Past and Future Trends in Architecture and Hardware

David Patterson
pattrsn@eecs.berkeley.edu

SOSP History Day October 3, 2015

Outline

Part I - Past
50 years of Computer Architecture History:
- 1960s: Computer Families / Microprogramming
- 1970s: CISC
- 1980s: RISC
- 1990s: VLIW
- 2000s: NUMA vs. Clusters

Part II – Future
HW Technology
- End of Moore's Law
- Flash vs. Disks
- Fast DRAM
- Crosspoint NVRAM

Open ISA & RISC-V
- Case for Open ISAs
- Tour of RISC-V ISA
- RISC-V Software Stack
- RISC-V Chips

IBM Compatibility Problem in early 1960s

By early 1960's, *IBM had*
4 incompatible lines of computers!

701	→	7094
650	→	7074
702	→	7080
1401	→	7010



Each system had its own
- Instruction set
- I/O system and Secondary Storage:
  magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche: business, scientific, real time, ...

⇒ *IBM System/360 – one ISA to rule them all*

IBM 360: A Computer Family

	Model 30	*. . .*	*Model 70*
Storage	8K - 64 KB		256K - 512 KB
Datapath	8-bit		64-bit
Circuit Delay	30 nsec/level		5 nsec/level
Registers	Main Store		Transistor Registers

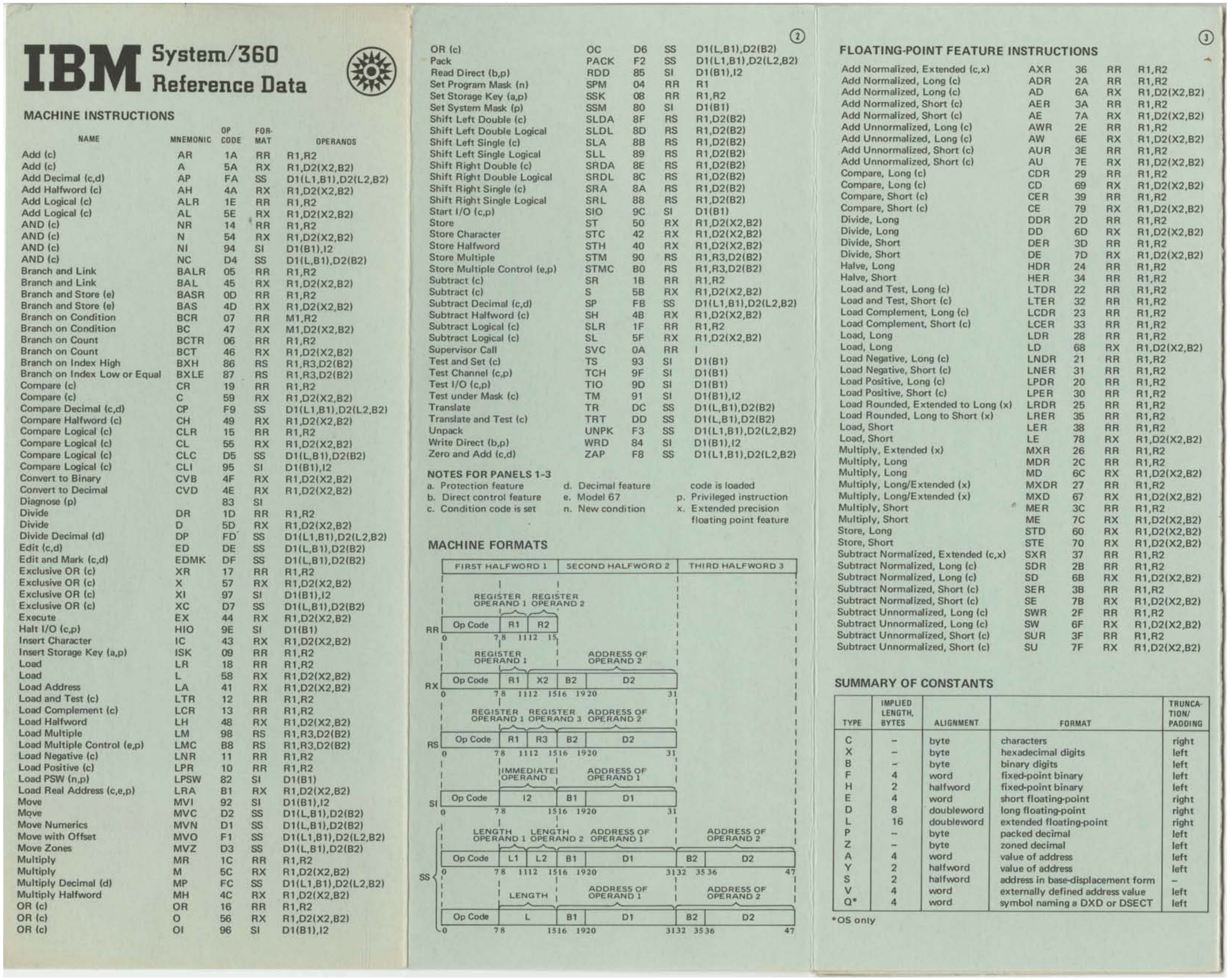
The IBM 360 is why bytes are 8-bits long today!

IBM 360 instruction set architecture (ISA) completely hid the underlying technological differences between various models.

Milestone: The first true ISA designed as portable hardware-software interface!

With minor modifications it still survives today!

IBM System/360 Reference Card ("Green card")



IBM System/360 Reference Data

MACHINE INSTRUCTIONS

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Add (c)	AR	1A	RR	R1,R2
Add (c)	A	5A	RX	R1,D2(X2,B2)
Add Decimal (c,d)	AP	FA	SS	D1(L1,B1),D2(L2,B2)
Add Halfword (c)	AH	4A	RX	R1,D2(X2,B2)
Add Logical (c)	ALR	1E	RR	R1,R2
Add Logical (c)	AL	5E	RX	R1,D2(X2,B2)
AND (c)	NR	14	RR	R1,R2
AND (c)	N	54	RX	R1,D2(X2,B2)
AND (c)	NI	94	SI	D1(B1),I2
AND (c)	NC	D4	SS	D1(L,B1),D2(B2)
Branch and Link	BALR	05	RR	R1,R2
Branch and Link	BAL	45	RX	R1,D2(X2,B2)
Branch and Store (e)	BASR	0D	RR	R1,R2
Branch and Store (e)	BAS	4D	RX	R1,D2(X2,B2)
Branch on Condition	BCR	07	RR	M1,R2
Branch on Condition	BC	47	RX	M1,D2(X2,B2)
Branch on Count	BCTR	06	RR	R1,R2
Branch on Count	BCT	46	RX	R1,D2(X2,B2)
Branch on Index High	BXH	86	RS	R1,R3,D2(B2)
Branch on Index Low or Equal	BXLE	87	RS	R1,R3,D2(B2)
Compare (c)	CR	19	RR	R1,R2
Compare (c)	C	59	RX	R1,D2(X2,B2)
Compare Decimal (c,d)	CP	F9	SS	D1(L1,B1),D2(L2,B2)
Compare Halfword (c)	CH	49	RX	R1,D2(X2,B2)
Compare Logical (c)	CLR	15	RR	R1,R2
Compare Logical (c)	CL	55	RX	R1,D2(X2,B2)
Compare Logical (c)	CLC	D5	SS	D1(L,B1),D2(B2)
Compare Logical (c)	CLI	95	SI	D1(B1),I2
Convert to Binary	CVB	4F	RX	R1,D2(X2,B2)
Convert to Decimal	CVD	4E	RX	R1,D2(X2,B2)
Diagnose (p)		83	SI	
Divide	DR	1D	RR	R1,R2
Divide	D	5D	RX	R1,D2(X2,B2)
Divide Decimal (d)	DP	FD	SS	D1(L1,B1),D2(L2,B2)
Edit (c,d)	ED	DE	SS	D1(L,B1),D2(B2)
Edit and Mark (c,d)	EDMK	DF	SS	D1(L,B1),D2(B2)
Exclusive OR (c)	XR	17	RR	R1,R2
Exclusive OR (c)	X	57	RX	R1,D2(X2,B2)
Exclusive OR (c)	XI	97	SI	D1(B1),I2
Exclusive OR (c)	XC	D7	SS	D1(L,B1),D2(B2)
Execute	EX	44	RX	R1,D2(X2,B2)
Halt I/O (c,p)	HIO	9E	SI	D1(B1)
Insert Character	IC	43	RX	R1,D2(X2,B2)
Insert Storage Key (a,p)	ISK	09	RR	R1,R2
Load	LR	18	RR	R1,R2
Load	L	58	RX	R1,D2(X2,B2)
Load Address	LA	41	RX	R1,D2(X2,B2)
Load and Test (c)	LTR	12	RR	R1,R2
Load Complement (c)	LCR	13	RR	R1,R2
Load Halfword	LH	48	RX	R1,D2(X2,B2)
Load Multiple	LM	98	RS	R1,R3,D2(B2)
Load Multiple Control (e,p)	LMC	B8	RS	R1,R3,D2(B2)
Load Negative (c)	LNR	11	RR	R1,R2
Load Positive (c)	LPR	10	RR	R1,R2
Load PSW (n,p)	LPSW	82	SI	D1(B1)
Load Real Address (c,e,p)	LRA	B1	RX	R1,D2(X2,B2)
Move	MVI	92	SI	D1(B1),I2
Move	MVC	D2	SS	D1(L,B1),D2(B2)
Move Numerics	MVN	D1	SS	D1(L,B1),D2(B2)
Move with Offset	MVO	F1	SS	D1(L1,B1),D2(L2,B2)
Move Zones	MVZ	D3	SS	D1(L,B1),D2(B2)
Multiply	MR	1C	RR	R1,R2
Multiply	M	5C	RX	R1,D2(X2,B2)
Multiply Decimal (d)	MP	FC	SS	D1(L1,B1),D2(L2,B2)
Multiply Halfword	MH	4C	RX	R1,D2(X2,B2)
OR (c)	OR	16	RR	R1,R2
OR (c)	O	56	RX	R1,D2(X2,B2)
OR (c)	OI	96	SI	D1(B1),I2
OR (c)	OC	D6	SS	D1(L,B1),D2(B2)
Pack	PACK	F2	SS	D1(L1,B1),D2(L2,B2)
Read Direct (b,p)	RDD	85	SI	D1(B1),I2
Set Program Mask (n)	SPM	04	RR	R1
Set Storage Key (a,p)	SSK	08	RR	R1,R2
Set System Mask (p)	SSM	80	SI	D1(B1)
Shift Left Double (c)	SLDA	8F	RS	R1,D2(B2)
Shift Left Double Logical	SLDL	8D	RS	R1,D2(B2)
Shift Left Single (c)	SLA	8B	RS	R1,D2(B2)
Shift Left Single Logical	SLL	89	RS	R1,D2(B2)
Shift Right Double (c)	SRDA	8E	RS	R1,D2(B2)
Shift Right Double Logical	SRDL	8C	RS	R1,D2(B2)
Shift Right Single (c)	SRA	8A	RS	R1,D2(B2)
Shift Right Single Logical	SRL	88	RS	R1,D2(B2)
Start I/O (c,p)	SIO	9C	SI	D1(B1)
Store	ST	50	RX	R1,D2(X2,B2)
Store Character	STC	42	RX	R1,D2(X2,B2)
Store Halfword	STH	40	RX	R1,D2(X2,B2)
Store Multiple	STM	90	RS	R1,R3,D2(B2)
Store Multiple Control (e,p)	STMC	B0	RS	R1,R3,D2(B2)
Subtract (c)	SR	1B	RR	R1,R2
Subtract (c)	S	5B	RX	R1,D2(X2,B2)
Subtract Decimal (c,d)	SP	FB	SS	D1(L1,B1),D2(L2,B2)
Subtract Halfword (c)	SH	4B	RX	R1,D2(X2,B2)
Subtract Logical (c)	SLR	1F	RR	R1,R2
Subtract Logical (c)	SL	5F	RX	R1,D2(X2,B2)
Supervisor Call	SVC	0A	RR	I
Test and Set (c)	TS	93	SI	D1(B1)
Test Channel (c,p)	TCH	9F	SI	D1(B1)
Test I/O (c,p)	TIO	9D	SI	D1(B1)
Test under Mask (c)	TM	91	SI	D1(B1),I2
Translate	TR	DC	SS	D1(L,B1),D2(B2)
Translate and Test (c)	TRT	DD	SS	D1(L,B1),D2(B2)
Unpack	UNPK	F3	SS	D1(L1,B1),D2(L2,B2)
Write Direct (b,p)	WRD	84	SI	D1(B1),I2
Zero and Add (c,d)	ZAP	F8	SS	D1(L1,B1),D2(L2,B2)

