Review: (Write on the board before class)
1) 80x86: First 16-bit microprocessor, evolved from segmented 16-bit dedicated register machine to 32-bit general register machine

Today Ch 4: DLX
DLX (Deluxe) similar to many machines annonced at end of 1980s

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

32-bit (GPRs); the value of R0 is always 0.
(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.

Memory is byte addressable in Big Endian mode with a 32-bit address.

- All instructions are 32 bits and must be aligned. data too

(Oppositve emphasis from other machines: same size to simplify decoding vs. saving code size)

See figures

Break
Pick project: Wednesday with partners!!
Quickly over go over projects
Discussion questions for Friday 5.21, 5.22

*Fallacy: There is such a thing as **a** typical program.*

Pitfall: Designing an architecture on the basis of small benchmarks or large benchmarks from a restricted application domain when the machine is intended to be general purpose.

Fallacy: An architecture with flaws cannot be successful.

> The IBM 360 is often criticized in the literature—the branches are not PC-relative, and the offset is too small in based addressing. Yet, the machine has been an enormous success because it did several new things properly. First, the architecture had a big enough address space. Second, it was byte addressed and handled bytes well. Third, it was a general-purpose register machine. Finally, it

was simple enough that it could be efficiently implemented across a wide performance and cost range.

The 8086 provides an even more dramatic example. The 8086 architecture is the only widespread architecture in existence today that is not truly a general-purpose register machine. Furthermore, the segmented address space of the 8086 causes major problems both for programmers and compiler writers. Despite these major difficulties, the 8086 architecture—because of its selection as the microprocessor in the IBM PC—has been enormously successful.

Fallacy: One can design a flawless architecture.

For example, in 1975 the VAX designers overemphasized the importance of code-size efficiency and underestimated how important ease of decoding and pipelining would be ten years later. Almost all architectures eventually succumb to the lack of sufficient address space. However, avoiding this problem in the long run would probably mean compromising the efficiency of the architecture in the short run.

Fallacy: In instruction mixes, time distribution and frequency distribution will be close.

Appendix D shows the time distributions for our benchmark programs and compares the time and frequency distributions. A simple example of where these distributions are very different is in the COBOLGO program on the 360. Figure 4.35 shows the top instructions by frequency and by time. The two highest occurring instructions are responsible for 33% of the instruction executions in COBOLGO, but only 4% of the execution time! Remember that time distributions are dependent on both the architecture and the **implementation** used for the measurement. Hence, time distributions may differ from model to model, while frequency distributions will be the same, provided neither the software nor the program changes. This large difference between time and frequency distributions does not exist for simpler load/store architectures, such as DLX.

Pitfall: Examining only the worst-case or average behavior of an instruction as design input.

The best example of this comes from the use of `MVC` on an IBM 360. The instruction can move overlapped fields of characters, but this occurs less than 1% of the time, and then usually to clear a field. The average length of a move as measured by Shustek was ten bytes, but more than three-quarters of the moves were either one byte or four bytes in length. Assuming worst-case behavior (overlapping strings) or average length can each lead to suboptimal design decisions.

Ch 5 Impementation

Determines 2 or three parameters of performance
Time = IC * CPI * CCT

Quickly go over the ideas, although assume that this is review (how many think it is review?)

1. Instruction fetch step:
   MAR ← PC; IR ← M[MAR]
2. Instruction decode/register fetch step:
   A ← Rs1; B ← Rs2; PC ← PC + 4
3. Execution/effective address step:

Memory reference:

MAR ← A + $(IR_{16})^{16}$##$IR_{16..31}$; MDR ← **B**

Note: This is a mistake Rd => B (p. 202); also p. 221 Fig 5.14; drop A<-Rd from Memory state; change Stores state from MDR <- A to MDR <- B

ALU instruction:

ALUoutput ← A *op* (B or $(IR_{16})^{16}$##$IR_{16..31}$)

Branch/Jump:

ALUoutput← PC+$(IR_{16})^{16}$##$IR_{16..31}$; cond←(A *op* 0)

4. Memory access/branch completion step: The only DLX instructions active in this step are loads, stores, branches, and jumps.

Memory reference:

MDR ← M[MAR] or M[MAR] ← MDR;

Branch:

if (cond) PC ← ALUoutput (branch)

5. Write-back step:
   Rd ← ALUoutput or MDR

Example instruction	Instruction name	Meaning
LW R1,30(R2)	Load word	$R1 \leftarrow_{32} M[30+R2]$
LW R1,1000(R0)	Load word	$R1 \leftarrow_{32} M[1000+0]$
LB R1,40(R3)	Load byte	$R1 \leftarrow_{32} (M[40+R3]_0)^{24}$ ## $M[40+R3]$
LBU R1,40(R3)	Load byte unsigned	$R1 \leftarrow_{32} 0^{24}$ ## $M[40+R3]$
LH R1,40(R3)	Load halfword	$R1 \leftarrow_{32} (M[40+R3]_0)^{16}$ ##$M[40+R3]$##$M[41+R3]$
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.

