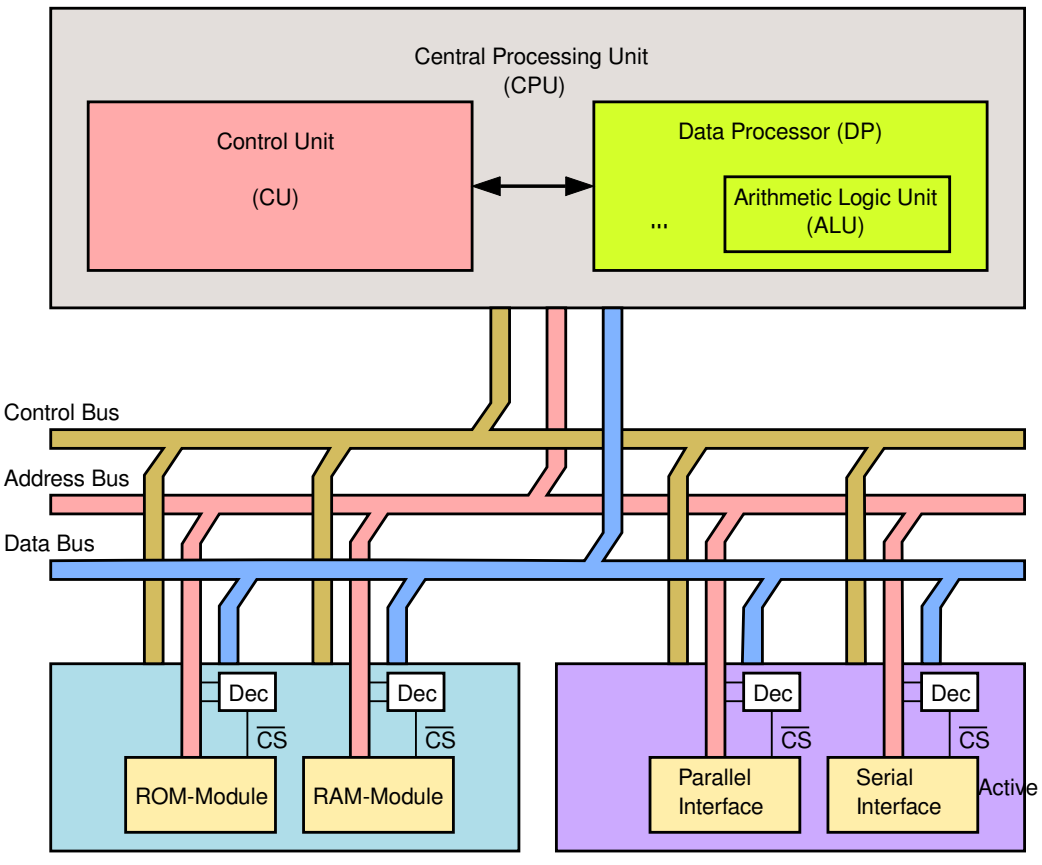
Maschinenbefehle
Machine Level Instructions

Vorlesung Mikroprozessortechnik

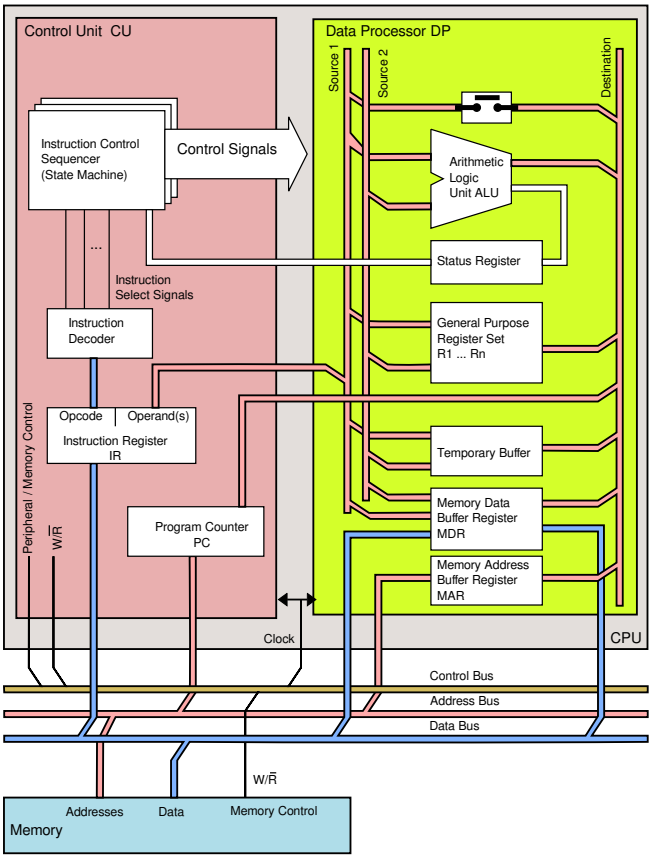
HAW Hamburg

31. Dezember 2017

von-Neumann-Architektur



CPU im Detail IV



Die CPU wird durch Maschinenbefehle gesteuert

Maschinenbefehle:

- ... sind elementare (unteilbare) Programmschritte
- ... sind binär codiert $\rightsquigarrow$ 10110000 01100001
- ... haben verschiedene Längen und bewirken komplexe Operationen von unterschiedlicher Dauer ($\rightsquigarrow$ CISC)
- oder haben einheitlich gleiche Längen und bewirken einfache Operationen von mit einheitlich kurzer Dauer ($\rightsquigarrow$ RISC)
- ... werden aus dem Speicher in die Control Unit geladen (Fetch)
- ... werden in der Control Unit decodiert (Decode)
- Jeder Maschinenbefehl löst eine besondere (kurze) Steuersequenz mit Steuersignalen aus, die eine Operation, Register und Verbindungen im Data Processor passend schalten

Maschinenbefehle und Assembler

- Ein Assembler ist ein Übersetzungsprogramm, dass ein in Assemblersprache geschriebenes Programm in die Maschinencode übersetzt.
- Häufig wird der Begriff Assembler sowohl für das Übersetzungsprogramm als auch für die Sprache verwendet.
- In der Assemblersprache sind die Maschinenbefehle als Mnemonik (leicht merkbare Abkürzung) geschrieben.
- In der Mehrzahl werden die Maschinenbefehle vom Mnemonik zu Maschinencode durch eine 1:1 Abbildung (programmierte Tabelle der Codes) übertragen.
- Assemblersprachen sind auf Prozessorfamilien abgestimmt. → nicht prozessorunabhängig
- Hochsprachen sind abstrakt bezüglich der Prozessoren. → möglichst prozessorunabhängig

Maschinenbefehle in Assembler-Syntax (ARM)

<op> {cond} {flags} Rd, Rn, Operand2

{cond} = An optional two-letter condition code, e.g. EQ or CS.

