Politechnika Świętokrzyska

Laboratorium

Mikrokontrolerów

Ćwiczenie 1

Programowanie w asemblerze

dr inż. Robert Kazała

Kielce 2015

Cel ćwiczenia

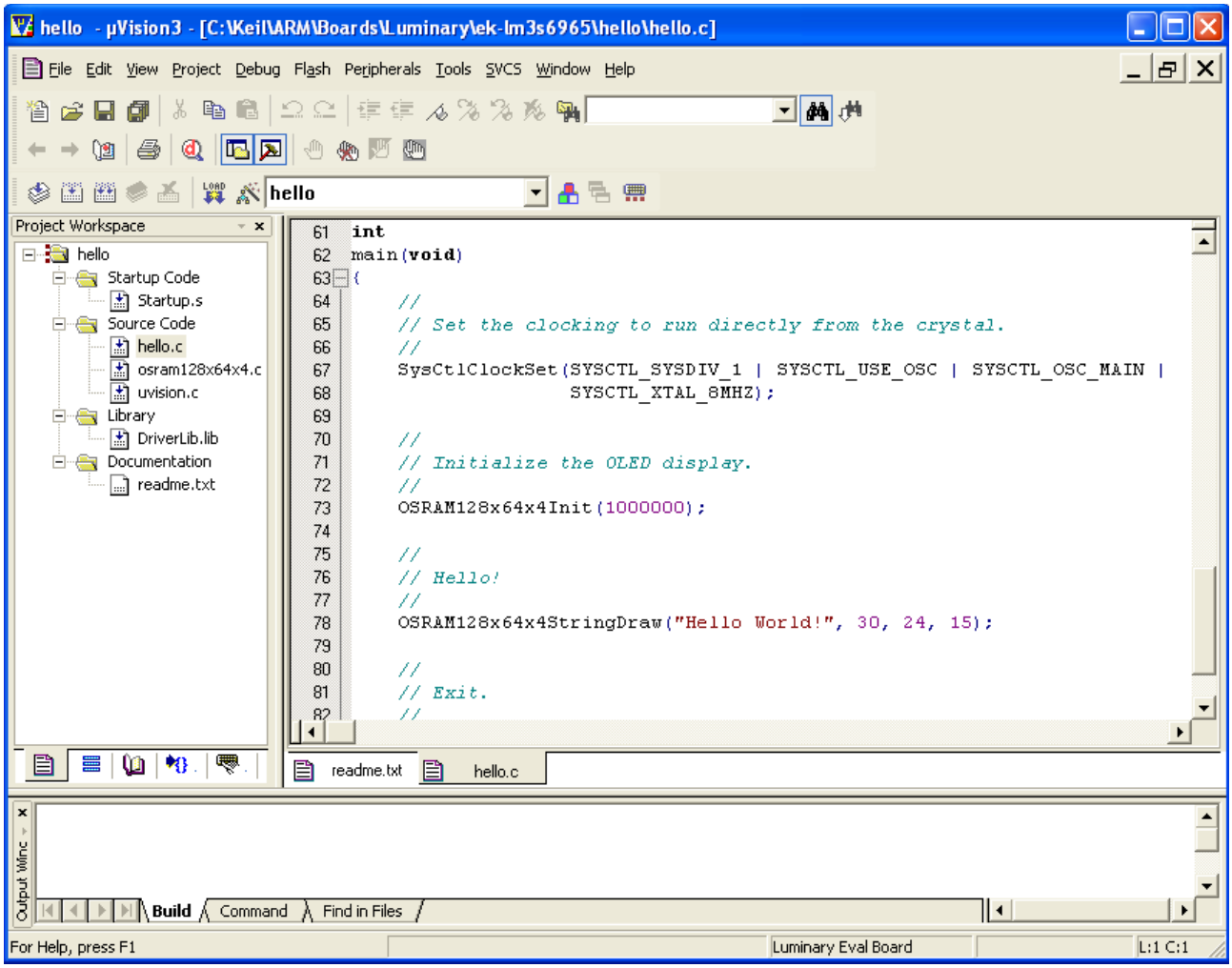
Celem ćwiczenia jest zdobycie umiejętności obsługi środowiska programistycznego Keil uVision dla procesorów z rdzeniem ARM. Poznanie struktury programu w języku asemblera oraz zapoznanie się z podstawowymi instrukcjami asemblera. Wykonanie przykładowych programów zamieszczonych w instrukcji oraz napisanie programów zadanych przez prowadzącego.

