Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hardwarenahe Softwareentwicklung

Instruktionssatz ARM V7M, Teil 2

V5.1, ©2023 roger.weber@bfh.ch

Lernziele

Sie sind in der Lage:

- Die Instruktionen eines ARM V7M Prozessors mit Hilfe von Unterlagen zu erklären und anzuwenden.
- Einfache Assemblerprogramme zu entwickeln.

Inhaltsverzeichnis

Arithmetische Instruktionen

Integer-Arithmetik

- Grundrechenoperationen sowie arithmetische Vergleichsoperationen.
- Operand N: Direktwert, ein Register oder ein geshiftetes Register.
- Einige Instruktionen:

Instruktion	**Operanden**	**Beschreibung**	**Operation**
ADD	Rd, Rn, N	32-Bit Addition	$Rd = Rn + N$
ADC	Rd, Rn, N	32-Bit Addition mit Übertrag	$Rd = Rn + N + Carryflag$
MUL	Rd, Rn, Rs	32-Bit Multiplikation	$Rd = Rn * Rs$
MLA	Rd, Rn, Rs, Rm	32-Bit Multiplikation / Addition	$Rd = Rn * Rs + Rm$
SUB	Rd, Rn, N	32-Bit Subtraktion	$Rd = Rn - N$

- Siehe auch: ARM Architecture Reference Manual, Thumb-2 Supplement

Beispiele Integer-Arithmetik

- Subtraktion r0 = r1 - r2

Vorher:

```
r0 = 0x00000000
r1 = 0x00000005
r2 = 0x00000002
```

```
SUB r0, r1, r2
```

Nachher:

```
r0 = 0x00000003
r1 = 0x00000005
r2 = 0x00000002
```

Beispiele Integer-Arithmetik

- Dekrementieren von Schleifenzählern

Vorher:

```
    Statusbits = nzcvq
    r1 = 0x00000005
```

```
loop:
    ...
    SUBS    r1,r1,#1      @ decrement r1
    BNE     loop          @ until r1 = 0, ---> Z-Bit set, branch if Z cleared
```

Nachher:

```
    Statusbits = nZCvq
    r1 = 0x00000000
```

Logische Instruktionen

Logische Instruktionen

- Logische Verknüpfungsoperationen UND, ODER, EXOR, Bit clear.
- Operationen auf Bitmuster: löschen, setzen oder komplementieren von Bits.
- Syntax der logischen Instruktionen:
  <Instruktion> <cond> S Rd, Rn, N

Instruktion	Operanden	Beschreibung	Operation
AND	Rd, Rn, N	Logische UND-Verknüpfung zweier 32-Bit Werte	$Rd = Rn \& N$
ORR	Rd, Rn, N	Logische ODER-Verknüpfung zweier 32-Bit Werte	$Rd = Rn \mid N$
EOR	Rd, Rn, N	Logische Exklusiv-ODER-Verknüpfung zweier 32-Bit Werte	$Rd = Rn \hat{} N$
BIC	Rd, Rn, N	Logisches Bit Nullsetzen (UND NICHT)	$Rd = Rn \& \sim N$

Beispiel Logische Instruktionen

- ORR-Instruktion / ODER Verknüpfung

Vorher:

```
r0 =0x00000000
r1 =0x10305070
r2 =0x02040608
```

```
ORR r0,r1,r2
```

Nachher:

```
r0 =0x12345678
r1 =0x10305070
r2 =0x02040608
```

Shift und Rotate

- Instruktionen für logische und arithmetische Schiebeoperationen.
- 2 Varianten:
  - Verwendung der MOV-Instruktion mit Barrel-Shifter
  - Verwendung der Instruktionen ASR, LSL, LSR und ROR
- Beispiel:

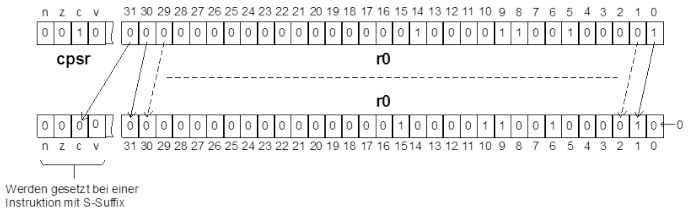
Vorher:

```
r0 =0x00004321
```

Nachher:

```
r0 =0x00008642
```

```
MOV  r0,r0,LSL #1 @ r0 = r0 << 1, using Barrel Shifter
LSL  r0,r0,#1     @ r0 = r0 << 1, using LSL Instruction
```



Programmverzweigungen

Programmverzweigungen

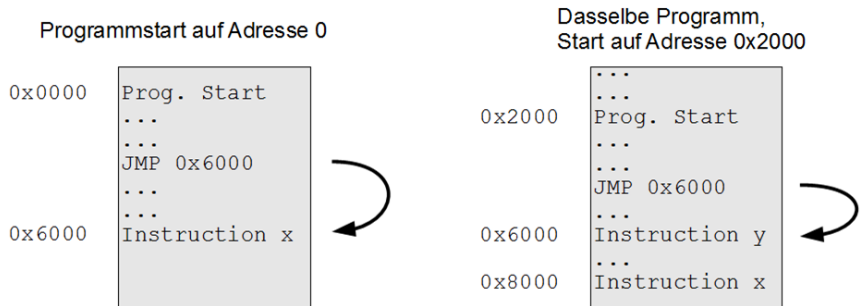
Kriterien für Verzweigungsbefehle:

- Absolute oder relative Verzweigung.
  - Absolute Verzweigung: Programm springt auf fixe, vorgegebene Adresse.
  - Relative Verzweigung: Programm springt um Offset von der aktuellen Adresse.

