Wykład 5

Instrukcje asemblera

Nazwy rejestrów

Zdefiniowane nazwy rejestrów:

- r0-r15 and R0-R15,
- a1-a4 (argument, result, or scratch registers, synonyms for r0 to r3),
- v1-v8 (variable registers, r4 to r11),
- sb and SB (static base, r9),
- ip and IP (intra-procedure-call scratch register, r12),
- sp and SP (stack pointer, r13),
- lr and LR (link register, r14),
- pc and PC (program counter, r15),
- cpsr and CPSR (current program status register),
- spsr and SPSR (saved program status register).

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 definującym stałą numeryczną.

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

Rozmiar instrukcji

W zestawie instrukcji Thumb-2 niektóre operacje mogą być wykonywane zarówno jako 16- jak i 32-bitowe. Przykładem może być dodawanie wartości liczbowej do rejestru:

```
ADDS R0, #5 ;Zostanie użyta 16 – bitowa instrukcja
Thumb (domyślna, dla mniejszego rozmiaru kodu)
ADDS.N R0, #5 ;Zostanie użyta również 16 – bitowa
instrukcja (N = Narrow)
ADDS.W R0, #5 ;Tutaj będzie zastosowana 32 –
bitowa instrukcja Thumb – 2 (W = Wide)
```

Bez użycia sufiksu W (wide) lub N (narrow) kompilator wybierze domyślny rodzaj instrukcji, z reguły będzie to 16-bitowa instrukcja z zestawu Thumb-2.

Instrukcje przesunięcia

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.

Stałe

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-bittowej 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.

Dyrektywy EQU i DCD

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

Podział instrukcji

Instrukcje procesorów ARM można podzielić na kilka grup:

- dostępu do pamięci (LDR, STR, ADR, PLD, PLI ,LDM, STM, PUSH, POP, RFE, SRS),
- przesyłania danych (MOV, MVN, MOVT),
- rozgałęzień (B, BL, BX, BLX, BXJ, CBZ, CBNZ),
- warunkowe (IT),
- generacji wyjątków (SWI),

- arytmetyczne (ADD, SUB, RSB, ADC, SBC, RSC),
- mnożące (MUL, MLA, MLS, UMULL, UMLAL),
- dzielenia (SDIV, UDIV),
- logiczne (AND, ORR, EOR, BIC, ORN),
- bitowe (TST, TEQ),
- przesuwania (ASR, LSL, LSR, ROR, RRX)

Warunkowe wykonywanie instrukcji

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

Warunkowe wykonywanie instrukcji

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

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.

Instrukcja IT

- W procesorach pracujących w trybie ARM możliwe jest warunkowe wykonanie większości instrukcji asemblera.
- W trybie Thumb-2 nie jest to możliwe.
- W trybie Thumb-2 w celu określenia warunków wykonania instrukcji można wykorzystać polecenie IT.
- Instrukcja IT realizuje wyrażenie If-Then dla czterech następujących po niej instrukcji, jest to tzw. blok IT.
- Warunki mogą być wszystkie takie same lub niektóre mogą być inwersją logiczną innych warunków.
- Instrukcja IT jest 16-bitowa i realizowana jest tylko w trybie Thumb-2.

Instrukcja IT

- Składnia instrukcji jest następująca

- `IT{x{y{z}}} {cond}`

- gdzie:
  - x jest warunkiem dla drugiej instrukcji w bloku IT .
  - y jest warunkiem dla trzeciej instrukcji w bloku IT.
  - z jest warunkiem dla czwartej instrukcji w bloku IT.
  - cond jest warunkiem dla pierwszej instrukcji w bloku IT.

Instrukcja IT

- Warunkiem dla drugiej, trzeciej i czwartej instrukcji w bloku IT może być także:
  - T Then. Warunek zastosowany do każdej z instrukcji jest cond.
  - E Else. Warunek zastosowany do instrukcji jest inwersją cond.
- 16-bitowe w bloku IT inne niż CMP, CMN i TST nie zmieniają znaczników warunków.
- Dla pozostałych instrukcji można zastosować IT z warunkiem AL.