Środowisko programistyczne Keil uVision

Środowisko programistyczne Keil uVision umożliwia programowanie różnych typów mikrokontrolerów. Podstawowe elementy tego środowiska to:

- edytor tekstu,
- kompilator i linker,
- symulator rdzenia procesora,
- symulator otoczenia procesora,
- graficzny debuger.

Po uruchomieniu programu wyświetla się okno przedstawione na rysunku.

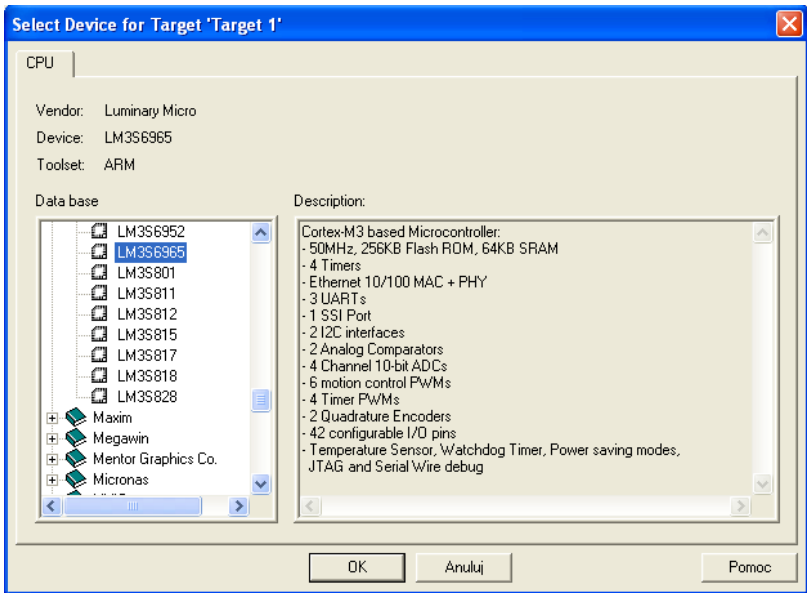


W oknie wyróżnić można:

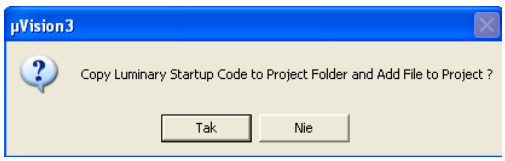
- menu,
- paski narzędziowe,
- okno projektu,
- edytor tekstu,
- okno komunikatów.

Tworzenie nowego projektu

W celu utworzenia nowego projektu należy z menu *Project* wybrać opcję *New* , a następnie wywołać *uVision Project....* Pojawi się wtedy okno pozwalające wpisać nazwę nowo tworzonego projektu. Po zatwierdzeniu wpisanej nazwy pojawi się okno dialogowe pozwalające wybrać typ mikrokontrolera. Okno to ma następujący wygląd.



W oknie tym należy wybrać producenta Luminary Micro oraz typ LM3S6965. Po zatwierdzeniu pojawi się zapytanie, czy ma być do projektu dołączony plik z kodem startowym.