{flags} = An optional additional flags. e.g. S.

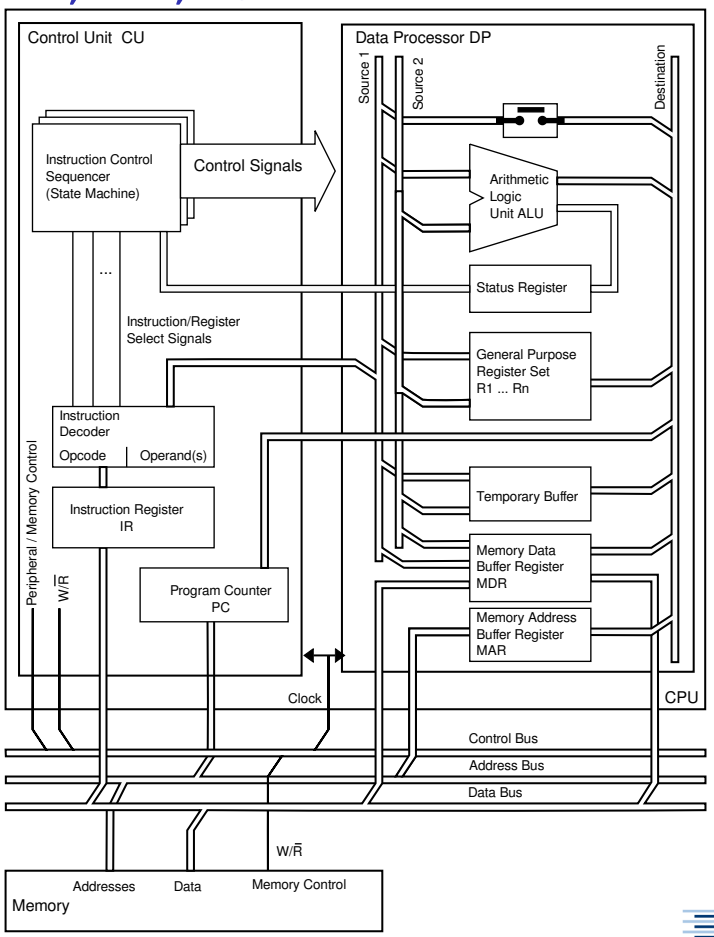
Rd = The destination register. (Leftmost register Rd is the destination.)

Rn = The first source register.

Operand2 = A flexible second operand.

