Zusammenfassung: Grundlagen der Informatik

Zahlensysteme, b-adische Darstellung, Umrechnung

Beispiel: Umrechnung von $(69.59375)_{10}$ ins Dualsystem

Rechnen im Quellsystem

Divisionsverfahren

69 / 2 = 34 Rest *1*
34 / 2 = 17 Rest *0*
17 / 2 = 8 Rest *1*
8 / 2 = 4 Rest *0*
4 / 2 = 2 Rest *0*
2 / 2 = 1 Rest *0*
1 / 2 = 0 Rest *1*
→ ganzzahliger Anteil: $(1000101)_2$

$0.59375 \cdot 2 = 1.1875$
$0.1875 \cdot 2 = 0.375$
$0.375 \cdot 2 = 0.75$
$0.75 \cdot 2 = 1.5$
$0.5 \cdot 2 = 1$
→ gebrochener Anteil: $(0.10011)_2$

Subtraktion der größten Potenzen

$69 - 2^6 = 5$
$5 - 2^2 = 1$
$1 - 2^0 = 0$
→ ganzzahliger Anteil $2^6 + 2^2 + 2^0 = (1000101)_2$

$0.59375 = 19 / 2^5$
$19 - 2^4 = 3$
$3 - 2^1 = 1$
$1 - 2^0 = 0$
→ $(19)_{10} = (10011)_2$ → gebrochener Anteil: $(0.10011)_2$

Potenzen

x	2^x	16^x
0	1	1
1	2	16
2	4	256
3	8	4096
4	16	65536
5	32	1048576
6	64	16777216
7	128	268435456
8	256	4294967296
9	512	68719476736
10	1024	
11	2048	
12	4096	
13	8192	
14	16384	
15	32768	
16	65536	

Hexziffern

A	10
B	11
C	12
D	13
E	14
F	15

Rechnen im Zielsystem

Mult.-/Additionsverfahren

$(1000101.10011)^2$
$= 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} = (69.59375)_{10}$

Ergebnis (in jedem Fall): $(69.59375)_{10} = (1000101.10011)_2$

Hornerschema

Beispiel: $14 \cdot 16^3 + 15 \cdot 16^2 + 0 \cdot 16^1 + 1 = ((14 \cdot 16 + 15) \cdot 16 + 0) \cdot 16 + 1 = 61185$
weiteres Beispiel: $x^2 + 3x + 4 = (x + 3) \cdot x + 4$

Zahlendarstellung

Zahlen mit Vorzeichen

- Darstellung mit Vorzeichen und Betrag
  - Bsp.: $-8 = (10001000)_2$ mit 8 Bit
- Exzessdarstellung
  - Bsp.: $-8 = (01111000)_2$ in Exzess-128 mit 8 Bit
- b-1-Komplement
  - Bsp.: $-8 = (11110111)_2$ im Einerkomplement mit 8 Bit
  - Berechnung des b-1-Komplment von a: $(b^N - 1) - a$
  - Bei Addition Einerrücklauf beachten (im Gegensatz zum b-Komplement)
- b-Komplement
  - Bsp.: $-8 = (11111000)_2$ im Zweierkomplement mit 8 Bit
  - Berechnung des b-Komplment von a: $(b^N) - a$
  - *b-Komplement = b-1-Komplement* + 1

Reelle Zahlen

- **Festkommadarstellung**

- feste Zahl von Bits steht für Vor- und Nachkommastellen zur Verfügung
- Vorzeichen wird in den Vorkommastellen dargestellt
- Bsp.: $(1110.001011)_2 = (0000\ 0000\ 1110\ 0010)_2$ mit 12 Bits für die Vorkomma- und 4 Bits für die Nachkommasstellen

