Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hardwarenahe Softwareentwicklung

Instruktionssatz ARM V7M, Teil 1

V5.1, ©2023 roger.weber@bfh.ch

Lernziele

Sie sind in der Lage:

- Die Instruktionen einer ARM V7M CPU mit Hilfe von Unterlagen zu erklären und anzuwenden.
- Einfache Assemblerprogramme zu entwickeln.
- Die Entwicklungsumgebung zu bedienen.

Inhaltsverzeichnis

Grundlagen

Instruktionen

- Befehlskürzel (Mnemonics) werden aus der Funktionalität des Befehls abgeleitet.
  Beispiel: **AD**d with **C**arry → ADC

- Thumb-2 Instruktionssatz:
  - 16-bit Befehle für hohe Codedichte
  - 32-bit Befehle für hohe Performance
  - Einige Befehle als 16-bit oder 32-bit

Syntax

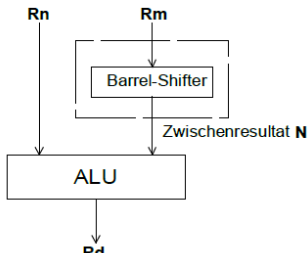
- Die Syntax einer Assemblerinstruktion ist wie folgt definiert:
  Opcode Rd, Rn ,N
  - Opcode: Assemblerinstruktion (ADD, MOV usw.)
  - Rd und Rn: Register r0 bis r15
  - N: Register, konstanter Wert oder geshiftetes Register
- Syntaxnotation:

Syntax	Kurzbeschreibung
[ ]	Indirekter, indizierter Zugriff
{ }	Fakultatives Element
< >	Element aus einer Menge, Aufzählung
|	Auswahl aus Aufzählung (oder)

- Beispiel: <MOV | MVN><cond>S Rd, Rn {, <shift>}

Operanden

- Instruktionen haben 1, 2 oder 3 Operanden
- 1. Operand ist das Zielregister
- 2. Operand und optional 3. Operand sind Quellen
- Operand N: Register, konstanter Wert oder geshiftetes Register
- Optionaler Shift:



- Beispiel:

```
ADD r6, r7, r5, LSL #2          @ r6 = r7 + (r5 * 4)
```

Barrel-Shifter Operationen

Befehl	Wirkung	Operation	Resultat	Schiebebereich
LSL	Logisches Schieben nach links	x LSL y	$x << y$	#0-31 oder Rs
LSR	Logisches Schieben nach rechts	x LSR y	$(unsigned)x >> y$	#1-31 oder Rs
ASR	Arithmetisches Schieben nach rechts	x ASR y	$(signed)x >> y$	#1-31 oder Rs
ROR	Rotieren nach rechts	x ROR y	$((unsigned)x >> y)\|(x << (32-y))$	#1-31 oder Rs
RRX	Erweitertes Rotieren nach rechts	x RRX	$(cflag << 31)\|((unsigned) >> 1)$	1 (fest gegeben)

Beispiel Instruktion MOV (immediate)

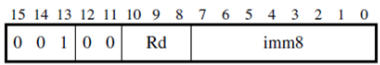
- Beispiel (aus ARM Architecture Reference Manual, Thumb-2 Suppl.)

MOV (immediate)

Move (immediate) writes an immediate value to the destination register. It can optionally update the condition flags based on the value.

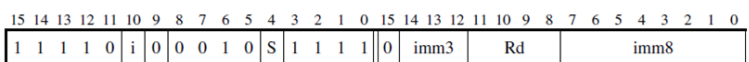
Encodings

T1 `MOVS <Rd>,#<imm8>` — Outside IT block.
`MOV<c> <Rd>,#<imm8>` — Inside IT block.




```
d = UInt(Rd);  setflags = !InITBlock();
(imm32, carry) = (ZeroExtend(imm8, 32), APSR.C);
```

T2 `MOV{S}<c>.W <Rd>,#<const>`




```
d = UInt(Rd);  setflags = (S == '1');
(imm32, carry) = ThumbExpandImmWithC(i:imm3:imm8, APSR.C);
if BadReg(d) then UNPREDICTABLE;
```

Befehlsübersicht

- Datentransfer
- Arithmetische und logische Instruktionen
  - Integer Arithmetik
  - Logische Befehle
  - Schiebe- und Rotationsbefehle
- Programmverzweigungen
  - Unbedingte Programmverzweigung
  - Bedingte Programmverzweigung
- Sonstige Instruktionen

Datentransfer

Übersicht Datentransfer

Folgende Transferbefehle werden unterstützt:

- Register → Register (MOV) Word
- Speicher → Register (LDR) Byte, Halfword, Word
- Register → Speicher (STR) Byte, Halfword, Word