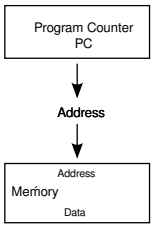
Maschinenbefehl ADD R4, R2,R3

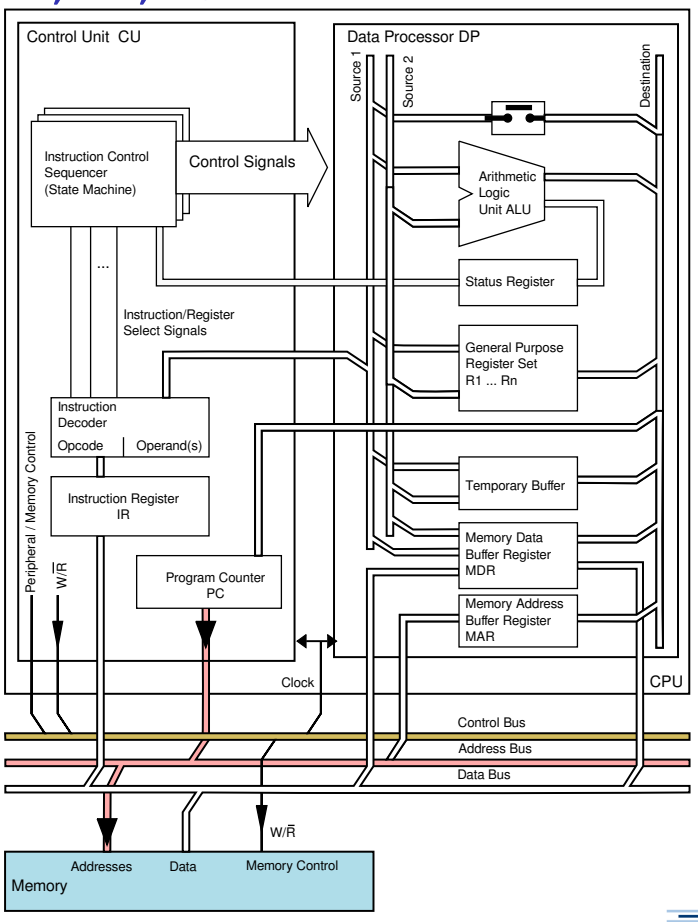
Fetch Decode Execute



Maschinenbefehl ADD R4,R2,R3

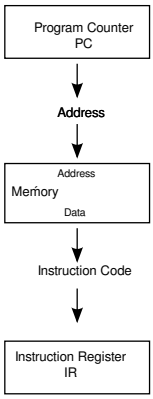
Fetch Decode Execute

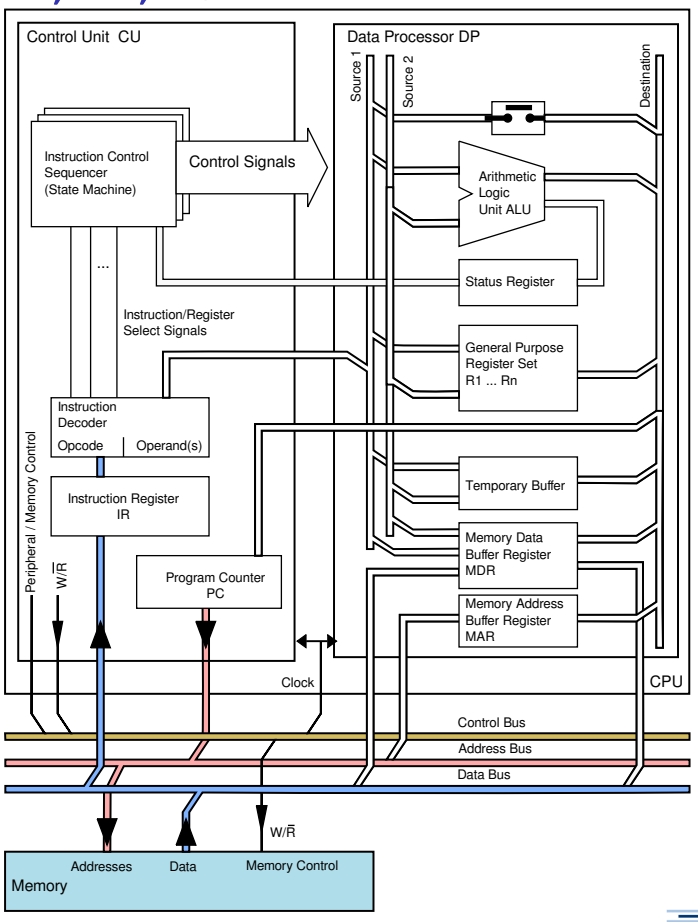




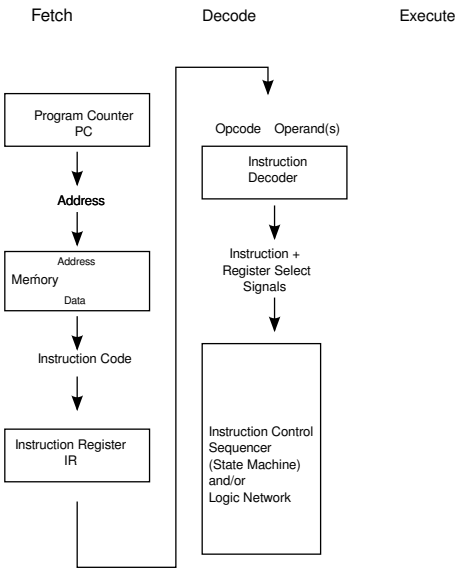
Maschinenbefehl ADD R4,R2,R3

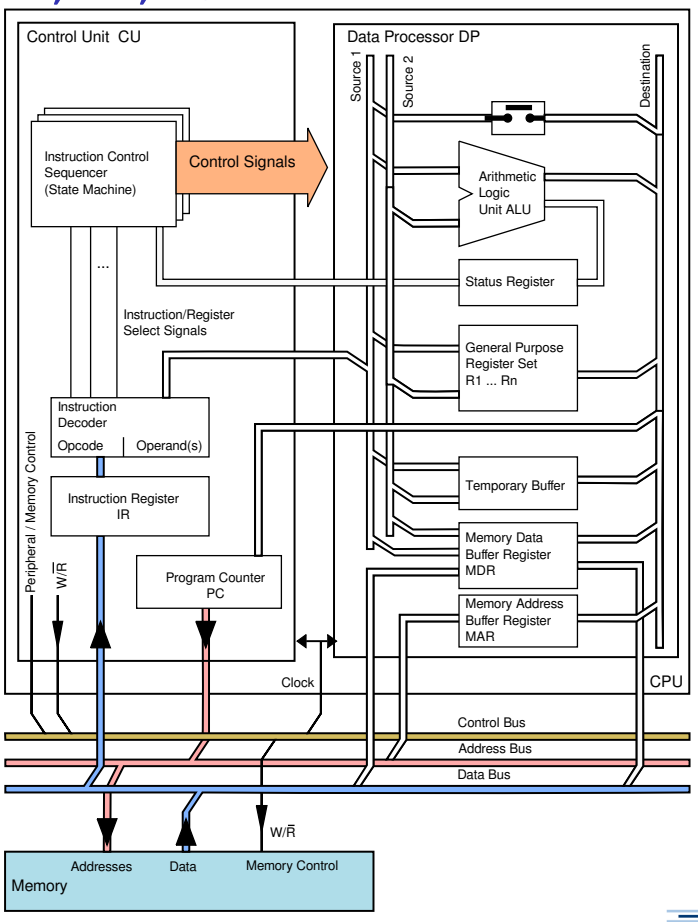
Fetch Decode Execute



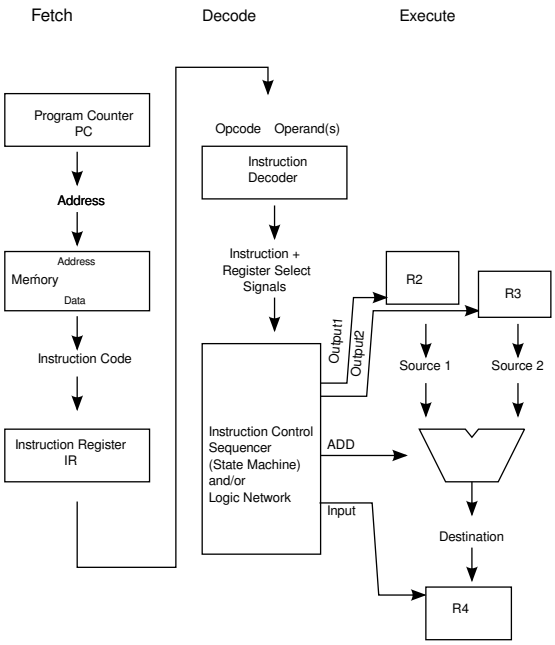


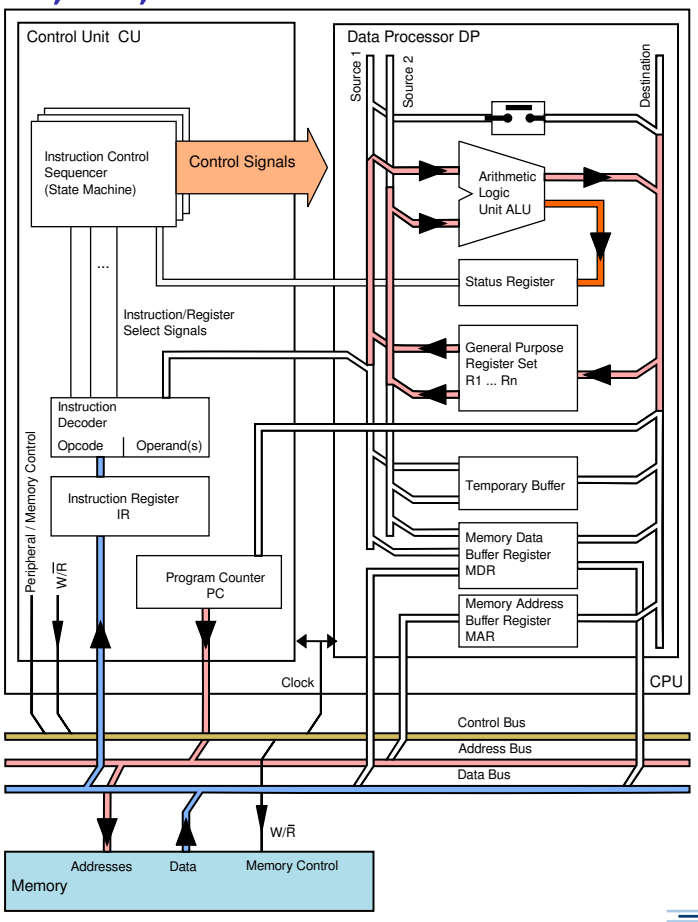
Maschinenbefehl ADD R4,R2,R3





Maschinenbefehl ADD R4,R2,R3





Fetch – Decode – Execute

FETCH-Phase (auch instruction fetch)

