PROWADZĄCY: **mgr inż. Piotr Radochoński** **LABORATORIUM Z PODSTAW TECHNIK MIKROPROCESOROWYCH**			
WYKONAWCY :	GRUPA : **1**	ROK AK. **II**	SEMESTR : **4**
Dawid Pichen Leszek Wiland	WYKONANO : **09.03.2005**	ODDANO : **20.03.2005**	OCENA :
NR ĆWICZ. **1**	TEMAT **:** **ZAPOZNANIE SIĘ Z ZESTAWEM DYDAKTYCZNYM ZD537, OPROGRAMOWANIEM µVISION 2 ORAZ OPERACJE NA PAMIĘCIACH**		

Mikrokomputer dydaktyczny ZD537

Na zajęciach laboratoryjnych zapoznaliśmy się z mikrokomputerem dydaktycznym ZD537. Sercem całego zestawu jest ośmiobitowy mikroprocesor 80C537 firmy Infineon. Mikroprocesor ten jest rozszerzoną wersją układu 8051, niemniej jednak jest z nim całkowicie kompatybilny. Na płycie głównej zestawu znajdują się następujące układy: pamięć RAM i ROM (zawiera program zarządzający), złącza portów szeregowych, układ zegara/kalendarza RTC oraz przycisk RESET. Ponadto zestaw zawiera płytę dodatkową na której znalazły się diody LED, klawisze sterujące, buzzer (miniaturowy piszczący głośnik), generator przebiegu prostokątnego oraz stabilizator napięcia zasilania (wymagany, ponieważ zestaw zasilany jest z zasilacza niestabilizowanego). Do dyspozycji mamy także klawiaturę 16-to klawiszową, moduł wyświetlacza LCD (2x16 znaków) oraz rzecz bardzo ważną, czyli przewód łączący zestaw dydaktyczny z komputerem osobistym wyposażonym w port szeregowy RS232. Owy przewód jest wykorzystywany podczas przesyłu programu użytkownika skompilowanego w środowisku µVision 2, do pamięci RAM (układ 431000 na płycie głównej) zestawu.

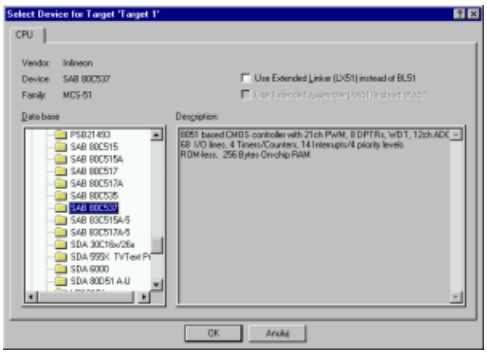
Oprogramowanie µVision 2

Program firmy Keil, to prawdziwy kombajn dla programistów mikroprocesorów serii ’51. Aplikacja umożliwia pisanie i edycję kodu źródłowego (edytor tekstu), kompilowanie kodu, konsolidację, przysłanie skompilowanego kodu do zestawu dydaktycznego oraz emulowanie różnych procesorów z serii ’51 (niezwykle cenny atrybut programu dla osób nieposiadających zestawu ZD537). Oprogramowanie umożliwia kompilację kodu źródłowego języka niskiego poziomu – asemblera, oraz języka wysokiego poziomu – C. Pisanie programów w jęz. C jest wskazane przy budowaniu bardzo dużych programów, gdyż pisząc w tym języku trudniej jest popełnić błąd. Jednakże podczas naszych ćwiczeń laboratoryjnych skupimy się na programowaniu w asemblerze.

Praca z programem

Po uruchomieniu aplikacji pojawia się główne okno programu. Aby utworzyć nowy projekt należy rozwinąć menu `Project` i wybrać opcję `New project...` Pojawi się okno `Create New Project`, należy wpisać nazwę projektu. Zostanie utworzony plik projektu o wskazanej nazwie z rozszerzeniem `uv2`. Następnie aplikacja zapyta o urządzenie

wyjściowe na którym ma być uruchamiany pisany przez nas program. Jak widać oprogramowanie firmy KEIL obsługuje bardzo dużo odmian procesorów ’51. Dla naszego zestawu dydaktycznego należy wskazać układ Infineon SAB 80C537. W tym momencie znowu powrócimy do okna głównego. Przed rozpoczęciem pisania kodu należy wskazać aplikacji informacje o dołączanym (emulowanym) układzie. W tym celu rozwijamy menu `Project` i wybieramy opcję `Options for Target 'Target 1'`.