NOTES FOR PANELS 1-3

a. Protection feature
b. Direct control feature
c. Condition code is set
d. Decimal feature
e. Model 67
n. New condition code is loaded
p. Privileged instruction
x. Extended precision floating point feature

MACHINE FORMATS

FLOATING-POINT FEATURE INSTRUCTIONS

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Add Normalized, Extended (c,x)	AXR	36	RR	R1,R2
Add Normalized, Long (c)	ADR	2A	RR	R1,R2
Add Normalized, Long (c)	AD	6A	RX	R1,D2(X2,B2)
Add Normalized, Short (c)	AER	3A	RR	R1,R2
Add Normalized, Short (c)	AE	7A	RX	R1,D2(X2,B2)
Add Unnormalized, Long (c)	AWR	2E	RR	R1,R2
Add Unnormalized, Long (c)	AW	6E	RX	R1,D2(X2,B2)
Add Unnormalized, Short (c)	AUR	3E	RR	R1,R2
Add Unnormalized, Short (c)	AU	7E	RX	R1,D2(X2,B2)
Compare, Long (c)	CDR	29	RR	R1,R2
Compare, Long (c)	CD	69	RX	R1,D2(X2,B2)
Compare, Short (c)	CER	39	RR	R1,R2
Compare, Short (c)	CE	79	RX	R1,D2(X2,B2)
Divide, Long	DDR	2D	RR	R1,R2
Divide, Long	DD	6D	RX	R1,D2(X2,B2)
Divide, Short	DER	3D	RR	R1,R2
Divide, Short	DE	7D	RX	R1,D2(X2,B2)
Halve, Long	HDR	24	RR	R1,R2
Halve, Short	HER	34	RR	R1,R2
Load and Test, Long (c)	LTDR	22	RR	R1,R2
Load and Test, Short (c)	LTER	32	RR	R1,R2
Load Complement, Long (c)	LCDR	23	RR	R1,R2
Load Complement, Short (c)	LCER	33	RR	R1,R2
Load, Long	LDR	28	RR	R1,R2
Load, Long	LD	68	RX	R1,D2(X2,B2)
Load Negative, Long (c)	LNDR	21	RR	R1,R2
Load Negative, Short (c)	LNER	31	RR	R1,R2
Load Positive, Long (c)	LPDR	20	RR	R1,R2
Load Positive, Short (c)	LPER	30	RR	R1,R2
Load Rounded, Extended to Long (x)	LRDR	25	RR	R1,R2
Load Rounded, Long to Short (x)	LRER	35	RR	R1,R2
Load, Short	LER	38	RR	R1,R2
Load, Short	LE	78	RX	R1,D2(X2,B2)
Multiply, Extended (x)	MXR	26	RR	R1,R2
Multiply, Long	MDR	2C	RR	R1,R2
Multiply, Long	MD	6C	RX	R1,D2(X2,B2)
Multiply, Long/Extended (x)	MXDR	27	RR	R1,R2
Multiply, Long/Extended (x)	MXD	67	RX	R1,D2(X2,B2)
Multiply, Short	MER	3C	RR	R1,R2
Multiply, Short	ME	7C	RX	R1,D2(X2,B2)
Store, Long	STD	60	RX	R1,D2(X2,B2)
Store, Short	STE	70	RX	R1,D2(X2,B2)
Subtract Normalized, Extended (c,x)	SXR	37	RR	R1,R2
Subtract Normalized, Long (c)	SDR	2B	RR	R1,R2
Subtract Normalized, Long (c)	SD	6B	RX	R1,D2(X2,B2)
Subtract Normalized, Short (c)	SER	3B	RR	R1,R2
Subtract Normalized, Short (c)	SE	7B	RX	R1,D2(X2,B2)
Subtract Unnormalized, Long (c)	SWR	2F	RR	R1,R2
Subtract Unnormalized, Long (c)	SW	6F	RX	R1,D2(X2,B2)
Subtract Unnormalized, Short (c)	SUR	3F	RR	R1,R2
Subtract Unnormalized, Short (c)	SU	7F	RX	R1,D2(X2,B2)

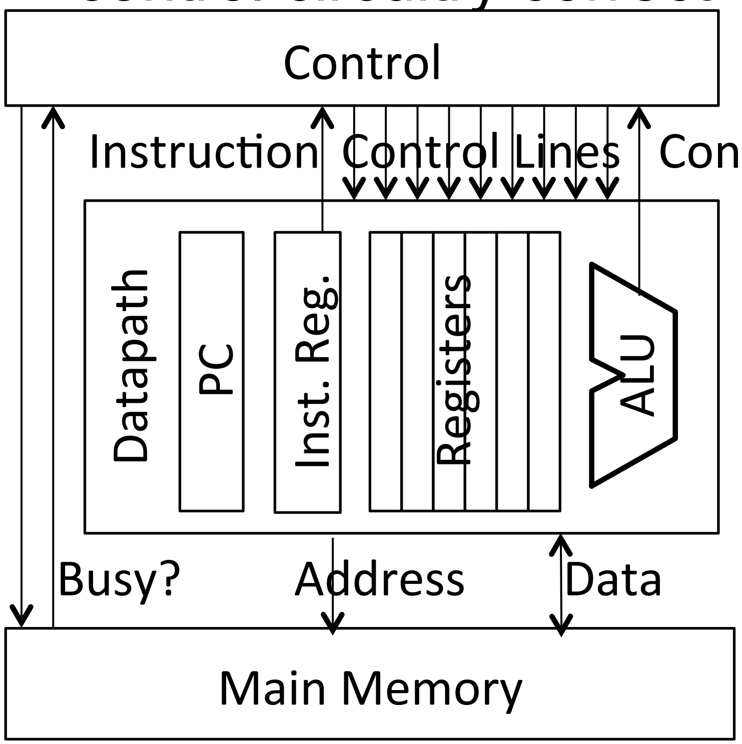
SUMMARY OF CONSTANTS

TYPE	IMPLIED LENGTH, BYTES	ALIGNMENT	FORMAT	TRUNCATION/PADDING
C	–	byte	characters	right
X	–	byte	hexadecimal digits	left
B	–	byte	binary digits	left
F	4	word	fixed-point binary	left
H	2	halfword	fixed-point binary	left
E	4	word	short floating-point	right
D	8	doubleword	long floating-point	right
L	16	doubleword	extended floating-point	right
P	–	byte	packed decimal	left
Z	–	byte	zoned decimal	left
A	4	word	value of address	left
Y	2	halfword	value of address	left
S	2	halfword	address in base-displacement form	–
V	4	word	externally defined address value	left
Q*	4	word	symbol naming a DXD or DSECT	left

*OS only

Control versus Datapath

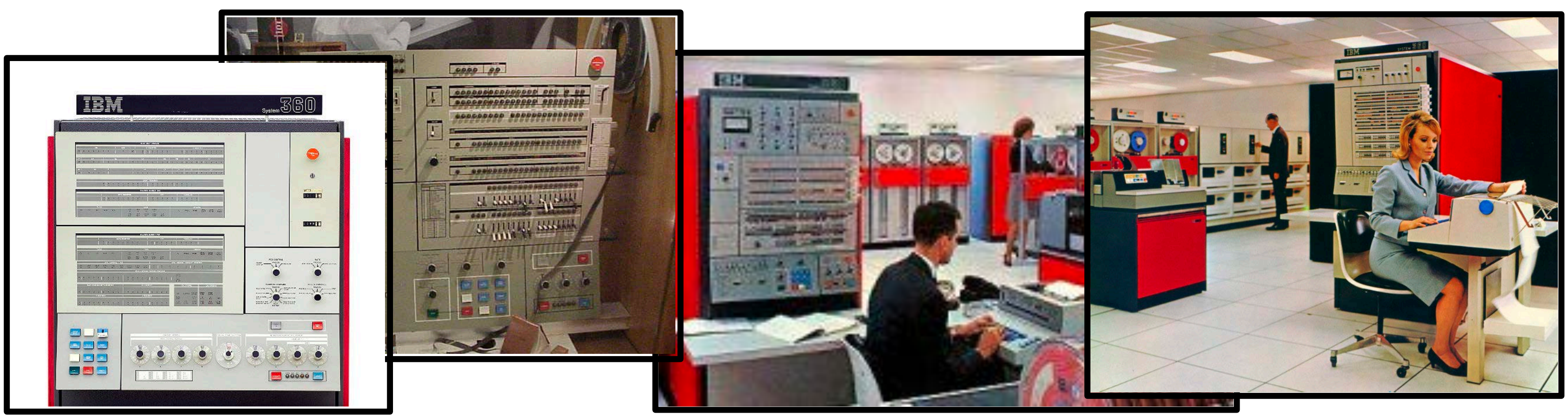
- Processor designs can be split between *datapath*, where numbers are stored and arithmetic operations computed, and *control*, which sequences operations on datapath
- Biggest challenge for early computer designers was getting control circuitry correct



- Maurice Wilkes invented the idea of *microprogramming* to design the control unit of a processor, 1958
  - Logic expensive compared to ROM or RAM
  - ROM cheaper than RAM
  - ROM much faster than RAM

Microprogramming in IBM 360

Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Microcode size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock cycle time (ROM)	750 ns	625 ns	500 ns	200 ns
Main memory cycle time	1500 ns	2500 ns	2000 ns	750 ns
Annual rental fee (1964 $)	$48,000	$54,000	$115,000	$270,000
Annual rental fee (2015 $)	$570,000	$650,000	$1,400,000	$3,200,000

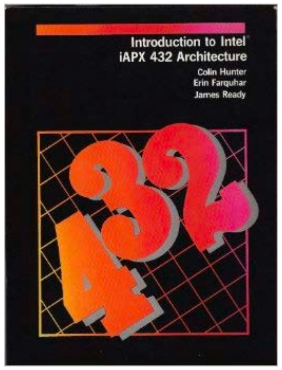


IC technology, Microcode, and CISC

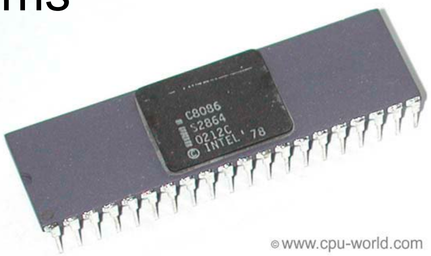
- Logic, RAM, ROM all implemented using MOS transistors
- Semiconductor RAM ≈ same speed as ROM
- With Moore's Law, memory for control store could grow
- Allowed more complicated instruction sets (CISC)
- Minicomputer (TTL server) Example: Digital Equipment VAX ISA in 1978

Microprocessor Evolution

- Rapid progress in 1970s, fueled by advances in MOS technology, imitated minicomputers and mainframe ISAs
- Intel i432
  - Most ambitious 1970s micro
  - started in 1975 - released 1981
  - 32-bit capability-based object-oriented architecture
  - Instructions variable number of bits long
  - Heavily microcoded
  - Severe performance, complexity, and usability problems

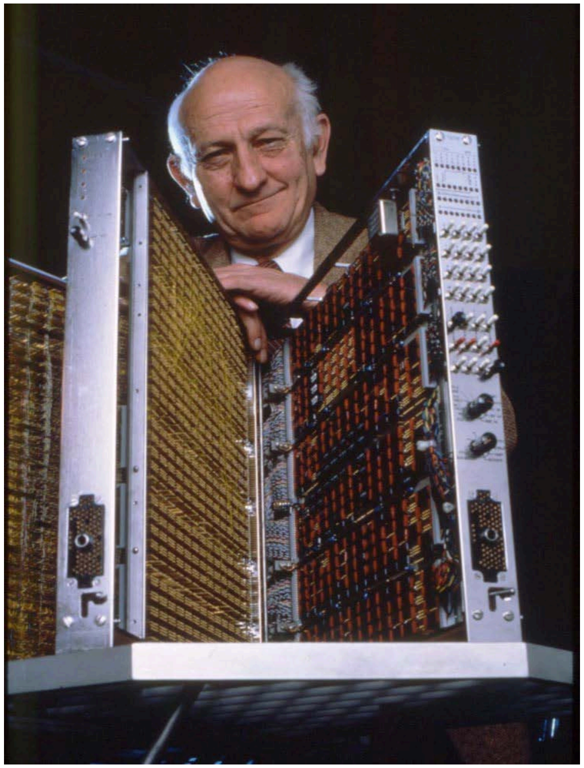


- Intel 8086 (1978, 8MHz, 29,000 transistors)
  - "Stopgap" 16-bit processor, architected in 10 weeks
  - Extended accumulator architecture
  - Assembly-compatible with 8080
  - 20-bit addressing through segmented addressing scheme