- **Allgemeine Gleitkommadarstellung**
  - Form: $z = m \cdot B^e$ (mit Mantisse *m*, Exponent *e* und fester Exponentenbasis *B)*
  - Mantisse: gebrochener Anteil, bestimmt Genauigkeit
    Bedingung für Normalisierung: $B^{-1} \leq m < 1$ (bei IEEE anders!)
  - Exponent: positive oder negative Ganzzahl (angegeben in Exzess-128), bestimmt den Bereich der Darstellbaren Zahlen
- **Unterschiede und Festlegungen der Gleitkommadarstellung im IEEE-Format**
  - Exponentenbasis: 2
  - „Single Precision" (32-Bit): 1 Bit für Vorzeichen, 23 Bit für Mantisse, 8 Bit für Exponent
  - Null: $e = (0...0)_2$, $m = (0...0)_2$
  - Unendlich: $e = (1...1)_2$, $m = (0...0)_2$, Vorzeichen entscheidet ob Plus- oder Minusunendlich
  - NaN: $e = (1...1)_2$, $m \neq (0...0)_2$
  - Normalisiertes Format: $e \neq (0...0)_2$ und $e \neq (1...1)_2$
    - Bedingung für Normalisierung der Mantisse: $1 \leq |m| \leq 2$
      $m = (1.M_1M_2M_3...M_{23})_2$
    - Exponent: in **Exzess-127** angegeben
  - Denormalisiertes Format: $e = (0...0)_2$ und $m \neq (1...1)_2$
    - Mantisse muss nicht normalisiert sein:
      $m = (0.M_1M_2M_3...M_{23})_2$
    - Exponent: auf -126 festgelegt

Ermitteln des Unterlauf- und Überlaufbereichs der Darstellung

- Unterlaufbereich: $|z| < m \cdot B^e$ (für alle Darstellbaren m > 0, e)
  - kleinstes *m* suchen, z. B. $m = (0.1)_2$ (allg. Darstellung) oder $m = 2^{-23}$ (IEEE 32-Bit, denormalisiertes Format)
  - kleinstes *e* suchen, z. B. $e = -128$ (für 8-Bit-Zweierkomplement) oder $e = -126$ (IEEE 32-Bit, denormalisiertes Format)
- Überlaufbereich: $|z| > m \cdot B^e$ (für alle Darstellbaren m > 0, e)
  - größte *m* suchen, z. B. $m = 1 - 2^{-15}$ (allg. Darstellung, 15-Bit Mantisse) oder $m = 2 - 2^{-23}$ (IEEE 32-Bit)
  - größte *e* suchen, z. B. $e = 127$ (für 8-Bit-Zweierkomplement) oder $e = 127$ (IEEE 32-Bit)

Ermitteln des minimalen Abstands zweier Werte in bestimmten Bereich

Beispiel: Bereich: 2^{-30} bis 2^{-29}, Mantissengröße: 15 Bit

1. Überprüfen, ob gilt $e_1 = e_2$:
   $$2^{-30} \leqslant z_1 \wedge z_2 < 2^{-29} \rightarrow (0.1)_2 \cdot 2^{-29} \leqslant z_1 \wedge z_2 < (1.0)_2 \cdot 2^{-29} \rightarrow e_1 = e_2 = 2^{-29}$$
   für IEEE-Format:
   $$2^{-30} \leqslant z_1 \wedge z_2 < 2^{-29} \rightarrow (1.0)_2 \cdot 2^{-30} \leqslant z_1 \wedge z_2 \leqslant (1.1\ldots1)_2 \cdot 2^{-30} \rightarrow e_1 = e_2 = 2^{-30}$$
2. Kleinsten Abstand Δm_{min} zwischen m_1 und m_2 ermitteln: in diesem Beispiel mit 15 Mantissenstellen $\Delta m_{min} = 2^{-15}$
3. Dann gilt folgende Formel:
   $$d_{min} = z_1 - z_2 = m_1 \cdot 2^{e_1} - m_2 \cdot 2^{e_2} = (m_1 - m_2) \cdot 2^{e_{1/2}} = \Delta m_{min} \cdot 2^{e_{1/2}} = 2^{-15} \cdot 2^{-29} = 2^{-44}$$

Darstellbarkeit einer Zahl überprüfen

Wann ist eine Zahl *nicht* darstellbar?

- Unterlauf, Überlauf
- Zahl der Mantissenstellen genügt nicht, um Wert präzise darzustellen

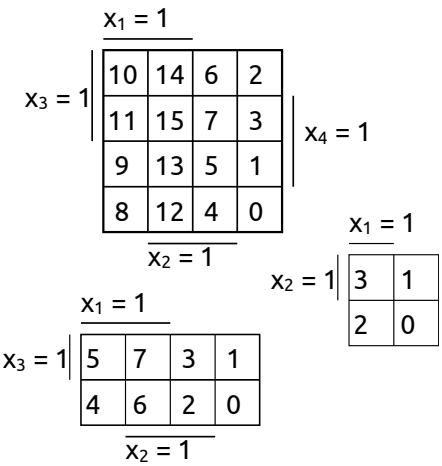
Beispiel (nicht IEEE):

$$z = 256 + \frac{17}{256} = (0.1)_2 \cdot 2^9 + (10001)_2 \cdot 2^{-8} = (0.1)_2 \cdot 2^9 + (0.00000000000010001)_2 \cdot 2^9 =$$
$$= (0.10000000000010001)_2 \cdot 2^9$$