W głównej zakładce `Target` należy zmienić częstotliwość taktowania układu ze standardowych 16 MHz na 12 MHz. W zakładce `Output` należy zaznaczyć `Debug Information` (zaznaczenie tej opcji powoduje dołączanie informacji potrzebnych dla debuggera do pliku obiektu, te informacje potrzebne podczas testowania programu w trybie emulatora). Następnie w zakładce `Debug` w zależności od trybu pracy należy wybrać odpowiednie opcje. Jeśli nie mamy do dyspozycji zestawu dołączanego do komputera, to należy zaznaczyć przełącznik `Use Simulator` (programowa emulacja mikroprocesora). Jeżeli mamy zestaw, to należy zaznaczyć i wybrać `Use: Keil Monitor-51 Driver` oraz zaznaczyć `Load Application at Startup` (opcja ta powoduje automatyczne przesłanie kodu do zestawu i uruchomienie go po skompilowaniu). Należy także wskazać programowi odpowiedni port szeregowy komputera do którego podłączony jest zestaw dydaktyczny. W tym celu przyciskamy na przycisk `Settings` i wybieramy właściwy port.
Kończymy ustawianie oprogramowania przyciskając na `OK`. W tym momencie możemy zacząć pisać kod źródłowy nowego programu.

Pisanie kodu źródłowego

Z menu `File` wybieramy opcję `New`. Pojawi się nowe okno edycyjne, w które wpisujemy nasz kod źródłowy. Po zakończeniu pisania zapisujemy plik kodu, pamiętając o tym, aby użyć rozszerzenia `asm`, `a` lub `a51`. Następnie należy dodać napisany przed chwilą plik z kodem źródłowym do projektu. Rozwijamy menu `Project` i wybieramy `Targets, Groups, Files...` Pojawi się nowe okno, w którym wybieramy zakładkę `Groups / Add Files`, po czym zaznaczamy `Source Group 1`, i przyciskamy na przycisku `Add Files to Group...` Teraz trzeba wskazać plik z kodem źródłowym (domyślną maską wyświetlania plików są pliki kodów źródłowych języka C, należy zmienić maskę na pliki asemblerowe). Dodany plik powinien pojawić się w głównym oknie programu, w lewym panelu, po rozwinięciu gałęzi `Target 1` oraz `Source Group 1`.
Możemy przystąpić do uruchomienia programu...

Uruchomienie programu

W pierwszym kroku kompilujemy nasz kod źródłowy. Wybieramy opcję `Build target` z menu `Project`. W tym momencie program zacznie kompilację kodu. Jeśli wszystko przebiegnie prawidłowo, program wyświetli informację:

```
Build target 'Target 1'
"1" - 0 Error(s), 0 Warning(s).
```

Jeśli pojawią się błędy lub ostrzeżenia, to znaczy, że w kodzie źródłowym znajdują się błędy i trzeba je poprawić.

Następnym etapem uruchamianie programu jest wejście w tryb debuggera (menu `Debug` → `Start/Stop Debug Session` lub poprzez klawisze skrótu Ctrl+F5). Widać, że po wejściu w ten tryb, zmienia się zawartość głównego okna aplikacji. W panelu umieszczonym po lewej stronie okna można kontrolować rejestry mikroprocesora.
Uruchomienie programu na zestawie dydaktycznym lub w trybie emulacji nastąpi po wybraniu opcji `Go` z menu `Debug` (klawisz skrótu F5) . Możemy zakończyć wykonywanie działania programu poprzez wybranie opcji `Stop Running` w menu `Debug` (lub klawisz Esc). Możliwe jest także uruchomienie aplikacji w trybie krok po kroku, umożliwiającego śledzenie na bieżąco stanów rejestrów i pamięci. Tryb ten jest szczególnie użyteczny podczas odpluskwiania, czyli znajdowania błędów w programie. Uruchamiamy go poleceniem `Step` z menu `Debug`.
Aby powrócić do edycji kodu należy wyjść z trybu debuggera (Crtl+F5).

