Ch 2.
Instruction Set Architecture
DLX

Datapath and Control

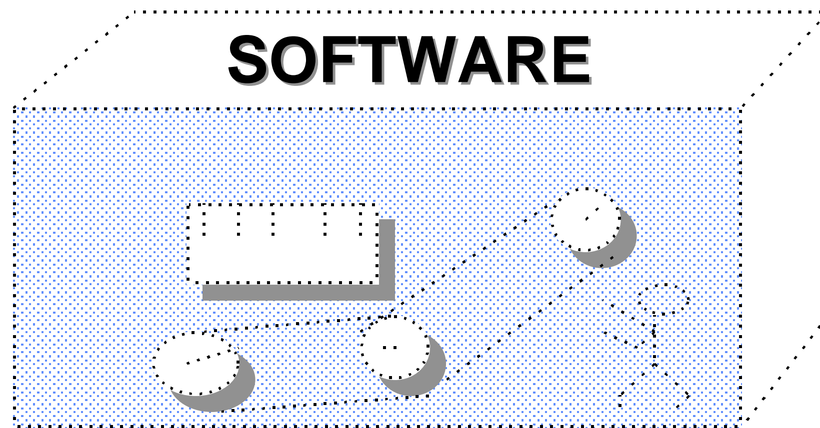
Computer Architecture Course?

- **1950s to 1960s:**
  - **Computer Arithmetic**
- **1970 to mid 1980s:**
  - **Instruction Set Design, especially ISA appropriate for compilers**
- **1990s:**
  - **Design of CPU, memory system, I/O system, Multiprocessors**

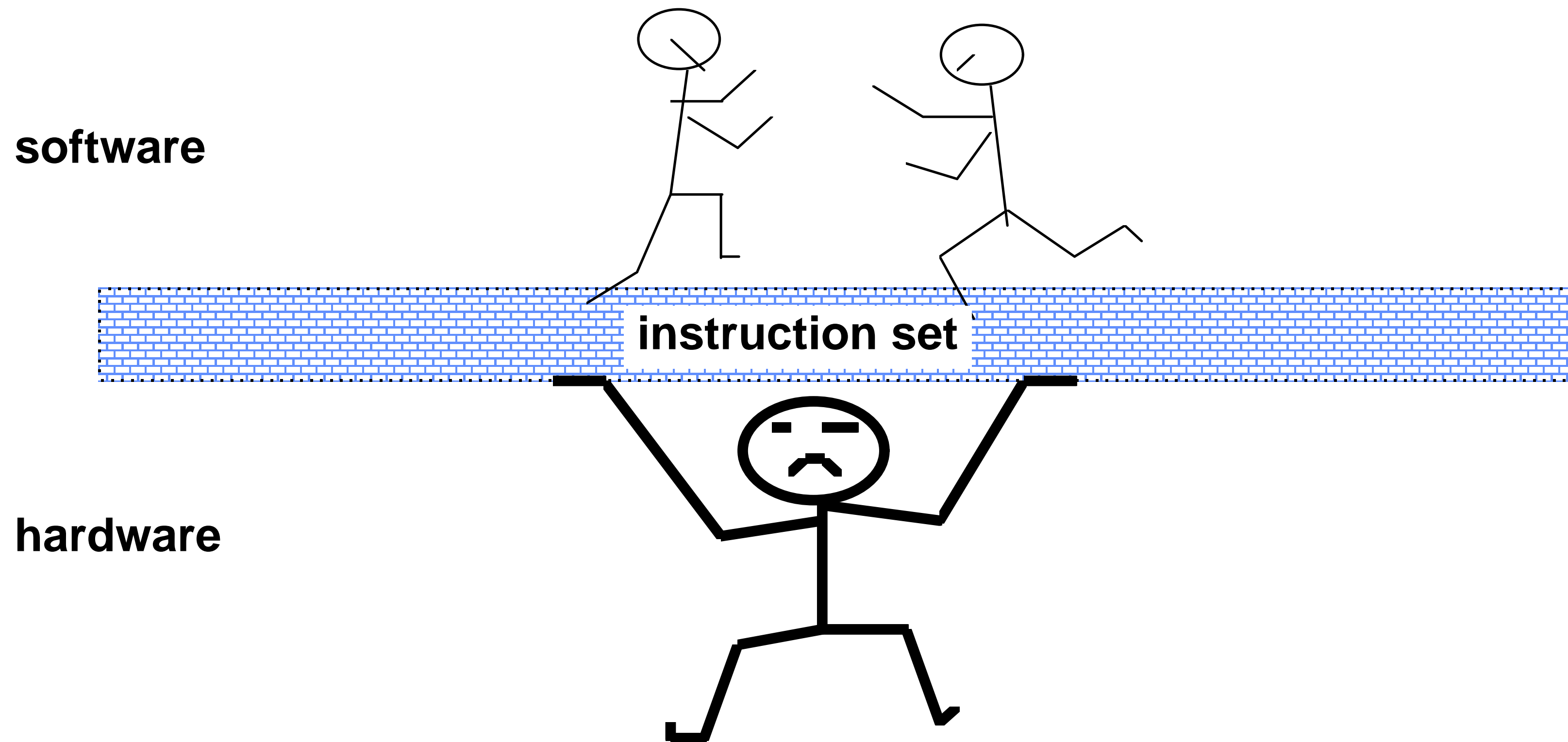
(Instruction Set) Computer Architecture?