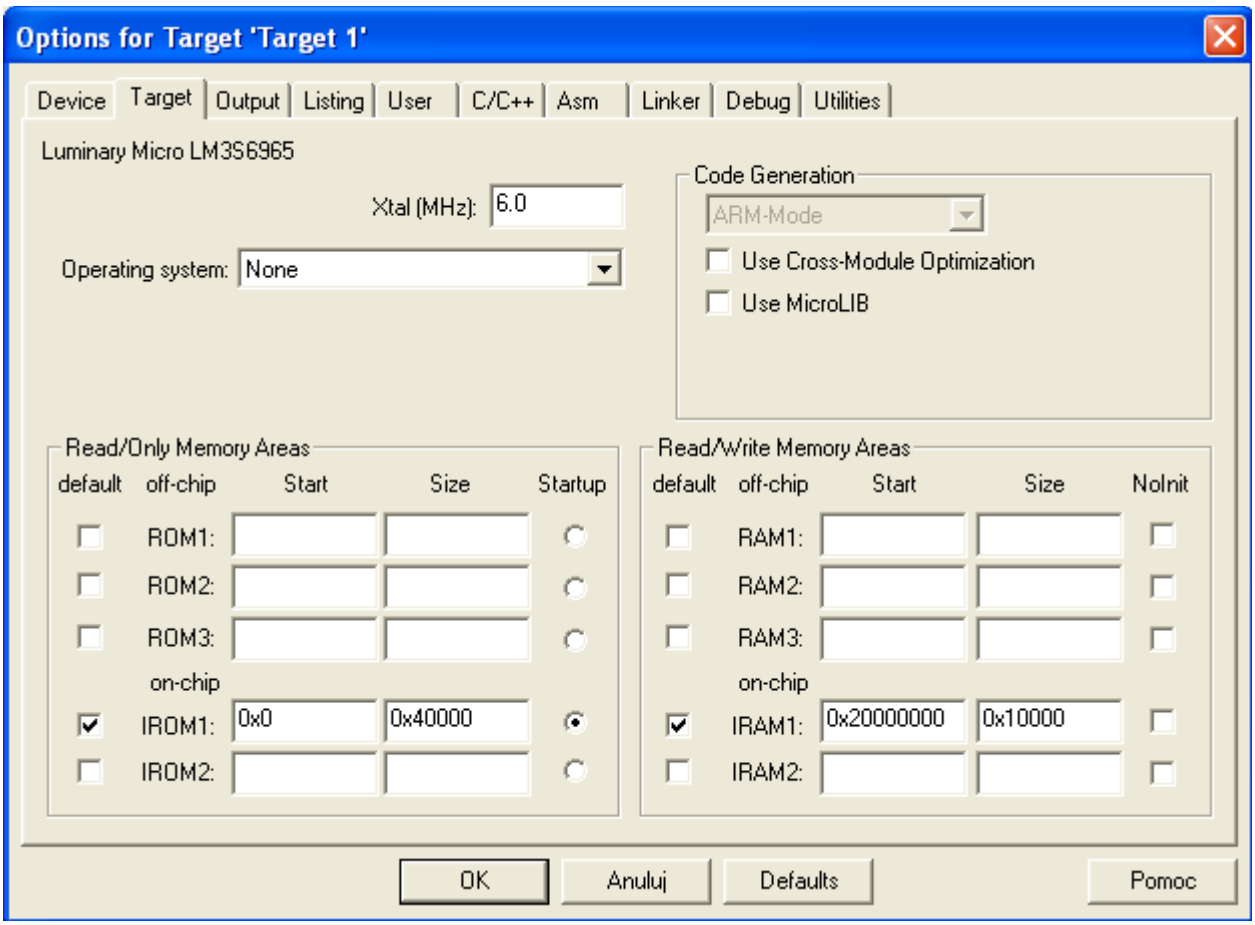
W oknie z zapytaniem należy potwierdzić dołączenie pliku do projektu. Po zatwierdzeniu środowisko jest wstępnie skonfigurowane.