Instruktion	**Operanden**	**Operation**	**Beschreibung**
MOV	Reg, Op2	Reg := Op2	Registerinhalt oder Konstante in Register kopieren
MVN	Reg, Op2	Reg := ~Op2	Registerkomplement oder Konstantenkomplement in Register kopieren
LDR	Reg, Address	Reg := [Address]	Speicherinhalt in Register kopieren
STR	Reg, Address	[Address] := Reg	Registerinhalt in Speicher kopieren
LDM	Rbase, {Rlist}	[Rbase] → {Rlist}	Mehrere Register vom Speicher laden
STM	Rbase, {Rlist}	{Rlist} → [Rbase]	Mehrere Register in Speicher kopieren
POP	{Rlist}	[SP] → {Rlist}	Mehrere Register vom Stack lesen
PUSH	{Rlist}	{Rlist} → [SP]	Mehrere Register auf den Stack kopieren

Registertransportbefehle MOV, MVN

- Datentransport von Registern untereinander.
- Laden von Registern mit konstanten Werten.
- Syntax: <**MOV|MVN**> <**cond**> **S Rd, N**
- MOV lädt einen Datenwert, MVN lädt den Komplementwert.
- Beispiele:

```
MOV     r0,r1       @ r0 = r1
MOVS    r0,#10      @ r0 = 10, S-Suffix, Statusbits werden angepasst
MVN     r1,r0       @ r1 = ~r0
MOV     pc,lr       @ Ruecksprung aus Subroutine
```

Lade- / Speicherbefehle LDR, STR

- Transfer von Datenwerten aus dem Speicher in ein Register und umgekehrt.
- Können für Words, Halfwords und Bytes verwendet werden (ausser LDRD und STRD).
- Die Syntax der LDR-STR Instruktionen lautet:

```
LDR{<cond>} {B|SB} Rd , <Adresse>
LDR{<cond>} {H|SH} Rd , <Adresse>
STR{<cond>} {B|H} Rd , <Adresse>
```

Lade- / Speicherbefehle LDR, STR

Instruktion	Beschreibung	Wirkung
LDR	Word in Register laden	Rd := mem32[Adresse]
STR	Word aus Register speichern	mem32[Adresse] := Rd
LDRB	Byte in Register laden	Rd := mem8[Adresse]
STRB	Byte aus Register speichern	mem8[Adresse] := Rd
LDRH	Halfword in Register laden	Rd := mem16[Adresse]
STRH	Halfword aus Register speichern	mem16[Adresse] := Rd
LDRSB	Vorzeichenbehaftetes Byte in Register laden	Rd := (signExtend) mem8[Adresse]
LDRSH	Vorzeichenbehaftetes Halfword in Register laden	Rd := (signExtend) mem16[Adresse]

Lade- / Speicherbefehle LDR, STR

Beispiele:

```
/* Register r5 mit Adresse laden */
LDR r5, =0x20000000

/*
Register r0 mit dem Wert laden, der im Speicher auf der Adresse in r5
    gespeichert ist. Indirekte Adressierung, in der Programmiersprache C:
    r0 = *r5
*/
LDR r0, [r5]

/*
Wert aus Register r0 in Speicherstelle schreiben, die durch r5 adressiert wird.
*/
STR r0, [r5]
```

Adressiermodi für Lade- und Speicherbefehle

- Vielfältige Adressiermöglichkeiten für den Speicherzugriff.
- Speicherplatzadresse setzt sich aus einer Basis und einem Offset zusammen.
- Übersicht der Adressiermodi (Indexing-Methoden):

Indexmethode	**Daten**	**Basisadressregister nach der Operation**	**Beispiel**
Preindex	mem[Basis+Offset]	Basis	LDR r0,[r1,#4]
Preindex mit Writeback	mem[Basis+Offset]	Basis+Offset	LDR r0,[r1,#4]!
Postindex	mem[Basis]	Basis+Offset	LDR r0,[r1],#4

Preindex

Beim Preindex wird die Speicherplatzadresse aus Basisregister und Offset berechnet. Nach dem Zugriff bleibt der Wert des Basisregisters unverändert.

Vorher:

```
r0 = 0x00000000
r1 = 0x20000000
mem32[0x20000000]=0x10101010
mem32[0x20000004]=0x20202020
```

```
LDR r0,[r1,#4]      @ address = 0x20000000 + 4
```

Nachher:

```
r0 = 0x20202020
r1 = 0x20000000
```

Preindex mit Writeback

Ähnlich Preindex, nach dem Zugriff wird die berechnete Speicherplatzadresse als neuer Wert in das Basisregister zurückgeschrieben.