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	$R1 \leftarrow 42\#\#0^{16}$
SLL 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; $(PC-2^{25}) \le name < (PC+2^{25})$
JAL	name	Jump and link	R31←PC+4; PC←name; $(PC-2^{25}) \le name < (PC+2^{25})$
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; $(PC-2^{15}) \le name < (PC+2^{15})$
BNEZ	R4,name	Branch not equal zero	if (R4!=0) PC←name; $(PC-2^{15}) \le name < (PC+2^{15})$

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."

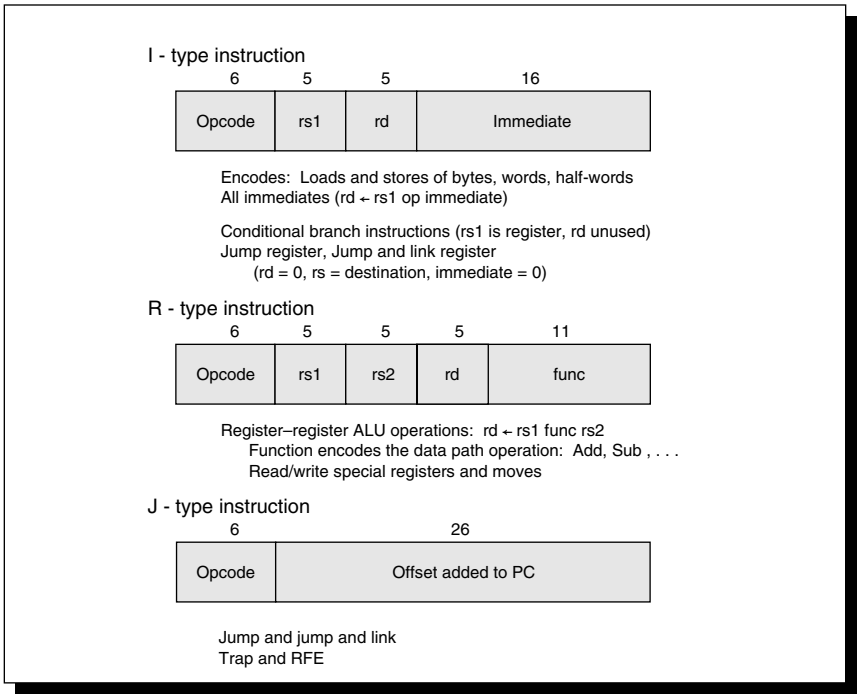


FIGURE 4.19 Instruction layout for DLX. All instructions are encoded in one of three types.

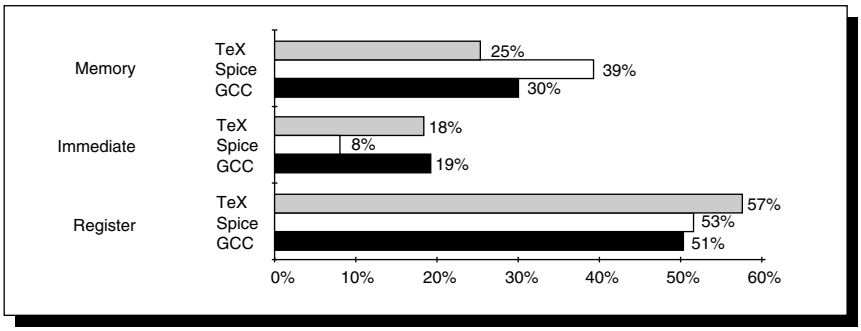


FIGURE 4.22 Breakdown of basic operand types for the three benchmarks on the VAX. The frequencies are very similar across programs, except for the low usage of immediates by Spice and its correspondingly higher use of memory operands. This probably arises because few floating-point constants are stored as immediates, but are instead accessed from memory. An operand is counted by the number of times it appears in an instruction, rather than by the number of references. Thus, the instruction `ADDL2 R1,45(R2)` counts as one memory reference and one register reference. The memory address modes in Figure 4.23 are counted in the same fashion. Wiecek [1982] reports that about 90% of the operand accesses are either a read or a write, and only about 10% of the accesses both read and write the same operand (such as `R1` in the `ADDL2`).