. . . the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964



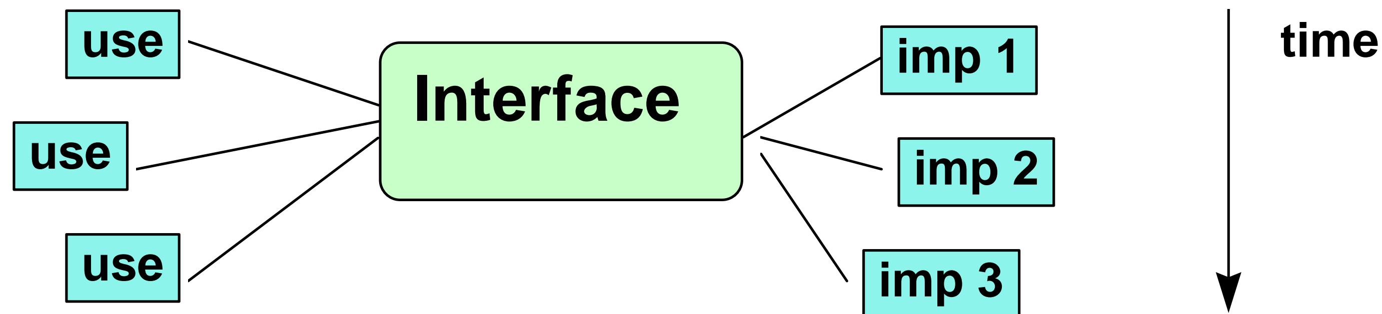
Towards Evaluation of ISA and Organization