→ hier würden mindestens 17 Mantissenstellen benötigt, um den Wert präzise darzustellen

KV-Diagramme

Minterme: jede Variable (bei 0 negiert); durch *und* verknüpft
Maxterme: jede Variable (bei 1 negiert); durch *oder* verknüpft
DNF: alle Minterme mit einschlägigen Index durch *oder* verknüpft
KNF: alle Maxterme mit nicht einschlägigen Index durch *und* verknüpft



Flipflops

RS-Flipflop

R	S	Q_{alt}	Q_{neu}	
0	0	0	0	Ruhe speichern
0	0	1	1	
0	1	0	1	Setzen
0	1	1	1	
1	0	0	0	Rücksetzen
1	0	1	0	

JK-Flipflop

J	K	Q_{alt}	Q_{neu}	
0	0	0	0	Ruhe speichern
0	0	1	1	
0	1	0	0	Rücksetzen
0	1	1	0	
1	0	0	1	Setzen
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

D-Flipflop

D	Q_{alt}	Q_{neu}	
0	0	0	Rücksetzen
0	1	0	
1	0	1	Setzen
1	1	1	

T-Flipflop

T	Q_{alt}	Q_{neu}	
0	0	0	Ruhe speichern
0	1	1	
1	0	1	Toggle
1	1	0	

MIPS-Architektur

Eckdaten: RISC (reduced instruction set computer), Load/Store-Architektur (Nur Speicherwerk mit Registern verbunden, Rechenwerk/ALU ließt und schreibt nur aus/zu Registern), 32 gewöhnliche Register (erstes immer Null), 2 Spezialregister (hi, lo)

Ablauf: Befehle stehen im Speicher, Programmschrittzähler (PC) zeigt auf aktuellen Befehl, Befehl wird geholt und ausgeführt, PC ← PC + 4, Befehl wird geholt und aus...

Dekodierung der Maschinenbefehle

- Format R (register): Opcode = $(000000)_2$
  - Rs: Nummer des Eingaberegisters 1 (5 Bit für Nummer 0-31)
  - Rt: Nummer des Eingaberegisters 2 (5 Bit)
  - Rd: Nummer des Ausgaberegisters (5 Bit)
  - Shamt: Zahl der Stellen, um die geschoben werden soll (Shift amount, 5 Bit, nur bei Schiebeoperationen verwendet)
  - Function: bestimmt Operation (6 Bit)
- Format J (jump): Opcode = $(00001x)_2$
  - Opcode (6 Bit) bestimmt Funktion, z. B. $(000010)_2$ absoluter Sprung
  - Restliche 26 Bit z. B. Sprungadresse
- Format I (immediate): Opcode = $(xxxxxx)_2$
  - Opcode (6 Bit) bestimmt Funktion, z. B. $(001000)_2$ Addieren
  - Rs: Quellregister (5 Bit)
  - Rt: Zielregister (5 Bit)
  - Wert (16 Bit)

Übersicht über die wichtigsten Maschinen-/Assemblerbefehle

Arithmetik

Addition: Rd = Rs + Rt
mit Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	Rd	00000	**100000**
Assembler	**add**	$Rd, $Rs, $Rt				

ohne Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	Rd	00000	**100001**
Assembler	**addu**	$Rd, $Rs, $Rt				

Addition eines unmittelbaren Wertes (im Zweierkomplement): Rt = Rs + Wert

mit Beachtung des Vorzeichens:

Format I	**001000**	Rs	Rt	Wert (16 Bit)
Assembler	**addi**	$Rt, $Rs, Wert		

ohne Beachtung des Vorzeichens:

Format I	**001001**	Rs	Rt	Wert (16 Bit)
Assembler	**addiu**	$Rt, $Rs, Wert		

Subtraktion: Rd = Rs – Rt

mit Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	Rd	00000	**100010**
Assembler	**sub**	$Rd, $Rs, $Rt				

ohne Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	Rd	00000	**100011**
Assembler	**subi**	$Rd, $Rs, $Rt				

Multiplikation zweiter 32-Bit-Werte zu einem 64-Bit-Wert: (hi, lo) = Rs · Rt

mit Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	00000	00000	**011010**
Assembler	**mult**	$Rs, $Rt				

ohne Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	00000	00000	**011011**
Assembler	**multu**	$Rs, $Rt				

