CS184b:
Computer Architecture (Abstractions and Optimizations)

Day 2: March 30, 2005

Instruction Set Architecture

Today

- Datapath review
- H&P view on ISA
- RISC Themes

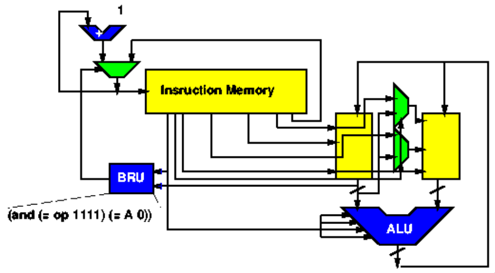
RISC?

- What does RISC mean?

CS184a Day 5

With Branch and Indirect

Instr: ALUOP Bsel Write Bsrc Asrc DST Baddr



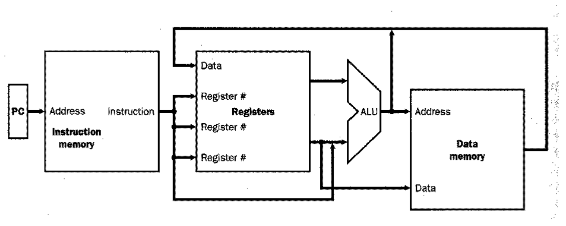
ISA

- Model based around Sequential Instruction Execution
- Visible-Machine-State × Instruction → New Visible-Machine-State
- New Visible-Machine-State identifies next instruction
  - PC←PC+1 or PC←branch-target

ISA

- Visible Machine State
  - Registers (including PC)
  - Memory
- Instructions
  - ADD R1, R2, R3
  - LD R3, R4
  - BNE R4, R2, loop

Datapath

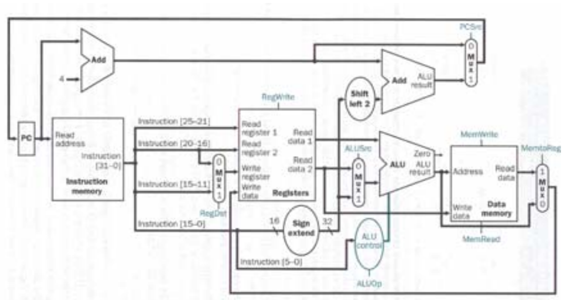


[Datapath from PH (Fig. 5.1)]

Instructions

- Primitive operations for constructing (describing) a computation
- Need to do?
  – Interconnect (space and time)
  – Compute (intersect bits)
  – Control (select operations to run)

Detail Datapath



[Datapath from PH (Fig. 5.13)]

H&P View

- ISA design done?
- Not many opportunities to completely redefine
- Many things mostly settled
  – at least until big technology perturbations arrive
- Implementation (uArch) is where most of the action is
- **Andre**: maybe we've found a nice local minima...

H&P Issues

- Registers/stack/accumulator
  – # operands, memory ops in instruction
- Addressing Modes
- Operations
- Control flow
- Procedure calls
- Primitive Data types
- Encoding

Addressing Modes

- Minimal:

– Immediate	#3
– Register	R1
– Register Indirect	(R1)

- Others:
  – displacement
  – indirect (double derefrence)
  – auto increment/decrement (p[x++]=y)
  – scaled

Addressing Modes

- More capable
  - less instructions
  - potentially longer instructions
    - bits and cycle time
  - many operations (complicate atomicity of instructions)
    - Add (R2)+,(R3)+,(R4)+

Address Space Quote

- *The Virtual Address eXtension of the PDP-11 architecture . . . provides a virtual address of about 4.3 gigabytes which, even given the rapid improvement of memory technology, should be adequate far into the future.*
- William Strecker, “VAX-11/780—A Virtual address Extension to the PDP-11 Family,” *AFIPS Proc., National Computer Conference*, 1978

Operations: Procedure call/return

- ? Save registers?
- Update PC
  - call target
  - return address
- Change stack and frame pointers
  - store old
  - install new

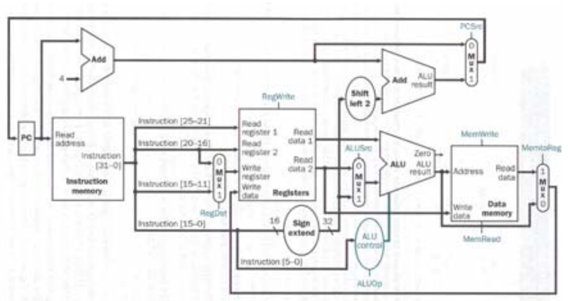
Operations: Procedure call/return

- **Question**: How much should instruction do?
- Lesson: High variance in work needs to be done
  - which registers need to save
  - best way to transfer arguments to procedures
  - better to expose primitives to the compiler and let it specialize the set of operations to the particular call