- Das binäre Befehlswort wird aus der durch den Programcounter (PC) adressierten Stelle im Speicher geholt.
- Das Befehlswort wird im Instruction Register (IR) abgelegt.
- Weil nur die Adresse, aber der Befehl noch nicht bekannt ist =¿ immer gleicher Ablauf

DECODE-Phase

- Art des Befehls wird durch Auswertung des OPCODES erkannt
- Weil noch nicht bekannt ist, welcher Befehl ansteht =¿ immer gleicher Ablauf

EXECUTE-Phase (hier nur typische Abläufe)

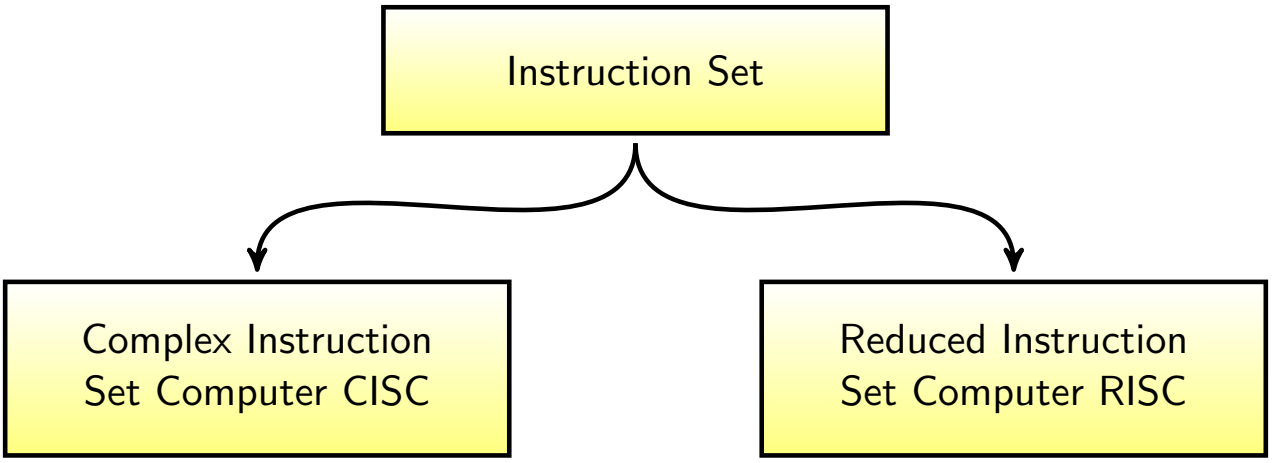
- Laden von Operanden aus einem General-Purpose-Register, Operation mit der ALU durchführen, Resultat in ein General-Purpose-Register leiten sowie das Statusregister aktualisieren
- oder Holen von Daten aus dem Speicher in ein General-Purpose-Register (LOAD = Data-fetch)
- oder Schreiben von Daten aus einem General-Purpose-Register in den Speicher (STORE = Writeback)
- Weil Befehle andersartig sind ⇒ hat jeder Befehl einen spezifischen Ablauf

Befehlssatzarchitektur = Instruction Set Architecture

Die Maschinenbefehle eines Prozessortyps / einer Prozessorfamilie

- ... sind weitgehend orthogonal codiert (alle Kombinationen von Teilfunktionen systematisch ausgenutzt, alle Operanden mit allen Operationen verbindbar, keine Redundanz)
- ... haben verschiedene Längen und bewirken komplexe Operationen von unterschiedlicher Dauer ($\rightsquigarrow$ CISC)
- oder haben einheitlich gleiche Längen und bewirken einfache Operationen von mit einheitlich kurzer Dauer ($\rightsquigarrow$ RISC)
- ... bilden eine Vorrat (Instruction Set), der für eine Prozessorfamilie charakteristisch und kompatibel ist
- ... werden durch Assemblertools aus der Quellcodes der Assemblersprache (Menschen bequem lesbare Symbole für die Maschinenbefehle $\rightsquigarrow$ Mnemonics)
- ... oder Compilertools aus Quellcodes der Hochsprache (Abstrakte, maschinenunabhängige Formulierung erzeugt)

CISC und RISC



Viele, komplexe Maschinenbefehle, vielfältige Adressierungsmöglichkeiten, unterschiedliche Größe und Dauer

Wenige, einheitliche Maschinenbefehle, wenige Adressierungsmodi, gleiche Größe und i.allg. auch gleiche, kurze Dauer

Maschinenbefehlsstruktur (Instruction Bit Fields)

Ein Maschinenbefehl besteht aus Bitfeldern mit folgender allgemeiner Gliederung

1) Feld OPC0DE, der die Operation festlegt

2) Feld Zieloperand R_n, i.allg. das Register in dem das Ergebniss gespeichert wird

3) Feld erster Quelloperanden R_s, i.allg. das Register in ein Operand steht

4a) Feld Für den zweiten Quelloperanden R_s, i.allg. das Register in ein Operand steht

4b) Feld für (kurze) Direkt-Operanden als zweiten Quelloperanden, Direkt-Operanden sind unmittelbar im Maschinenbefehl codierte Zahlenwerte

5) optional ein Feld für abzuprüfende Bedingungen (Conditions = Statusflags, die gesetzt sind) um den Maschinenbefehl auszuführen oder nicht auszuführen

Speicherformate - Endianness

Endianness: Festlegung des zu verwendenden Speicherungsformats, wenn mehrere adressierbare Speicherzellen (Bytes) für eine Datenblock oder Zahlenwert benutzt werden.