Operacje na pamięciach

Skompilowany kod programu umieszczany jest w pamięci RAM w przestrzeni adresowej CODE i dostępny jest począwszy od adresu 0x0000. Miejsce od adresu 0x8000 do adresu 0xdeff przeznaczone jest na dane programu. Miejsce to znajduje się w zewnętrznej pamięci RAM (XRAM). W przestrzeni adresowej od 0xdf00 do 0xfeff znajduje się program MONITOR oraz jego zmienne (program w pamięci stałej ROM a zmienne w pamięci RAM). Porty wejścia/wyjścia są dostępne w przestrzeni adresowej od 0xff00 do 0xffff.

Tryby adresowania

Tryby adresowania mówią nam o tym jak wskazywać lokalizację w pamięci. Mikroprocesor 8051 umożliwia wskazywanie pamięci za pomocą 5 różnych sposobów:

- adresowanie natychmiastowe (ang. *immediate addressing*),
- adresowanie bezpośrednie (ang. *direct addressing*),
- adresowanie pośrednie (and. *indirect addressing*),
- adresowanie zewnętrzne bezpośrednie/indeksowe (ang. *external direct addressing*),
- adresowanie zewnętrzne pośrednie (ang. *external indirect addressing*).

Przykład adresowania natychmiastowego:

```
MOV A, #10h
```

Rozkaz ten spowoduje wstawienie wartości 16 (10 szesnastkowo) do akumulatora.

Przykład adresowania bezpośredniego:

```
MOV A, 30h
```

Instrukcja ta odczyta bajt o adresie 0x30 umieszczony w pamięci wewnętrznej i skopiuje do akumulatora.

Przykład adresowania pośredniego:

```
MOV A, @R1
```

Polecenie podobne do poprzedniego, jednakże adres bajtu, co do którego nastąpi operacja czytania znajduje się pod adresem zapisanym w rejestrze R1.

Przykład adresowania zewnętrznego bezpośredniego.
Są tylko dwa polecenia wykorzystujące zewnętrzne adresowanie bezpośrednie:

```
MOVX A,@DPTR
MOVX @DPTR,A
```

Obydwa rozkazy używają rejestru DPTR. DPTR to wskaźnik danych (ang. *data pointer*), jest to jedyny 16 bitowy rejestr mikroprocesora 8051. 16 bitów pozwala na zaadresowanie 64kB danych. Z tego względu wskaźnik ten służy głównie do wskazywania obszarów w pamięci zewnętrznej. Pierwszy napisany rozkaz spowoduje skopiowanie zawartości obszaru pamięci wskazywanej przez DPTR do akumulatora, a drugi operację odwrotną, czyli przepisanie zawartości akumulatora do miejsca wskazywanego przez wskaźnik danych.

Przykład adresowania zewnętrznego pośredniego:

```
MOVX @R2, A
```

Zewnętrzna pamięć może być także adresowana w sposób pośredni. Ta forma adresowania jest używana wyłącznie w małych programach, które wymagają niewiele zewnętrznej pamięci. Rozkaz powyższy skopiuje zawartość akumulatora do miejsca wskazywanego przez rejestr R2. Ponieważ wszystkie rejestry oprócz DPTR są ośmiobitowe, zatem polecenie to potrafi zaadresować wyłącznie 256 bajtów pamięci zewnętrznej.

Przykłady programów działających na różnych pamięciach systemu dydaktycznego

Program № 1

```
$NOMOD51
$INCLUDE (REG517.INC)
CSEG AT 0    ;segment kodu zaczyna się od adresu 0x0000

MOV R0,#10   ;zmienna 1.
MOV R1,#5    ;zmienna 2.
MOV A,R0
ADD A,R1     ;suma
MOV R2,A     ;wynik w R2
MOV A,R0
SUBB A,R1    ;różnica
MOV R3,A     ;wynik w R3
MOV A,R0
MOV B,R1
PUSH B       ;zawartość rej. B wstawiona na stos
MUL AB       ;iloczyn
MOV R4,A     ;wynik w R4
MOV A,R0
POP B        ;do rej. B wstawiana zawartość ze stosu
DIV AB       ;iloraz
MOV R5,A     ;wynik w R5
JMP $

END
```