Ganzzahlige Division mit Rest: hi = Rs mod Rt, lo = Rs / Rt

mit Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	00000	00000	**011010**
Assembler	**div**	$Rs, $Rt				

ohne Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	00000	00000	**011011**
Assembler	**divu**	$Rs, $Rt				

Verschieben von hi: Rd = hi

Format R	**000000**	00000	00000	Rd	00000	**010000**
Assembler	**mfhi**	$Rd				

Verschieben von lo: Rd = lo

Format R	**000000**	00000	00000	Rd	00000	**010010**
Assembler	**mflo**	$Rd				

Logische Operationen und Vergleichsoperationen

Kleiner als: wenn Rs < Rt, dann Rd = 1, sonst Rd = 0

mit Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	Rd	00000	**101010**
Assembler	**slt**	$Rd, $Rs, $Rt				

ohne Beachtung des Vorzeichens:

Format R	**000000**	Rs	Rt	Rd	00000	**101011**
Assembler	**sltu**	$Rd, $Rs, $Rt				

Kleiner als mit unmittelbarem Wert: wenn Rs < Wert, dann Rd = 1, sonst Rd = 0

mit Beachtung des Vorzeichens:

Format I	**001010**	Rs	Rt	Wert (16 Bit)
Assembler	**slti**	$Rt, $Rs, Wert		

ohne Beachtung des Vorzeichens:

Format I	**001011**	Rs	Rt	Wert (16 Bit)
Assembler	**sltiu**	$Rt, $Rs, Wert		

Bitweise UND-Verknüpfung: Rd = Rs und Rt

Format R	**000000**	Rs	Rt	Rd	00000	**100100**
Assembler	**and**	$Rd, $Rs, $Rt				

Bitweise ODER-Verknüpfung: Rd = Rs oder Rt

Format R	**000000**	Rs	Rt	Rd	00000	**100101**
Assembler	**or**	$Rd, $Rs, $Rt				

Bitweise XOR-Verknüpfung: Rd = Rs xor Rt

Format R	**000000**	Rs	Rt	Rd	00000	**100110**
Assembler	**xor**	$Rd, $Rs, $Rt				

Bitweise NOR-Verknüpfung: Rd = Rs nor Rt

Format R	**000000**	Rs	Rt	Rd	00000	**100111**
Assembler	**nor**	$Rd, $Rs, $Rt				

Bitweise UND-Verknüpfung mit unmittelbarem Wert: Rd = Rs und Rt

Format I	**001100**	Rs	Rt	Wert (16 Bit)
Assembler	**andi**	$Rt, $Rs, Wert		

Bitweise ODER-Verknüpfung mit unmittelbarem Wert: Rd = Rs oder Rt

Format I	**001101**	Rs	Rt	Wert (16 Bit)
Assembler	**ori**	$Rt, $Rs, Wert		

Bitweise XOR-Verknüpfung mit unmittelbarem Wert: Rd = Rs xor Rt

Format I	**001110**	Rs	Rt	Wert (16 Bit)
Assembler	**xori**	$Rt, $Rs, Wert		

Sprünge

Bedingter relativer Sprung: Springe, wenn eine Bedingung erfüllt ist, um die als Wert angegebene Anzahl von *32-Bit-Wörtern*.

Springe bei Gleichheit der Inhalte von Rs und Rt:

Format I	**000100**	Rs	Rt	Wert (16 Bit)
Assembler	**beq**	$Rs, $Rt, Marke		

Springe bei Ungleichheit der Inhalte von Rs und Rt:

Format I	**000101**	Rs	Rt	Wert (16 Bit)
Assembler	**bne**	$Rs, $Rt, Marke		

Absoluter Sprung: Springe zur angegebenen Adresse, wobei

- die letzten beiden Ziffern immer 0 sind und daher weggelassen werden
- und die ersten vier Bits aus Platzgründen auch weggelassen werden und dafür der Wert der Adresse des Sprungsbefehles angenommen wird.

Format J	**000010**	Adresse (26 Bit, siehe Bemerkung)
Assembler	**j**	Marke

Gleicher Befehl mit vorherigem Speichern der Addresse der nächsten Instruktion in $ra:

Format J	**000011**	Adresse (26 Bit, siehe Bemerkung)
Assembler	**jal**	Marke

Absoluter Sprung mittels indirekter Adressierung: Interpretiere den Wert von Register Rs als Zieladresse und springe dorthin.

Format R	**000000**	Rs	00000	00000	00000	**001000**
Assembler	**jr**	$Rs				

Speicheroperationen

„Load Word“: Hole das Wort von der Adresse im Speicher, die sich aus dem Inhalt von Rs plus dem Versatz ergibt und speichere es in Rt.