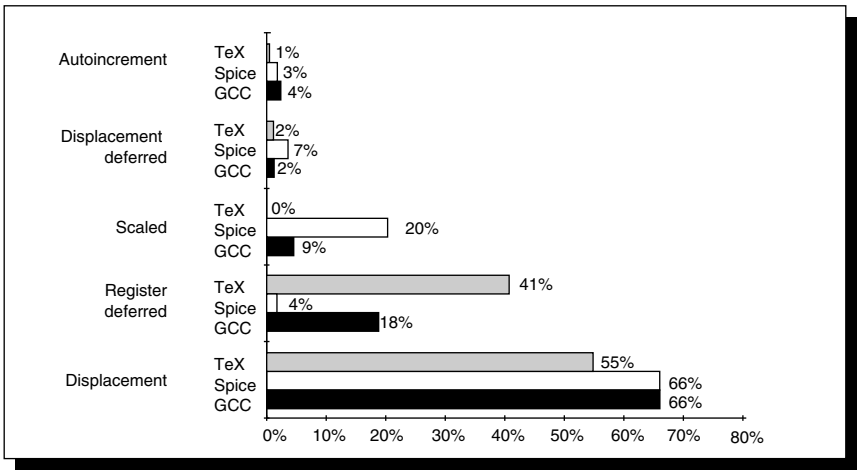


FIGURE 4.23 Use of VAX memory addressing modes, which account for about 31% of the operand references, in the three programs. Spice again stands out because of the low frequency of register deferred. In Spice, nonzero displacement values occur much more frequently. The use of arrays rather than pointers probably influences this. Likewise, Spice uses the scaled mode to access array elements. The displacement deferred mode is used to access actual parameters in a FORTRAN subroutine. These modes account for all the memory operands in GCC and TeX and all but 1% (which are autodecrement mode) in Spice. Remember that PC-based addressing is not included here—use of PC-based addressing can be measured by branch frequency.

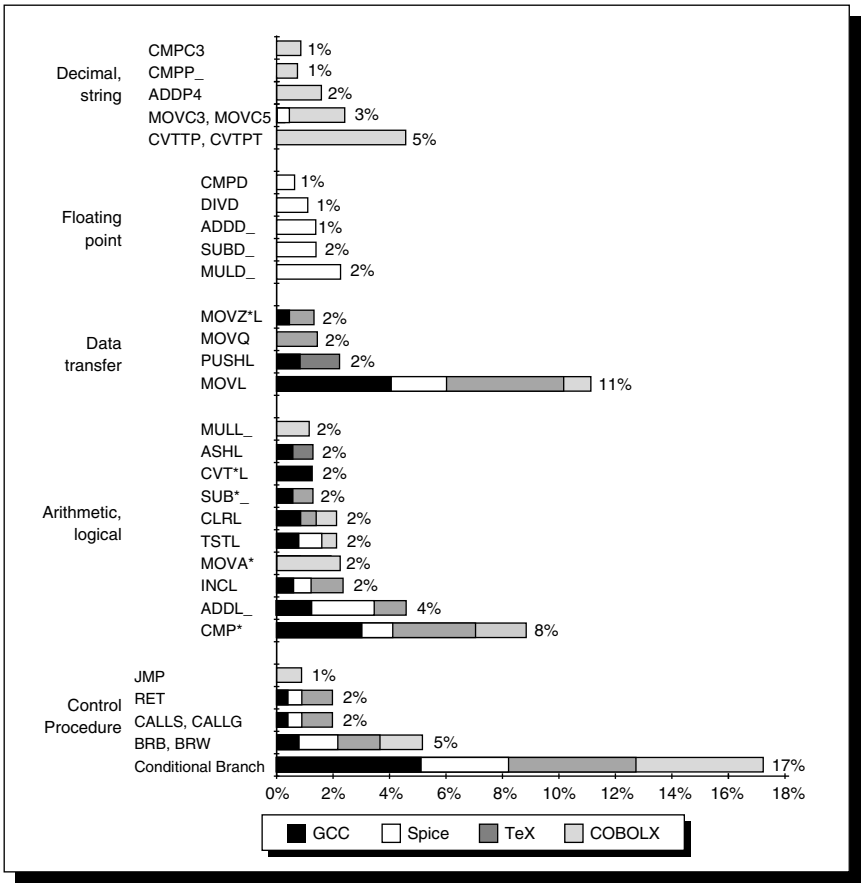


FIGURE 4.24 The VAX instruction frequencies combined graphically. The total size of each bar shows the behavior that would be seen on a machine that ran these four programs with equal frequency. The segments of the bar show what percentage of the usage of that instruction would come from each of the programs. This illustrates that some portions of the instruction set need to be there for only one class of applications. Overall, only a small number of instructions outside of the control, data transfer, and integer arithmetic instructions are heavily used.

Tools

P1. Instruction statistics tools for 370 or 80x86. (Exercise 4.22)

P2. Link-time instrumentation for cache simulation.

P3. Improving DLX tools.