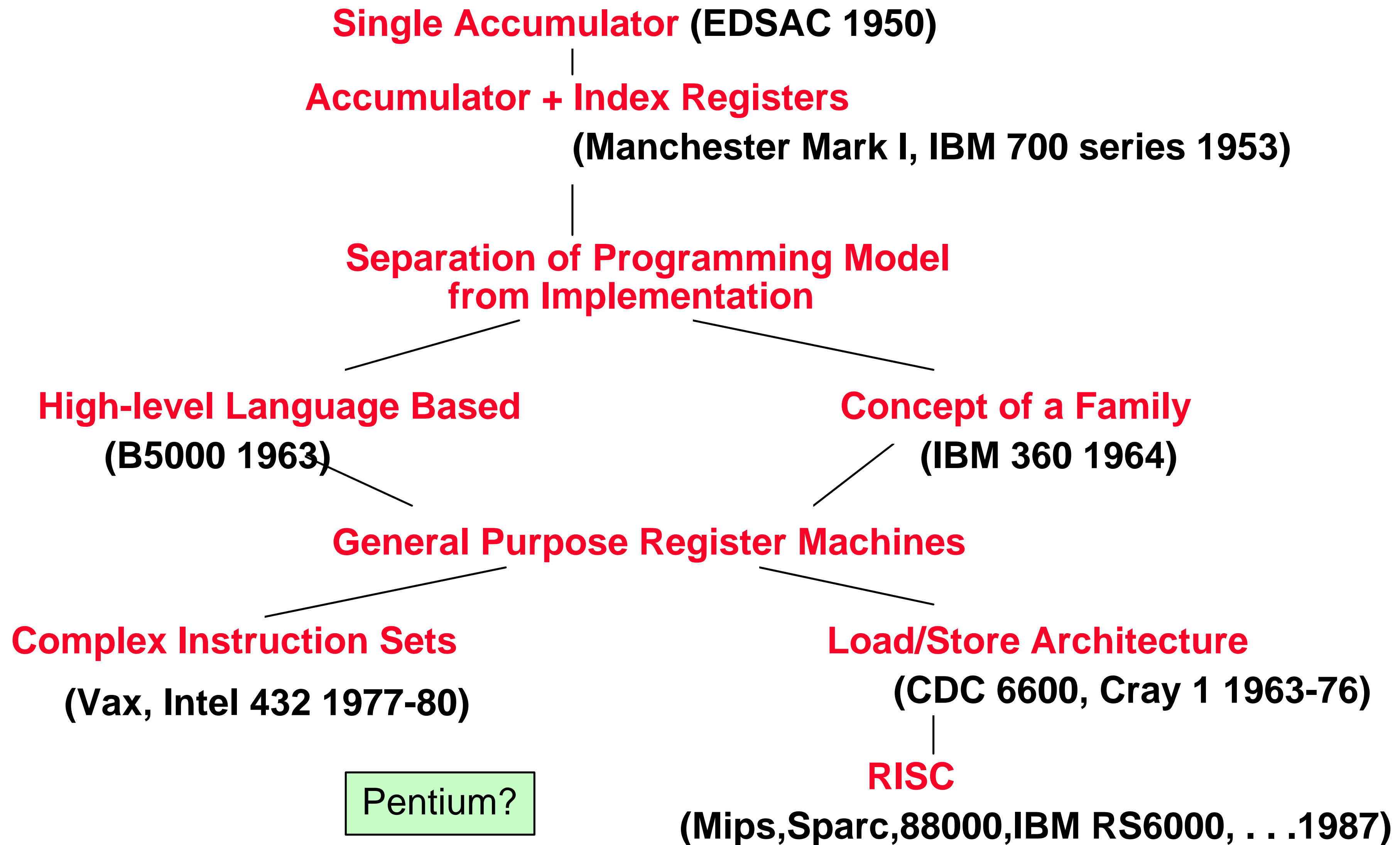
Interface Design

A good interface:

- Lasts through many implementations (portability, compatability)
- Is used in many differeny ways (generality)
- Provides convenient functionality to higher levels
- Permits an efficient implementation at lower levels



Evolution of Instruction Sets



Evolution of Instruction Sets

- Major advances in computer architecture are typically associated with landmark instruction set designs
  - Ex: Stack vs. GPR (System 360)
- Design decisions must take into account:
  - technology
  - machine organization
  - programming languages
  - compiler technology
  - operating systems
- And they in turn influence these

Design Space of ISA

Five Primary Dimensions

• Number of explicit operands	(0, 1, 2, 3)
• Operand Storage	Where besides memory?
• Effective Address	How is memory location specified?
• Type & Size of Operands	byte, int, float, vector, . . .
	How is it specified?
• Operations	add, sub, mul, . . .
	How is it specifed?