Instrukcja IT

- Następujące instrukcje nie są dozwolone w bloku IT:
  - IT,
  - rozgałęzień warunkowych,
  - CBZ i CBNZ,
  - TBB i TBH,
  - CPS, CPSID i CPSIE,
  - SETEND.
- Dodatkowo rozgałęzienie bezwarunkowe jest dozwolone w bloku IT, jeżeli jest ostatnią instrukcją w bloku.

Instrukcja IT

- Przykłady
  - `ITTE NE`
  - `ANDNE r0,r0,r1`
  - `ADDSNE r2,r2,#1`
  - `MOVEQ r2,r3`
  - `ITT AL`
  - `ADDAL r0,r0,r1`
  - `SUBAL r2,r2,#1`
  - `ADD r0,r0,r1`
  - `IT NE`
  - `ADD r0,r0,r1` ;błąd składni:brak warunku przy instrukcji w bloku IT

Przykłady wykorzystania instrukcji IT

```
   ITTET NE ; assemblers can allow IT to be
omitted
   ANDNE r0,r0,r1 ; 16-bit AND, not ANDS
   ADDSNE r2,r2,#1 ; 32-bit ADDS (16-bit ADDS
does not set flags in IT block)
   MOVEQ r2,r3 ; 16-bit MOV(CPY)
   MOVNE r2,r3 ; 16-bit MOV(CPY)
```

Przykłady wykorzystania instrukcji IT

```
    28:           SUBS r4,r2,r2
0x00004118 1A94       SUBS     r4,r2,r2
    29:           MULEQ r4,r0,r0
    30:
0x0000411A BF06       ITTE     EQ
0x0000411C FB00F400   MULEQ    r4,r0,r0
    31:           ADDEQ r1, r2, r3
0x00004120 18D1       ADDEQ    r1,r2,r3
```

Instrukcje dostępu do pamięci

- Architektura systemów z rdzeniami ARM nie umożliwia bezpośredniego wykonywania instrukcji arytmetycznych na danych znajdujących się w pamięci.
- W związku z tym do przesyłania danych z pamięci do rejestrów procesora konieczne jest wykorzystanie instrukcji przesyłania.

Instrukcje LDR i STR

- Instrukcje LDR i STR umożliwiają odczyt i zapis danych z rejestrów do pamięci. Składnia ich jest następująca

- op{type}{T}{cond} Rd, {Rd2,} [Rn {, #offset}]
- op{type}{T}{cond} Rd, {Rd2,} [Rn , #offset]!
- op{type}{T}{cond} Rd, {Rd2,} [Rn], #offset
- op{type}{T}{cond} Rd, {Rd2,} [Rn], #offset
- op{type}{cond} {Rd, {Rd2,}} [Rn, +/-Rm {, shift}]{!}
- op{type}{T}{cond} Rd, {Rd2,} [Rn], +/-Rm {, shift}
- LDR{type}{cond}{.W} Rd, {Rd2,} label

Instrukcje LDR i STR

gdzie:

- op jest typem instrukcji (LDR odczyt z pamięci lub STR zapis do pamięci).
- type określa typ danej:
  - B unsigned Byte
  - SB Signed Byte (tylko LDR)
  - H unsigned Halfword
  - SH Signed Halfword (tylko LDR)
  - - pomijane dla Word
  - D Doubleword,

Instrukcje LDR i STR

T jest opcjonalne i umożliwia określenie trybu dostępu do pamięci,

- cond jest warunkiem wykonania instrukcji,
- Rd jest zapisywanym lub odczytywanym rejestrem,
- Rd2 jest drugim rejestrem do zapisu i odczytu dla podwójnych słów (tylko dla type = D),
- Rn jest rejestrem zawierającym adres w pamięci,
- offset jest przesunięciem. Jeżeli jest pominięty to instrukcja ma zerowe przesunięcie,
- ! jest opcjonalny. Jeżeli ! jest użyty adres jest przed wykonaniem pobrania zwiększany o wartość przesunięcia, Rn nie może być w tym przypadku tym samym rejestrem co Rd lub Rd2.

