

Inhaltsverzeichnis

1	Codierung und Informationstheorie	4
1.1	Codes und Codierung	6
1.2	Eigenschaften von Codes	7
1.3	Spezielle Codes	7
1.4	Informationsgehalt und Entropie	12
1.5	Optimalcodierung	15
1.6	Nachrichtenkanal und Leitungscodierung	19
2	Zahlendarstellung	22
2.1	Einfache Binärdarstellung	23
2.2	Darstellung negativer Zahlen	25
2.2.1	Addition im 1-Komplement	26
2.2.2	Addition im 2-Komplement	28
2.2.3	Multiplikation von Binärzahlen	29
2.3	Darstellung rationaler Zahlen	30
2.3.1	Festpunktdarstellung	30
2.3.2	Gleitpunktdarstellung	32
3	Logische Schaltungen	34
3.1	Aussagenlogik	34
3.1.1	Aussagenlogische Operationen	35
3.1.2	Aussagenlogische Grundbegriffe	37
3.1.3	Normalformen	38
3.1.4	Verknüpfungsbasen	41
3.2	Boolesche Algebra	42
3.3	Schaltnetze	44
3.3.1	Logische Gatter	46
3.3.2	Komposition von Schaltnetzen	47
3.3.3	Arithmetische Schaltnetze	51
3.4	Schaltungsminimierung	54
3.4.1	Gebietsdarstellung	54

3.4.2	Karnaugh-Diagramm	56
3.4.3	Das Verfahren von Quine-McCluskey	57
3.4.4	Graphisches Verfahren mit Karnaugh-Diagrammen	62
3.5	Schaltwerke	64
3.5.1	Zeitverhalten	65
3.5.2	Das Huffman-Modell	67
3.5.3	Das R - S -Flip-Flop	68
3.5.4	Das D -Flip-Flop	69
3.5.5	Das J - K -Flip-Flop	71
3.5.6	Parallelregister	73
3.5.7	Schieberegister	74
3.5.8	Binärzähler	74
4	Rechnerarchitektur	78
4.1	Eine kleine Computergeschichte	78
4.2	Die von Neumann-Architektur	80
4.3	Der Universalrechenautomat	82
4.4	Die Architektur des Befehlssatzes	84
4.4.1	Maschinenarchitektur	84
4.4.2	Speicheradressierung	88
4.4.3	Operationen des Befehlssatzes	91
4.4.4	Operandentypen	94
4.4.5	Befehlssatzcodierung	94
4.4.6	Befehlssatz und Compiler	95
4.5	Architekturperformance	96
5	Die DLX-Maschine	99
5.1	Der DLX-Befehlssatz	99
5.1.1	Register	99
5.1.2	Datentypen	100
5.1.3	Adressierungsmodi	100
5.1.4	Befehlsformat	100
5.1.5	Befehlsfunktionen	101
6	Pipelining	105
6.1	Einfache DLX-Implementierung	106
6.2	Die DLX-Pipeline	109
6.3	Pipeline Hazards	112
6.3.1	Structure Hazards	112
6.3.2	Data Hazards	113
6.3.3	Static Scheduling	118

6.3.4	Dynamic Scheduling	120
6.3.5	Control Hazards	126
6.3.6	Static Scheduling	128
6.3.7	Dynamic Branch Prediction	132
7	Interrupts	139
7.1	Exceptions in der DLX-Pipeline	142

Kapitel 5

Die DLX–Maschine

Mit den Überlegungen des vorigen Kapitels wollen wir nun einen konkreten Prozessor konstruieren. Die wesentlichen Designkriterien umfassen:

- Einfache *Load–Store*–Architektur
- Befehlssatz für effiziente Pipeline (Befehlscodes fester Länge)
- Befehlssatz für effiziente Compiler

Die sich daraus ergebende Maschine ist zwar eine hypothetische (nicht kommerzielle erhältliche), doch stellt sie einen repräsentativen Durchschnitt vieler realer Prozessoren dar. Versucht man dies (in nicht ganz ernster Weise) zu formalisieren, so ergibt sich die Gleichung (Hennessy and Patterson, 1996) (AMD 29K, DECstation 3100, HP 850, IBM 801, Intel i860, MIPS M/120A, MIPS M/1000, Motorola 88K, RISC I, SGI 4D/60, SPARCstation–1, Sun–4/110, Sun–4/260) / 13 = 560 = DLX.

