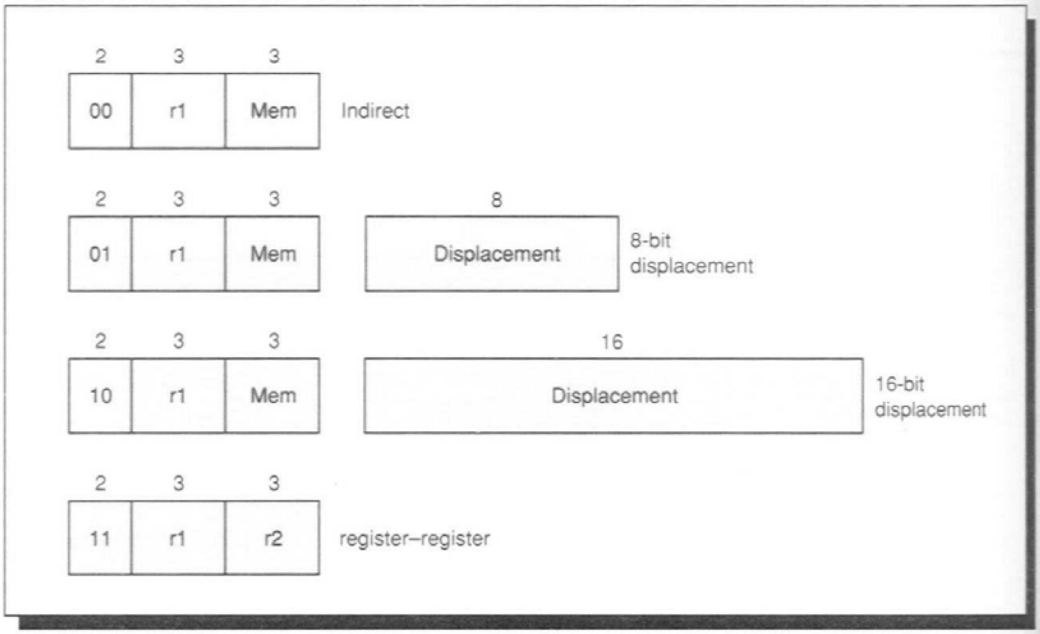

FIGURE 4.14 There are four postbyte encodings on the 8086 designated by a 2-bit tag. The first three indicate a register–memory instruction, where Mem is the base register. The fourth form is register–register.

4.5 The DLX Architecture

In many places throughout this book we will have occasion to refer to a computer's "machine language." The machine we use is a mythical computer called "MIX." MIX is very much like nearly every computer in existence, except that is, perhaps, nicer ... MIX is the world's first polyunsaturated computer. Like most machines, it has an identifying number—the 1009. This number was found by taking 16 actual computers which are very similar to MIX and on which MIX can be easily simulated, then averaging their number with equal weight:

$$\lfloor (360 + 650 + 709 + 7070 + U3 + SS80 + 1107 + 1604 + G20 + B220 + S2000 + 920 + 601 + H800 + PDP\text{-}4 + 11)/16 \rfloor = 1009.$$

The same number may be obtained in a simpler way by taking Roman numerals.

Donald Knuth, *The Art of Computer Programming. Volume I: Fundamental Algorithms*

In this section we will describe a simple load/store architecture called DLX (pronounced "Deluxe"). The authors believe DLX to be the world's second polyunsaturated computer—the average of a number of recent experimental and commercial machines that are very similar in philosophy to DLX. Like Knuth, we derived the name of our machine from an average expressed in Roman numerals:

(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.

The architecture of DLX was chosen based on observations about the most frequently used primitives in programs. More sophisticated (and less performance-critical) functions are implemented in software with multiple instructions. In Section 4.9 we discuss how and why these architectures became popular.

Like most recent load/store machines, DLX emphasizes

- A simple load/store instruction set
- Design for pipelining efficiency (discussed in Chapter 6)
- An easily decoded instruction set
- Efficiency as a compiler target

DLX provides a good architectural model for study, not only because of the recent popularity of this type of machine, but also because it is an easy architecture to understand.

DLX—Our Generic Load/Store Architecture

In this section, the DLX instruction set is defined. We will use this architecture again in Chapters 5 through 7, and it forms the basis for a number of exercises and programming projects.