Data Types

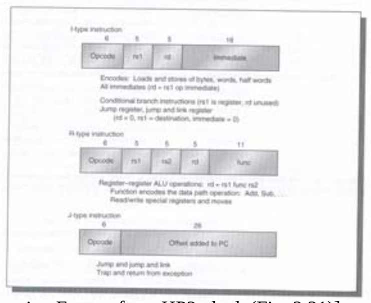
- Powers of two from bytes to double words?
  - 8, 16, 32, 64
  - (very implementation driven decision)
- Floating Point types
- Are pointers integers?
- Alignment requirements

Detail Datapath



[Datapath from PH (Fig. 5.13)]

Encoding: RISC/Modern



I-type instruction
6
5
5
16
Opcode
rs1
rd
Immediate
Encodes: Loads and stores of bytes, words, half words
All immediates (rd ← rs1 op immediate)
Conditional branch instructions (rs1 is register, rd unused)
Jump register, jump and link register
(rd = 0, rs1 = destination, immediate = 0)
R-type instruction
6
5
5
5
11
Opcode
rs1
rs2
rd
func
Register–register ALU operations: rd ← rs1 func rs2
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves
J-type instruction
6
26
Opcode
Offset added to PC
Jump and jump and link
Trap and return from exception

[DLX Instruction Format from HP2nd ed. (Fig. 2.21)]

Operation Complexity

- Contradiction?
  - Providing primitives
  - Including floating point ops

Local Minima?

- Self-Fulfilling?
  - How would we quantitatively validate need for a new operation?
  - [cue: bridge story]
  - This is what we use as primitives
  - Funny, we don't find a need for other primitives…

RISC Themes

- Common case fast
- Provide primitives (building blocks)
- Let compiler specialize to particular operation
- Make decode/operation simple so implementation is fast

Compilers

- 1960$\rightarrow$1990 shift
  - increasing capability and sophistication of compilers
  - e.g.
    - inter-procedural optimization
    - register assignment (register usage)
    - strength reduction
    - dataflow analysis and instruction reordering
    - (some progress) alias analysis

Compilers

- Gap between programmer and Architecture
- Increasingly bridged by compiler
- Less need to make assembly language human programmable
- More opportunity for compiler to specialize, partial evaluate
  - (do stuff at compile time to reduce runtime)
- RISC: “Relegate Interesting Stuff to Compiler”

ISA Driven by

1. Implementation costs
2. Compiler technology
3. Application structure

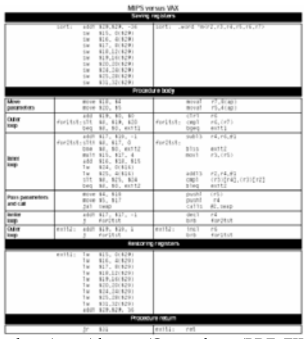
- Can't do good architecture in isolation from any of these issues.

RISC?

VAX Instructions

- Vary in length 1 to 53 bytes
- Some very complex
  – Powerful call routines
  – Polynomial evaluate (polyf)
  – Calculate CRC (crc)

VAX / MIPS procedure


MIPS versus VAX

	Saving registers	
	sort: addi $29,$29,-36 sw $15, 0($29) sw $16, 4($29) sw $17, 8($29) sw $18,12($29) sw $19,16($29) sw $20,20($29) sw $24,24($29) sw $25,28($29) sw $31,32($29)	sort: .word ^m<r2,r3,r4,r5,r6,r7>
	Procedure body	
Move parameters	move $18, $4 move $20, $5	moval r7,8(ap) moval r5,4(ap)
Outer loop	add $19, $0, $0 for1tst: slt $8, $19, $20 beq $8, $0, exit1	clrl r6 for1tst: cmpl r6,(r7) bgeq exit1
Inner loop	addi $17, $19, -1 for2tst: slti $8, $17, 0 bne $8, $0, exit2 muli $15, $17, 4 add $16, $18, $15 lw $24, 0($16) lw $25, 4($16) slt $8, $25, $24 beq $8, $0, exit2	subl3 r4,r6,#1 for2tst: blss exit2 movl r3,(r5) addl3 r2,r4,#1 cmpl (r3)[r4],(r3)[r2] bleq exit2
Pass parameters and call	move $4, $18 move $5, $17 jal swap	pushl (r5) pushl r4 calls #2,swap
Inner loop	addi $17, $17, -1 j for2tst	decl r4 brb for2tst
Outer loop	exit2: addi $19, $19, 1 j for1tst	exit2: incl r6 brb for1tst
	Restoring registers	
	exit1: lw $15, 0($29) lw $16, 4($29) lw $17, 8($29) lw $18,12($29) lw $19,16($29) lw $20,20($29) lw $24,24($29) lw $25,28($29) lw $31,32($29) addi $29,$29, 36	
	Procedure return	
	jr $31	exit1: ret