- IBM PC uses Intel 8088 for 8-bit bus (and Motorola 68000 was late)
  - Estimated sales of 250,000; 100,000,000s sold

Analyzing Microcoded Machines 1980s



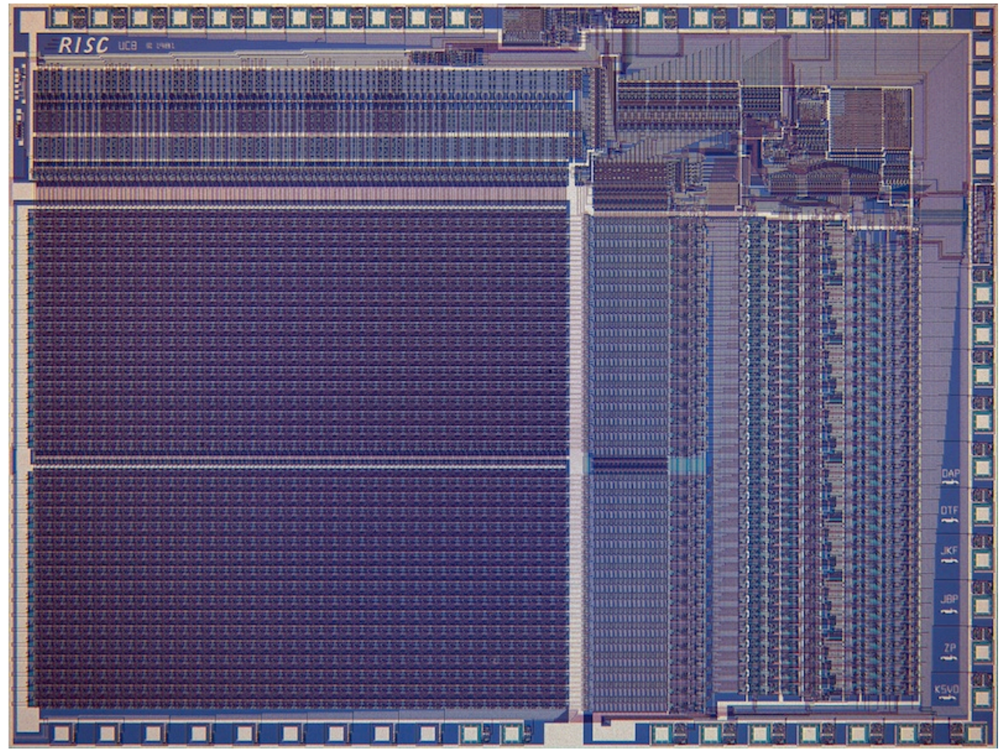
- John Cocke and group at IBM
  - Working on a simple pipelined processor, 801 minicomputer (ECL server), and advanced compilers inside IBM
  - Ported experimental PL.8 compiler to IBM 370, only used simple register-register and load/store instructions similar to 801
  - Code ran faster than other existing compilers that used all 370 instructions!
  - Up to 6 MIPS whereas 2 MIPS considered good before
- Emer and Clark at DEC
  - Found 20% of VAX instructions responsible for 60% of microcode, but only account for 0.2% of execution time!
- Patterson 1979 sabbatical at DEC
  - VAX microcode bugs ⇒ field repair, but field-repairable chips don't make sense

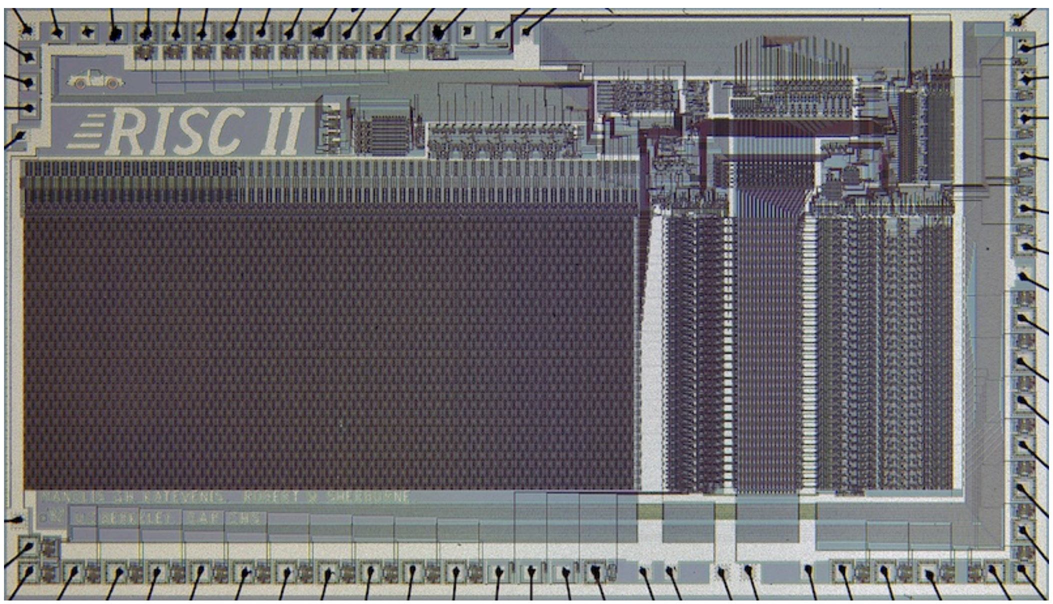
From CISC to RISC

- Use fast RAM to build fast instruction *cache* of user-visible instructions, not fixed hardware microroutines
  - Contents of fast instruction memory change to fit what application needs right now
- Simple ISA => hardwired pipelined implementation
  - Compiled code only used a few CISC instructions
  - Simpler encoding allowed pipelined implementations
- Further benefit with integration
  - In early ‘80s, could finally fit 32-bit datapath + small caches on a single chip
    - No chip crossings in common case allows faster operation

Berkeley RISC Chips

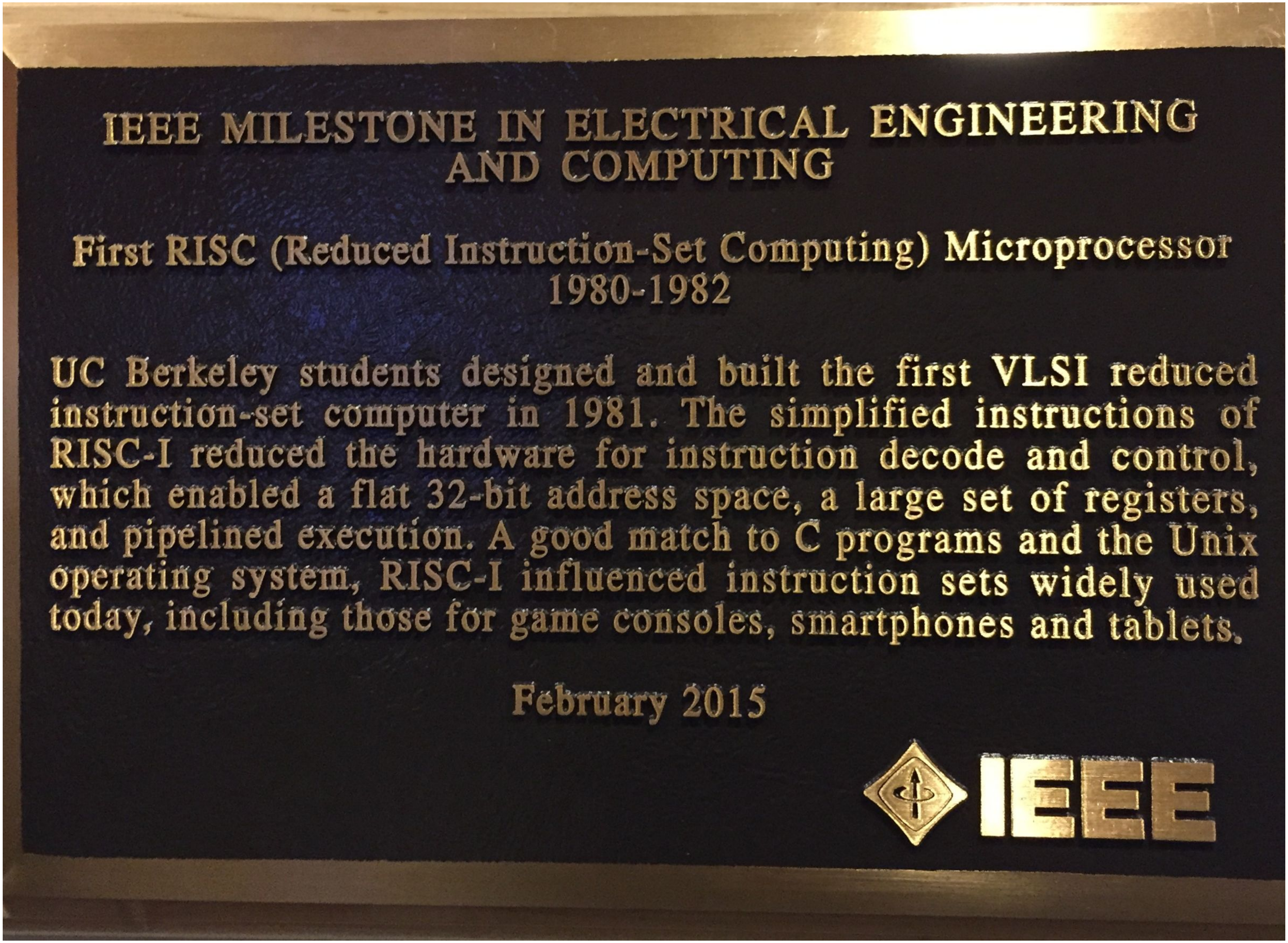
RISC-I (1982) Contains 44,420 transistors, fabbed in 5 µm NMOS, with a die area of 77 mm², ran at 1 MHz.





RISC-II (1983) contains 40,760 transistors, was fabbed in 3 µm NMOS, ran at 3 MHz, and the size is 60 mm²

Stanford built some too…

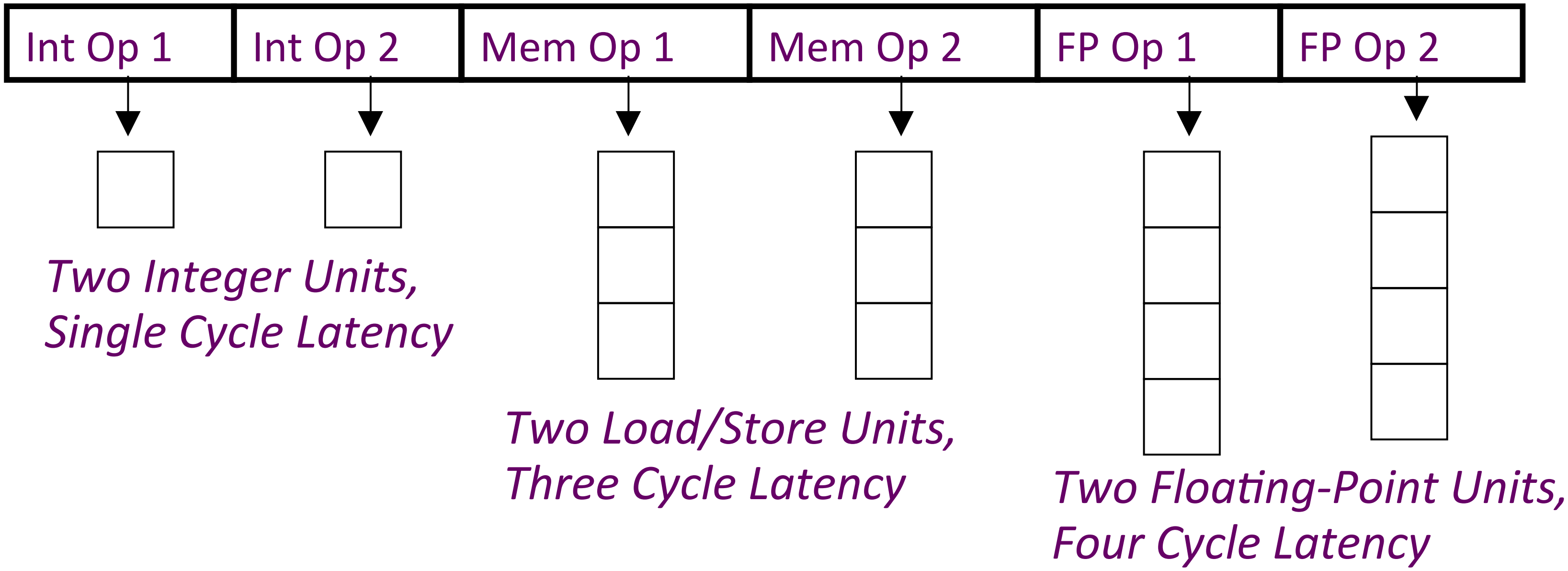

IEEE MILESTONE IN ELECTRICAL ENGINEERING AND COMPUTING
First RISC (Reduced Instruction-Set Computing) Microprocessor
1980-1982
UC Berkeley students designed and built the first VLSI reduced instruction-set computer in 1981. The simplified instructions of RISC-I reduced the hardware for instruction decode and control, which enabled a flat 32-bit address space, a large set of registers, and pipelined execution. A good match to C programs and the Unix operating system, RISC-I influenced instruction sets widely used today, including those for game consoles, smartphones and tablets.
February 2015
IEEE