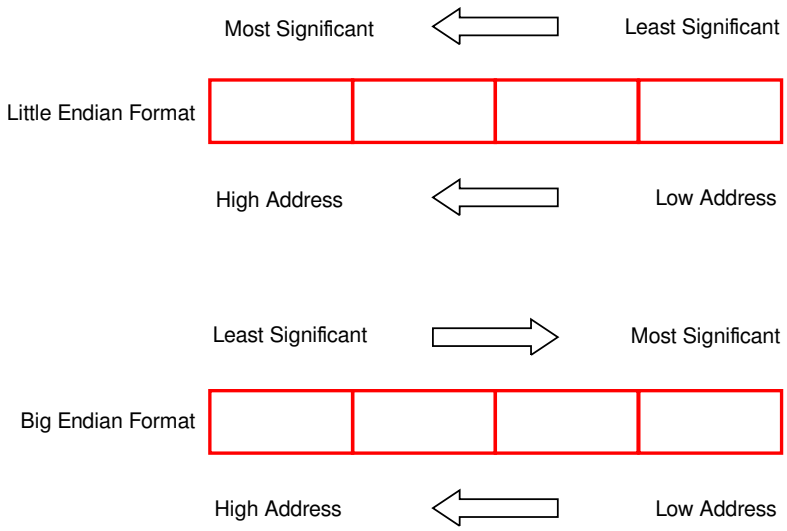
Byte-Order: Little Endian

- Byte-Reihenfolge: Byte mit den niederwertigen Bits zuerst bzw. an den Stellen mit der niedrigsten Adresse
- Intel PC, Windows

Byte-Order: Big Endian

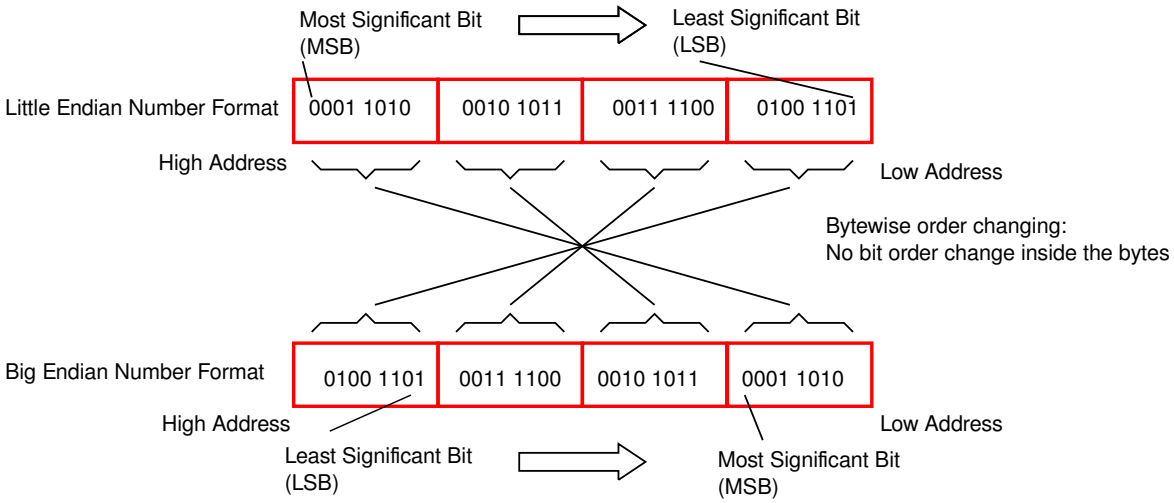
- Byte-Reihenfolge: Byte mit den höchstwertigen Bits zuerst bzw. an den Stellen mit der niedrigsten Adresse
- Motorola, IBM (Mainframes), Internetprotokolle mit Network Byte Order Big-Endian, Unix

Big Endian & Little Endian



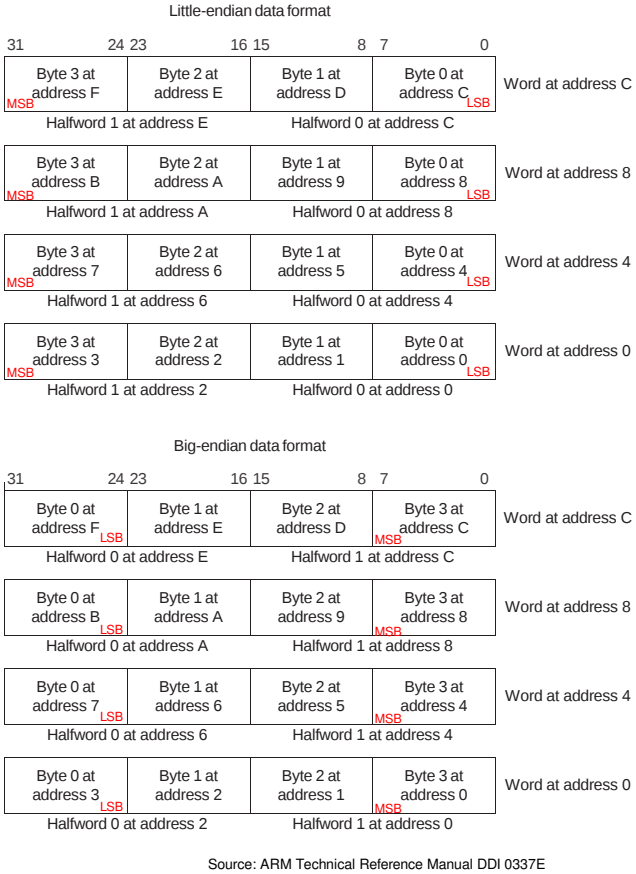
Die Bytes einer Zahl oder eines Datenblocks werden entweder absteigend oder aufsteigend in der Reihenfolge der Adressen gespeichert

Big Endian & Little Endian



Die Reihenfolge ist byteweise umgekehrt, im Byte selbst bleibt die Reihenfolge der Bits erhalten. Dabei hat Bit 7 den höchsten Wert und Bit 0 den niedrigsten.