W oknie Project Workspace w grupie Source Group 1 powinien być widoczny plik startup.s. Plik ten można otworzyć klikając na nim dwukrotnie myszą.

W kolejnym kroku należy utworzyć własny plik z programem. W tym celu należy wybrać z menu *File* pozycję *New....* Pojawi się wtedy w oknie edytora plik o nazwie *Text1*. Plik ten należy zapisać za pomocą opcji *Plik/Save as...* Plik może mieć dowolną nazwę z rozszerzeniem *„s”*dla programów pisanych w asemblerze i rozszerzenie *„c”* dla programów pisanych w języku C. Po zapisaniu plik ten należy dodać do projektu klikając prawym klawiszem myszy w oknie *Project Workspace* na grupie *Source Group 1* i w wyświetlonym menu podręcznym wybrać opcję *Add Files to Group*. Pojawi się wtedy okno wyboru pliku. W oknie tym należy ustawić typ pliku na *ASM Source file* dla programów w asemblerze i następnie wybrać utworzony wcześniej plik. W tym momencie środowisko jest przygotowane do rozpoczęcia pisania programu.

Kompilacja programu

Przed przystąpieniem do kompilacji programu należy ustawić parametry projektu wybierając w menu Project pozycję Options for Target. Pojawia się wtedy następujące okno dialogowe.



W oknie tym w pozycji *Code Generation* należy zaznaczyć opcję *Use MicroLib*. Dodatkowo można ustawić wartość oscylatora w pozycji *Xtal(MHz)* na 8. Zapewni to zgodność odmierzanego czasu z systemem uruchomieniowym wykorzystywanym do zajęć.
W celu użycia symulatora do debugowania programu należy wyświetlić zakładkę *Debug* i ustawić opcję *Use Simulator.*

Struktura modułów asemblera.

Moduły napisane w języku asemblera są przetwarzane przez ASM asembler do postaci kodu obiektowego. Pliki mogą zawierać programy lub funkcje. Pojedyncza sekcja kodu wystarczy do utworzenia aplikacji.

Pliki przetwarzane są do formatu ELF i następnie łączone są w linkerze do postaci kodu wynikowego. Zasady rozmieszczania kodu i danych definiowane są na poziomie kodu asemblera i zapisywane w plikach źródłowych.

Każdy moduł przetworzony do formatu ELF jest niezależny, ma swoją nazwę i zawiera kod lub dane, które będą rozmieszczone przez linker w pamięci mikrokontrolera. Sekcje kodu są najczęściej sekcjami z atrybutem tylko do odczytu, sekcje danych umożliwiają zapis i odczyt.

Linia programu asemblera ma następującą postać

{etykieta} {instrukcja|dyrektywa|} {;komentarz}

Uwaga! Jeżeli w kodzie nie ma etykiety instrukcje muszą być poprzedzone spacją lub tabulatorem.

Instrukcje i dyrektywy mogą być pisane małymi lub dużymi literami. Nie mogą być jednak mieszane.

Linie, które są za długie mogą być dzielone przy wykorzystaniu znaku \ (backslash) umieszczanego na końcu linii. Znak ten nie może być oddzielony od poprzedzającej go instrukcji za pomocą spacji lub tabulatora. W takim przypadku jest on pomijany.

Etykiety

Etukiety reprezentują adresy. Adresy wskazywane przez etykiety są obliczane w czasie asemblacji kodu. Etykiety mogą być definiowane jednokrotnie.

Etykiety lokalne

Etykiety lokalne rozpoczynają się liczbą od 0-99. W przeciwieństwie do etykiet mogą być definiowane wiele razy. Jeżeli asembler znajdzie skok do takiej etykiety to wykonuje skok do najbliższej etykiety o wskazanej nazwie.
Widoczność etykiet lokalnych jest ograniczona do jednego obszaru AREA.

Komentarze

Pierwszy średnik umieszczony w linii programu oznacza rozpoczęcie komentarza. Koniec linii jest zakończeniem komentarza. Dopuszczalne są linie tylko z samym komentarzem.