Es werden also Elemente aller dieser Maschinen verwendet, um zur DLX (die Zahl 560 in römischen Ziffern) zu gelangen.

5.1 Der DLX–Befehlssatz

Wir wollen nun die Details der DLX–Maschine im einzelnen präsentieren.

5.1.1 Register

Für die DLX werden 32 32–Bit Register (*General Purpose Register*, GPR) vorgesehen, die mit R0...R31 bezeichnet werden. Zusätzlich existieren 32 32–Bit Floating Point Register (FPR), die auch im 64–Bit Modus angesprochen werden können (*Double Precision*). Die FPRs werden mit F0...F31

bezeichnet. Falls sie als Double verwendet werden, so können alle geraden Registernummer verwendet werden, wobei die ungeraden dann das LSWord der Zahl im Double-Format beinhalten. Eine spezielle Rolle übernimmt das Register R0, das *immer* den Wert 0 hat. Ein Laden dieses speziellen Registers bleibt ohne Wirkung. Mit dieser kleinen Besonderheit lassen sich einige wichtige Befehle sehr einfach auf andere zurückführen, was eine Einsparung von Befehlen bedeutet.

5.1.2 Datentypen

Es können alle gängigen Datentypen – Byte, Halbwort, Wort, Doppelwort – adressiert werden. Die Operationen der DLX arbeiten aber ausschliesslich mit Operanden mit 32-Bit (Integer oder Float) und 64-Bit (Double). Werden Bytes oder Halbworte geladen, so werden die unberührten Stellen des Registers abhängig vom speziellen Befehl entweder mit 0 aufgefüllt (*unsigned*), oder vorzeichenrichtig erweitert (*signed*).

5.1.3 Adressierungsmodi

Es werden nur die beiden Adressierungen Displacement und Immediate mit jeweils 16-Bit im Adressfeld vorgesehen. Zwei weitere Modi lassen sich daraus einfach ableiten. Eine indirekte Adressierung über ein Register entspricht einfach einem Displacement von 0. Eine absolute Adressierung (allerdings nur mit 16 Bit) entspricht einem Displacement relativ zum Register R0. Der DLX-Speicher ist 32-Bit byteadressierbar, ausgerichtet (aligned) und verwendet das Big Endian Format.

5.1.4 Befehlsformat

Alle Befehle werden mit einer festen Länge von 32-Bit codiert. Der Opcode ist immer 6 Bit lang. Da es nur zwei Adressierungsmodi gibt, können diese im Opcode codiert werden. Das Format der Adressfelder kann abhängig vom jeweiligen Befehl in drei Gruppen unterteilt werden: I-, R-, und J-Instruktionen. Das Befehlsformat einer I-Instruktion ist in Abb. 5.1 dargestellt.

6	5	5	16
Opcode	rs	rd	Immediate

Abbildung 5.1: Instruktionstyp I der DLX.

Das Adressfeld **rs** bezeichnet das Quellregister (*Source*) und das Adressfeld **rd** das Zielregister (*Destination*). Befehle aus dieser Gruppe umfassen: Loads und Stores (Immediate ist dann das Displacement), alle Immediate-Befehle ($\text{rd} \leftarrow \text{rs op immediate}$), Branches (**rd** bleibt unbenutzt, Immediate ist PC-relative Sprungadresse), und Jumps (mit **rd** = 0 und **immediate** = 0).

Abb. 5.2 zeigt das Befehlsformat einer R-Instruktion.