P4. DLX instruction scheduler (Exercise 6.22)

P5. Adding vector instructions to the DLX simulator (Exercise 7.13)

P6. Adding vector checking to the DLX compiler (Exercise 7.14)

Input/Output

P7 Collecting statistics on disk arm motion

P8 Measuring I/O patterns in supercomputers

P9 I/O overhead experiments

P10 I/O performance experiments (Exercise 9.15)

P11 Generalized scaling formulas

Cost and Performance

P12 Evaluate a "stall" cache

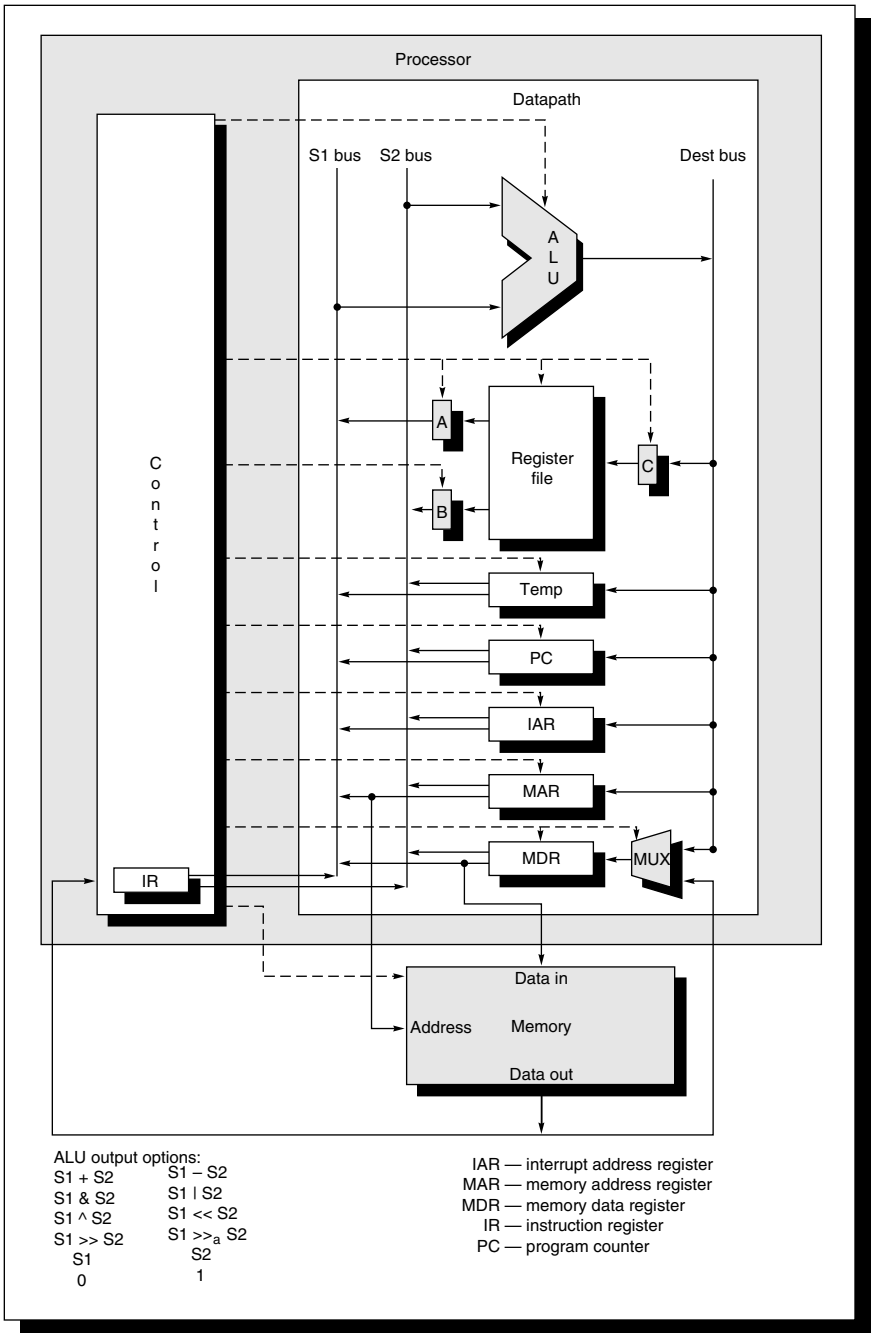
P13 Distributed system cost

P14 Benchmarking Cray instruction set architecture vs. newer architectures

P15 Generalized Benchmarks

P16 Synthetic SPEC benchmark

Remaining ideas


Processor
Datapath
S1 bus
S2 bus
Dest bus
A
L
U
A
B
Register file
C
Temp
PC
IAR
MAR
MDR
MUX
IR
Control
Data in
Address
Memory
Data out
ALU output options:
S1 + S2
S1 – S2
S1 & S2
S1 | S2
S1 ^ S2
S1 << S2
S1 >> S2
S1 >>a S2
S1
S2
0
1
IAR — interrupt address register
MAR — memory address register
MDR — memory data register
IR — instruction register
PC — program counter