The DLX Architecture

CS448

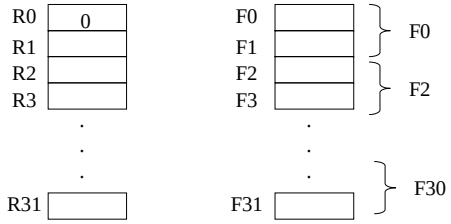
Chapter 2

DLX (Deluxe)

- Pedagogical “world’s second polyunsatured computer” via load-store architecture
- Goals
  - Optimize for the common case
    - Less common cases via software
    - Provide primitives
  - Simple load-store instruction set
    - Entire instruction set fits on a page
  - Efficient pipeline via fixed instruction set encoding
  - Compiler efficiency
    - Lots of general purpose registers

DLX Registers

- 32 GPRs, can be used for int, float, double
- 32 bits for R0..R31, F0..F31. 64 bits for F0,F2…
- Extra status register
- R0 always 0
  - Loads to R0 have no effect



DLX Data Types

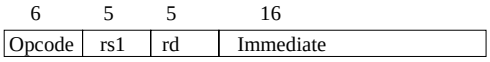
- 32 bit words
- Byte-addressable memory
- 16-bit “half words” also addressable
- 32 bit floats – single precision
- 64 bit floats – double precision
  - Use IEEE 754 format for SP and FP
- Loaded bytes/half-bytes are sign-extended to fill all 32 bits of the register
- Note big-endian format will be used

DLX Addressing

- Support for Displacement, Immediate ONLY
  - Recall previous discussion, these are the most commonly used modes
  - Other modes can be accomplished through these types of addressing with a bit of extra work
    - Absolute: Use R0 as base
    - Indirect: Use 0 as the displacement value
- All memory addresses are aligned

DLX Instruction Format

- All instructions 32 bits, two addressing modes
- I-Type



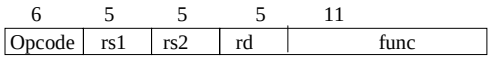
Loads & Stores
rd ← rs op immediate

Conditional Branches
rs1 is the condition register checked, rd unused, immediate is offset

JR, JALR (Jump Register, Jump and link Register)
rs1 holds the destination address, rd & immediate = 0 (unused)

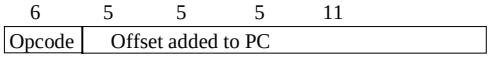
DLX Instruction Format Cont'd

- R-Type Instruction



Register-To-Register operations
All non-immediate ALU operations R-to-R only
rd ← rs1 func rs2

- J-Type Instruction



Jump and Jump and Link
Trap and return from exception

DLX Move Instructions

- LB, LBU, SB - load byte, load byte unsigned, store byte
- LH, LHU, SH - same as above but with halfwords
- LW, SW - load or store word
- LF, SF – load or store single precision float via F Regs
- LD, SD – load or store double precision float via FD Regs
- MOVI2S - move from GPR to a special register
- MOVS2I - move from special register to a GPR
- MOVFP2I - move 32- bits from an FPR to a GPR
- MOVI2FP - move 32- bits from a GPR to an FPR

- How could we move data to/from the D Registers?

Instruction Format and Notation

- LW R1, 30(R2) Load Word
  - Regs[R1]$\leftarrow_{32}$ Mem[30+Regs[R2]]
    - Transfer 32 bits at address added to Mem Loc 30
  - What do we get if we use R0?
- SW R3, 500(R4) Store Word
  - Mem[500 + Regs[R4]] $\leftarrow_{32}$ Regs[R3]
- LB R1, 40(R3) Load Byte
  - Regs[R1]$\leftarrow_{32}$ (Mem[40+Regs[R3]]$_0$)^{24} ## Mem[40+Regs[R3]]
    - Subscript 0 is MSB (Remember Big Endian!)
    - 24 is to replicate value for 24 bits (Sign extends first bit of the byte)
    - ## is concatenation

More Move Examples

- LBU R1, 40(R3) Load Byte Unsigned
  - Regs[R1]$\leftarrow_{32}$ 0^{24} ## (Mem[40+Regs[R3])
- LH R1, 40(R3) Load Half word
  - Regs[R1]$\leftarrow_{32}$ (Mem[40+Regs[R3]]$_0$)^{16} ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]]
    - Sign extend 16 bit quantity, get next 16 bits in two byte chunks
    - Note that MEM can reference byte, word, etc.
- SF 40(R3), F0 Store Float
  - M[40+ R3] $\leftarrow_{32}$ F0
    - Can store values using addressing modes too

And More Move Examples

- LD F0, 50(R3) Load Double
  - Regs[F0] ## Regs[F1] $\leftarrow_{64}$ Mem[50+Regs[R3]]
  - Must use F0, F2, F4, etc.
- SW 500(R4), F0 Store Double
  - Mem[500 + Regs[R4]] $\leftarrow_{32}$ Regs[F0]
  - Mem[504 + Regs[R4]] $\leftarrow_{32}$ Regs[F1]
  - Note the book has the 500(R4) reversed with F0; WinDLX requires it in the direction shown here
  - Will normally use labels in a data segment:

```
            .data
            .align      4       ; Align memory
Storage:    .space      4

            SW Storage(R0), F0
```