Przykład 1

Przykładowy program napisany w asemblerze.

```
      AREA Przyk1, CODE, READONLY   ; Nazwa bloku kodu Przyk1

      ENTRY                    ; Oznaczenie początku programu
start
      MOV r0, #10              ; Zapis wartości 10 do rejestru r0
      MOV r1, #3               ; Zapis wartości 3 do rejestru r1
      ADD r0, r0, r1           ; r0 = r0 + r1

stop  B stop                   ; Skok do etykiety stop

      END               ; Koniec pliku
```

W pliku źródłowym dyrektywa AREA oznacza rozpoczęcie sekcji. Ta dyrektywa umożliwia podanie nazwy sekcji oraz jej atrybutów. Atrybuty umieszczane są po nazwie sekcji i oddzielone przecinkami. Nazwa może być dowolnym zestawem znaków. W przypadku znaków rozpoczynających spoza alfabetu wymaga otoczenia kreskami np. |1_ObszarDanych|. W przykładowym kodzie zdefiniowana jest jedna sekcja o nazwie ARMex, która zawiera kod i jest oznaczona jako tylko do odczytu READONLY.

Dyrektywa ENTRY oznacza pierwszą instrukcję, która będzie wykonywana. W pliku zdefiniowano dodatkowo dwie etykiety start i stop, które nie są wymagane. Pomiędzy etykietami zapisany jest zasadniczy kod sekcji.

Po wykonaniu kodu wykonywana jest instrukcja skoku do etykiety stop.

Dyrektywa END informuje asembler o zakończeniu pliku źródłowego.

Własności drugiego argumentu instrukcji

Wiele instrukcji przetwarzania danych ARM i Thumb-2 ma elastyczny drugi argument.
Może on przyjać dwie następujące formy:

#constant

Rm{, shift}

gdzie:
constant jest wyrażeniem definiującym stałą numeryczną.

Rm jest rejestrem przechowującym daną. Bity w rejestrze mogą być przesuwane na różne sposoby.

Przesunięcie może być jednym z następujących typów:

ASR #n	arytmetyczne przesunięcie w prawo o n bitów. $1 \leq n \leq 32$.
LSL #n	logiczne przesunięcie w lewo o n bitów. $0 \leq n \leq 31$.
LSR #n	logiczne przesunięcie w prawo o n bitów. $1 \leq n \leq 32$.
ROR #n	rotacja w prawo o n bitów. $1 \leq n \leq 31$.
RRX	rotacja w prawo o jeden bit z uzupełnieniem.

Argument określający przesunięcie może być rejestrem Rs, ale tylko dla zestawu instrukcji ARM.

W zestawie instrukcji ARM stała może mieć dowolną wartość jaką można uzyskać przez rotację 8-bitowej wartości w ramach 32-bitowego słowa.
W 32-bitowym zestawie instrukcji Thumb-2 stała może być:
• dowolna stała wyznaczona przez przesunięcie 8-bitowej wartości w lewo o dowolną liczbę bitów w słowie 32-bitowym,
• dowolna stała postaci 0x00XY00XY,
• dowolna stała postaci 0xXY00XY00,
• dowolna stała postaci 0xXYXYXYXY.
Dodatkowo niewielka ilość instrukcji pozwala na szerszy zakres wartości
Stałe wyznaczone przez przesunięcie 8-bitowej wartości w prawo o 2, 4, lub 6 bitów są dostępne w zestawie instukcji ARM, lecz nie w Thumb-2. Wszystkie inne stałe ARM są dostępne w Thumb-2.

Instrukcje dostępu do pamięci oraz dyrektywy deklarujące stałe, zmienne i obszary danych.