http://jbsim.cs.pku.edu.cn/users/chengxu/Org_web_ext/PDF_FILES/webext3_vax.pdf

RISC

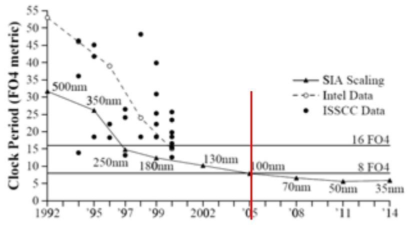
- Reduced Instruction Set Computers
- Idea:
  – Provide/expose minimal primitives
  – Make sure primitives fast
  – Compose primitives to build functionality
  – Provide orthogonal instructions

RISC Equation

- Time= CPI $\times$ Instructions $\times$ CycleTime
- CISC:
  – Minimize: Instructions
  – Result in High CPI
  – Maybe High CycleTime
- RISC:
  – Target single-cycle primitives (CPI~1)
  – Instruction Count increases
  – Simple encoding, ops $\rightarrow$ reduced Cycle Time

Cycle Time

- How many gate delays in 300ps?



[Agarwal et al./ISCA 2000]

VAX Data

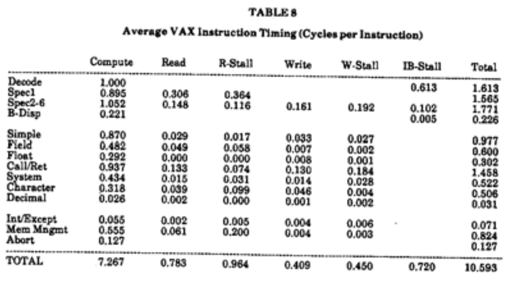


TABLE 8

Average VAX Instruction Timing (Cycles per Instruction)

	Compute	Read	R-Stall	Write	W-Stall	IB-Stall	Total
Decode	1.000					0.613	1.613
Spec1	0.895	0.306	0.364				1.565
Spec2-6	1.052	0.148	0.116	0.161	0.192	0.102	1.771
B-Disp	0.221					0.005	0.226
Simple	0.870	0.029	0.017	0.033	0.027		0.977
Field	0.482	0.049	0.058	0.007	0.002		0.600
Float	0.292	0.000	0.000	0.008	0.001		0.302
Call/Ret	0.937	0.133	0.074	0.130	0.184		1.458
System	0.434	0.015	0.031	0.014	0.028		0.522
Character	0.318	0.039	0.099	0.046	0.004		0.506
Decimal	0.026	0.002	0.000	0.001	0.002		0.031
Int/Except	0.055	0.002	0.005	0.004	0.006		0.071
Mem Mngmt	0.555	0.061	0.200	0.004	0.003		0.824
Abort	0.127						0.127
TOTAL	7.267	0.783	0.964	0.409	0.450	0.720	10.593

[Emer/Clark, ISCA 1984]

RISC Enabler 1

- "large", fast On-Chip SRAM
  - Large enough to hold kernel exploded in RISC Ops ~ 1--10K 32b words?
- Previous machines:
  - Off-chip memory bandwidth bottleneck
  - Fetch single instruction from off chip
  - Execute large number of microinstructions from on-chip ROM
    - ROM smaller than SRAM
- Small/minimal machine → make room for cache

RISC Enable 2

- High Level Programming
  - Bridge semantic gap by compiler
  - As opposed to providing powerful building blocks to assembly language programmer

Fit Problem

- "A great deal depends on being able to fit an entire CPU design on a single chip."
- "RISC computers benefit from being realizable at an earlier date."

Common Case

- "wherever there is a system function that is expensive or slow in all its generality, but where software can recognize a frequently occurring degenerate case (or can move the entire function from runtime to compile time) that function is moved from hardware to software, resulting in lower cost and improved performance." – 801 paper

Measurement Good

- Don’t assume you know what’s going on – measure
- Tune your intuition
- "Boy, you ruin all our fun -- you have data.” – DEC designers in response to a detailed quantitative study [Emer/Clark Retrospective on 11/780 performance characterization]