Vorher:

```
r0 = 0x00000000
r1 = 0x20000000
mem32[0x20000000]=0x10101010
mem32[0x20000004]=0x20202020
```

```
LDR r0,[r1,#4]!      @ address = 0x20000000 + 4
```

Nachher:

```
r0 = 0x20202020
r1 = 0x20000004
```

Postindex

Bei Postindex wird ebenfalls die Adresse in das Basisregister übernommen, aber erst nachdem der Speicherzugriff erfolgt ist.

Vorher:

```
r0 = 0x00000000
r1 = 0x20000000
mem32[0x20000000]=0x10101010
mem32[0x20000004]=0x20202020
```

```
LDR r0,[r1],#4      @ address = 0x20000000
```

Nachher:

```
r0 = 0x10101010
r1 = 0x20000004
```

Varianten zur Offsetberechnung

- Direktwerte, Register und geschobene Registerwerte.
- Untenstehende Tabelle gilt für Word. Für Byte und Halfword gelten Einschränkungen.

Adressiermodus	**Syntax**
Preindex mit konstantem Offsetwert	[Rn,#±<offset12>]
Preindex mit Registeroffset	[Rn,±Rm]
Preindex mit skaliertem Registeroffset	[Rn,±Rm, shift #<shift_wert>]
Preindex Writeback mit konstantem Offsetwert	[Rn,#±<offset8>]!
Postindex mit konstantem Offsetwert	[Rn],#±<offset8>

Beispiele Adressiermodi

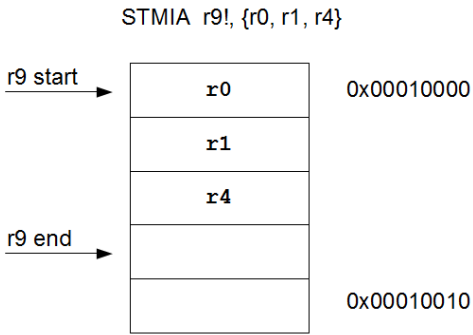
```
LDR r2,[r0,#1024]       @ Preindex mit konstantem Offset
                        @ r2 := [r0 + 1024]

LDR r2,[r0,r1]          @ Preindex mit Registeroffset
                        @ r2 := [r0 + r1]

LDR r2,[r0,r1,LSL#2]    @ Preindex mit skaliertem Regisgeroffset
                        @ r2 := [r0 + 4*r1]

LDR r2,[r0,#252]!       @ Preindex Writeback mit konstantem Offsetwert
                        @ r2 := [r0 + 252]; r0 := r0 + 252;

LDR r2,[r0],#252        @ Postindex mit konstantem Offsetwert
                        @ r2 := [r0]; r0 := r0 + 252

LDREQB r1,[r2,#5]       @ Bedingtes Laden r1 in D0 .. D7 aus [r2+5].
                        @ D8 .. D31 werden mit Null gefuellt .
```

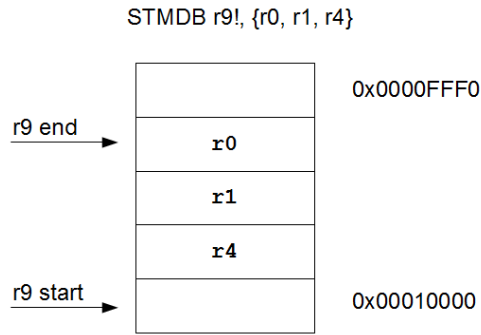
Mehrfach Lade- Speicherbefehle LDM, STM

- Transferieren von ganzen Register- oder Speicherblöcken mit einer Instruktion.
- **<LDM|STM><cond><Adressmodus> Rn!, <Registerblock>**

Adressmodus	**Beschreibung**	**Startadresse**	**Endadresse**	**Rn!**
IA	Increment after	Rn	$Rn + 4 * N - 4$	$Rn + 4 * N$
DB	Decrement before	$Rn - 4 * N$	$Rn - 4$	$Rn - 4 * N$

Mehrfach Lade- Speicherbefehle LDM, STM





Stackoperationen

- Stackzugriffe werden über PUSH und POP-Instruktionen ausgeführt.
- Als Variante sind auch LDM und STM möglich
- Gemäss AAPCS wird ein Full Descending Stacks verwendet
  - Stack wächst zu tieferen Adressen
  - Stackpointer zeigt am Schluss auf zuletzt geschriebenen Wert.
- Beispiel:

```
PUSH {r4-r6}        @ push r4 to r6 to stack
POP  {r4-r6}        @ pop r4 to r6 from stack
```