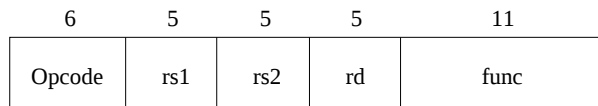


Abbildung 5.2: Instruktionstyp R der DLX.

Dieser Befehlstyp beinhaltet alle ALU Operationen, wobei die Operanden in den beiden Quellregistern **rs1** und **rs2** stehen, und das Resultat in **rd** abgelegt wird. In **func** wird die Operation spezifiziert, daher ist dieses Feld eigentlich Teil des Opcodes.

Das Format der J-Instruktionen zeigt Abb. 5.3.

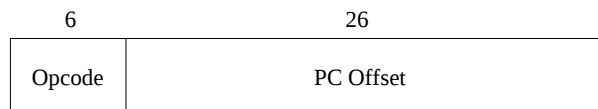


Abbildung 5.3: Instruktionstyp J der DLX.

Dieser Befehlstyp wird ausschliesslich für Jumps und Traps verwendet, wobei der Offset im Adressfeld zum PC addiert wird, um zur Sprungadresse zu gelangen. Bei einem *Jump and Link* wird zusätzlich die Rücksprungadresse in das Register *R31* geschrieben.

5.1.5 Befehlsfunktionen

Wir betrachten nun die einzelnen DLX-Befehle und geben einige Beispiele zu ihrer Verwendung an.

Datentransfer

LB, LBU, SB...lädt und speichert Bytes signed oder unsigned

LBU R1, 100(R2) ; hole Byte von R2 + 100 in R1

LH, LHU, SH...lädt und speichert Halbwörter signed oder unsigned

LH R3, 100(R1) ; hole Halbwort von R1 + 100 nach R3
 LW, SW...lädt und speichert Wörter
 SW R1, 100(R2) ; speichere Wort aus R1 nach R2 + 100
 LF, LD, SF, SD...lädt und speichert Floats und Doubles
 LD F2, 100(R2) ; hole Double von R2 + 100 nach F2 und F3
 MOVIS2S, MOVIS2I...transferiert Daten zwischen GPRs und Spezialregister
 MOVF, MOVD...transferiert Daten zwischen FPRs
 MOVF F3, F7 ; kopiere F7 nach F3
 MOVFP2I, MOVIS2FP...transferiert Daten zwischen FPRs und GPRs
 MOVFP2I R1, F2 ; kopiere F2 nach R1

Arithmetik und Logik

ADD, ADDI, ADDU, ADDUI...Addiert Register signed, unsigned, immediate
 ADD R3, R2, R1 ; R1 + R2 nach R3 (signed)
 SUB, SUBI, SUBU, SUBUI...Subtrahiert signed, unsigned, immediate
 SUBI R3, R2, #42 ; R2 - 42 nach R3 (signed)
 MULT, MULTU, DIV, DIVU...multipliziert/dividiert FPRs signed, unsigned
 MULT F3, F2, F1 ; F1 * F2 nach F3 (signed)
 AND, ANDI...Logisches UND (auch mit immediate)
 ANDI R3, R2, #16 ; Bit 4 von R2 nach R3
 OR, ORI, XOR, XORI...Logisches ODER, XOR (immediate)
 LHI...Lade oberes Halbwort immediate
 LHI R3, #12564 ; zur Generierung von 32-Bit Adressen
 SLL, SRL, SLLI, SRLI, SRA, SRAI...logisches, arithmetisches Schieben
 SLLI R3, #3 ; verschiebe R3 um 3 Stellen nach links, 0 von rechts
 SRA R3, R2 ; verschiebe R3 um R2 nach rechts, Vorzeichen von links
 S_, S_I...Prüfen einer Bedingung LT, GT, LE, GE, EQ, NE
 SEQ R3, R2, R1 ; setze R3, wenn R1 = R2
 SLTI R3, R2, #10 ; setze R3, wenn R2 <= 10

Steuerung (Control)