- The architecture has thirty-two 32-bit general-purpose registers (GPRs); the value of R0 is always 0. Additionally, there are a set of floating-point registers (FPRs), which can be used as 32 single-precision (32-bit) registers, or as even-odd pairs holding double-precision values. Thus, the 64-bit floating-point registers are named F0, F2, ..., F28, F30. Both single- and double-precision operations are provided. There are a set of special registers used for accessing status information. The FP status register is used for both compares and FP exceptions. All movement to/from the status register is through the GPRs; there is a branch that tests the comparison bit in the FP status register.

- Memory is byte addressable in Big Endian mode with a 32-bit address. All memory references are through loads or stores between memory and either the GPRs or the FPRs. Accesses involving the GPRs can be to a byte, to a halfword, or to a word. The FPRs may be loaded and stored with single-precision or double-precision words (using a pair of registers for DP). All memory accesses must be aligned. There are also instructions for moving between a FPR and a GPR.
- All instructions are 32 bits and must be aligned.
- There are also a few special registers that can be transferred to and from the integer registers. An example is the floating-point status register, used to hold information about the results of floating-point operations.

Operations

There are four classes of instructions: loads and stores, ALU operations, branches and jumps, and floating-point operations.

Example instruction	Instruction name	Meaning
LW R1,30(R2)	Load word	$Rl \leftarrow_{32} M[30+R2]$
LW R1,1000(R0)	Load word	$Rl \leftarrow_{32} M[1000+0]$
LB R1,40(R3)	Load byte	$Rl \leftarrow_{32} (M[40+R3]_0)^{24}$ ## $M[40+R3]$
LBU R1,40(R3)	Load byte unsigned	$Rl \leftarrow_{32} 0^{24}$ ## $M[40+R3]$
LH R1,40(R3)	Load halfword	$Rl \leftarrow_{32} (M[40+R3]_0)^{16}$ ##$M[40+R3]$##$M[41+R3]$
LHU R1,40(R3)	Load halfword unsigned	$Rl \leftarrow_{32} 0^{16}$ ##$M[40+R3]$##$M[41+R3]$
LF F0,50(R3)	Load float	$F0 \leftarrow_{32} M[50+R3]$
LD F0,50(R2)	Load double	$F0$##$F1 \leftarrow_{64} M[50+R2]$
SW 500(R4),R3	Store word	$M[500+R4] \leftarrow_{32} R3$
SF 40(R3),F0	Store float	$M[40+R3] \leftarrow_{32} F0$
SD 40(R3),F0	Store double	$M[40+R3] \leftarrow_{32} F0$; $M[44+R3] \leftarrow_{32} F1$
SH 502(R2),R3	Store half	$M[502+R2] \leftarrow_{16} R3_{16..31}$
SB 41(R3),R2	Store byte	$M[41+R3] \leftarrow_{8} R2_{24..31}$

FIGURE 4.15 The load and store instructions in DLX. All use a single addressing mode and require that the memory value be aligned. Of course, both loads and stores are available for all the data types shown.

Any of the general-purpose or floating-point registers may be loaded or stored, except that loading R0 has no effect. There is a single addressing mode, base register + 16-bit signed offset. Halfword and byte loads place the loaded object in the lower portion of the register. The upper portion of the register is filled with either the sign extension of the loaded value or zeros, depending on the opcode. Single-precision floating-point numbers occupy a single floating-point register, while double-precision values occupy a pair. Conversions between single and double precision must be done explicitly. The floating-point format is IEEE 754 (see Appendix A). Figure 4.15 gives an example of the load and store instructions. A complete list of the instructions appears in Figure 4.18 (page 165).

All ALU instructions are register–register instructions. The operations include simple arithmetic and logical operations: add, subtract, AND, OR, XOR, and shifts. Immediate forms of all these instructions, with a 16-bit sign-extended immediate, are provided. The operation `LHI` (load high immediate) loads the top half of a register, while setting the lower half to 0. This allows a full 32-bit constant to be built in two instructions. (We sometimes use the mnemonic `LI`, standing for Load Immediate, as an abbreviation for an add immediate where one of the source operands is R0; likewise, the mnemonic `MOV` is sometimes used for an `ADD` where one of the sources is R0.)

There are also compare instructions, which compare two registers ($=,\neq,<,>,\leq,\geq$). If the condition is true, these instructions place a 1 in the destination register (to represent true); otherwise they place the value 0. Because these operations "set" a register they are called set-equal, set-not-equal, set-less-than, and so on. There are also immediate forms of these compares. Figure 4.16 gives some examples of the arithmetic/logical instructions.