CISC vs. RISC Today

- PC Era
- Hardware translates x86 instructions into internal RISC instructions
- Then use any RISC technique inside MPU
- > 350M / year !
- x86 ISA eventually dominates servers as well as desktops

- PostPC Era: Client/Cloud
- IP in SoC vs. MPU
- Value die area, energy as much as performance
- > 16B / year in 2014!
- 98% RISC Processors:
  - 12.0B ARM (Advanced RISC Machine)
  - 2.0B Tensilica
  - 1.5B ARC (Argonaut RISC Core)
  - 0.8B MIPS

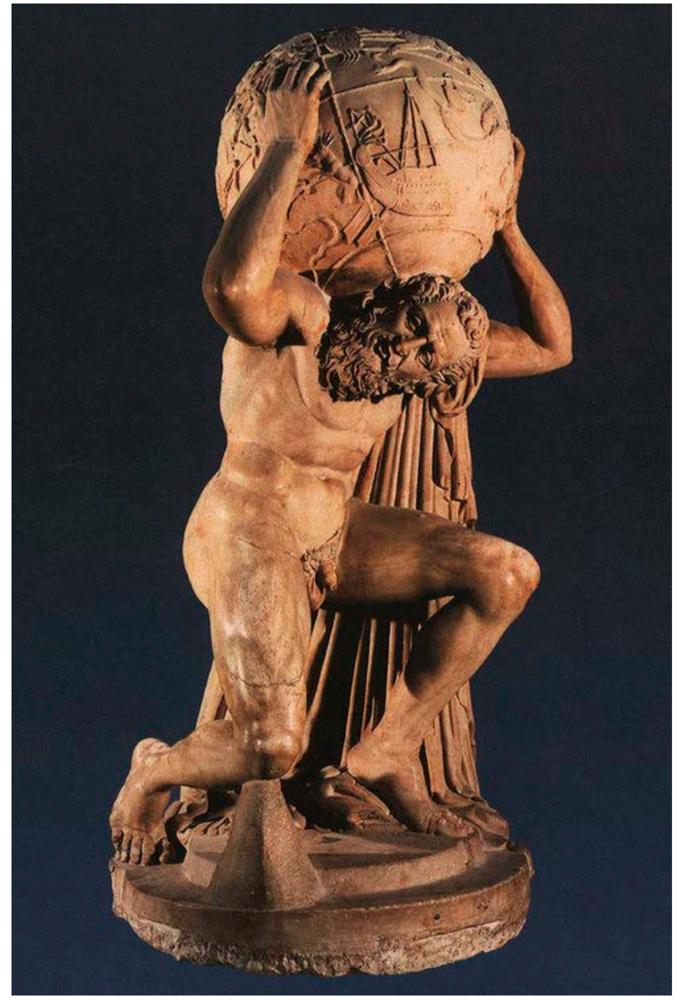
VLIW: Very Long Instruction Word



- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
  - Parallelism within an instruction => no cross-operation RAW check
  - No data use before data ready => no data interlocks

VLIW Compiler Responsibilities

- Schedule operations to maximize parallel execution
- Guarantees intra-instruction parallelism
- Schedule to avoid data hazards (no interlocks)
  - Typically separates operations with explicit NOPs



Loop Unrolling

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Unroll inner loop to perform 4 iterations at once

```
for (i=0; i<N; i+=4)
{
    B[i]     = A[i] + C;
    B[i+1] = A[i+1] + C;
    B[i+2] = A[i+2] + C;
    B[i+3] = A[i+3] + C;
}
```