BEQZ, BNEQZ...Branch, wenn GPR gleich/ungleich 0

BNEQZ R3, loop ; springe nach Marke "loop", wenn R3 ungleich 0

BFPT, BFPF...Branch zufolge Compare Bit in FP Status Register

BFPT loop ; springe nach Marke "loop", wenn FPSR true

J, JR...Sprung absolut (26-Bit) oder Register

JR R1 ; springe nach Adresse in R1

JAL, JALR...Sprung, speichere PC+4 in R31 (link)

JAL proc ; Rufe Prozedur "proc", Rücksprungadresse in R31

TRAP...Springe auf systemabhängige Stelle (Betriebssystem)

RFE...Rücksprung aus Exception (Trap)

Floating Point

ADDF, ADDD...addiere Floats, Doubles

ADDD F8, F4, F6 ; addiere Doubles in F4 und F6 nach F8

SUBF, SUBD...subtrahiere Floats, Doubles

MULTF, MULTD...multipliziere Floats, Doubles

DIVF, DIVD...dividiere Floats, Doubles

CVT_x2_y...konvertiere Integer, Float, Double (x != y)

CVTF2I F2, F3 ; wandle Float in F3 zu Integer in F2

__F, __D...Vergleiche Floats, Doubles LT, GT, LE, GE, EQ, NE

GED F4, F6 ; F4 >= F6? wenn ja, setze Compare Bit in FPSR

Besonderheiten

Die Multiplikation und Division von Zahlen wird ausnahmslos von der Floating Point Unit durchgeführt. Die Zahlen müssen daher in FPRs stehen.

Die Rücksprungadresse eines Prozeduraufrufs wird mit dem Befehl `JAL` immer in `R31` abgelegt.

Zwei wichtige Operationen sind nicht explizit im Befehlssatz der DLX: Laden einer Konstanten in ein Register und das Verschieben eines Datums von einem GRP in ein anderes (*Register Move*). Beide Befehle können mit dem Nullregister `R0` synthetisiert werden.

```
ADDI R1, R0, #5    ; lade R1 mit 5
ADD R1, R0, R2      ; kopiere R2 nach R1
```

Selbst auf einer RISC-Architektur werden abhängig vom verwendeten Compiler meist nur eine kleine Untermenge von Befehlen verwendet. Benchmarkmessungen von Integer-Programmen ergaben für die DLX-Maschine, dass 80% aller Befehle von nur fünf Befehlen abgedeckt wird: Load (26%), Branch (17%), Add (14%), Compare (14 %), Store (9%) (Hennessy and Patterson, 1996).

Literaturverzeichnis

- Ash, R. (1965). *Information Theory*. Wiley, New York, 1st edition.
- Broy, M. (1993). *Rechnerstrukturen und maschinennahe Programmierung*. Informatik, Eine grundlegende Einführung, Teil II. Springer, Berlin.
- Carlson, A. B. (1975). *Communication Systems*. McGraw–Hill, Tokyo, 2nd edition.
- Dworatschek, S. (1970). *Schaltalgebra und digitale Grundsaltungen*. de Gruyter, Berlin.
- Hennessy, J. L. and Patterson, D. A. (1996). *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, San Francisco, 2nd edition.
- Huffman, D. A. (1952). A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE*, 40(10):1098–1101.
- INFOTEC (1998). <http://www.infotec.org/history.htm>. WWW Repository, Association of Information Technology Professionals.
- Küpfmüller, K. (1954). Die Entropie der deutschen Sprache. *FTZ*, 7(6):265–272.
- Lochmann, D. (1995). *Digitale Nachrichtentechnik*. Verlag Technik, Berlin, 1st edition.
- Mendelson, E. (1982). *Boolesche Algebra und Logische Schaltungen*. Schaum's Outline. McGraw–Hill, Hamburg.
- Newald and Lindner (1985). Digitale Schaltwerke. Laborunterlagen, Institut für Datenverarbeitung, Technische Universität Wien.
- Volkert, J. (1999). Digitale Rechenanlagen. Vorlesungsunterlagen, Institut für Technische Informatik und Telematik, Universität Linz.