Instrukcje LDR i STR

Dozwolony zakres offsetu:

- –4095 to +4095 for ARM Word or Byte instructions,
- –255 to +255 for ARM Signed Byte, Halfword, Signed Halfword, and
- Doubleword instructions,
- –255 to +4095 for all Thumb-2 instructions without the T suffix, except
- Doubleword instructions,
- –1020 to +1020 for Thumb-2 Doubleword instructions. Must be a multiple of 4.
- 0 to +255 for Thumb-2 instructions with the T suffix.
- Przyrostek T suffix jest niedostępny dla instrukcji ARM lub Thumb-2 Doubleword. Jest dostępny dla pozostałych instrukcji Thumb-2.

Tryby adresowania

- Offset addressing
- Wartość przesunięcia jest dodawana lub odejmowana od adresu znajdującego się w rejestrze bazowym. Wynik jest wykorzystywany jako adres dostępu do pamięci. Rejestr bazowy pozostaje niezmodyfikowany. Składnia asemblera dla tej instrukcji jest następująca:
- op{type}{T}{cond} Rd, {Rd2,} [<Rn>,<offset>]

W każdym przypadku:

- Rn jest rejestrem bazowym,
- <offset> jest stałą lub rejestrem.

Tryby adresowania

- Pre-indexed addressing
- Wartość przesunięcia jest dodawana lub odejmowana od adresu znajdującego się w rejestrze bazowym. Wynik jest wykorzystywany jako adres dostępu do pamięci i zapisywany następnie do rejestru bazowego. Składnia asemblera dla tej instrukcji jest następująca:
- op{type}{T}{cond} Rd, {Rd2,} [<Rn>,<offset>]!

Tryby adresowania

- Post-indexed addressing
- Jako adres dostępu do pamięci wykorzystywana jest wartość znajdująca się w rejestrze bazowym. Po wykonaniu operacji do adresu jest dodawana lub odejmowana wartość przesunięcia i zapisywana w rejestrze bazowym. Składnia asemblera dla tej instrukcji jest następująca:
- op{type}{T}{cond} Rd, {Rd2,} [<Rn>], <offset>

Instrukcje LDR i STR

```
LDR r8,[r10] ; loads r8 from the address in r10.
```

- `LDRNE r2,[r5,#960]! ; (conditionally) loads r2 from a word`
- `; 960 bytes above the address in r5, and`
- `; increments r5 by 960.`
- `STR r2,[r9,#consta-struc] ; consta-struc is an expression evaluating`
- `; to a constant in the range 0-4095.`
- `LDR r0,localdata ; loads a word located at label localdata`
- `STR r5,[r7],#-8 ; stores a word from r5 to the address`
- `; in r7, and then decrements r7 by 8.`

Instrukcja ADR

- Do ładowania adresu do rejestru można wykorzystać instrukcję ADR o następującej składni:

```
ADR{cond}{.W} register,label
```

- gdzie:
  - cond jest kodem warunku,
  - .W określa rodzaj instrukcji w trybie Thumb-2,
  - register jest ładowanym rejestrem,
  - label jest etykietą, której adres jest pobierany.

Instrukcja ADR

- Dostępne zakresy adresowania są następujące:

- ARM ±255 bajtów od adresu wyrównanego do bajtu lub pół-słowa, ±1020 dla adresu wyrównanego do słowa.
- 32-bit Thumb-2 ±4095 bajtów dla każdego adresu.
- 16-bit Thumb 0 to 1020 bajtów. Etykieta musi być wyrównana do słowa. Można do tego użyć dyrektywy ALIGN.

Instrukcje PUSH i POP

- Instrukcje PUSH i POP umożliwiają odłożenie na stosie i zdjęcie zawartości rejestrów.

```
PUSH{cond} reglist
POP{cond} reglist
```

- gdzie:
  - cond jest warunkiem wykonania,
  - reglist jest listą rejestrów lub zakresem umieszczonymi w nawiasie i oddzielonymi przecinkami.