VAX/RISC Compare

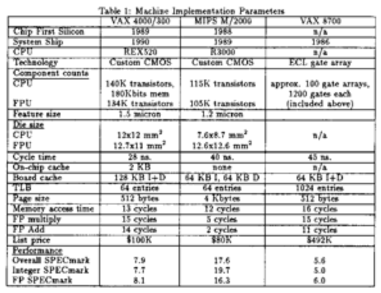


Table 1: Machine Implementation Parameters

	VAX 4000/300	MIPS M/2000	VAX 8700
Chip First Silicon	1989	1988	n/a
System Ship	1990	1989	1986
CPU	REX520	R3000	n/a
Technology	Custom CMOS	Custom CMOS	ECL gate array
Component counts			
CPU	140K transistors, 180Kbits mem	115K transistors	approx. 100 gate arrays, 1200 gates each (included above)
FPU	134K transistors	105K transistors	
Feature size	1.5 micron	1.2 micron	
Die size			
CPU	12x12 mm^2	7.6x8.7 mm^2	n/a
FPU	12.7x11 mm^2	12.6x12.6 mm^2	
Cycle time	28 ns.	40 ns.	45 ns.
On-chip cache	2 KB	none	n/a
Board cache	128 KB I+D	64 KB I, 64 KB D	64 KB I+D
TLB	64 entries	64 entries	1024 entries
Page size	512 bytes	4 Kbytes	512 bytes
Memory access time	13 cycles	12 cycles	16 cycles
FP multiply	15 cycles	5 cycles	15 cycles
FP Add	14 cycles	2 cycles	11 cycles
List price	$100K	$80K	$492K
Performance			
Overall SPECmark	7.9	17.6	5.6
Integer SPECmark	7.7	19.7	5.0
FP SPECmark	8.1	16.3	6.0

[Bhandarkar/Clark ASPLOS 1991]

VAX/RISC Compare

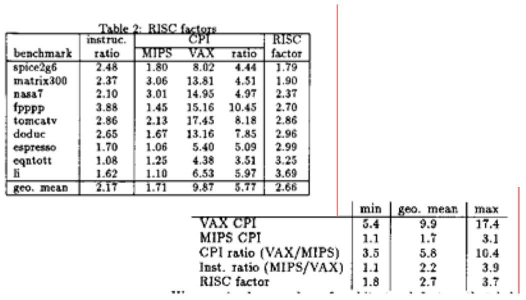


Table 2: RISC factors

benchmark	instruc. ratio	CPI MIPS	CPI VAX	CPI ratio	RISC factor
spice2g6	2.48	1.80	8.02	4.44	1.79
matrix300	2.37	3.06	13.81	4.51	1.90
nasa7	2.10	3.01	14.95	4.97	2.37
fpppp	3.88	1.45	15.16	10.45	2.70
tomcatv	2.86	2.13	17.45	8.18	2.86
doduc	2.65	1.67	13.16	7.85	2.96
espresso	1.70	1.06	5.40	5.09	2.99
eqntott	1.08	1.25	4.38	3.51	3.25
li	1.62	1.10	6.53	5.97	3.69
geo. mean	2.17	1.71	9.87	5.77	2.66

	min	geo. mean	max
VAX CPI	5.4	9.9	17.4
MIPS CPI	1.1	1.7	3.1
CPI ratio (VAX/MIPS)	3.5	5.8	10.4
Inst. ratio (MIPS/VAX)	1.1	2.2	3.9
RISC factor	1.8	2.7	3.7

[Bhandarkar/Clark ASPLOS 1991]

VAX

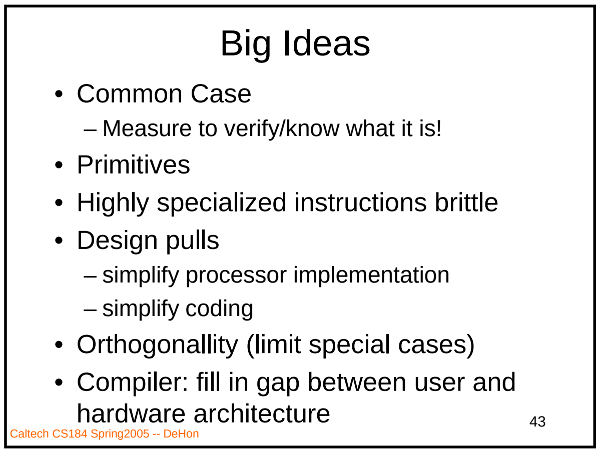
- Smoking gun?:
  - 3-operand instructions
  - One cycle per operand field
  - If field a mem-op, wash with ld/st in RISC
  - If register-op, costs more
- …long way to supporting gap…

ISA Growth

- Can add instructions (upward compatibility)
- Do we ever get to remove any?

RISC

- Everyone believe RISC
  - X86 only one left
  - ...and it’s a RISC core…
- …but do they understand it?
  - Today’s processors pretty complicated
  - Who’s managing instruction scheduling?
  - What mean to FPGAs?



Big Ideas

- Common Case
  - Measure to verify/know what it is!
- Primitives
- Highly specialized instructions brittle
- Design pulls
  - simplify processor implementation
  - simplify coding
- Orthogonallity (limit special cases)
- Compiler: fill in gap between user and hardware architecture