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:  fld f1, 0(x1)
       fld f2, 8(x1)
       fld f3, 16(x1)
       fld f4, 24(x1)
       add x1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       fsd f5, 0(x2)
       fsd f6, 8(x2)
       fsd f7, 16(x2)
       fsd f8, 24(x2)
       add x2, 32
       bne x1, x3, loop
```

Schedule →

	Int1	Int 2	M1	M2	FP+	FPx
loop:			fld f1			
			fld f2			
			fld f3			
	add x1		fld f4		fadd f5	
					fadd f6	
					fadd f7	
					fadd f8	
			fsd f5			
			fsd f6			
			fsd f7			
	add x2	bne	fsd f8			

How many FLOPS/cycle?

4 fadds / 11 cycles = 0.36

Intel Itanium, EPIC IA-64

- EPIC is the style of architecture
  - "Explicitly Parallel Instruction Computing"
  - A binary object-code-compatible VLIW
  - Developed jointly with HP
- IA-64 was Intel's chosen 64b ISA successor to 32b x86
  - IA-64 = Intel Architecture 64-bit
  - AMD wouldn't be able to make, unlike x86
- Intel Merced was first Itanium implementation
  - 1st customer shipment expected 1997 (actually 2001)
  - McKinley, 2nd implementation, 180 nm, shipped in 2002
  - Poulson, most recent, 8 cores, 32 nm, shipped in 2012

VLIW Issues and an "EPIC Failure"

- Unpredictable branches
- Variable memory latency (unpredictable cache misses)
- Code size explosion
- Compiler complexity: "*The Itanium approach...was supposed to be so terrific—until it turned out that the wished-for compilers were basically impossible to write*."
  - Donald Knuth, Stanford
- Columnist Ashlee Vance noted delays and under performance of Itanium "*turned the product into a joke in the chip industry*"

Itanimum => "Itanic" (like infamous ship *Titanic*)

2000s: How Should We Build Scalable Multiprocessors?

1. Shared Memory with "Non Uniform Memory Access" time (NUMA) using loads and stores
   - Distributed directory remembers sharing for coherency and consistency
   - DASH/FLASH projects at Stanford (1992-2000)
2. Message passing Cluster with separate address space per processor using RPC (or MPI)
   - Collection of independent computers connected by LAN switches to provide a common service
   - Network of Workstations project at Berkeley (1993-1998)

SGI Origin 2000 NUMA vs. Sun Enterprise 10000 SMP

- A pure NUMA
- Scales up to 2048 CPUs
- Scalable bandwidth is crucial to Origin
- Designed for scientific computation

- A pure UMA
- Up to 64 CPUs
- $4.7M = 64 CPUs, 64 GB SDRAM memory, 868 18GB disk, 12X CD, 1yr service
- Designed for commercial processing

NUMA Advantages

- Ease of programming when communication patterns are complex or vary dynamically during execution
- Ability to develop apps using familiar SMP model
- Lower communication overhead, better use of BW for small items due to implicit communication
- HW-controlled caching to reduce remote communication by caching of all data

Cluster Drawbacks

- Cost of administering a cluster of N machines ~ administering N independent machines vs. cost of administering a shared address space N processors multiprocessor ~ administering 1 big machine
- Clusters usually connected using I/O bus, whereas multiprocessors usually connected on memory bus
- Cluster of N machines has N independent memories and N copies of OS and code, but a shared address multi-processor allows 1 program to use almost all memory

Cluster Advantages

- Error isolation: separate address space limits contamination of error
- Repair: Easier to replace a machine without bringing down the system
- Scale: easier to expand the system
- Cost: Large scale machine has low volume => fewer machines to spread development costs vs. leverage high volume off-the-shelf switches and computers
- Inktomi first then Amazon, AOL, Google, Hotmail, WebTV, Yahoo … relied on clusters of PCs to provide services used by millions of people every day

Review: Networking

- Clusters +: fault isolation and repair, scaling, cost
- Clusters -: maintenance, network interface performance, memory efficiency
- Google as cluster example:
  - scaling (6000 PCs, 1 petabyte storage)
  - fault isolation (2 failures per day yet available)
  - repair (replace failures weekly/repair offline)
  - Maintenance: 8 people for 6000 PCs
- Cell phone as portable network device
  - # Handsets >> # PCs
  - Universal mobile interface?
- Is future services built on Google-like clusters delivered to gadgets like cell phone handset?

Outline

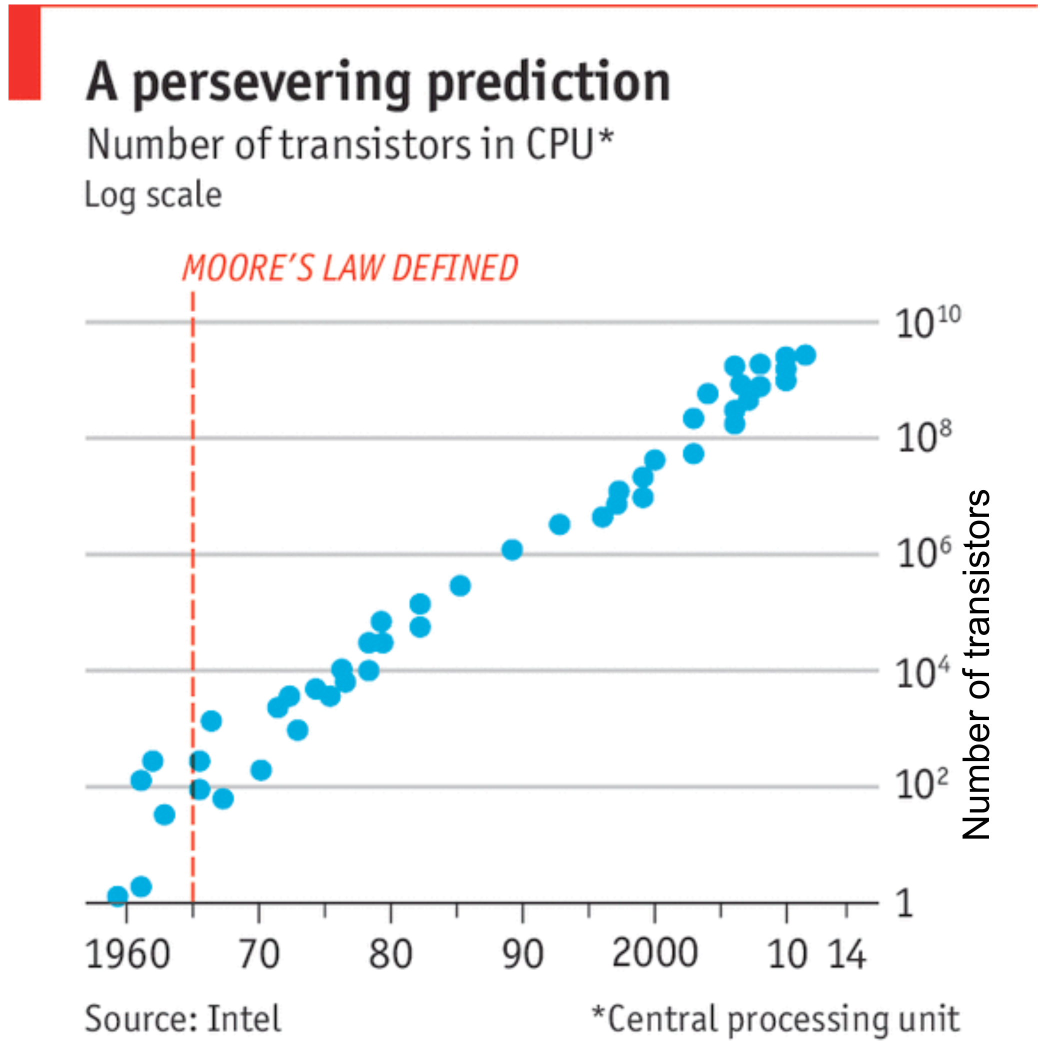
Part I - Past
50 years of Computer Architecture History:
- 1960s: Computer Families / Microprogramming
- 1970s: CISC
- 1980s: RISC
- 1990s: VLIW
- 2000s: NUMA vs. Clusters

Part II – Future
HW Technology
- End of Moore's Law
- Flash vs. Disks
- Fast DRAM
- Crosspoint NVRAM

Open ISA / RISC-V
- Case for Open ISAs
- Tour of RISC-V ISA
- RISC-V Software Stack
- RISC-V Chips

Moore's Law Slowing Down

- Stated 50 years ago by Gordon Moore
  - -Number of transistors on microchip double every 1-2 years
  - -Today 2.5-3? years



Economist.com

CPU Performance Improvement

- Number of cores: +18-20%
- Per core performance: +10%
- Aggregate improvement: +30-32%

Memory Price/Byte Evolution

- 1990-2000: -54% per year
- 2000-2010: -51% per year
- 2010-2015: -32% per year
- (http://www.jcmit.com/memoryprice.htm)

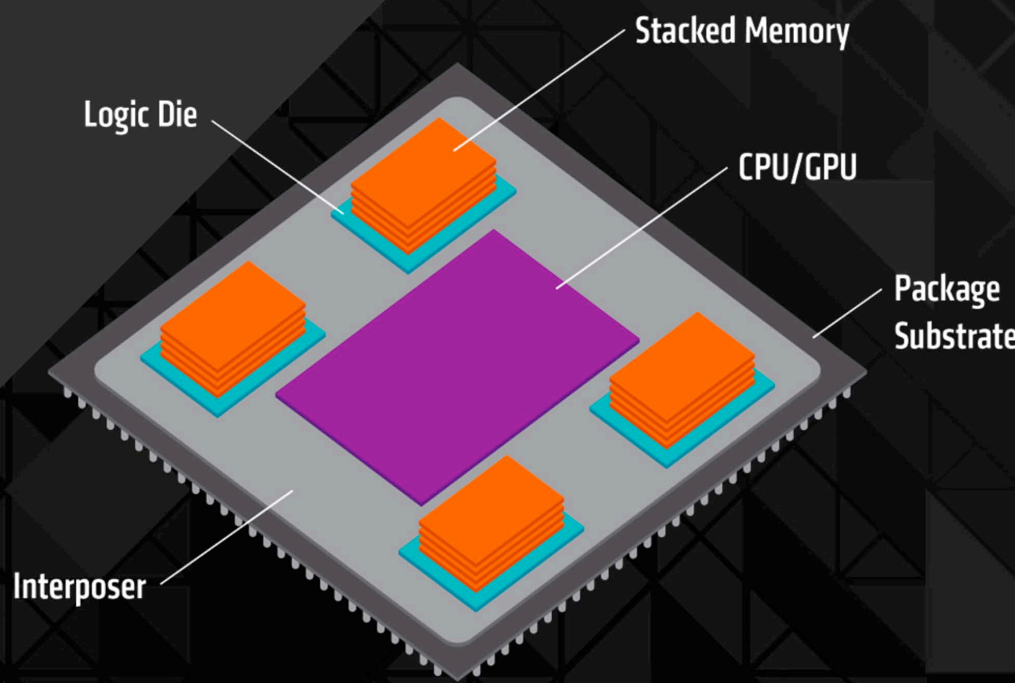
High Bandwidth Memory

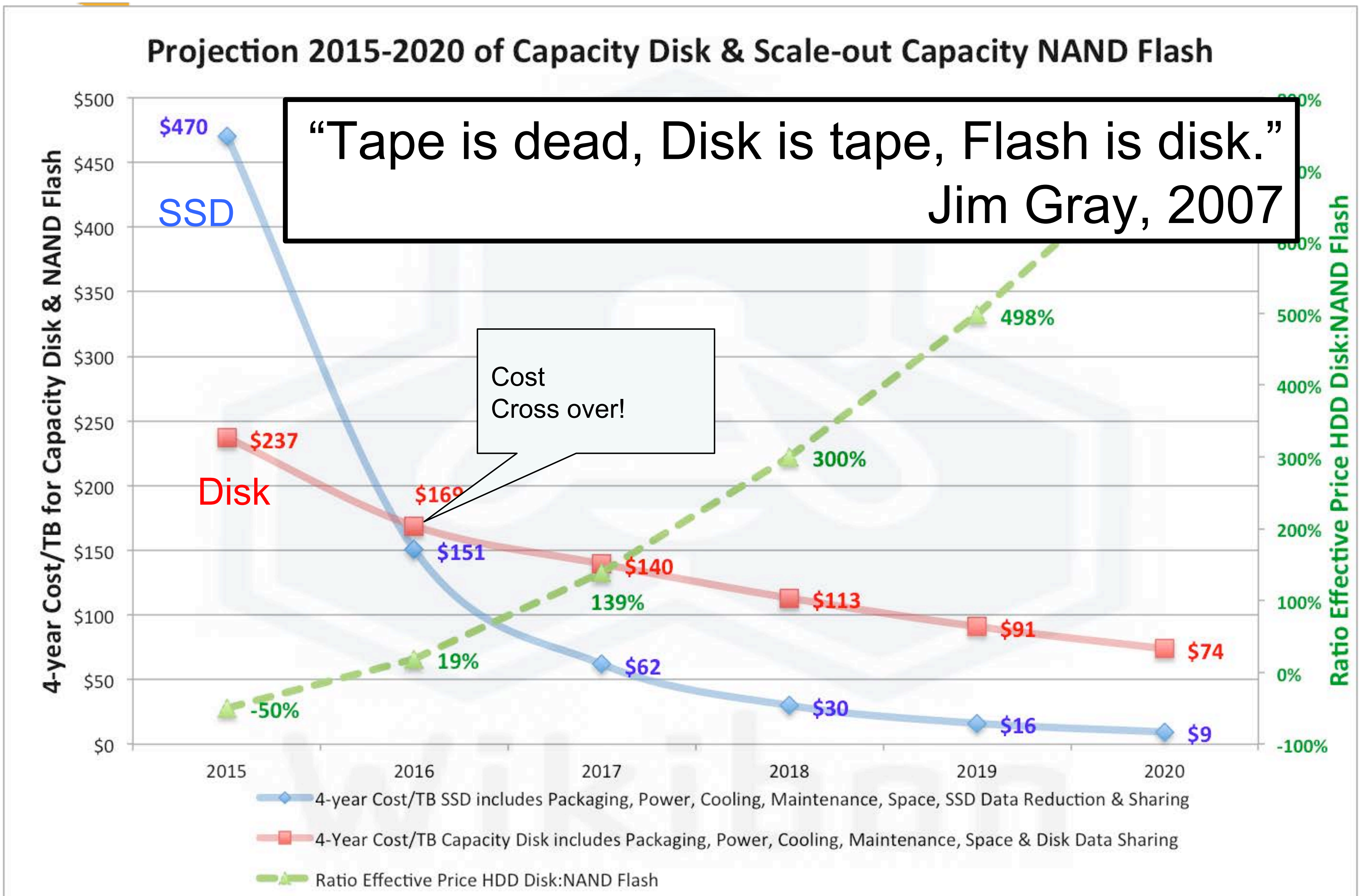
THE INTERPOSER

THE NEXT STEP IN INTEGRATION

AMD

- Brings DRAM as close as possible to the logic die
- Improving proximity enables extremely wide bus widths
- Improving proximity simplifies communication and clocking
- Improving proximity greatly improves bandwidth per watt
- Allows for integration of disparate technologies such as DRAM
- AMD developed industry partnerships with ASE, Amkor & UMC to develop the first high-volume manufacturable interposer solution





Source: © Wikibon 2015. 4-Year Cost/TB Magnetic Disk & SSD, including Packaging, Power, Maintenance, Space, Data Reduction & Data Sharing

3D XPoint Technology

- Developed by Intel and Micron
  - Announced July 28, 2015!

- Exceptional characteristics:
  - Non-volatile memory
  - 1000x more resilient than SSDs
  - 8-10x density of DRAM
  - Performance in DRAM ballpark!
    - 2-3x slower reads, 4x-6x slower writes

Future Memory Hierarchy Deeper

- Storage hierarchy gets more and more complex:
  - L1 cache
  - L2 cache
  - L3 cache
  - Fast DRAM (on interposer with CPU)
  - 3D XPoint based storage
  - SSD
  - (HDD)
- Need to design software to take advantage of this hierarchy

Consensus on ISAs Today

- Not CISC: no new commercial CISC ISAs in 30+ years
- Not VLIW: Despite several attempts,
  VLIW has failed in general-purpose computing arena
  - Complex VLIW architectures close to in-order superscalar in complexity, no real advantage on large complex apps
  - Although some VLIWs successful in embedded DSP market (Simpler VLIWs, more constrained, friendlier code)
- RISC! Widespread agreement (still) that RISC principles are best for general purpose ISA

So…

If there is widespread agreement on ISA principles …

Why isn't there a free, open, industry-standard ISA?

ISAs Should Be Free and Open

While ISAs may be proprietary for historical or business reasons, there is no good technical reason for the lack of free, open ISAs:

- It's not an error of omission
- Nor is it because the companies do most of the software development
- Neither do companies exclusively have the experience needed to design a competent ISA
- Nor are the most popular ISAs wonderful ISAs
- Neither can only companies verify ISA compatibility
- Nor does it protect you from patent lawsuits