W programach pisanych w asemblerze często istnieje potrzeba inicjowania rejestrów za pomocą stałych wartości liczbowych lub ciągów znaków. Bezpośrednia zapis wartości argumentów dla większości instrukcji jest ograniczony do zakresu 0-255, na którym można dodatkowo wykorzystać operację przesunięcia.

Instrukcje MOV, MOVT, MOV32

W celu przekazania stałej wartości można wykorzystać instrukcje MOV, która pozwala przekazać 16-bitową wartość do rejestry. W celu wypełnienia bardziej znaczących bitów można wykorzystać instrukcję MOVT. Użycie obydwu instrukcji pozwala zapisać wartość do całego 32-bitowego słowa. W celu uproszczenia tego zadania wprowadzona została pseudo-instrukcja MOV32

pozwalająca w jednej linii programu zainicjować wartość 32-bitową.

Jeżeli potrzebne są deklaracje większej ilości danych i celowe jest nadanie zmiennym nazw symbolicznych wygodnie jest zadeklarować te wartości z wykorzystaniem dyrektyw asemblera EQU i DCD.

Dostęp do danych zapisanych w pamięci możliwy jest poprzez wykorzystanie instrukcji
LDR – odczyt danej z pamięci i zapisanie do rejestru
STR – zapis danej z rejestru do pamięci

Instrukcje warunkowe

W trybie ARM i Thumb-2 większość instrukcji ma możliwość ustawiania flag ALU w rejestrze CPSR, w zależności od wyniku operacji. W trybie Thumb flagi są zawsze ustawiane i nie ma możliwości ich nie ustawiania.

Rejestr CPSR zawiera następujące flagi ALU:
- N ustawiana gdy wynik operacji jest ujemny,
- Z ustawiana gdy wynik operacji jest zero,
- C ustawiana jeżeli wystąpiło przeniesienie
- V ustawiane jeżeli wystąpiło przepełnienie.

Przeniesienie występuje, jeżeli wynik dodawania jest większy lub równy 2^32, jeżeli wynik odejmowania jest dodatni. Przepełnienie występuje jeżeli dodawanie, odejmowanie lub porównywanie jest większu lub równe 2^31 lub mniejsze niż -2^31

W procesorach z rdzeniami ARM istnieje kilka sposobów realizacji instrukcji warunkowych:
- warunkowe wykonywanie instrukcji w trybie ARM,
- zastosowanie instrukcji IT w trybie Thumb-2,
- instrukcje rozgałęzień umożliwiające sprawdzanie warunków.

Prawie wszystkie instrukcje w trybie ARM mogą być warunkowo wykonywane w zależności od stanu flag w rejestrze CPSR.
W trybie Thumb-2 instrukcje mogą być warunkowo wykonywane poprzez zastosowanie instrukcji IT.

Kody warunków są podane w tabeli.

Suffix	Flaga	Znaczenie
EQ	Z set	Equal
NE	Z clear	Not equal
CS/HS	C set	Higher or same (unsigned >=)
CC/LO	C clear	Lower (unsigned <)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned >)
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V differ	Signed <
GT	Z clear, N and V the same	Signed >

LE	Z set, N and V differ	Signed <=
AL	Any	Always. Ten suffix jest zazwyczaj pomijany.

Instrukcje skoków i rozgałęzień

W procesorach ARM dostępne są następujące instrukcje rozgałęzień:

- *B, BL, BX, BLX, and BXJ* (Branch, Branch with Link, Branch and exchange instruction set, Branch with Link and exchange instruction set, Branch and change instruction set to Jazelle),
- *CBZ and CBNZ* (Compare against zero and branch).
- *TBB and TBH* (Table Branch Byte or Halfword).

Instrukcje B, BL,

Instrukcje B, BL mają taką następującą składnię wywołania
op{cond}{.W} label
op{cond} Rm

gdzie:

op jest nazwą instrukcji
cond jest warunkiem wykonania, który nie zawsze może być użyty.