Instrukcje PUSH i POP

- Przykłady

- 16-bit
  - `PUSH {r0,r3,r5}`
  - `PUSH {r1,r4-r7} ; pushes r1, r4, r5, r6, and r7`
  - `PUSH {r0,LR}`
  - `POP {r2,r5}`
  - `POP {r0-r7,pc} ; pop and return from subroutine`

Instrukcje PUSH i POP

- Przykłady

- ARM and 32-bit Thumb-2
  - `PUSH {r0,r3,r5}`
  - `PUSH {r1,r4-r11}`
  - `PUSH {r0,LR}`
  - `POP {r8,r12}`
  - `POP {r0-r10,pc}`

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.

MOV and MVN

Move and Move Not.

Syntax
MOV{S}{cond} Rd, Operand2
MVN{S}{cond} Rd, Operand2

where:
S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation.
cond is an optional condition code.
Rd is the destination register.
Operand2 is a flexible second operand. See also Wide constants.

Usage
The MOV instruction copies the value of Operand2 into Rd.
The MVN instruction takes the value of Operand2, performs a bitwise logical NOT operation on the value, and places the result into Rd.
In certain circumstances, the assembler can substitute MVN for MOV, or MOV for MVN. Be aware of this when reading disassembly listings.

MOV and MVN wide constant

Wide constants

In MOV instructions, Operand2 can take any value in the range 0-65535, in addition to the normal range of Operand2 values.

These wide constants cannot be used with MVN instructions.

You cannot use the S suffix with a wide constant.

You cannot use r15 as Rd with a wide constant.

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 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),

Instrukcje B, BL

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.

Instrukcje CBZ i CBNZ

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

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 instrukcje BXJ.

Instrukcja BL:
odkłada adres powrotu w rejestrze
ustawia PC na adres wywoływanej funkcji

Wywoływanie 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 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
```

Instrukcje TBB i TBH

- Instrukcje TBB i TBH umożliwiają wykonywanie rozgałęzień w ramach tablicy bajtów lub pół-słów.
- Mają one następującą składnię:

```
TBB [Rn, Rm]
TBH [Rn, Rm, LSL #1]
```

- gdzie:
  - Rn to rejestr bazowy zawierający adres tablicy,
  - Rm jest rejestrem indeksowym zawierającym liczbę całkowitą wskazującą na bajt docelowy.

Instrukcje TBB i TBH

- Przesunięcie w ramach tablicy zależy od rodzaju instrukcji:
  - TBB jest równy indeksowi,
  - TBH jest podwójna wartością indeksu.
- Instrukcje te są dostępne tylko w trybie Thumb-2.
- Instrukcje te umożliwiają wykonanie rozgałęzienia do przodu względem PC z wykorzystaniem przesunięcia odczytanego z tablicy bajtów lub pół-słów.
- Odległość rozgałęzienia jest podwójną wartością odczytaną z tablicy.

Przykład TBB

```
ADR.W  R0, BranchTable_Byte

    TBB   [R0, R1]          ; R1 is the index, R0 is the base address of
the

                            ; branch table

Case1

; an instruction sequence follows

Case2

; an instruction sequence follows

Case3

; an instruction sequence follows

BranchTable_Byte

    DCB    0                    ; Case1 offset calculation

    DCB    ((Case2-Case1)/2)    ; Case2 offset calculation

    DCB    ((Case3-Case1)/2)    ; Case3 offset calculation
```

Przykład TBH

```
TBH    [PC, R1, LSL #1]   ; R1 is the index, PC is used as base of the
                                ; branch table
BranchTable_H
    DCI    ((CaseA - BranchTable_H)/2)  ; CaseA offset calculation
    DCI    ((CaseB - BranchTable_H)/2)  ; CaseB offset calculation
    DCI    ((CaseC - BranchTable_H)/2)  ; CaseC offset calculation

CaseA
; an instruction sequence follows
CaseB
; an instruction sequence follows
CaseC
; an instruction sequence follows
```