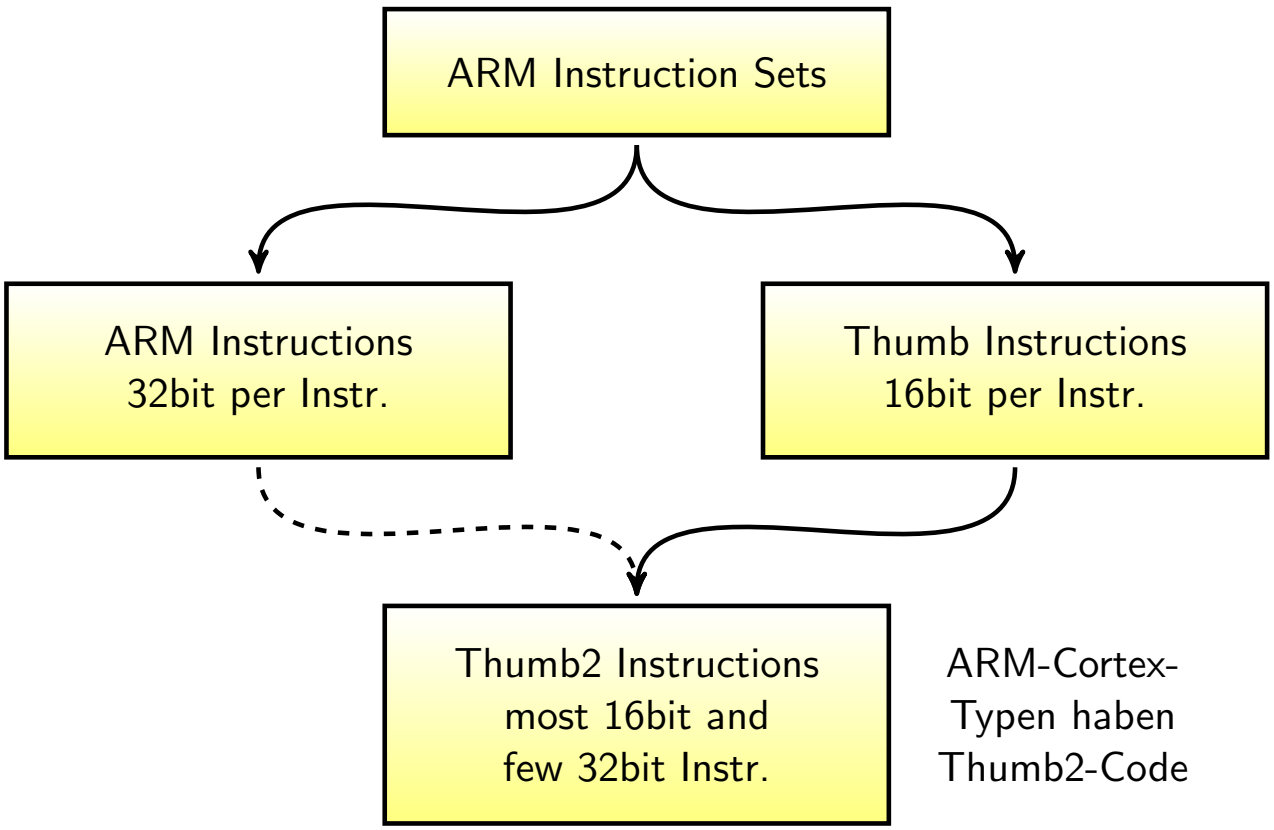
Sowohl Big Endian als auch Little Endian bei ARM



Source: ARM Technical Reference Manual DDI 0337E

Statische Umschaltung: Nach Reset/Power-On durch Abfrage eines Pins

ARM und Thumb Instruction Sets



ARM-Cortex-Typen haben Thumb2-Code

Warum Thumb Code ?

Thumb is a subset of the ARM instruction set encoded in 16-bit-wide instructions.

- Requires 70% of the space of ARM code.
- Uses 40% more instructions than equivalent ARM code.
- With 32-bit memory:

  ARM code is 40% faster than Thumb code.
- With 16-bit memory::

  Thumb code is 45% faster than ARM code.
- Uses 30% less external memory power than ARM code.

Source: www.davespace.co.uk/arm/

Thumb Instruction Set Format (ARM Cortex)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			*Move shifted register*
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			*Add/subtract*
3	0	0	1	Op		Rd			Offset8								*Move/compare/add /subtract immediate*
4	0	1	0	0	0	0	Op				Rs			Rd			*ALU operations*
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			*Hi register operations /branch exchange*
6	0	1	0	0	1	Rd			Word8								*PC-relative load*
7	0	1	0	1	L	B	0	Ro			Rb			Rd			*Load/store with register offset*
8	0	1	0	1	H	S	1	Ro			Rb			Rd			*Load/store sign-extended byte/halfword*
9	0	1	1	B	L	Offset5					Rb			Rd			*Load/store with immediate offset*
10	1	0	0	0	L	Offset5					Rb			Rd			*Load/store halfword*
11	1	0	0	1	L	Rd			Word8								*SP-relative load/store*
12	1	0	1	0	SP	Rd			Word8								*Load address*
13	1	0	1	1	0	0	0	0	S	SWord7							*Add offset to stack pointer*
14	1	0	1	1	L	1	0	R	Rlist								*Push/pop registers*
15	1	1	0	0	L	Rb			Rlist								*Multiple load/store*
16	1	1	0	1	Cond				Soffset8								*Conditional branch*
17	1	1	0	1	1	1	1	1	Value8								*Software Interrupt*
18	1	1	1	0	0	Offset11											*Unconditional branch*
19	1	1	1	1	H	Offset											*Long branch with link*
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Opcodes + Mnemonics I

Mnemonic	Instruction	1. Reg. Operand	2. Reg. Operand	Condition Flags set
ADC	Add with Carry	✓	✓	
ADD	Add	✓	✓	✓
AND	AND	✓	✓	
ASR	Arithmetic Shift Right	✓	✓	
B	Unconditional branch	✓		
Bxx	Conditional branch	✓		
BIC	Bit Clear	✓	✓	
BL	Branch and Link			
BX	Branch and Exchange	✓	✓	
CMN	Compare Negative	✓		✓
CMP	Compare	✓	✓	✓
EOR	Exclusive Or	✓		✓
•	•	•	•	•

Opcodes + Mnemonics II

Mnemonic	Instruction	1. Reg. Operand	2. Reg. Operand	Condition Flags set
•	•	•	•	•
LDMIA	Load multiple	✓		
LDR	Load word	✓		✓
LDRB	Load byte	✓		
LDRH	Load halfword	✓		
LSL	Logical Shift Left	✓		✓
LDSB	Load signed byte	✓		
LDSH	Load signed halfword	✓		
LSR	Logical Shift Right	✓		✓
MOV	Move Register	✓	✓	✓
MUL	Multiply	✓		✓
MVN	Move Negative Register	✓		✓
NEG	Negate	✓		✓
ORR	OR	✓		✓
•	•	•	•	•