Znaczenie poszczególnych instrukcji jest następujące:

- B rozgałęzienie,
- BL rozgałęzienie z zapamiętaniem adresu w rejestrze r14 (lr, link register),

Wszystkie instrukcje umożliwiają wykonanie skoku do etykiety lub adresu znajdującego się w rejestrze.

Przykłady

```
B loopA
BLE ng+8
BL subC
BLLT rtX
BEQ {pc}+4 ; #0x8004
```

Instrukcje CBZ i CBNZ

Instrukcje CBZ i CBNZ umożliwiają wykonanie skoku jeżeli wartość w rejestrze jest równa zero lub różna od zera. Składnia tych instrukcji jest następująca:

CBZ Rn, label
CBNZ Rn, label
gdzie:
Rn jest rejestrem przechowującym argument operacji,
label jest celem rozgałęzienia.

Instrukcja ta jest dostępna tylko w trybie Thumb-2 i nie może być wykonywana w bloku IT.

Instrukcja CBZ Rn, poza tym że nie ustawia flag, odpowiada następującemu wywołaniu:
CMP Rn, #0
BEQ label

Instrukcja CBNZ Rn, poza tym że nie ustawia flag, odpowiada następującemu wywołaniu:
CMP Rn, #0
BNE label

Ograniczeniem tej instrukcji związane jest z tym, że adres skoku musi sie zawierać od 4 do 130 bajtów za instrukcją

Instrukcje ADD, SUB, RSB, ADC, SBC, RSC.

Sposób wywołania

op{S}{cond} {Rd}, Rn, Operand2

op{cond} {Rd}, Rn, #imm12 ; dla Thumb-2 tylko ADD i SUB

imm12 – jest dowolną wartością z zakresu 0-4095

Instrukcja	Opis
ADD	Dodaje wartość Rn i Operand2
SUB	Odejmuje wartość Operand2 od Rn
RSB	Odejmuje wartość Rn od Operand2
ADC	Dodaje wartość Rn i Operand2 oraz bit przeniesienia
SBC	Odejmuje wartość Operand2 od Rn. Jeżeli bit przeniesienia jest wyzerowany to wynik jest zmniejszany o jeden.
RSC	Odejmuje wartość Rn od Operand2. Jeżeli bit przeniesienia jest wyzerowany to wynik jest zmniejszany o jeden.

Przykłady wywołań

ADD r2, r1, r3

SUBS r8, r6, #240

RSB r4, r4, #1280

ADCHI r11, r0, r3

SBC r5, r8, r11

Wywoływanie funkcji

W celu wywołania funkcji należy wykonać instrukcję skoku

BL cel

gdzie cel jest etykietą opisującą pierwszą instrukcję wywoływanej funkcji. Do wywołania funkcji można wykorzystać także instrukcBXJ.

Istrukcja BL:

- odkłada adres powrotu w rejestrze
- ustawia PC na adres wywoływanej funkcji

Po wykonaniu funkcji należy wywołać instrukcję

B lr

lub

MOV pc, lr

Rejestry r0 do r3 są zazwyczaj wykorzystywane do przekazywania parametrów do funkcji, po wywołaniu funkcji r0 jest wykorzystywane do zwracania wyniku.

Przykład 2

Przykładowy kod funkcji

```
      AREA subrout, CODE, READONLY  ; Nazwa programu

      ENTRY                   ; Oznaczenie początku programu

start       MOV r0, #10       ; Zapis wartości 10 do rejestru r0
      MOV r1, #3              ; Zapis wartości 3 do rejestru r0
      BL doadd                ; Wywołanie funkcji
stop  B stop

doadd       ADD r0, r0, r1          ; Kod funkcji
      BX lr                   ; Powrót z funkcji

      END               ; Koniec pliku
```