- Finally, proprietary ISAs are not guaranteed to last, and many actually disappear

Why Open ISA Now?

1. Switch from microprocessors of PC Era to IP in SoC of PostPC Era
   ⇒ Can offer designs (as ARM does) without offering chips (as Intel does)
2. Ending of Moore's Law
   ⇒ Cost/performance/energy advance via architectural innovation vs. semiconductor process improvements
   ⇒ Renaissance for domain specific coprocessor (e.g., image processor, DSP, GPU, …)
   ⇒ Want a minimal, open ISA to run standard software with domain specific coprocessors

RISC-V Origin Story

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, time to look at ISA for next set of projects
- x86 and ARM obvious choices, but complex ISAs and serious IP issues
- MIPS64 – not enough opcodes left if try to extend
- So we started "3-month project" in summer 2010 to develop our own clean-slate ISA
- Four years later, we released frozen base user spec
  - Also many tape outs and research publications
- Why are Outsiders complaining about changes to RISC-V in Berkeley classes???

Modest RISC-V Goal

Become an industry-standard ISA for all computing devices

RISC-V Base Plus Standard Extensions

- Three base integer ISAs, one per address width
  - RV32I, RV64I, RV128I
  - Minimal: <50 hardware instructions needed
- Modular: Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision floating-point
  - Q: Quad-precision floating-point
  - C: Compressed instruction encoding (16b and 32b)
- Reserved opcode space for SoC unique instructions
- All the above in fairly standard RISC encoding

RV32I

Category	Name	Fmt	RV32I Base	
Base Integer Instructions: RV32I, RV				
Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Word	I	LW	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	LHU	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	SW	rs1,rs2,imm
Shifts	Shift Left	R	SLL	rd,rs1,rs2
	Shift Left Immediate	I	SLLI	rd,rs1,shamt
	Shift Right	R	SRL	rd,rs1,rs2
	Shift Right Immediate	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Logical	XOR	R	XOR	rd,rs1,rs2
	XOR Immediate	I	XORI	rd,rs1,imm
	OR	R	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches	Branch =	SB	BEQ	rs1,rs2,imm
	Branch ≠	SB	BNE	rs1,rs2,imm
	Branch <	SB	BLT	rs1,rs2,imm
	Branch ≥	SB	BGE	rs1,rs2,imm
	Branch < Unsigned	SB	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	SB	BGEU	rs1,rs2,imm
Jump & Link	J&L	UJ	JAL	rd,imm
	Jump & Link Register	UJ	JALR	rd,rs1,imm
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
System	System CALL	I	SCALL	
	System BREAK	I	SBREAK	
Counters	ReaD CYCLE	I	RDCYCLE	rd
	ReaD CYCLE upper Half	I	RDCYCLEH	rd
	ReaD TIME	I	RDTIME	rd
	ReaD TIME upper Half	I	RDTIMEH	rd
	ReaD INSTR RETired	I	RDINSTRET	rd
	ReaD INSTR upper Half	I	RDINSTRETH	rd

32-bit Instruction Formats

	31	30 – 25	24 – 21	20	19 – 15	14 – 12	11 – 8	7	6 – 0
R	funct7		rs2		rs1	funct3	rd		opcode
I	imm[11:0]				rs1	funct3	rd		opcode
S	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
SB	imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode
U	imm[31:12]						rd		opcode
UJ	imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode

+ 12 for 64I /128I

+ 30 for C

14 Privileged

+ 8 for M

+ 11 for A

+ 34 for F, D, Q

+ 4 for 64M /128M

+ 11 for 64A /128A

+ 6 for 64F/ 128F, 64D/ 128D, 64Q/ 128Q

RV32I / RV64I / RV128I + C, M, A, F, D,& Q
RISC-V "Green Card"

Base Integer Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base	+RV{64,128}
Loads	Load Byte	I	LB rd,rs1,imm	
	Load Halfword	I	LH rd,rs1,imm	
	Load Word	I	LW rd,rs1,imm	L{D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm	
	Load Half Unsigned	I	LHU rd,rs1,imm	L{W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm	
	Store Halfword	S	SH rs1,rs2,imm	
	Store Word	S	SW rs1,rs2,imm	S{D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL rd,rs1,rs2	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI rd,rs1,shamt	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL rd,rs1,rs2	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI rd,rs1,shamt	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI rd,rs1,shamt	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUB{W|D} rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Logical	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Compare	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Branches	Branch =	SB	BEQ rs1,rs2,imm	
	Branch ≠	SB	BNE rs1,rs2,imm	
	Branch <	SB	BLT rs1,rs2,imm	
	Branch ≥	SB	BGE rs1,rs2,imm	
	Branch < Unsigned	SB	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm	
Jump & Link	J&L	UJ	JAL rd,imm	
	Jump & Link Register	UJ	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
System	System CALL	I	SCALL	
	System BREAK	I	SBREAK	
Counters	ReaD CYCLE	I	RDCYCLE rd	
	ReaD CYCLE upper Half	I	RDCYCLEH rd	
	ReaD TIME	I	RDTIME rd	
	ReaD TIME upper Half	I	RDTIMEH rd	
	ReaD INSTR RETired	I	RDINSTRET rd	
	ReaD INSTR upper Half	I	RDINSTRETH rd	

RV Privileged Instructions

Category	Name	RV mnemonic
CSR Access	Atomic R/W	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	CSRRC rd,csr,rs1
	Atomic R/W Imm	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	CSRRCI rd,csr,imm
Change Level	Env. Call	ECALL
	Environment Breakpoint	EBREAK
	Environment Return	ERET
Trap Redirect	to Supervisor	MRTS
	Redirect Trap to Hypervisor	MRTH
	Hypervisor Trap to Supervisor	HRTS
Interrupt	Wait for Interrupt	WFI
MMU	Supervisor FENCE	SFENCE.VM rs1

Optional Compressed (16-bit) Instruction Extension: RVC

Category	Name	Fmt	RVC	RVI equivalent
Loads	Load Word	CL	C.LW rd′,rs1′,imm	LW rd′,rs1′,imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD rd′,rs1′,imm	LD rd′,rs1′,imm*8
	Load Double SP	CI	C.LDSP rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ rd′,rs1′,imm	LQ rd′,rs1′,imm*16
	Load Quad SP	CI	C.LQSP rd,imm	LQ rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1′,rs2′,imm	SW rs1′,rs2′,imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Store Double	CS	C.SD rs1′,rs2′,imm	SD rs1′,rs2′,imm*8
	Store Double SP	CSS	C.SDSP rs2,imm	SD rs2,sp,imm*8
	Store Quad	CS	C.SQ rs1′,rs2′,imm	SQ rs1′,rs2′,imm*16
	Store Quad SP	CSS	C.SQSP rs2,imm	SQ rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Word	CR	C.ADDW rd,rs1	ADDW rd,rd,imm
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm	ADDIW rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd′,imm	ADDI rd′,sp,imm*4
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1′,imm	BEQ rs1′,x0,imm
	Branch≠0	CB	C.BNEZ rs1′,imm	BNE rs1′,x0,imm
Jump	Jump	CJ	C.J imm	JAL x0,imm
	Jump Register	CR	C.JR rd,rs1	JALR x0,rs1,0
Jump & Link	J&L	CJ	C.JAL imm	JAL ra,imm
	Jump & Link Register	CR	C.JALR rs1	JALR ra,rs1,0
System	Env. BREAK	CI	C.EBREAK	EBREAK

Optional Multiply-Divide Instruction Extension: RVM

Category	Name	Fmt	RV32M (Multiply-Divide)	+RV{64,128}
Multiply	MULtiply	R	MUL rd,rs1,rs2	MUL{W|D} rd,rs1,rs2
	MULtiply upper Half	R	MULH rd,rs1,rs2	
	MULtiply Half Sign/Uns	R	MULHSU rd,rs1,rs2	
	MULtiply upper Half Uns	R	MULHU rd,rs1,rs2	
Divide	DIVide	R	DIV rd,rs1,rs2	DIV{W|D} rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2	
Remainder	REMainder	R	REM rd,rs1,rs2	REM{W|D} rd,rs1,rs2
	REMainder Unsigned	R	REMU rd,rs1,rs2	REMU{W|D} rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV32A (Atomic)	+RV{64,128}
Load	Load Reserved	R	LR.W rd,rs1	LR.{D|Q} rd,rs1
Store	Store Conditional	R	SC.W rd,rs1,rs2	SC.{D|Q} rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W rd,rs1,rs2	AMOSWAP.{D|Q} rd,rs1,rs2
Add	ADD	R	AMOADD.W rd,rs1,rs2	AMOADD.{D|Q} rd,rs1,rs2
Logical	XOR	R	AMOXOR.W rd,rs1,rs2	AMOXOR.{D|Q} rd,rs1,rs2
	AND	R	AMOAND.W rd,rs1,rs2	AMOAND.{D|Q} rd,rs1,rs2