Opcodes + Mnemonics III

Mnemonic	Instruction	1. Reg. Operand	2. Reg. Operand	Condition Flags set
•	•	•	•	•
POP	Pop registers	✓		
PUSH	Push registers	✓		
ROR	Rotate Right	✓		✓
SBC	Subtract with Carry	✓		✓
STMIA	Store Multiple	✓		
STR	Store word	✓		
STRB	Store byte	✓		
STRH	Store halfword	✓		
SWI	Software Interrupt			
SUB	Subtract	✓		✓
TST	Test bits	✓		✓

Arithmetische Befehle in Assembler-Syntax

<operation> {cond} {S} Rd, Rn, <Operand2>

<operation>

ADD -add → Rd := Rn + Operand2

ADC - add with carry → Rd := Rn + Operand2 + Carry

SUB - subtract → Rd := Rn - Operand2

SBC - subtract with carry → Rd := Rn - Operand2 -NOT(Carry)

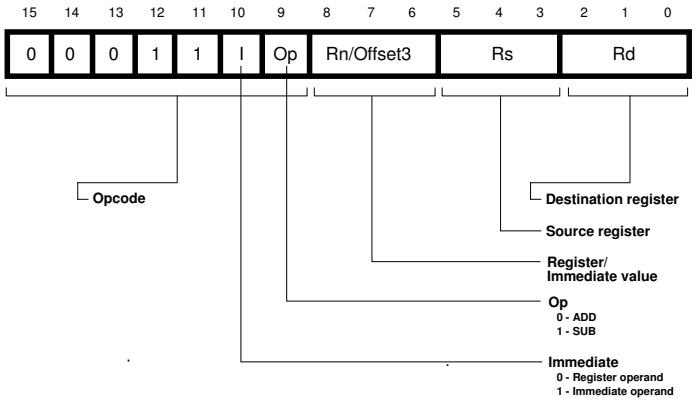
RSB - reverse subtract → Rd := Operand2 - Rn

RSC - reverse subtract w. carry → Rd := Operand2-Rn-NOT(Carry)

Beispiele

```
ADD r0, r1, r2 ; R0 = R1 + R2
SUB r5, r3, #10 ; R5 = R3 - 10
RSB r2, r5, #0xFF00 ; R2 = 0xFF00 - R5
```


3-Operanden-Befehle mit ADD/SUB-Operationen



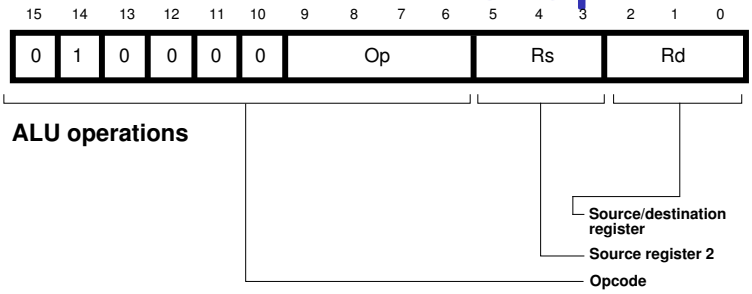
Op	I	THUMB assembler	ARM equivalent	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

Quelle: ARM7TDMI Data Sheet

3-Operanden-Befehle mit ADD/SUB-Operationen

```
1   ;  Examples: Add/Subtract Operations as Three
2   ;  Operand Instructions
3   ;  Option second Operand Register R1...
4   ;  Option second Operand imediate 3Bit Value
5
6   ;  Source ARM7TDMI Data Sheet Page 5-8
7
8
9   ADD R0, R3, R4 ; R0 := R3 + R4
10                 ;and set condition codes on
11                 ; the result.
12
13  SUB R6, R2, #6 ; R6 := R2 - 6
14                 ; and set condition codes.
```


2-Operanden-Befehle mit ALU-Operationen



OP	THUMB assembler	ARM equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = -Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

2-Operanden-Befehle mit ALU-Operationen

```
1  ; Examples: ALU_Operations
2  ; as Two Operand Instructions
3
4  ; Source ARM7TDMI Data Sheet Page 5-18
5
6  EOR R3, R4          ; R3 := R3 EOR R4
7                      ; EOR = Exclusive-Or and set condition codes
8
9  ROR R1, R0          ; Rotate Right R1 by the value in R0, store
10                     ; the result in R1 and set condition codes
11
12 NEG R5, R3          ; Subtract the contents of R3 from zero,
13                     ; store the result in R5. Set condition codes
14                     ; ie R5 = -R3
15
16 CMP R2, R6          ; Set the condition codes on the result of
17                     ; R2 - R6
18
19 MUL R0, R7          ; R0 := R7 * R0 and set condition codes
```


Data Transfer in Assembler-Syntax

<operation> {cond} {size} Rd, <address>

<operation>

LDR - load→ Rd := value at ¡address¿

STR - store → value at <address> := Rd

{size} is specified only to transfer bytes or half-words

<operation>B: unsigned byte

<operation>SB: signed byte

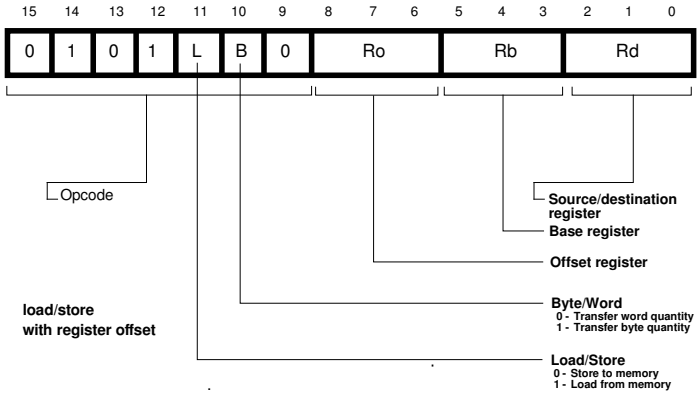
<operation>H: unsigned half-word

<operation>SH:signed half-word

Beispiele

```
LDR r0,[r1] ; Load word addressed by R1 into
R0.
LDRB r0,[r1] ; The same as above but loads a
byte.
```