Przykład 3

Kopiowanie ciągów znaków zadeklarowanych jako tablice bajtów.

```
       AREA StrCopy, CODE, READONLY  ; Nazwa bloku kodu

       ENTRY ; Pierwsza wykonywana instrukcja
start
       LDR r1, =srcstr ; Wskaźnik na pierwszy ciąg
       LDR r0, =dststr ; Wskaźnik na drugi ciąg
       BL strcopy ; wywołanie funkcji kopiującej
stop
       B stop
strcopy
       LDRB r2, [r1],#1 ; Załadowanie bajtu i zwiększenie wskaźnika
       STRB r2, [r0],#1 ; Zapisanie bajtu i zwiększenie wskaźnika
       CMP r2, #0 ; Sprawdzenie warunku
       BNE strcopy ; Jeżeli nie spełniony to wykonaj skok
       MOV pc,lr ; Powrót z funkcji


srcstr DCB "First string - source",0

       AREA Strings, DATA, READWRITE

dststr DCB "Second string – destination",0

       END
```

Przykład 4

Kopiowanie bloku danych

```
        AREA Word, CODE, READONLY ; Nazwa bloku kodu

num     EQU 20 ; Ustawienie liczby kopiowanych znaków

        ENTRY ; Zaznaczenie pierwszej wykonywanej instrukcji
start
        LDR r0, =src ; r0 = wskaźnik na blok źródłowy
        LDR r1, =dst ; r1 = wskaźnik na blok docelowy
        MOV r2, #num ; r2 = liczba kopiowanych znaków

wordcopy
        LDR r3, [r0], #4 ; Załadowanie znaku ze źródła
        STR r3, [r1], #4 ; Zapisanie znaku
        SUBS r2, r2, #1 ; Zmniejszenie licznika
        BNE wordcopy ; ... dalsze kopiowanie
stop
        B stop

src     DCD 1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4

        AREA BlockData, DATA, READWRITE


dst     DCD 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

        END
```

Przebieg ćwiczenia

1. Uruchomić program uVision i zapoznać się z opcjami menu, paskami narzędziowymi oraz obsługą programu.

2. Utworzyć następujący program

```
    THUMB                    ;use THUMB instruction set

    EXPORT  __main

    AREA    Program, CODE, READONLY

__main

;     kod programu
start
      mov r0, #10
      mov r1, #3
      add r0, r0, r1
stop
      B stop

    END
```

3. Dokonać kompilacji programu poleceniem *Build target*. Jeżeli kompilacja nie będzie miała błędów uruchomić program w symulatorze wywołując *Start/Stop Debug Session*. Pojawi się wtedy komunikat o braku urządzenia. Należy go potwierdzić przyciskiem *Ok*. Po potwierdzeniu wyświetli się okno symulatora.
   W oknie utworzonego programu założyć pułapkę na pierwszej linii z instrukcją mov. Następnie uruchomić program poleceniem Run. Po zatrzymaniu wykonywania programu na pułapce poleceniem Step wykonać kolejne instrukcje. W tym czasie należy obserwować zawartość rejestrów procesora.
   Następnie zmienić wartości liczbowe i ponownie sprawdzić działanie programu.

4. Wykonać i przeanalizować przykłady 2, 3, 4.
5. Uruchomić i przeanalizować program

```
    THUMB                    ;use THUMB instruction set
    EXPORT  __main
    AREA    Program, CODE, READONLY
__main
;     kod programu

      ldr  R1, Value
      LDR  R2, =Result
      str  R1, [R2]

    B       __main

Value DCW &C123

      AREA Dane, DATA, READWRITE
Result DCW  0
    END
```

6. Napisać programy prezentujące różne sposoby zadawania drugiego parametru instrukcji.
7. Przeanalizować działania instrukcji LDR i STR i napisać własne programy wykorzystujące te instrukcje.

8. Napisać program znajdujący większa z dwóch wartości.
9. Napisać program wyznaczający sumę elementów w tablicy.
10. Napisać program znajdujący największą wartość w tablicy.
11. Napisać program porównujący dwa ciągi znakowe i informujący czy są jednakowe