

Podstawy Programowania Systemów Wbudowanych
Instrukcja do ćwiczeń laboratoryjnych z ARM
2017
Mirosław Żołądź
Piotr Otfinowski

Spis treści

1. Mikrokontroler	2
2. Rejestry robocze.....	2
3. Warunkowe wykonywanie instrukcji	2
4. Warunkowa modyfikacja rejestru statusu	2
5. Tworzenie projektu.	2
Ćwiczenie 1	3
Ćwiczenie 2	3
Ćwiczenie 3	3
Ćwiczenie 4	3
Ćwiczenie 5	3
Ćwiczenie 6	4
Ćwiczenie 7	4
Ćwiczenie 8	4
Ćwiczenie 9	5
Ćwiczenie 10	5

1. Mikrokontroler

Część ARM ćwiczeń laboratoryjnych z przedmiotu PPSW odbywa się z użyciem zestawu uruchomieniowego używanego na przedmiocie MITP. Wspomniany zestaw jest wyposażony w procesor ARM z rdzeniem 7TDMI. Należy przyjąć, że procesor pracuje w trybie z oryginalnym zestawem instrukcji ARM (nie Thumb).

Szczegółowe informacje: KEIL -> Help -> Assembler User Guide -> Overview of the ARM Architecture
-> Instruction set overview -> ARM and Thumb Instructions

2. Rejestry robocze

Ze względu na tryb pracy procesora (User Mode, ustawiany przez kod startowy, patrz „Tworzenie projektu”) należy korzystać z rejestrów roboczych R0-R12.

Szczegółowe informacje: ... -> ARM and Thumb Instructions-> ARM registers oraz General-purpose registers

3. Warunkowe wykonywanie instrukcji

Procesory ARM pozwalają na warunkowe wykonywanie instrukcji (nie mylić z ich pomijaniem przy pomocy skoków). To czy instrukcja będzie wykonana uzależnione jest od stanu określonych bitów w rejestrze statusu. Które to bity oraz jaką wartość powinny mieć aby instrukcja została wykonana określa suffix.

Przykład:

ADD r0, r1, r2 ; r0 = r1 + r2

ADDCS r0, r1, r2 ; If C flag set then r0 = r1 + r2

Szczegółowe informacje dotyczące warunkowej modyfikacji rejestru statusu oraz warunkowego wykonywania instrukcji znajdują się w: KEIL -> Help -> Assembler User Guide -> Condition Codes

4. Warunkowa modyfikacja rejestru statusu

Procesory ARM pozwalają instrukcjom na warunkową modyfikację flag w rejestrze statusu. Jeżeli do instrukcji dołączony jest sufiks "S" instrukcja może zmodyfikować rejestr statusu (np. ustawić flagę "Carry" w przypadku przepełnienia). W przeciwnym przypadku (brak suffixu "S") rejestr statusu nie jest modyfikowany.


Przykład:

ADD r0, r1, r2 ; r0 = r1 + r2, don't update flags

ADDS r0, r1, r2 ; r0 = r1 + r2, and update flags

Szczegółowe informacje : ... -> Condition Codes -> Updates to the ALU status flags

5. Tworzenie projektu.

1. Utwórz nowy projekt KEILa, wybierz mikrokontroler „LPC2131”, dodaj kod startowy.
2. Skopiuj do katalogu projektu a następnie dodaj do projektu plik „arm_skeleton.s”.
4. W ustawieniach projektu () wybierz: „Output -> Create HEX File” oraz „Linker -> Use Memory Layout from Target Dialog”.
5. Sprawdź, czy projekt kompiluje się poprawnie (zignoruj ostrzeżenie: „L6314W: No section matches pattern *(InRoot\$\$Sections)”).

Ćwiczenie 1

Napisać pętlę, która wykona się 1000 razy.

W tym celu zapoznać się z instrukcjami:

- Odejmowanie – SUB (zwróć uwagę na modyfikację rejestru statusu, patrz punkt 4 instrukcji)
- Skoku – B (zwróć uwagę na pole {cond}, patrz punkt 3 instrukcji)
- Ładowania stałej – LDR (w wersji z pseudoinstrukcją LDR Rd, =value)

Nie zagnieżdżać pętli (rejestry 32 bitowe).

KEIL -> Help:

- Assembler User Guide -> Overview of the ARM Architecture -> Instruction set overview -> ARM and Thumb Instructions
- ARM Instruction Set User's Guide
- Instruction set overview -> ARM and Thumb Instructions -> Memory access instructions -> LDR pseudo-instruction

Ćwiczenie 2

Zmodyfikować kod z poprzedniego ćwiczenia tak aby czas jego wykonania wynosił $R0 \times 1\text{ms}$ (z dokładnością do kilku cykli procesora). Użyć dodatkowego rejestru oraz instrukcji mnożenia. Liczbę cykli potrzebnych na wykonanie poszczególnych instrukcji w pętli wyznaczyć przy pomocy symulatora (okno „registers” -> Internal -> States)

Ćwiczenie 3

Zamknąć kod z poprzedniego ćwiczenia do podprogramu „delay_in_ms” zgodnie z KEIL -> Help -> Assembler User Guide -> Writing ARM Assembly Language -> Register usage in subroutine calls

Poprawne działanie podprogramu sprawdzić wywołując go w pętli głównej (wcześniej dodać pętlę główną).

