

DLX ARCHITECTURE

The architecture of DLX was chosen based on observations about most frequently used primitives in programs. 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 easy to understand. Like most recent load/store machines, DLX emphasizes

- A simple load/store
- Design for pipelining efficiency
- An easily decoded instruction set
- Efficiency as a compiler target

Registers for DLX

- Thirty-two 32-bit general purpose registers (GPRs), named R0, R1, ..., R31. The value of R0 is always 0.
- Thirty-two floating-point registers (FPRs), which can be used as :
32 single precision (32-bit) registers or
Even-odd pairs holding double-precision values. Thus, the 64-bit FPRs are named F0,F2,...,F30
- A few special registers can be transferred to and from the integer registers.

Data types for DLX

- For integer data
 - 8-bit bytes
 - 16-bit half words
 - 32-bit words
- For floating point
 - 32-bit single precision
 - 64-bit double precision

The DLX operations work on 32-bit integers and 32- or 64-bit floating point. Bytes and half words are loaded into registers with either zeros or the sign bit replicated to fill the 32 bits of the registers.

Memory

- Byte addressable
- Big Endian mode
- 32-bit address
- Two addressing modes (immediate and displacement). Register deferred and absolute addressing with 16-bit field are accomplished using R0.
- Memory references are load/store between memory and GPRs or FPRs
- Access to GPRs can be to a byte, to a halfword, or to a word
- All memory accesses must be aligned

- There are instructions for moving between a FPR and a GPR

Instructions

- Instruction layout for DLX
- Complete list of instructions in DLX
- 32 bits(fixed)
- Must be aligned

Operations

There are four classes of instructions:

Load/Store

Any of the GPRs or FPRs may be loaded and stored except that loading R0 has no effect.

ALU Operations

All ALU instructions are register-register instructions.

The operations are :

- add
- subtract
- AND
- OR
- XOR
- shifts

Compare instructions compare two registers (=,!=,<,>,<=,>=).

If the condition is true, these instructions place a 1 in the destination register, otherwise they place a 0.

Branches/Jumps

All branches are conditional. The branch condition is specified by the instruction, which may test the register source for zero or nonzero.

Floating-Point Operations

- add
- subtract
- multiply
- divide

DLX INSTRUCTION SET

Complete list of the instructions in DLX.	
Instruction type/opcode	Instruction meaning
Data transfers	Move data between registers and memory, or between the integer and FP or special register; 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 (SP - single precision, DP - double precision)
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 register 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, XOP, XOPI	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
BFPT, BPPF	Test comparison bit in the FP status register and branch; 16-bit offset from PC
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

RFE	Return to user code from an exception; restore user code
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: CVT _x 2 _y converts from type <i>x</i> to type <i>y</i> , where <i>x</i> and <i>y</i> 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; set comparison bit in FP status register.

DLX INSTRUCTION FORMAT

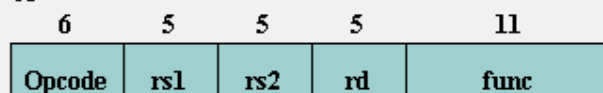
I - type instruction



Encodes: Loads and stores of bytes, words, half-words
All immediates (rd <- rs1 op immediate)

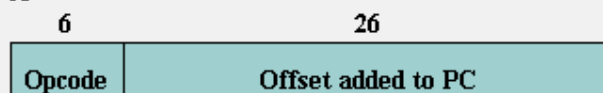
Conditional branch instructions (rs1 is register, rd unused)
Jump register, Jump and link register
 (rd = 0, rs = destination, immediate = 0)

R - type instruction



Register - register ALU operations: rd <- rs1 func rs2
Function encodes the data path operation: Add, Sub,...
Read/Write special registers and moves

J - type instruction



Jump and Jump and link
Trap and RFE

EXAMPLES OF INSTRUCTIONS ON DLX

To understand these tables we need to introduce notations of the description language.

- A subscript is appended to the symbol <- whenever the length of the datum being transferred might not be clear. Thus, <- n mean transfer an n-bit quantity.
- A subscript is used to indicate selection of a bit from a field. Bits are labeled from the most-significant bit starting at 0. The subscript may be a single digit (e.g. Regs[R4] 0 yields the sign bit of R4) or a subrange (e.g. Regs[R3] 24..31 yields the least-significant byte of R3).
- The variable Mem, used as an array that stands for main memory, is indexed by a byte address and may transfer any number of bytes.
- A superscript is used to replicate a fields (e.g. 0 24 yields a fiels of zeros of length 24 bits).
- The symbol ## is used to concatenate two fields and may appear on either side of a data transfer.

Examples of arithmetic/logical instructions on DLX		
Example instruction	Instruction name	Meaning
ADD R1, R2, R3	Add	Regs[R1] <- Regs[R2]+Regs[R3]
ADDI R1, R2, #3	Add immediate	Regs[R1] <- Regs[R2] + 3
LHI R1, #42	Load high immediate	Regs[R1] <- 42##0 ¹⁶
SLLI R1, R2, #5	Shift left logical immediate	Regs[R1] <- Regs[R2] << 5
SLT R1, R2, R3	Set less than	if (Regs[R2]<Regs[R3]) Regs[R1] <- 1 else Regs[R1] <- 0

Typical control-flow instructions in DLX		
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	Regs[R31]<-PC+4; PC , Regs[R2]
JR R3	Jump register	PC <- Regs[R3]
BEQZ R4, name	Branch equal zero	if (Regs[R4]==0) PC<-name; ((PC+4)-2 ¹⁵)<=name<((PC+4)+2 ¹⁵)
BNEZ R4, name	Branch not equal zero	if (Regs[R4]!=0) PC<-name; ((PC+4)-2 ¹⁵)<=name<((PC+4)+2 ¹⁵)

The load and store instructions in DLX

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

BIBLIOGRAPHY

GURPUR PRABHU - COMPUTER ARCHITECTURE TUTORIAL