	OR	R	AMOOR.W rd,rs1,rs2	AMOOR.{D|Q} rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W rd,rs1,rs2	AMOMIN.{D|Q} rd,rs1,rs2
	MAXimum	R	AMOMAX.W rd,rs1,rs2	AMOMAX.{D|Q} rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W rd,rs1,rs2	AMOMINU.{D|Q} rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W rd,rs1,rs2	AMOMAXU.{D|Q} rd,rs1,rs2

Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVQ

Category	Name	Fmt	RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)	+RV{64,128}
Move	Move from Integer	R	FMV.{H|S}.X rd,rs1	FMV.{D|Q}.X rd,rs1
	Move to Integer	R	FMV.X.{H|S} rd,rs1	FMV.X.{D|Q} rd,rs1
Convert	Convert from Int	R	FCVT.{H|S|D|Q}.W rd,rs1	FCVT.{H|S|D|Q}.{L|T} rd,rs1
	Convert from Int Unsigned	R	FCVT.{H|S|D|Q}.WU rd,rs1	FCVT.{H|S|D|Q}.{L|T}U rd,rs1
	Convert to Int	R	FCVT.W.{H|S|D|Q} rd,rs1	FCVT.{L|T}.{H|S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{H|S|D|Q} rd,rs1	FCVT.{L|T}U.{H|S|D|Q} rd,rs1
Load	Load	I	FL{W,D,Q} rd,rs1,imm	
Store	Store	S	FS{W,D,Q} rs1,rs2,imm	
Arithmetic	ADD	R	FADD.{S|D|Q} rd,rs1,rs2	
	SUBtract	R	FSUB.{S|D|Q} rd,rs1,rs2	
	MULtiply	R	FMUL.{S|D|Q} rd,rs1,rs2	
	DIVide	R	FDIV.{S|D|Q} rd,rs1,rs2	
	SQuare RooT	R	FSQRT.{S|D|Q} rd,rs1	
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q} rd,rs1,rs2,rs3	
	Multiply-SUBtract	R	FMSUB.{S|D|Q} rd,rs1,rs2,rs3	
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q} rd,rs1,rs2,rs3	
	Negative Multiply-ADD	R	FNMADD.{S|D|Q} rd,rs1,rs2,rs3	
Sign Inject	SiGN source	R	FSGNJ.{S|D|Q} rd,rs1,rs2	
	Negative SiGN source	R	FSGNJN.{S|D|Q} rd,rs1,rs2	
	Xor SiGN source	R	FSGNJX.{S|D|Q} rd,rs1,rs2	
Min/Max	MINimum	R	FMIN.{S|D|Q} rd,rs1,rs2	
	MAXimum	R	FMAX.{S|D|Q} rd,rs1,rs2	
Compare	Compare Float =	R	FEQ.{S|D|Q} rd,rs1,rs2	
	Compare Float <	R	FLT.{S|D|Q} rd,rs1,rs2	
	Compare Float ≤	R	FLE.{S|D|Q} rd,rs1,rs2	
Categorization	Classify Type	R	FCLASS.{S|D|Q} rd,rs1	
Configuration	Read Status	R	FRCSR rd	
	Read Rounding Mode	R	FRRM rd	
	Read Flags	R	FRFLAGS rd	
	Swap Status Reg	R	FSCSR rd,rs1	
	Swap Rounding Mode	R	FSRM rd,rs1	
	Swap Flags	R	FSFLAGS rd,rs1	
	Swap Rounding Mode Imm	I	FSRMI rd,imm	
	Swap Flags Imm	I	FSFLAGSI rd,imm	

32-bit Instruction Formats

	31	30–25	24–21	20	19–15	14–12	11–8	7	6–0
R	funct7		rs2		rs1	funct3	rd		opcode
I	imm[11:0]				rs1	funct3	rd		opcode
S	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
SB	imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode
U	imm[31:12]						rd		opcode
UJ	imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode

16-bit (RVC) Instruction Formats

	15–13	12	11–10	9–7	6–5	4–2	1–0
CR	funct4		rd/rs1		rs2		op
CI	funct3	imm	rd/rs1		imm		op
CSS	funct3	imm			rs2		op
CIW	funct3	imm				rd′	op
CL	funct3	imm		rs1′	imm	rd′	op
CS	funct3	imm		rs1′	imm	rs2′	op
CB	funct3	offset		rs1′	offset		op
CJ	funct3	jump target					op

RISC-V Ecosystem

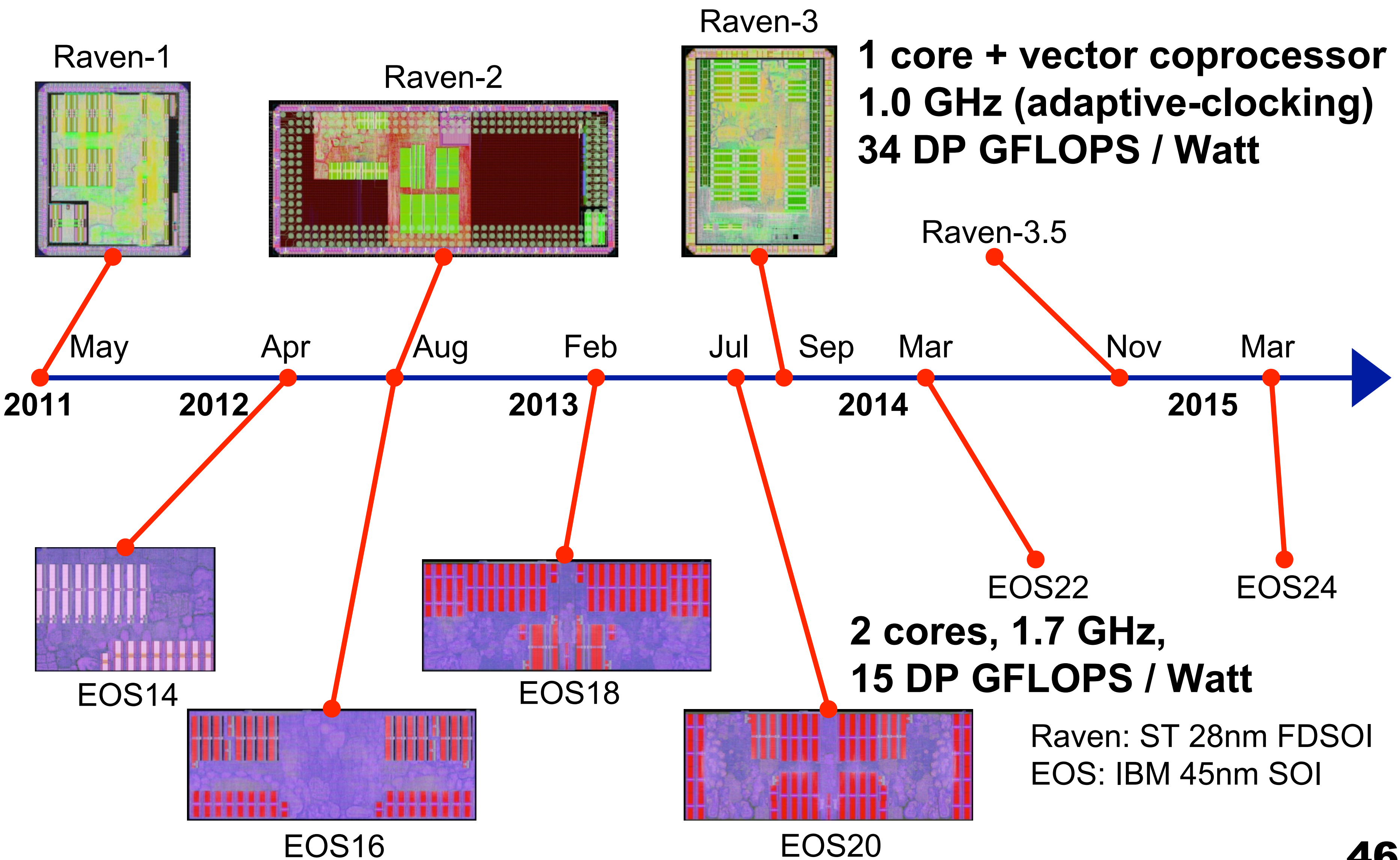
www.riscv.org

- ***Documentation***
  - User-Level ISA Spec v2.0 (Released 5/6/14)
  - Privileged ISA Spec v1.7 (Released 5/9/15)
  - Compressed Instr. v1.7 (Released 5/29/15)
- ***Software Tools***
  - GCC/glibc/GDB
  - LLVM/Clang
  - Linux
  - Yocto
  - Verification Suite
- ***Hardware Tools***
  - Zynq FPGA Infrastructure
  - Chisel
  - Interfaces to ARM buses
  - Debugger interface (underway)
- ***Hardware Implementations***
  - Rocket Chip Generator
    - RV64G single-issue in-order pipe
  - Zscale Chip Generator
  - Zscale core also in Verilog
  - Sodor Processor Collection
- ***Software Implementations***
  - ANGEL, JavaScript ISA Sim.
  - Spike, In-house ISA Sim.
  - QEMU ISA Sim.

RISC-V as Customizable Computer using FPGAs

- $250 Zed FPGA board ⇒ working computer with full SW stack to customize as desired in ≈1 hour @ 50 – 100 MHz
- ≈1 minute on real hardware processor ⇒ ≈1 hour of FPGA vs ≈1 month on SW simulator
- 32 node FPGA cluster for ≈$10,000

Four 28nm & Six 45nm RISC-V Chips taped out so far



Cost for 100 2x2mm 28 nm dies?

designlines SoC

Blog

Agile Design for Hardware, Part II

David Patterson and Borivoje Nikolić, UC Berkeley
7/30/2015 07:00 AM EDT

In the second of a three-part series, two Berkeley professors suggest its time to apply Agile design techniques to hardware.

We asked readers of Part I to guess the cost of a prototype run of 28 nm chips, as Agile development relies on a sequence of interim prototypes versus the One Big Tapeout of the traditional Waterfall process. Here are the results:

Prototype Category	Reader Average	Reader St. Dev.	Actual
Smallest die	2.3 x 2.3 mm	1.1 x 1.1 mm	1.57 x 1.57 mm
Fewest dies	190	280	80 to 100
Avg. cost / untested die	$690	$470	$300 to $375
Total cost	$170,000	$250,000	$30,000

$30,000!
Any project can afford to build hardware!

See “Is Agile Development Feasible for Hardware? Part II,” by David Patterson and Borivoje Nikolić, *EE Times*, 8/1/2015

RISC-V Beyond Berkeley

- Adopted as “standard ISA” for **India**
  - IIT-Madras building 6 different open-source cores, from microcontrollers to servers ($80M)
- **LowRISC** project based in Cambridge, UK producing open-source RISC-V based SoCs
  - Led by a founder of Raspberry Pi, privately funded
  - Adding capability-based security
  - Make and distribute ≈200,000 LowRISC SoCs
- **U. Maryland** research: Privacy preserving processor*

*Liu, Chang, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, and Elaine Shi. "GhostRider: A hardware-software system for memory trace oblivious computation." In *Proc. Int’l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2015. Best paper award.

RISC-V Big Ideas: An ISA for SoCs

- Base of <50 RISC instrs run can full SW stack
  - Just need to get simple ISA working
- Optional standard extensions to include or omit
  - Save area/energy by using only what needed
- Reserved opcodes to tailor SoC to apps
  - Secret sauce per SoC yet run SW stack
- Free ISA: $0, 0 paperwork, anyone can use
  - vs. if lucky, 6+ months negotiation + royalty
- Foundation will evolve RISC-V slowly for technical reasons determined by votes
  - vs. fast for business & technical reasons

Learning More about RISC-V

- Sign up for mailing lists/twitter at riscv.org to get announcements
- 1st RISC-V workshop was January 14-15 in Monterey
  - -Slides & videos: riscv.org/workshop-jan2015.html
  - -Sold out: 144 (33 companies & 14 universities)
- 2nd RISC-V workshop was June 29-30 at UC Berkeley
  - -Slides & videos: riscv.org/workshop-jun2015.html
  - -Sold out: 120 (30 companies & 20 universities)
- 3rd RISC-V workshop Jan 5-6 at Oracle Redwood City
  - -Free to academics & RISC-V sponsors; $149 others
    - -Will likely sell out too, so sign up soon
  - -Sign up www.regonline.com/riscvworkshop3

Outline

Part I - Past
50 years of Computer Architecture History:
- 1960s: Computer Families / Microprogramming
- 1970s: CISC
- 1980s: RISC
- 1990s: VLIW
- 2000s: NUMA vs. Clusters

Part II – Future
HW Technology
- End of Moore's Law
- Flash vs. Disks
- Fast DRAM
- Crosspoint NVRAM

Open ISA & RISC-V
- Case for Open ISAs
- Tour of RISC-V ISA
- RISC-V Software Stack
- RISC-V Chips

Questions?

BACKUP SLIDES

RISC-V ISA vs. ARMv8 ISA

Category	RISC-V	ARMv8	ARM/RISC
Year announced	2011	2011	--
Address sizes	32 / 64 / 128	32 / 64	--
Instruction formats	6 / 12†	53	4X-8X
Data addressing modes	1	8	8X
Instructions	177†	1,070	6X
Min number instructions to run Linux, gcc, LLVM	57	359	6X
Backend gcc compiler size	10K LOC	47K LOC	5X
Backend LLVM compiler size	10K LOC	22K LOC	2X
ISA manual size	181 pages	5,428 pages	30X

MIPS manual 700 pages
80x86 manual 3,600 pages

†With optional Compressed RISC-V ISA extension