Load/Store-Befehle



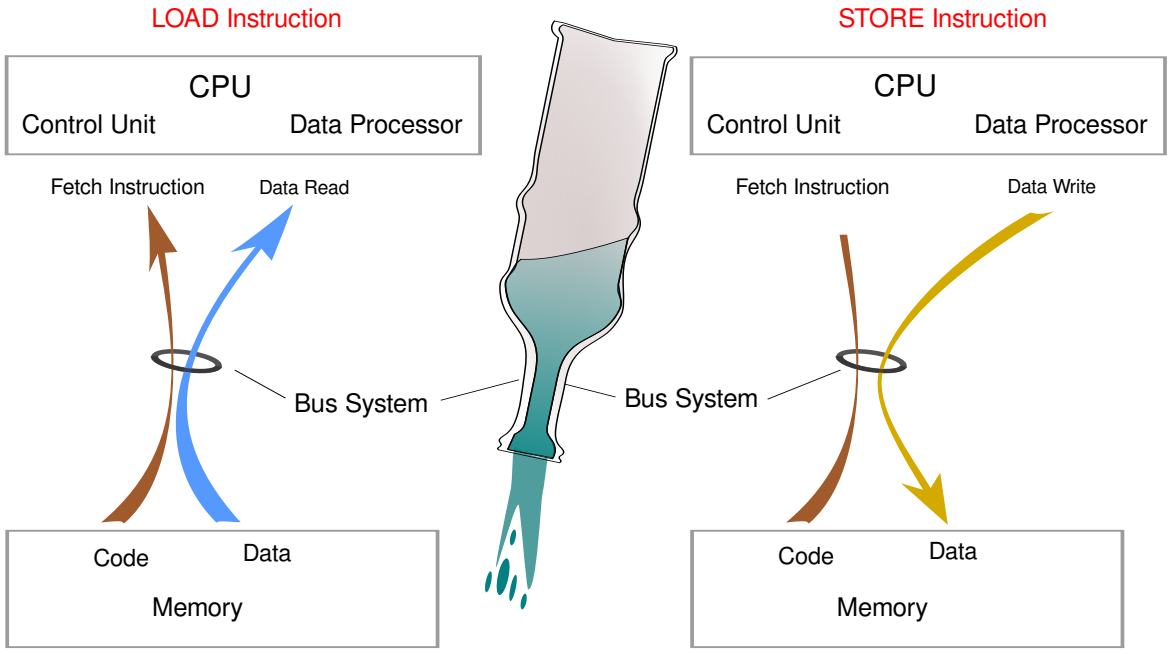
L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

Quelle: ARM7TDMI Data Sheet

2-Operanden-Befehle mit ALU-Operationen

```
1   ;  Examples: Add/Subtract Operations as Three
2   ;  Operand Instructions
3   ;  Option second Operand Register R1...
4   ;  Option second Operand immediate 3Bit Value
5
6   ;  Source ARM7TDMI Data Sheet Page 5-8
7
8
9   ADD R0, R3, R4 ; R0 := R3 + R4
10                 ;and set condition codes on
11                 ; the result.
12
13  SUB R6, R2, #6 ; R6 := R2 - 6
14                 ; and set condition codes.
```


Von-Neumann-Bottleneck verlangsamt bei Load und Store Befehlen



Die Speicherbefehle benötigen für mehrfache Speicherzugriff mehrere Takt-Zyklen.

Branch in Assembler-Syntax

<operation> {**cond**} **<address>**

`<operation>`

einfacher Sprungbefehl

B - branch → `PC := <address>`

bedingte ausgeführter Sprungbefehl

Bxx - if (Flag for Condition xx in Statusregister is set) → `PC := <address>`

Unterprogrammaufruf = Call (Branch with link back)

BL - branch with link (call) → `R14 := address of next instruction` and `PC := <address>`

How return from the subroutine called by BL ?

`MOV pc, r14` or **`BX r14`**

Befehl Unconditional Branch

Unconditional Branch

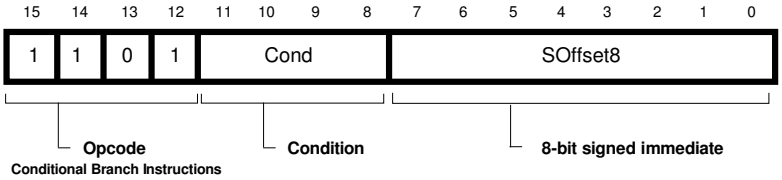
THUMB assembler	ARM equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

Quelle: ARM7TDMI Data Sheet

Unconditional Branch - One Immediate Operand - Destination PC-Relative

```
1   ;  Example: Unconditional Branch Operation as
2   ;  One Operand Instruction
3   ;  Another Name is 'Unconditional Jump'
4   ;  Operand is an Immediate 11Bit Value
5   ;  This instruction performs a PC-relative Branch
6   ;  The Operand is shifted by one and added to PC-Content
7   ;  The Range of Branches is PC +/- 2048 bytes
8   ;  Used with symbolic labels (arbitary identifier name)
9
10  ;  Source ARM7TDMI Data Sheet Page 39 (modified)
11
12
13          B label_xy ; Branch to labelx ('label_xy' is a identifier name).
14
15                 ... ;    Note that the Thumb opcode will
16                     ;    contain the number of halfwords
17                     ;    to offset.
18  label_xy       ... ;    Destination must be halfword aligned
19
20
21  here       B here      ; Branch onto itself.
22                         ; Assembles to 0xE7FE.
23                         ; similar to C-loop 'while(1);'
```

Befehle Conditional Branch



Conditional Branch Instructions

Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

Quelle: ARM7TDMI Data Sheet

Conditional Branch - Condition Code + One Immediate Operand - Destination PC-Relative

```
1   ;  Example: Conditional Branch Operation as
2   ;  Condition Code plus One Operand Instruction
3   ;  Another Name is 'Conditional Jump'
4   ;  Operand is an Immediate 8Bit Value
5   ;  This instruction performs a PC-relative Branch
6   ;  The Operand is shifted by one and added to PC-Content
7   ;  The Range of Branches is PC +/- 256 bytes
8   ;  Used with symbolic labels (arbitary identifier name)
9
10  ;  Source ARM7TDMI Data Sheet Page 36/37 (modified)
11
12                        ; Branch to "over" if R0 > 45.
13        CMP R0, #45     ; Step 1: Test instruction (like SUB
14                        ; delivers status register Flag Z (Zero-Flag)
15                        ; Step 2: Branch to label "over" if condition is fullfilled
16         BGT over       ; Branch if Z clear, and either N set
17                        ; and V set or N clear and V clear
18                        ;  (greater than)
19                        ; Note that the Thumb inmstruction code will contain
20        ...             ; the number of halfwords to offset.
21        ...
22        ...
23   over ...             ; Must be halfword aligned.
24        ...
```