Program Behavior

• Fundamental Computer Engineering Principle: Make the common case fast.

• Goal: Understand the properties of the "common case".

• Caution: Because of the wide variability in computations -- even different runs of the same computation -- and because of the complex interactions between programming languages, compilers, operating systems, instruction sets, architectures and techology, NO ONE has any absolute idea of what the characteristics of a workload are, and of course, computers must be designed to work well across a wide range of work loads!

• Treat the numbers presented as "reasonable estimates".

Instruction Coding

The encoding of instructions (design of the ISA) and their dynamic usage determine how many bytes of memory must be fetched to realize a give "quantity of computation."

Compare the encoding of `X := X - Y;` for different architectures:

L/S	R/M	R+M
`LD.W R8,X`	`LD.W R8,X`	`Sub.W X,X,Y`
`LD.W R9,Y`	`Sub.W R8,Y`	
`Sub.W R8,R8,R9`	`ST.W X,R8`	
`ST.W X,R8`		
$4 \times 4 = 16$B	$3 \times 4 = 12$B	$1 \times 6 = 6$B

Code Density

A "tighter" encoding of instructions can reduce the memory bandwidth needed for fetching instructions.

Architectures	2B RR	4B MR	6B MM	Ave	Inst Ex'ed*
L/S	0%	100%	0%	4.0B	200
R/M					
Sci/Sys Load	40%	50%	0%	3.2B	180
Commercial	30%	50%	20%	3.8B	
R+M					
Sci/Sys Load	25%	50%	25%	4.0B	120
Commercial	30%	50%	20%	3.8B	

* Estimates of dynamic instruction executions for "100 Higher Level Language Operations"

Characteristics of Loads

Though no program is "typical", program execution can be predicted to have a variety of features.

For a UNIX workstation environment, program execution divides up as follows

User time -- performing task, etc. -- 60%

System time -- setting up page tables, etc. -- 30%

Library -- sine(), printf(), etc. -- 10%

Conclusion: If different parts of the computation have different characteristics, then optimizing for one does not guarantee improved performance.

Procedure Calls

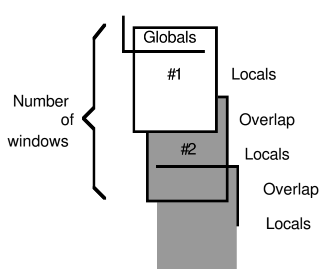
Procedures are encouraged as a programming device, but their invocation/return introduce overhead.

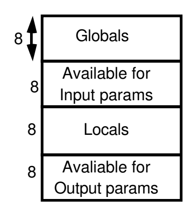
Reasonable Conclusions ...
- Call frequency low 0.5%-1% of inst'ns.
- Variance is large: 10%-12% possible, but under 1% is too.
- The overhead of a call is nontrivial.
- Procedural call instructions architecture independent.
- Instruction count, but not work reducible.
- Register windows reduce, but don't eliminate the procedural call cost.

L/S Architecture	Inst'ns
Procedure Calls	
Call/return	2.0
Fix Stack Frame	2.0
Param Passing	1.5
Reg Save/Restore	4.8
Return Value	0.6
Total	10.9

Register Windows (Digression)

Because procedures often do not use all the GPRs and quickly call other procedures, registers can be managed as a sliding window:





Cf. Section 2.6.5

System Calls

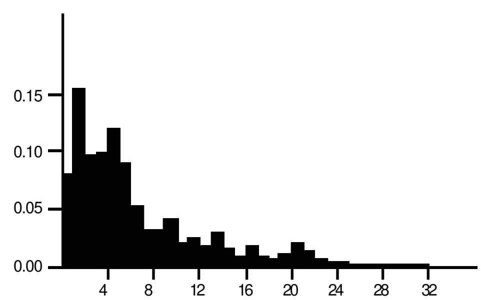
A substantial amount of time is spent in the OS. Entering and exiting the operating system is a usually more expensive than a procedure call.

Operation	VAX	R2000	SPARC	i860
OS call/return	12	84	128	86
Trap/interrupt	14	103	145	155
Fix Page Tab	11	36	15	559
Context Sw	9	135	326	618

Memory	VAX	S/390	R2000	SPARC	i860
GPRegs	16	16	32	136	32
FP Regs	0	8	32	32	32
Other(PSW..)	1	2	5	6	9

Instruction Run Length

How many instructions are executed typically before a branch?



Notice: The mean distance between brances is 7.5 instructions, but 66% of the sequences are shorter than that.