Move Examples

- MovI2FP f2, r3 Move Int to FP
  - Regs[F2] $\leftarrow$ Regs[R3]
  - No value conversion performed, just copy bits
- MovFP2I r5, f0 Move FP to Int
  - Regs[R5] $\leftarrow$ Regs[F0]

ALU Instructions

- Add, subtract, AND, OR, XOR, Shifts, Add, Subtract, Multiply, Divide
- Integer Arithmetic
  - ADD, ADDI, ADDU, ADDUI
    - Add, Add Immediate, Add Unsigned, Add Unsigned Immediate
  - SUB, SUBI, SUBU, SUBUI
    - Subtract, Subtract Immediate, Subtract Unsigned, Subtract Immediate Unsigned
  - MULT, MULTU, DIV, DIVU
    - Multiply and Divide for signed, unsigned.
    - Book: Operands must be in FP registers
    - WinDLX: Operands must be in R registers

ALU Integer Arithmetic Examples

- ADD R1, R2, R3
  - Regs[R1] ← Regs[R2] + Regs[R3]
- ADD R1, R2, R0
  - Result?
- ADDI R1, R2, #0xFF
  - Regs[R1] ← Regs[R2] + 0xFF
- MULT R5, R2, R1
  - Regs[R1] ← Regs[R2] * Regs[R1]

Other Integer ALU Instructions

- Logical
  - AND, ANDI, OR, ORI, XOR, XORI
  - Operate on register or immediate
- LHI Load High Immediate
  - loads upper half of register with immediate value
  - Note a full 32- bit immediate constant will take 2 instructions
- Shifts
  - SLLL, SRL, SRA, SLLI, SRLI, SRAI
  - Shift left/right logical, arithmetic, for immediate or register

Other Integer ALU Instructions

- Set Conditional Codes
  - S__, S__I
- Sets a register to hold some condition
- __ may equal LT, GT, LE, GE, EQ, NE
- Puts 1 or 0 in destination register
- I for immediate, no I for register as operaand
  - E.g. SLTI R1, R2, #55 ; Sets R1 if R2 < 55
  - E.g. SEQ R1, R2, R3 ; Sets R1 if R2 = R3
- Convenience of any register can hold condition codes
- Used for branches; test if zero or nonzero

DLX Control

- Jump and Branch
  - Jump is unconditional, branch is conditional. Relative to PC.
- J label
  - Jump to PC+ 4 + 26 bit offset
- JAL label
  - Jump and Link to label, save return address: Regs[31]←PC+4
  - See any potential problems here?
- JALR Reg
  - Jump and Link to address stored in Reg, save PC+4
- BEQZ Reg, label BNEZ Reg, label
  - Branch to label if Regs[REG]==0, otherwise no branch
  - Branch to label if Regs[REG]!=0, otherwise no branch
- Trap, RFE – will see later (invoke OS, return from exception)

DLX Floating Point

- Arithmetic Operations
  - ADDD, ADDF Dest, Src1, Src2
  - SUBD, SUBF
  - MULTD, MULTF, DIVD, DIVF
    - Add, subtract, multiply, or divide DP (D) or SP (F) numbers
    - All operands must be registers
- Conversion
  - CVTF2D, CVTF2I, DVTD2F, CVT2DI, CVTI2F, CVTI2D take Dest, Source registers
    - Converts types, I=Int, F=Float, D=Double
- Comparison
  - __D, __F Src Register 1, Src Register 2
  - Compare, with __ = LT, GT, LE, GE, EQ, NE
  - Sets FP status register based on the result

Is DLX a good architecture?

- See book for specs on SPECint92 and SPECfp92
  - Ideally should have somewhat of an even distribution among instructions
- Architecture allows a low CPI, but simplicity means we need more instructions
  - Compared to VAX, programs on average are twice as large on DLX, but CPI is six times shorter
  - Implies a threefold performance advantage

Sample DLX Assembly Program

```
        .data
        .align      2
n:      .word       6
result: .word       0

        .text
        .global main
main:
        ;some initializations
        addi        r1, r0, 0
        addi        r2, r0, 1
        lw          r3,  n(r0)
        lw          r10, n(r0)
```

```
Top:    slei    r11, r10, #1
        bnez    r11, Exit
        add     r3, r1, r2
        addi    r1, r2, #0
        addi    r2, r3, #0
        subi    r10, r10, #1
        j       Top

Exit:   sw      result(r0), r3
        trap    0
```

Can you figure out what this does?