Program powyższy dokonuje podstawowych działań arytmetycznych na dwóch liczbach (10 i 5) umieszczonych w rejestrach R0 i R1. Wyniki tych działań (suma, różnica, iloczyn, iloraz) umieszczane są kolejno w rejestrach od R2 do R5. W programie wykorzystałem stos, ażeby nie stracić zawartości rejestru B w wyniku mnożenia. Program posługuje się podczas działania wyłącznie pamięć wewnętrzną.

Program № 2

```
$NOMOD51
$INCLUDE (REG517.INC)
CSEG AT 0

MOV DPTR,#TABLICA   ;ustawiamy wskaźnik na tablicę
MOV R0,#01H         ;rejestr R0 zawiera adres rejestru R1

PETLA:              ;pętla kopiująca
CLR A               ;zerowanie akumulatora
MOVC A,@A+DPTR      ;akumulator=wartość pod adr. DPTR
JZ KONIEC           ;jeśli A=0, to skok do KONIEC
MOV @R0, A          ;wpisz A pod adres umieszczony w R0
INC R0              ;inkrementuj R0
INC DPTR            ;inkrementuj wskaźnik DPTR
JMP PETLA           ;skok do PETLA

KONIEC:
JMP $

TABLICA: DB 'Dawid',0
JMP $
END
```

Program ten kopiuje zawartość tablicy umieszczonej w pamięci kodu (CODE) do wewnętrznej pamięci danych. Kopiowane dane umieszczane są począwszy od adresu 0x01 (czyli rejestru R1). Pętla przerywa kopiowanie wtedy, kiedy wartość pobrana z pamięci kodu jest równa zeru, czyli mówi o tym, że nastąpił koniec łańcucha znaków.

Program № 3

```
$NOMOD51
$INCLUDE (REG517.INC)
CSEG AT 0

MOV DPTR,#TABLICA   ;wskaźnik na tablicę
MOV P2, #80H        ;ustawianie adr. 0x8000 w pamięci
                    ;zewnętrznej (sterownie portem P2)
MOV R0,#00H         ;rejestr R0 zawiera przesunięcie adresowe
                    ;(offset) w pamięci zewnętrznej

PETLA:              ;pętla kopiująca
CLR A               ;zerowanie akumulatora
MOVC A,@A+DPTR      ;akumulator=wartość pod adr. DPTR
JZ KONIEC           ;jeśli A=0, to skok do KONIEC
MOVX @R0, A         ;wpisz A do pamięci zewnętrznej pod adres
                    ;0x8000 plus przesunięcie, będące
                    ;wartością umieszczoną w rejestrze R0
INC R0              ;inkrementuj R0
INC DPTR            ;inkrementuj wskaźnik DPTR
```

```
JMP PETLA           ;skok do PETLA

KONIEC:
JMP $

TABLICA: DB 'Leszek',0
JMP $
END
```

Tym razem kopiujemy tablicę z pamięci kodu do pamięci zewnętrznej. Jeśli pamięć zewnętrzna jest podłączona, to w naszym zestawie dydaktycznym wartość wysłana przez port P2 wskazuje starszy bajt adresu pamięci. Ponieważ przestrzeń począwszy od adresu 0x8000 przeznaczona jest na zmienne programu, zatem używamy jej jako miejsce kopiowanych danych. Używamy adresowania zewnętrznego pośredniego, gdyż tablica danych jest mniejsza niż 256 bajtów.