Format I	**100011**	Rs	Rt	Versatz (16 Bit)
Assembler	**lw**	$Rt, Versatz($Rs)		

„Load Byte“: Hole das Byte von der Adresse (siehe oben) und speichere es in den rechten 8 Bit von Rt.

mit Beachtung des Vorzeichens:

Format I	**100000**	Rs	Rt	Versatz (16 Bit)
Assembler	**lb**	$Rt, Versatz($Rs)		

ohne Beachtung des Vorzeichens:

Format I	**100100**	Rs	Rt	Versatz (16 Bit)
Assembler	**lbu**	$Rt, Versatz($Rs)		

„Store Word“: Schreibe den Inhalt von Rt an die Speicheradresse (Berechnung siehe oben).

Format I	**101011**	Rs	Rt	Versatz (16 Bit)
Assembler	**sw**	$Rt, Versatz($Rs)		

„Store Byte“: Schreibe die rechtesten 8 Bit von Rt an die berechnete Speicheradresse.

Format I	**101000**	Rs	Rt	Versatz (16 Bit)
Assembler	**sb**	$Rt, Versatz($Rs)		

Load „upper word immediate“: Linke 16 Bit von Rt werden auf Wert gesetzt, die restlichen 16 Bit auf 0. Die rechten 16-Bit können mit *ori* in einem zweiten Schritt gesetzt werden.

Format I	**001111**	0000	Rt	Wert (16 Bit)
Assembler	**lui**	$Rt, Wert		

Schiebeoperationen

Schieben nach links (der Zahl in $Rt, um shamt Stellen, Ergebnis speichern in $Rd):

Format R	**000000**	00000	Rt	Rd	shamt	**000000**
Assembler	**sll**	$Rd, $Rt, shamt				

Schieben nach rechts (der Zahl in $Rt, um shamt Stellen, Ergebnis speichern in $Rd):

mit Beachtung des Vorzeichens:

Format R	**000000**	00000	Rt	Rd	shamt	**000010**
Assembler	**srl**	$Rd, $Rt, shamt				

ohne Beachtung des Vorzeichens:

Format R	**000000**	00000	Rt	Rd	shamt	**000011**
Assembler	**sra**	$Rd, $Rt, shamt				

Marken und Pseudobefehle

Notation von Marken: „name_der_marke:“; Marke kann dann anstelle einer Adresse verwendet werden.

li	$Rd, Wert	Belegung eines Registers mit einem 32-Bit-Wert
la	$Rd, Marke	Laden der *Adresse* einer Marke
move	$Rd, $Rs	Kopieren des Wertes von $Rs nach $Rd
not	$Rd, $Rs	Einerkomplement des Wertes von $Rs in $Rd speichern
b	Marke	unbedingter (relativer) Sprung

Programmcodebereiche und Datenbereiche

Programmcode wird eingeleitet durch „.text“, Datenbereiche durch „.data“. Direktiven in Datenbereichen:

.byte Wert1, Wert2, …, WertN	Bytes, die in dieser Reihenfolge im Speicher abgelegt werden
.word Wert1, Wert2, …, WertN	Wörter, die in dieser Reihenfolge im Speicher abgelegt werden
.space n	Reserviert einen n Byte langen Bereich
.ascii Zeichenkette	Bytes, die eine Zeichenkette bilden
.asciiz Zeichenkette	Bytes, die eine null-terminierte Zeichenkette bilden

Assemblersymbole

Direkte Bezeichnung	Symbolische Bezeichnung	Bedeutung
$0	$zero	immer 0
$1	$at	Assembler nutzt dies temporär
$2, $3	$v0, $v1	Ergebnisse (values) von Unterprogrammen
$4, …, $7	$a0, $a1, $a2, $a3	Aufrufparameter für Unterprogramme
$8, …, $15	$t0, …, $t7	temporäre Werte; können vom Unterprogramm geändert werden
$16, …, $23	$s0, …, $s7	gespeicherte Werte; zurückstellen vor Rückkehr
$24, $25	$t8, $t9	weitere temporäre Werte
$26, $27	$k0, $k1	reserviert für spezielle Ereignisse
$28	$gp	Globalspeicherzeiger (Pointer)
$29	$sp	Stapelspeicherzeiger (Pointer)
$30	$s8/$fp	Framepointer bzw. weitere s-Variable
$31	$ra	Rücksprungadresse für Unterprogramme