrukcję SJMP i wszystkie instrukcje skoków warunkowych. Zakres wartości: od -128 do +127 względem pierwszego bajta instrukcji

bit – bezpośrednio adresowany bit w ramach wewnętrznej pamięci RAM lub w ramach bloku rejestrów specjalnych (SFR)