Control is handled through a set of jumps and a set of branches. The four jump instructions are differentiated by the two ways to specify the destination address and by whether or not a link is made. Two jumps use a 26-bit signed offset added to the program counter (of the instruction sequentially following the jump) to determine the destination address; the other two jump instructions specify a register that contains the destination address. There are two flavors of jumps: plain jump, and jump and link (used for procedure calls). The latter places the return address in R31.

Example instruction	Instruction name	Meaning
`ADD R1,R2,R3`	Add	R1←R2+R3
`ADDI R1,R2,#3`	Add immediate	R1←R2+3
`LHI R1,#42`	Load high immediate	$R1 \leftarrow 42\#\#0^{16}$
`SLLI R1,R2,#5`	Shift left logical	R1←R2<<5
`SLT R1,R2,R3`	Set less than	if (R2<R3) R1←1 else R1←0

FIGURE 4.16 Examples of arithmetic/logical instructions on DLX, both with and without immediates.

Example instruction	Instruction name	Meaning
J name	Jump	PC←name; ((PC+4)-2^{25}) ≤ name < ((PC+4)+2^{25})
JAL name	Jump and link	R31←PC+4; PC←name; ((PC+4)-2^{25}) ≤ name < ((PC+4)+2^{25})
JALR R2	Jump and link register	R31←PC+4; PC←R2
JR R3	Jump register	PC←R3
BEQZ R4,name	Branch equal zero	if (R4==0) PC←name; ((PC+4)-2^{15}) ≤ name < ((PC+4)+2^{15})
BNEZ R4,name	Branch not equal zero	if (R4!=0) PC←name; ((PC+4)-2^{15}) ≤ name < ((PC+4)+2^{15})

FIGURE 4.17 Typical control-flow instructions in DLX. All control instructions, except jumps to an address in a register, are PC-relative. If the register operand is R0, the branch is unconditional, but the compiler will usually prefer to use a jump with a longer offset over this "unconditional branch."

All branches are conditional. The branch condition is specified by the instruction, which may test the register source for zero or nonzero; this may be a data value or the result of a compare. The branch target address is specified with a 16-bit signed offset that is added to the program counter. Figure 4.17 gives some typical branch and jump instructions.

Floating-point instructions manipulate the floating-point registers and indicate whether the operation to be performed is single or double precision. Single-precision operations can be applied to any of the registers, while double-precision operations apply only to an even-odd pair (e.g., F4, F5), which is designated by the even register number. Load and store instructions for the floating-point registers move data between the floating-point registers and memory both in single and double precision. The operations `MOVF` and `MOVD` copy a single-precision (`MOVF`) or double-precision (`MOVD`) floating-point register to another register of the same type. The operations `MOVFP2I` and `MOVI2FP` move data between a single floating-point register and an integer register; moving a double-precision value to two integer registers require two instructions. Integer multiply and divide that work on 32-bit floating-point registers are also provided, as are conversions from integer to floating point and vice versa.

The floating-point operations are add, subtract, multiply, and divide; a suffix `D` is used for double precision and a suffix `F` is used for single precision (e.g., `ADDD`, `ADDF`, `SUBD`, `SUBF`, `MULTD`, `MULTF`, `DIVD`, `DIVF`). Floating-point compares set a bit in the special floating-point status register that can be tested with a pair of branches: `BFPT` and `BFPF`, branch floating point true and branch floating point false.

Figure 4.18 contains a list of all operations and their meaning.