Other Aspects

• Successor	How is it specified?
• Conditions	How are they determined?
• Encodings	Fixed or variable? Wide?
• Parallelism	

ISA Metrics

Aesthetics:

- **Orthogonality**
  - No special registers, few special cases, all operand modes available with any data type or instruction type
- **Completeness**
  - Support for a wide range of operations and target applications
- **Regularity**
  - No overloading for the meanings of instruction fields
- **Streamlined**
  - Resource needs easily determined

Ease of compilation (programming?)

Ease of implementation

Scalability

Basic ISA Classes

Accumulator:

1 address	add A	acc ← acc + mem[A]
1+x address	addx A	acc ← acc + mem[A + x]

Stack:

0 address	add	tos ← tos + next

General Purpose Register:

2 address	add A B	EA(A) ← EA(A) + EA(B)
3 address	add A B C	EA(A) ← EA(B) + EA(C)

Load/Store:

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

Stack Machines

- **Instruction set:**

  +, -, *, /, . . .

  push A, pop A

- **Example: a*b - (a+c*b)**

  push a

  push b

  *****

  push a

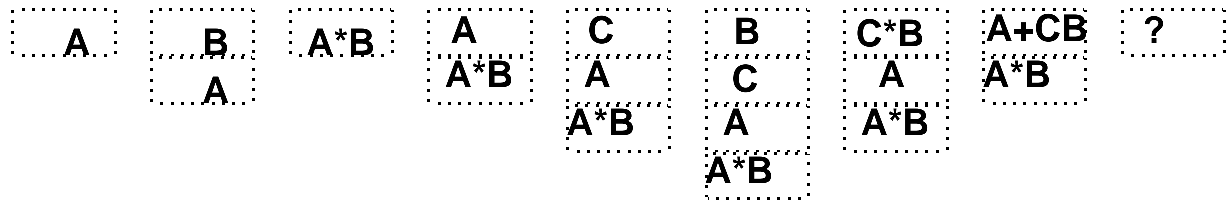
  push c

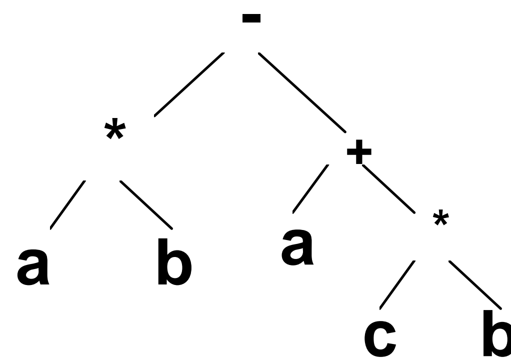
  push b

  *****

  +

  -





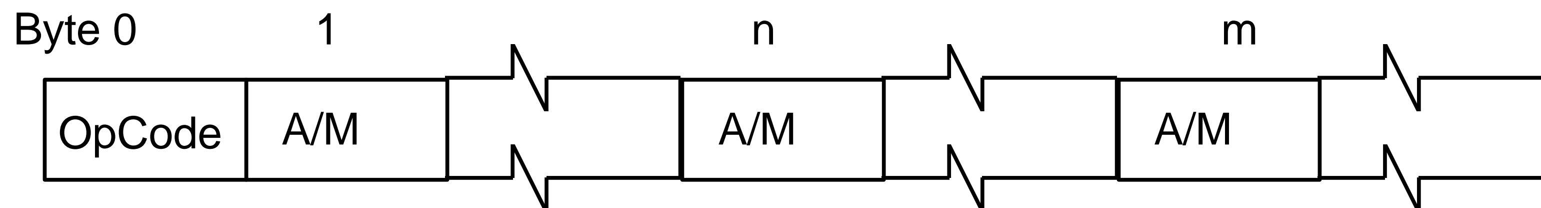
The Case Against Stacks

- Performance is derived from the existence of several fast registers, not from the way they are organized
- Data does not always “surface” when needed
  - Constants, repeated operands, common subexpressions

  so TOP and Swap instructions are required
- Code density is about equal to that of GPR instruction sets
  - Registers have short addresses
  - Keep things in registers and reuse them
- Slightly simpler to write a poor compiler, but not an optimizing compiler