- Unbedingte oder bedingte Verzweigung.
  - Unbedingte Verzweigung wird immer ausgeführt.
  - Bedingte Verzweigung wird abhängig von den Statusbits ausgeführt.

Absolute Verzweigung

- Z.B. Intel x86, Motorola 68x, jedoch nicht ARM!

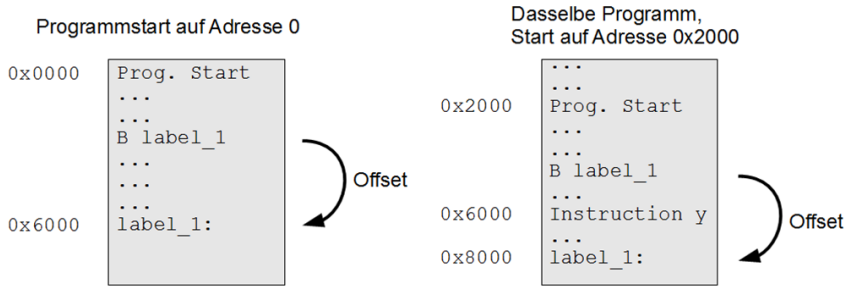


Merke

→ **Programme mit absoluten Verzweigungen können nicht beliebig im Speicher verschoben werden.**

Relative Verzweigung

- Z.B. ARM, Intel x86, Motorola 68x



Merke

→ Programme mit relativen Verzweigungen können beliebig im Speicher verschoben werden.

Unbedingte Programmverzweigungen

- Syntax für unbedingte Sprungbefehle:
  <B | BL> <label>
- Label: Symbolische Konstante für die Sprungadresse

Instruktion	Sprungdistanz	Beschreibung	Wirkung
B	± 16 MB	Generelle Programmverzweigung	pc = pc + Offset(label)
BL	± 16 MB	Unterprogrammaufruf (Subroutine). Programmverzweigung mit Speichern der Rücksprungadresse im Link-Register	pc = pc + Offset(label); lr = Adresse der nächsten Instruktion nach BL

Beispiel unbedingte Programmverzweigungen

- Sprünge vor- und rückwärts

```
    B       label1
    ADD   r1 , r2 ,#4
    ADD   r0 , r6 ,#2
    ADD   r3 , r7 ,#4
label1 :
    SUB   r1 , r2 ,#4
label2 :
    ADD   r1 , r2 ,#4
    SUB   r1 , r2 ,#4
    ADD   r4 , r6 , r7
    B     label2
```

- Welches ist die Reihenfolge der ausgeführten Befehle?
- Welche Befehle werden nie ausgeführt?

Beispiel unbedingte Programmverzweigungen

- Aufruf eines Unterprogramms (Subroutine)

```
    ...
    BL    subroutine
    ADD   r0,r1,r2          @ Ruecksprungadresse nach
                            @ Subroutinen-Aufruf
    ...
subroutine:
    ...
    MOV   pc,lr
```

- Welches ist die Reihenfolge der ausgeführten Befehle?

Bedingte Programmverzweigungen

- Syntax für bedingte Sprungbefehle:
  <B | BL><cond> label
- <cond> ist eine Bedingung als Suffix.

Bedingte Programmverzweigungen

Bedingung	Beschreibung	Flags
EQ	Equal	Z
NE	Not Equal	!Z
CS, HS	Unsigned higher or same	C
CC, LO	Unsigned lower	!C
MI	Minus, Negative	N
PL	Positive or zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	C && !Z
LS	Unsigned lower or same	!C && Z
GE	Signed greater or equal	(N == V)
LT	Signed less than	(N != V)
GT	Signed greather than	!Z und (N == V)
LE	Less or equal	Z oder (N != V)

Beispiel bedingte Programmverzweigungen

```
main:
    MOV     r0,#5       @ Beispiel 5!
    BL      Fakultaet   @ Aufruf Subroutine, (r0!) berechnen
    B       main        @ Endlosschleife
# Subroutine zur Berechnung der Fakultaet durch fortgesetzte Multiplikation
# n! = n * (n-1) * (n-2) * ... * 3 * 2
# Input     r0: Argument n
# Output    r0: Resultat n!
# Benutzte Register: r1 als Zaehler
Fakultaet:
    MOVS    r1,r0       @ r1 ist Zaehlerregister, initialisiert mit Startwert
    BGE     l1          @ Startwert n >= 0? dann Sprung zu l1
    MOV     r0,#0       @ nein, n < 0, Resultat ist 0 (Konvention)
    B       l2          @
l1: MOV     r0,#1       @ Resultat von 0! oder Startwert fuer loop
l2: BLE     endFakultaet @ Abbruch falls n <= 0

loop:
    MUL     r0,r1,r0    @ Fortgesetzte Multiplikation n*(n-1)*(n-2)*...*3*2
    SUB     r1,r1,#1    @ Zaehlerregister dekrementieren
    CMP     r1,#1       @ Zaehlerregister > 1: naechster loop
    BGT     loop
endFakultaet:
    MOV     pc,lr       @ Ruecksprung
```