WinDLX Assembly Summary (1)

- ADD Rd,Ra,Rb Add
- ADDI Rd,Ra,Imm Add immediate (all immediates are 16 bits)
- ADDU Rd,Ra,Rb Add unsigned
- ADDUI Rd,Ra,Imm Add unsigned immediate
- SUB Rd,Ra,Rb Subtract
- SUBI Rd,Ra,Imm Subtract immediate
- SUBU Rd,Ra,Rb Subtract unsigned
- SUBUI Rd,Ra,Imm Subtract unsigned immediate

WinDLX Assembly Summary (2)

- MULT Rd,Ra,Rb Multiply signed
- MULTU Rd,Ra,Rb Multiply unsigned
- DIV Rd,Ra,Rb Divide signed
- DIVU Rd,Ra,Rb Divide unsigned
- AND Rd,Ra,Rb And
- ANDI Rd,Ra,Imm And immediate
- OR Rd,Ra,Rb Or
- ORI Rd,Ra,Imm Or immediate
- XOR Rd,Ra,Rb Xor
- XORI Rd,Ra,Imm Xor immediate

WinDLX Assembly Summary (3)

- LHI Rd,Imm Load high immediate - loads upper half of register with immediate
- SLL Rd,Rs,Rc Shift left logical
- SRL Rd,Rs,Rc Shift right logical
- SRA Rd,Rs,Rc Shift right arithmetic
- SLLI Rd,Rs,Imm Shift left logical 'immediate' bits
- SRLI Rd,Rs,Imm Shift right logical 'immediate' bits
- SRAI Rd,Rs,Imm Shift right arithmetic 'immediate' bits

WinDLX Assembly Summary (4)

- S__ Rd,Ra,Rb Set conditional: "__" may be EQ, NE, LT, GT, LE or GE
- S__I Rd,Ra,Imm Set conditional immediate: "__" may be EQ, NE, LT, GT, LE or GE
- S__U Rd,Ra,Rb Set conditional unsigned: "__" may be EQ, NE, LT, GT, LE or GE
- S__UI Rd,Ra,Imm Set conditional unsigned immediate: "__" may be EQ, NE, LT, GT, LE or GE
- NOP No operation

WinDLX Assembly Summary (5)

- LB Rd,Adr Load byte (sign extension)
- LBU Rd,Adr Load byte (unsigned)
- LH Rd,Adr Load halfword (sign extension)
- LHU Rd,Adr Load halfword (unsigned)
- LW Rd,Adr Load word
- LF Fd,Adr Load single-precision Floating point
- LD Dd,Adr Load double-precision Floating point

WinDLX Assembly Summary (6)

- SB Adr,Rs Store byte
- SH Adr,Rs Store halfword
- SW Adr,Rs Store word
- SF Adr,Fs Store single-precision Floating point
- SD Adr,Fs Store double-precision Floating point
- MOVI2FP Fd,Rs Move 32 bits from integer registers to FP registers
- MOVI2FP Rd,Fs Move 32 bits from FP registers to integer registers

WinDLX Assembly Summary (7)

- MOVF Fd,Fs Copy one Floating point register to another register
- MOVD Dd,Ds Copy a double-precision pair to another pair
- MOVI2S SR,Rs Copy a register to a special register (not implemented!)
- MOVS2I Rs,SR Copy a special register to a GPR (not implemented!)

WinDLX Assembly Summary (8)

- BEQZ Rt,Dest Branch if GPR equal to zero; 16-bit offset from PC
- BNEZ Rt,Dest Branch if GPR not equal to zero; 16-bit offset from PC
- BFPT Dest Test comparison bit in the FP status register (true) and branch; 16-bit offset from PC
- BFPF Dest Test comparison bit in the FP status register (false) and branch; 16-bit offset from PC

WinDLX Assembly Summary (9)

- J Dest Jump: 26-bit offset from PC
- JR Rx Jump: target in register
- JAL Dest Jump and link: save PC+4 to R31; target is PC-relative
- JALR Rx Jump and link: save PC+4 to R31; target is a register
- TRAP Imm Transfer to operating system at a vectored address; see Traps.
- RFE Dest Return to user code from an execption; restore user mode (not implemented!)

WinDLX Assembly Summary (10)

- ADDD Dd,Da,Db Add double-precision numbers
- ADDF Fd,Fa,Fb Add single-precision numbers
- SUBD Dd,Da,Db Subtract double-precision numbers
- SUBF Fd,Fa,Fb Subtract single-precision numbers.
- MULTD Dd,Da,Db Multiply double-precision Floating point numbers
- MULTF Fd,Fa,Fb Multiply single-precision Floating point numbers

WinDLX Assembly Summary (11)

- DIVD Dd,Da,Db Divide double-precision Floating point numbers
- DIVF Fd,Fa,Fb Divide single-precision Floating point numbers
- CVTF2D Dd,Fs Converts from type single-precision to type double-precision
- CVTD2F Fd,Ds Converts from type double-precision to type single-precision
- CVTF2I Fd,Fs Converts from type single-precision to type integer
- CVTI2F Fd,Fs Converts from type integer to type single-precision

WinDLX Assembly Summary (12)

- CVTD2I Fd,Ds Converts from type double-precision to type integer
- CVTI2D Dd,Fs Converts from type integer to type double-precision
- __D Da,Db Double-precision compares: "__" may be EQ, NE, LT, GT, LE or GE; sets comparison bit in FP status register
- __F Fa,Fb Single-precision compares: "__" may be EQ, NE, LT, GT, LE or GE; sets comparison bit in FP status register