Instruction type / opcode	Instruction meaning
Data transfers	**Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR**
LB,LBU,SB	Load byte, load byte unsigned, store byte
LH,LHU,SH	Load halfword, load halfword unsigned, store halfword
LW,SW	Load word, store word (to/from integer registers)
LF,LD,SF,SD	Load SP float, load DP float, store SP float, store DP float
MOVI2S, MOVS2I	Move from/to GPR to/from a special register
MOVF, MOVD	Copy one floating-point register or a DP pair to another register or pair
MOVFP2I,MOVI2FP	Move 32 bits from/to FP registers to/from integer registers
Arithmetic / Logical	**Operations on integer or logical data in GPRs; signed arithmetics trap on overflow**
ADD,ADDI,ADDU,ADDUI	Add, add immediate (all immediates are 16 bits); signed and unsigned
SUB,SUBI,SUBU,SUBUI	Subtract, subtract immediate; signed and unsigned
MULT,MULTU,DIV,DIVU	Multiply and divide, signed and unsigned; operands must be floating-point registers; all operations take and yield 32-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LHI	Load high immediate—loads upper half of register with immediate
SLL, SRL, SRA, SLLI, SRLI, SRAI	Shifts: both immediate (S__I) and variable form (S__); shifts are shift left logical, right logical, right arithmetic
S__,S__I	Set conditional: "__" may be LT,GT,LE,GE,EQ,NE
Control	**Conditional branches and jumps; PC-relative or through register**
BEQZ,BNEZ	Branch GPR equal/not equal to zero; 16-bit offset from PC+4
BFPT,BFPF	Test comparison bit in the FP status register and branch; 16-bit offset from PC+4
J, JR	Jumps: 26-bit offset from PC (J) or target in register (JR)
JAL, JALR	Jump and link: save PC+4 to R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address; see Chapter 5
RFE	Return to user code from an exception; restore user mode; see Chapter 5
Floating point	**Floating-point operations on DP and SP formats**
ADDD,ADDF	Add DP, SP numbers
SUBD,SUBF	Subtract DP, SP numbers
MULTD,MULTF	Multiply DP, SP floating point
DIVD,DIVF	Divide DP, SP floating point
CVTF2D, CVTF2I, CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convert instructions: CVTx2y converts from type x to type y, where x and y are one of I (Integer), D (Double precision), or F (Single precision). Both operands are in the FP registers
___D,___F	DP and SP compares: "__" may be LT,GT,LE,GE,EQ,NE; sets comparison bit in FP status register

FIGURE 4.18 Complete list of the instructions in DLX. The formats of these instructions are shown in Figure 4.19. This list can also be found in the back inside cover.

Instruction Format

All instructions are 32 bits with a 6-bit primary opcode. Figure 4.19 shows the instruction layout.

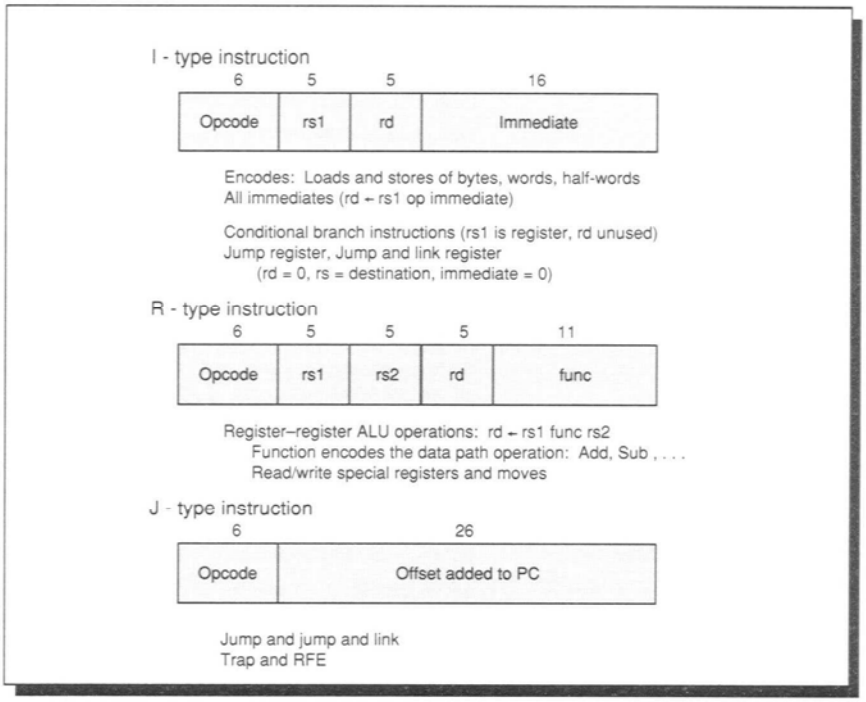

FIGURE 4.19 Instruction layout for DLX. All instructions are encoded in one of three types.

Machines Related to DLX

Between 1985 and 1990 many load/store machines were announced that are similar to DLX. Figure 4.20 describes the major features of these machines. All have 32-bit instructions and are load/store architectures; the figure lists their differences. These machines are all very similar—if you're not convinced, try making a table such as this one comparing these machines to the VAX or 8086.

DLX bears a close resemblance to all the other load/store machines shown in Figure 4.20. (See Appendix E for a detailed description of four load/store machines closely related to DLX.) Thus, the measurements in the next section will be reasonable approximations of the behavior of any of the machines. In fact, some studies suggest that compiler differences are more significant than architectural differences among these machines.