Ćwiczenie 4

Wstawić przed pętlą główną ustawienie kierunku pinów 4-7 portu 0 na wyjściowy.

Skorzystać z adresów rejestrów portu zdefiniowanych w pliku „LPC213x.s” (dodać plik do pliku main dyrektywą “GET LPC213x.s”).

Skorzystać z przykładu zapisu do rejestru VPBDIV znajdującego się w kodzie startowym dołączonym do projektu (Startup.s)

Ćwiczenie 5

Wyświetlić wybraną cyfrę na wybranym wyświetlaczu.

Przyjąć, że segmenty A-G zostały podłączone do pinów 16-23 portu 1 oraz że wyświetlacze 0-3 zostały podłączone do pinów 16-19 portu 0 za pośrednictwem tranzystorów tak samo jak w przypadku AVR.

Użyć rejestrów od R4 wzwyż (Rejestry R0-3 powinny być używane do przekazywania argumentów do podprogramów).

Ćwiczenie 6

Zmodyfikować kod tak aby wybrana (ta sama) cyfra wyświetlała się po kolei cyklicznie na wszystkich wyświetlaczach.

Jako licznika cyfr użyć rejestru R12 (zdefiniować etykietę CURRENT_DIGIT za pomocą dyrektywy RN).

Do przesuwania stałej użyć instrukcji mov z odpowiednim operandem KEIL -> Help-> ARM Instruction Set User's Guide -> ARM Instruction Set

Do inkrementacji z modulo użyć instrukcji: add, cmp, eor z warunkowym wykonaniem.

Na początku każdej grupy instrukcji dać komentarz z odpowiadającej jej pseudokodem i/lub opisem

```
; IO0CLR=0xF0000 wygaszenie cyfr
ldr          r5, =IO0CLR
ldr          r4, =0xF0000
str          r4, [r5]
```

Pseudo kod pętli głównej:

mainn_loop:

```
IO0CLR = 0xf0000 // wygaszenie wszystkich wyświetlaczy

IO0SET = 0x80000 >> CURRENT_DIGIT

CURRENT_DIGIT = (CURRENT_DIGIT+1)%4 // inkrementacja licznika cyfr,

R0=500; // opóźnienie
Delay(R1)
```

jmp main_loop

Ćwiczenie 7

Dodać do pętli głównej (bezpośrednio przed inkrementacją licznika cyfr) fragment kodu, który będzie zamieniał licznik cyfr na jej kod siedmiosegmentowy.

Deklaracja tablicy stałych będących kodami siedmiosegmentowymi poszczególnych cyfr powinna znaleźć się na końcu programu (użyć dyrektywy DCB, użyć wartości: 0x3f,0x06,0x5B,0x4F,0x66,0x6d,0x7D,0x07,0x7f,0x6f)

Kod siedmiosegmentowy powinien znaleźć się w rejestrze R6.

Użyć instrukcji: add, ldrb, adr

Ćwiczenie 8

Dodać do pętli głównej (bezpośrednio przed inkrementacją licznika cyfr) fragment kodu, który będzie wstawiał zawartość R6, czyli kod siedmiosegmentowy z poprzedniego ćwiczenia (8 najmłodszych bitów) na piny sterujące segmentami wyświetlaczy. W efekcie na wyświetlaczu powinna być widoczna liczba „3210”

Kod programu powinien wyglądać następująco:

```
; ustawienie pinów sterujących wyświetlaczem na wyjściowe
; wyzerowanie licznika cyfr
main_loop
; włączenie cyfry o numerze podanym w CURR_DIG,
; zamiana numeru cyfry (CURR_DIG) na kod siedmiosegmentowy (R6)
; wpisanie kodu siedmiosegmentowego (R6) do segmentów
; inkrementacja licznika cyfr (CURR_DIG) modulo 4
; opóźnienie
B main_loop

; podprogram delay_in_ms
; tablica kodów siedmiosegmentowych
```

Ćwiczenie 9

Zdefiniować etykiety DIGIT_0 .. DIGIT_3 -> R8 .. R11. Zmodyfikować program tak aby zawartość wyświetlacza odpowiadała zawartości DIGIT_0..3. W tym celu:

- zmodyfikować zamianę liczby na kod siedmiosegmentowy tak aby zamianie ulegała zawartość rejestru R6 a nie licznika cyfr

- wstawić przed zamianą liczby na kod siedmiosegmentowy kod, który wykona następującą operację:
R6 <= DIGIT_X, gdzie X=CURR_DIG (instrukcje cmp i mov)

Ćwiczenie 10

Wstawić bezpośrednio przed inkrementację licznika cyfr inkrementację DIGIT_0..3 (licznik dekadowy)

Kod programu powinien wyglądać następująco:

```
        ; ustawienie pinów sterujących wyświetlaczem na wyjściowe
        ; inicjalizacja licznika dekadowego
        ; wyzerowanie licznika cyfr
main_loop

        ; włączenie cyfry o numerze podanym w CURR_DIG
        ; R6 <= DIGIT_X, gdzie X=CURR_DIG
        ; zamiana R6 na kod siedmiosegmentowy (R6)
        ; wpisanie kodu siedmiosegmentowego (R6) do segmentów
        ; inkrementacja licznika dekadowego (DIGIT_0 .. DIGIT_3)
        ; inkrementacja licznika cyfr (CURR_DIG) modulo 4
main_loop

; delay_in_ms
; tablica kodów siedmiosegmentowych
```