Zmodyfikowany program testowy zestawu ZD537

Postanowiliśmy zmodyfikować program testowy, tak aby na wyświetlaczu pojawiły się nasze imiona. Zmieniliśmy także miejsce wyświetlania znaku klawiszu oraz zachowania na naciśnięcie klawisza (tzn. wyłączenie buzzera, zmiana zapalonej diody).
Oto zmodyfikowany kod:

```
$NOMOD51
$INCLUDE (REG517.INC)
$INCLUDE (LCD.INC)
$INCLUDE (KEYB.INC)
CSEG AT 0

JMP PROGRAM_MAIN
ORG 0Bh         ;obsługa przerwania od timera
CPL P3.2
RETI

ORG 100h
PROGRAM_MAIN:
ANL P6, # NOT 10h
ORL P6, # 01h
CALL LCDinit
CALL LCDcursorOff
MOV DPTR, #LCD_SZABLON1
CALL LCDputs_C
CALL LCDhome2
MOV DPTR, #LCD_SZABLON2
CALL LCDputs_C
MOV C, P1.7
MOV 0, C
MOV 21, #' '
MOV 22, #0
MOV IEN0, #82h
MOV TMOD, #02h
```

```
MOV TH0, #06h
ANL P6, # NOT 40h

PETLA:
CALL GetKey
CJNE A, #0FFh, KLAW_DEK
MOV A, #' '
JMP KLAW_WYSW
KLAW_DEK:
MOV DPTR, #TABZNAKOW
MOVC A, @A+DPTR
KLAW_WYSW:
CJNE A, 21, KLAW_WYSW2
JMP KLAW_P3
KLAW_WYSW2:
MOV 21, A
CALL LCDhome1
MOV B, #12
CALL LCDrmoveB
CALL LCDputchar

CJNE A, #'*', BUZP32_OFF
SETB TR0
JMP RELAY
BUZP32_OFF:
SETB P3.2
CLR TR0

RELAY:
CJNE A, #'0', RELAY_OFF
ORL P6, #40h
JMP BUZP64
RELAY_OFF:
ANL P6, # NOT 40h

BUZP64:
CJNE A, #'#', BUZP64_OFF
ORL P6, #10h
JMP REDLED
BUZP64_OFF:
ANL P6, # NOT 10h

REDLED:
CJNE A, #'D', REDLED_OFF
ANL P6, # NOT 01h
JMP KLAW_P3
REDLED_OFF:
ORL P6, # 01h

KLAW_P3:
MOV A, P3
```

```
JNB TR0, P3_DEK
ORL A, #04h
P3_DEK:
CPL A
RR A
RR A
ANL A, #0Fh
CJNE A, 22, P3_WYSW
JMP P1LICZNIK
P3_WYSW:
MOV 22, A
CALL LCDhome2
MOV B, #12
CALL LCDrmoveB
CALL LCD_CyfraHex

P1LICZNIK:
MOV C, P1.7
MOV 1, C
MOV C, 0
ANL C, /1
MOV 2, C
MOV C, 1
ANL C, /0
ORL C, 2
JNC PETLA
MOV C, 1
MOV 0, C
MOV A, P1
DEC A
ANL A, #7Fh
ANL P1, #80h
ORL P1, A
JMP PETLA

LCD_CyfraHex:
PUSH ACC
CLR C
SUBB A, #0Ah
JC MNIEJ10
ADD A, #'A'
SJMP WYSW
MNIEJ10:
ADD A,#('0' + 0Ah)
WYSW:
LCALL LCDputchar
POP ACC
RET

TABZNAKOW: DB '*0#D789C456B123A'
LCD_SZABLON1: DB 'Dawid Pichen', 0
```

```
LCD_SZABLON2: DB 'Leszek Wiland', 0
END
```

Uwagi i wnioski

Zapoznaliśmy się z zestawem dydaktycznym ZD537 oraz z oprogramowaniem µVision 2 firmy KEIL. Mimo, że mikroprocesory ’51 są jednymi z najprostszych procesorów dostępnych na rynku, to ich programowanie nie jest wcale łatwe. Praktycznie niezbędna jest znajomość architektury mikrokomputera. Pisząc programy należy szczególnie uważać na zmiany wartości rejestrów w trakcie wykonywania rozkazów. Programowanie w asemblerze nie jest tak intuicyjne, jak np. w języku C. Programista musi przewidywać zachowania poszczególnych rozkazów.
Poznaliśmy tryby adresowania pamięci oraz napisaliśmy programy umożliwiające pracę na różnych jej rodzajach (p. kodu, p. wewnętrzna danych, p. zewnętrzna). Dowiedzieliśmy się, że zewnętrzna pamięć jest adresowana dwa portami równoległymi mikroprocesora (portami P0 i P2).