VAX-11

Variable format, 2 and 3 address instruction



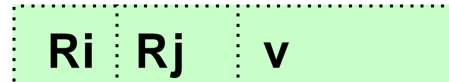
- 32-bit word size, 16 GPR (four reserved)
- Rich set of addressing modes (apply to any operand)
- Rich set of operations
  - bit field, stack, call, case, loop, string, poly, system
- Rich set of data types (B, W, L, Q, O, F, D, G, H)
- Condition codes

Kinds of Addressing Modes

- **Register direct** Ri
- **Immediate (literal)** v
- **Direct (absolute)** M[v]
- **Register indirect** M[Ri]
- **Base+Displacement** M[Ri + v]
- **Base+Index** M[Ri + Rj]
- **Scaled Index** M[Ri + Rj*d + v]
- **Autoincrement** M[Ri++]
- **Autodecrement** M[Ri --]
- **Memory Indirect** M[M[Ri]]
- **[Indirection Chains]**

memory

reg. file



A "Typical" RISC

Double Precision

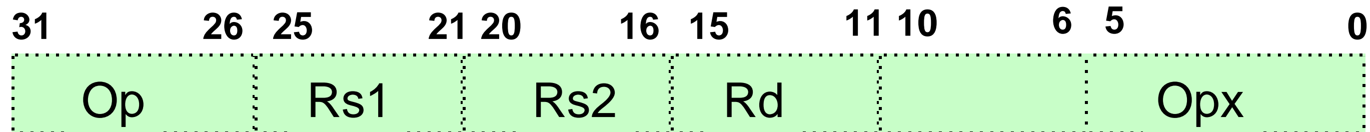
- **32-bit fixed format instruction (3 formats)**
- **32 32-bit GPRs (R0 contains zero, DP take pair)**
- **3-address, reg-reg arithmetic instruction**
- **load-and-store**
- **Single address mode for load/store: base + displacement**
  - **no indirection**
- **Simple branch conditions**
- **Delayed branch**

KISS - Keep It Simple, Stupid!

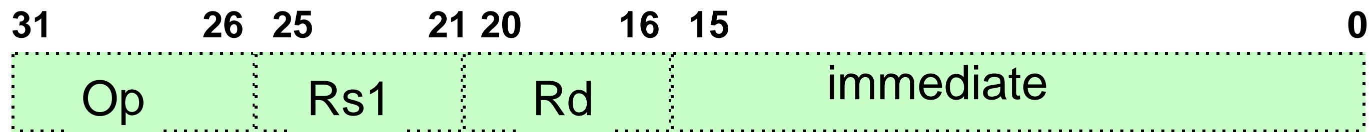
see: SPARC, MIPS, MC88100, AMD2900, i960, i860
PARisc, DEC Alpha, Clipper,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS