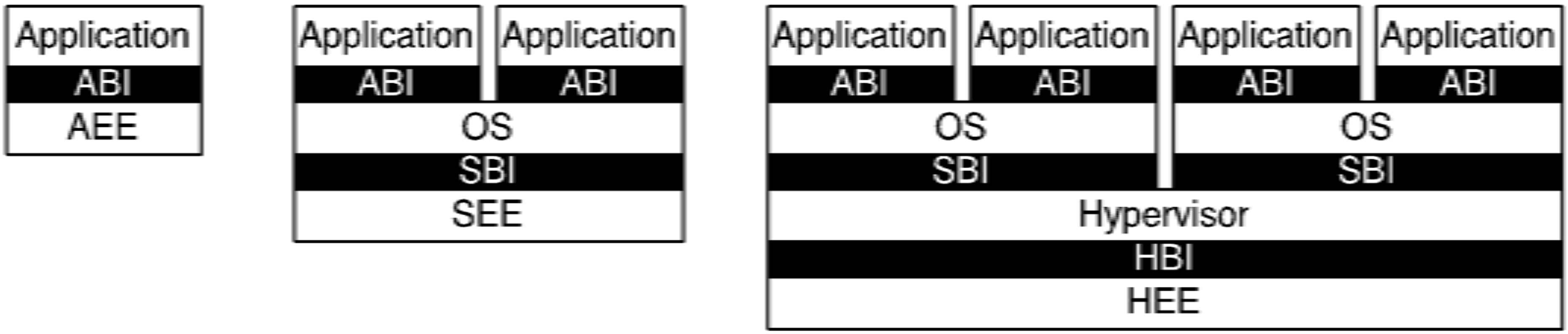
And it's still growing! ARM v8.1

- "The ARM architecture, in line with other processor architectures, is evolving with time. ARMv8.1 is the first set of changes ..."*
- Add a set of atomic read-write instructions
- Add a set of load & store instruction limited to configurable address regions
- More SIMD and scalar Multiply-Add instructions
  - "Signed Saturating Rounding Doubling Multiply Accumulate/Subtract, Returning High Half"
- Add a new protection mode
- Add a dirty bit for virtual address translation
- Expand Virtual Machine ID register

…

*"The ARMv8-A architecture and its ongoing development," by David Bash, 12/2/2014

RISC-V Privileged Architecture



- Application communicates with Application Execution Environment (AEE) via Application Binary Interface (ABI)
  - -ABI: user ISA + calls to AEE
- OS communicates via Supervisor Execution Environment (SEE) via System Binary Interface (SBI)
  - -SBI: user ISA + privileged ISA + calls to SEE
- Hypervisor communicates via Hypervisor Binary Interface (HBI) to Hypervisor Execution Environment (HEE)
- All levels of ISA designed to support virtualization

RISC-V Foundation

- Mission statement

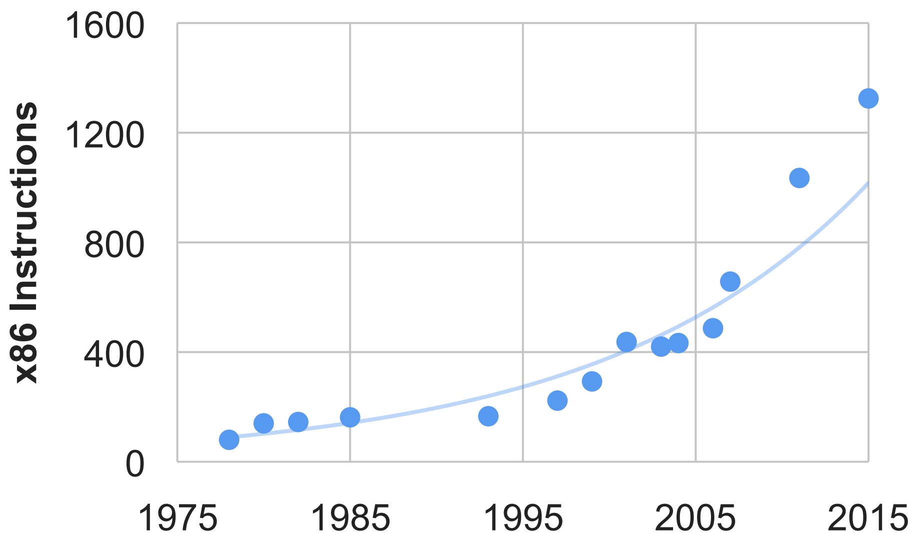
  "to standardize, protect, and promote the free and open RISC-V instruction set architecture and its hardware and software ecosystem for use in all computing devices."
- Established 7/31/2015 as a 501(c)(6) foundation
- Rick O'Connor is Executive Director
- Currently recruiting "founding" member companies
  - 7 signed up so far; to be revealed at workshop

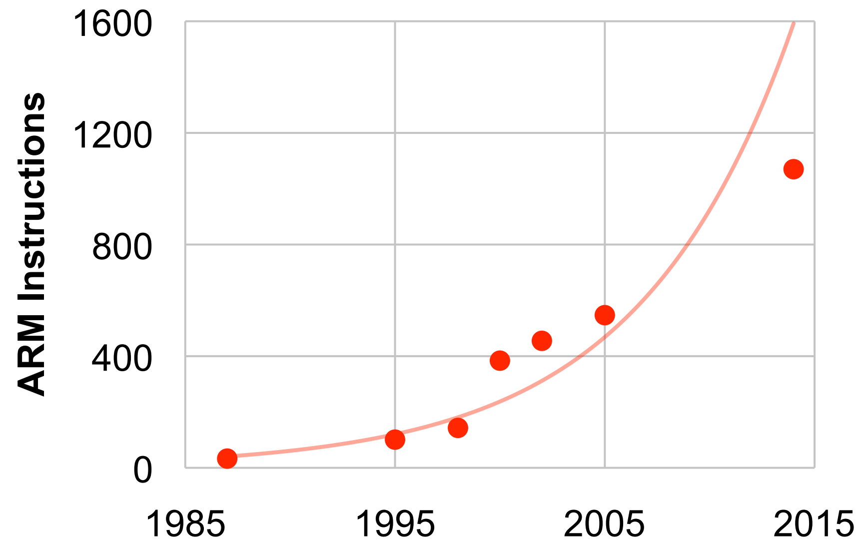
SSDs vs. HDDs

- SSDs will soon become cheaper than HDDs
- Transition from HDDs to SSDs will accelerate
  - -Already most instances in Amazon Web Service have SSDs
- Going forward we can assume SSD-only clusters

"Tape is dead, Disk is tape, Flash is disk."
Jim Gray, 2007

Evolution of Proprietary ISAs by company for business & technical reasons

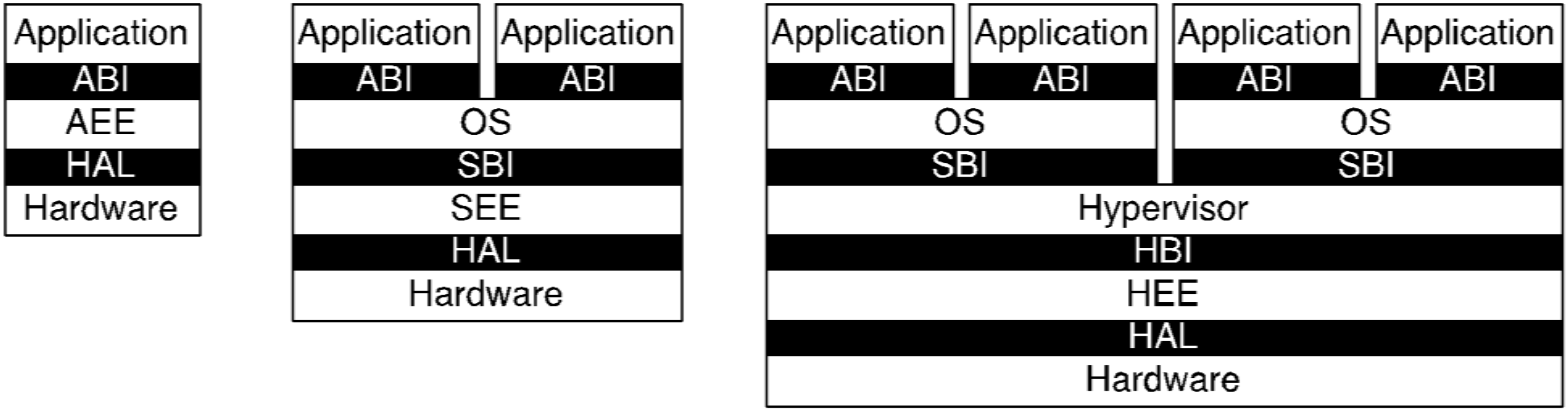




2 new x86 instructions per month for 38 years

2 new ARM instructions per month for 28 years

RISC-V Hardware Abstraction Layer



- HW requires more features beyond system ISA to support execution environments
- Separate features for HW platform from EE in HAL
  - Execution environments communicate with HW platforms via Hardware Abstraction Layer (HAL)
  - Details of execution environment and hardware platforms isolated from OS/Hypervisor ports

Four Supervisor Architectures

- Mbare
  - Bare metal, no translation or protection
- Mbb
  - Base and bounds protection
- Sv32
  - Demand-paged 32-bit VA space
- Sv39
  - Demand-paged 39-bit VA space
- Sv48
  - Demand-paged 48-bit VA space
- Page sizes: 4 KB, 2 MB, 1 GB
- Designed to support current popular operating systems
- Draft spec released May 7, 2015 for feedback

"Iron Law" of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Clock cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Clock cycle}}$$

- Instructions per program depends on source code, compiler technology, and ISA
- Clock cycles per instructions (CPI) depends on ISA and underlying microarchitecture
- Time per clock cycle depends upon the microarchitecture and base technology
- RISC executes more instructions per program, but many fewer clock cycles per instruction (CPI) ⇒ RISC faster than CISC

RISC-V ISA and Patents?

- Patents last 20 years, ISAs since 1950s
  ⇒ patent ISA quirks
- MIPS sued Lexra ISA clone for load/store word left/right (unaligned data)
  - US patent 4,814,976 (expired 2006)
- ≈35 RISC ISAs ≤1995
- 100 expired RISC patents
  - ≈25 expire in 2016 ...
- 100% coverage RISC-V?
  - Genealogy poster?

Year	Research / Commercial RISC ISA
1980	IBM 801
1981	Berkeley RISC-I, RISC-II
1982	Stanford MIPS
1983	Pyramid Technology 90X
1984	Berkeley SOAR ("RISC-III")
1985	ARMv1, MIPS I, Alliant FX(vector), Convex C1(vector)
1986	Sun SPARC v7, HP PA-RISC, IBM RT-PC
1987	Berkeley SPUR (SMP) ("RISC-IV")
1988	AMD 29000, Intel i960, Motorola 88000
1989	Intel i860 (SIMD), National CompactRISC
1990	DLX, IBM POWER, Sun SPARC v8, MIPS II
1991	MIPS III (64b address), Hitachi SH-1
1992	IBM PowerPC, ARMv6, DEC Alpha (64b), SH-2
1993	IBM POWER2, Sun SPARC v9 (64b), SH-3
1994	ARM Thumb (16b instr), HP PA-RISC (SIMD)
1995	MIPS16e (16b instr)

RISC-V Instruction Set Lineage

2015 RISC V	1981 RISC I RISC II	1984 SOAR	1984 Intel i960	1987 ARMv2	1988 SPUR	1990 DLX	1990 SPARCv8	1992 DEC Alpha	1992 MIPS III	1992 IBM PowerPC	1994 MIPS IV
LUI	LDHI					LHI	STHI		LUI		LUI
AUIPC				ADD[2]							
JAL		CALL	BAL	BL	JUMP/CALL	JAL	JMPL		JAL	BL	JAL
JALR		CALL	BAL	BL	JUMP_REGISTER	JALR	JMPL		JALR	BLR	JALR
BEQ	JMPR	SKIP+CALL	BE	BEQ	CMP_BRANCH_LIKELY	BEQ	BICC	BEQ	BEQ	BEQ	BEQ
BNE	JMPR	SKIP+CALL	BNE	BNE	CMP_BRANCH_LIKELY	BNE	BICC	BNE	BNE	BNE	BNE
BLT	JMPR	SKIP+CALL	BL	BLT	CMP_BRANCH_LIKELY		BICC	BLT		BLT	
BGE	JMPR	SKIP+CALL	BGE	BGE	CMP_BRANCH_LIKELY		BICC	BGE		BGE	
BLTU	JMPR	SKIP+CALL			CMP_BRANCH_LIKELY					BLT	
BGEU	JMPR	SKIP+CALL			CMP_BRANCH_LIKELY					BGE	
LB	LDBS		LDIB	LDRB		LB	LDSB		LB	LBZ	LB
LH	LDS	LOADC	LDIS			LH	LDSH	LDL	LH	LHZ	LH
LW	LDL	LOAD	LD	LDRB	LOAD_32	LW	LD	LDQ	LW	LWZ	LW
LBU	LDBU		LDOB			LBU	LDUB		LBU		LBU
LHU	LDSU		LDOS			LHU	LDUH		LHU	LHA	LHU
SB	STB		STIB	STRB		SB	STB		SB	STB	SB
SH	STS		STIS			SH	STH	STL	SH	STH	SH
SW	STL	STORE	ST	STR	STORE_32	SW	ST	STQ	SW	STW	SW
ADDI	ADD[1]	ADD		ADD	ADD	ADDI	ADD	ADD	ADDI	ADDI	ADDI
SLTI						SLTI			SLTI		SLTI
SLTIU									SLTIU		SLTIU
XORI	XOR	XOR		EOR	XOR	XORI	XOR	XOR	XORI	XORI	XORI
ORI	OR	OR		OR	OR	ORI	OR	BIS	ORI	ORI	ORI
ANDI	AND	AND		AND	AND	ANDI	AND	AND	ANDI	ANDI	ANDI
SLLI	SLL	SLA		LSL	SLL	SLLI	SLL			SLW	
SRLI	SRL	SRL		LSR	SRL	SRLI	SRL			SRW	
SRAI	SRA	SRA		ASR	SRA	SRAI	SRA			SRAWI	
ADD	ADD	ADD	ADDI	ADD	ADD	ADD	ADD	ADD	ADD	ADDI	ADD
SUB	SUB/SUBR	SUB	SUBI	SUB	SUBTRACT	SUB	SUB	SUB	SUB	SUB	SUB
SLL	SLL	SLA	SHLI	LSL	SLL	SLL	SLL	SLL	SLL	SLW	SLL
SLT						SLT			SLT		SLT
SLTU									SLTU		SLTU
XOR	XOR	XOR	XOR	EOR	XOR	XOR	XOR	XOR	XOR	XORI	XOR
SRL	SRL	SRL	SHRO	LSR	SRL	SRL	SRL	SRL	SRL	SRW	SRL
SRA	SRA	SRA	SHRI	ASR	SRA	SRA	SRA	SRA	SRA	SRAW	SRA
OR	OR	OR	OR	ORR	OR	OR	OR	BIS	ORI	ORI	ORI
AND	AND	AND	AND	AND	AND	AND	AND	AND	AND	ANDI	AND
FENCE								MB	SYNC	SYNC	SYNC
FENCE.I								CALL_PAL IMB		ISYNC	
SCALL		TRAP	CALLS		CALL_KERNEL	TRAP	TRAP		SYSCALL	SC	SYSCALL
SBREAK			RET		RETURN_KERNEL	RFE	RETT			RFI	
RDCYCLE							RDASR	RPCC			
RDCYCLEH											
RDTIME							RDASR				
RDTIMEH											
RDINSTRET							RDASR				
RDINSTRETH											
MUL			MULI	MUL		MULT	SMUL	MUL	MULT[5]	MULLW	MULT[5]
MULH							SMUL		MULT	MULHW	MULT
MULHSU											
MULHU							UMUL	UMULH	MULTU	MULHWU	MULTU
DIV			DIVI			DIV	SDIV		DIV	DIVW	DIV
DIVU			DIVO			DIVU	UDIV		DIVU	DIVWU	DIVU
REMU			REMO								
LR.W							LDSTUB	LDL_L	LL	LWARX	LL
SC.W							LDSTUB	STL_C	SC	STWCX	SC
AMOSWAP.W							SWAP				
AMOADD.W			ATADD								
AMOXOR.W											
AMOAND.W											
AMOOR.W											
AMOMIN.W											
AMOMAX.W											
AMOMINU.W											
AMOMAXU.W											
FLW				LDF	LOAD_SINGLE	LF	LDF	LDS	LWC1	LFS	LWC1
FSW				STF	STORE_SINGLE	SF	STF	STS	SWC1	STFS	SWC1
FMADD.S										FMADDS	MADD.S
FMSUB.S										FMSUBS	MSUB.S
FNMSUB.S										FNMSUBS	NMSUB.S
FNMADD.S										FNMADDS	NMADD.S
FADD.S			ADDR	ADF	FADD	ADDF	FADDs	ADDS	ADD.S	FADDS	ADD.S
FSUB.S				SUF	FSUB	SUBF	FSUBs	SUBS	SUB.S	FSUBS	SUB.S
FMUL.S			MULR	MUF	FMUL	MULTF	FMULs	MULS	MUL.S	FMULS	MUL.S
FDIV.S			DIVR	DVF	FDIV	DIVF	FDIVs	DIVS	DIV.S	FDIVS	DIV.S
FSQRT.S			SQRTR	SQT			FSQRTs		SQRT.S		SQRT.S
FSGNJ.S			CPYSRE[6]					CPYS			
FSGNJN.S			CPYRSRE[6]		FNEGATE			CPYSN			
FSGNJX.S											