COMPUTER ARCHITECTURE LAB

ESSENTIALS OF DLX INSTRUCTION SET

General description

The DLX processor is a RISC architecture for educational purposes whose name is obtained by averaging the denominations of different commercial and experimental processors, expressed in Roman numerals (DLX = 560). This procedure mimics what Knuth used for the MIX processor, explained in the classic book The Art of Computer Programming and the parents of the DLX honor this author by applying the naming procedure in a similar way to this processor.

This document it is an excerpt where we only show the essential instructions for doing the designed practices. In **bold** the specific instructions needed for the activities in the course.

The most important features of DLX are:

1. It uses 32 General Purpose Registers (GPR) for 32-bit integers: R0 to R31. The word is, therefore, 32 bits (4 bytes). R0 is always 0 and does not support writting any value to it.

2. It has 32 floating point registers (FP per Floating Point) called F0 to F31 32 bits. These can work as individual registers of Simple Precision (SP) or in pairs of Double Precision (64 bits, DP) in which case they are referenced with the even numbers of the registers (F0, F2, F4 ... F30). The floating point representation format is IEEE754.

3. Its memory is byte-addressabled, in BIG ENDIAN mode. Addresses are 32 bits long (address bus), and all instructions occupy the same size (one word) and they are all aligned to word boundary.

Data Transfer Instructions (LOAD STORE)

These instructions move data between a Register (GPR) and a Memory location where the memory address is computed adding a GPR register + immediate 16-bit offset (Addressing mode “Register relative” or “Displacement”

LW	**SW**	Load Word / Store Word (from/to integer GPRs) (word = integer)
LD	**SD**	Load Floating Point / Store Floating Point (from/to FP GPRs) in Double Precision

Processing Instructions (integers)

Logical and Arithmetic operations (except multiplication and division) with integer or logical data in GPR registers. In signed operations, in case of overflow, a TRAP is generated. Operands are GPRs or one GPR and an immediate (constant) operand. Destination is always a GPR. Immediate data is 16 bits long.

ADD / ADDI	Integer addition of two GPRs / a GPR and immediate data
SUB / SUBI	Integer subtraction of two GPRs / a GPR and immediate data
AND / OR / **XOR** **ANDI** / ORI / XORI	AND OR y XOR between GPRs AND OR y XOR with Immediate
SLL, SRL, **SLLI**, SRLI, SRA SRAI	Logical Shift to the Left (SLL) or Right (SRL), GPRs only Logical Shift to the Left (SLL) or Right (SRL), with immediate Arithmetic shifts to Right with GPR /Immediate

Multiplying and Division (integers)

Require FP registers. In case of overflow a TRAP is generated.

MULT, DIV	Multiply or Divide with and without sign. Operands must be in FP registers; all operations use 32 bit values

Double Precision Floating Point instructions (FP)

ADDD, SUBD	Add, Subtract in DP
MULTD, DIVD	Multiply, Divide in DP
CVTD2I, CVTI2D	Conversion: CVTx2y converts type x to type y. I=integer D=DP
ltD, gtD,leD.eqD.neD	Compares registers and sets compare bit in status FP register.

Control flow Instructions

Jump and Branch instructions. Condictonal and Unconditional. Effective address obtaind using mode Relative to PC with offset in immediate mode, or Register Direct

BEQZ, BNEZ	Conditional Branch if named GPR equals or not equals Zero; max. 16 bit offset from PC+4
J, JR	Jump (unconditional): 26 bits offset from PC (J) or destination in Register (JR)
TRAP	Transfer to OS's vector address
BFPT, BFPF	Test (True/False) for compare bit in status FP register. and jump to address PC+4 plus 16 bits offset

Examples

Notation:

- Bit numbering in registers follows BIG ENDIAN model:
- bit 0 is most significant left bit or sign bit
- bit 31 is least significant rightmost bit
- ## menas concatenation
- $\leftarrow_{32}$ stands for 32 bit transfers
- $(R3_0)^{24}$ replicates 24 times R3 bit 0

Instrucción	***Descripción***	***Acción***
LW R1,30(R2)	Load word	R1 $\leftarrow_{32}$ M[30+R2] (transferencia de 32 bits)
LW R1,100(R0)	Load word	R1 $\leftarrow_{32}$ M[100+0]
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
SD 40(R3),F0	Store floating DP	M[40+R3] $\leftarrow_{32}$ F0; M[44+R3] $\leftarrow_{32}$ F1
J dest	Jump	PC $\leftarrow$ dest; ((PC+4)-2^{15})$\leq$dest< ((PC+4)+2^{15})
JR R31	Jump to register	PC $\leftarrow$ R31 (may implement "return")
BEQZ R4, dest	Branch if equal to 0	If (R4==0) PC $\leftarrow$ dest; ((PC+4)-2^{15})$\leq$ dest < ((PC+4)+2^{15})
BNEZ R4, dest	Branch if not equal to 0	If (R4!=0) PC $\leftarrow$ dest; ((PC+4)-2^{15})$\leq$ dest < ((PC+4)+2^{15})
ADD R1,R2,R3	Add	R1 $\leftarrow$ R2 + R3
ADDI R1,R2,#3	Add with immediat	R1 $\leftarrow$ R2 + 3
SLLI R1,R2,#5	Logical left shift	R1 $\leftarrow$ R2 << 5

Tips

If you notice, multiplication of integers implies the use of floating point registers. Very often, we can multiply by a fraction of the time taken by this instruction if we use the left shift operation instead. Multiplying by 2 is nothing more than moving the operand one place to the left: **slli R7, R7,#1**

In a loop implementation it is usually more advantageous to maintain a descending counter, then decrement it and check if it is not equal to 0 to branch to the beginning of the loop again or to exit the loop and continue with the next instruction after the branch.

To read data from a list, it is usual to use a register as an index which, added to the starting address of the first element, allows the access to subsequent elements. This is used inside a loop

and the register is incremented (or decremented) by the size of the data (ex: 4 for integers) as an index in order to access one element of the list in each iteration.

It is possible to combine the loop counter and index function using a single register. Its value must be 4 times that of the initial counter and the decrease of 4 in 4 taking into account that the first element of the list will be accessed from the last one (in the case of integers, 8 for double etc.).