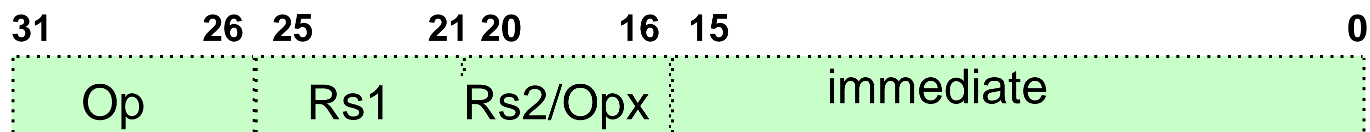
Register-Register



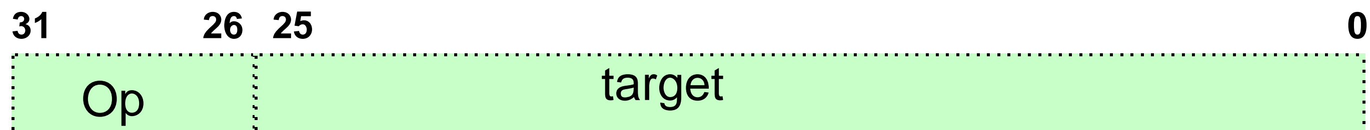
Register-Immediate



Branch



Jump / Call



DLX
Instruction Set Architecture

DLX

Five Primary Dimensions

• Number of explicit operands	(0, 1, 2, 3)
• Operand Storage	Where besides memory?
• Effective Address	How is memory location specified?
• Type & Size of Operands	byte, int, float, vector, . . .
	How is it specified?
• Operations	add, sub, mul, . . .
	How is it specified?

DLX principles

- simple load-and-store architecture
  - general purpose registers
- efficient pipeline
  - fixed instruction length, etc etc
  - easy to decode
- pipeline is easy to optimize by a compiler
  - pipeline is visible to the compiler!

DLX-registers

- fig. 2.3 --> reg-reg architecture
- load-and-store architecture with general purpose registers (GPR)
- helps to optimize pipeline (i.e., speed)
- 32 32bit GPRs: R0,...,R31
  - R0 = 0 (constant)
- 32 32bit floating point regs (FPR): F0,...,F31
  - can be used as 16 double prec regs: F0,F2, F4, ..., F30
- why not more?
  - too long address?
- why not less?
  - at least 16 needed to produce good, efficient code
  - enough for optimising compiler to do good register allocation

DLX Data types

- int: byte (8 bit), half word (16 bit), word (32 bit)
  - register operations always 32 bits
  - aligned data, see fig. 2.4
- float: single (32 bit), double (64 bit)
- No chars, bits, strings, quad words, ...
- support those most needed
  - others may take more time
  - others handled with software

Amdahl

DLX Addressing Modes

- **figs. 2.5, 2.6, 2.8, 2.9**
- **only 2 modes:**
  - **immediate** — **16bit operand in instruction**
  - **displacement** — **Ri + 16bit offset**
    - » **register deferred (offset = 0)**
    - » **absolute addressing (Ri = R0 = 0)**
- **addressing features**
  - **byte addressing**
  - **Big-Endian**
  - **32 bit address**
  - **data must be aligned**
    - » **"word address" mod 4 = 0**

	0x1200:	0x1201:	0x1202:	0x1203:
0x1200:			1	03

word(0x1200) = $257_{10} = 103_{16}$ = 0x103

DLX Instruction Format

- figs 2.11, 2.12, 2.13, 2.14
- three instruction types
  - with 16 bit
    - » immeadiate
    - » displacement addressing
    - » PC-relative branches
  - register to register ALU
  - jumps with 26 bit offset
- fig. 2.21

DLX operations

- load, store, fig. 2.22
- std ALU operations, fig. 2.23
  - MUL and DIV using floating point regs!
- FP operations
  - ADDF, SUBF, MULF, DIVF, converts, compares
  - ADDD, SUBD, ...
- (conditional) branch, (unconditional) jump, fig. 2.24
- all instructions: fig. 2.25
- which instrcutions used by SpecFP92: fig. 2.29
- which instructions used by SpecInt92: fig. 2.28

Is DLX approach fast?

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

i.e.,

CPU time = Instruction count x CPI x Clock Cycle Time

- **if same clock cycle time, how does RISC approach (DLX, MIPS, ...) compare to CISC?**
  - **more but <u>much faster</u> instructions**
  - **fig. 2.30**
- **what if clock cycle times are not the same?**