COL216
Computer Architecture

Architecture Space – contd.
27th Jan, 2022

RISC and CISC

- Concept introduced by Patterson & Ditzel in 1980
- Followed by development of MIPS at Stanford and Berkeley RISC at UCB
- Virtually all new instruction sets since 1982 have been RISC
- Patterson & Hennessey were given Touring Award in 2018 for their “pioneering work on computer chip design”

"The case for the Reduced Instruction Set Computer"

by Patterson & Ditzel, ACM Sigarch, 1980

- *" .. the next generation of VLSI computers may be more efficiently implemented as RISC's than CISC's."*
- Reasons for increased complexity
- Usage and consequences of CISC instructions
- RISC and VLSI technology

Why processors became complex?

- Factors that motivated design of complex processors
- Factors that facilitated design of complex processors

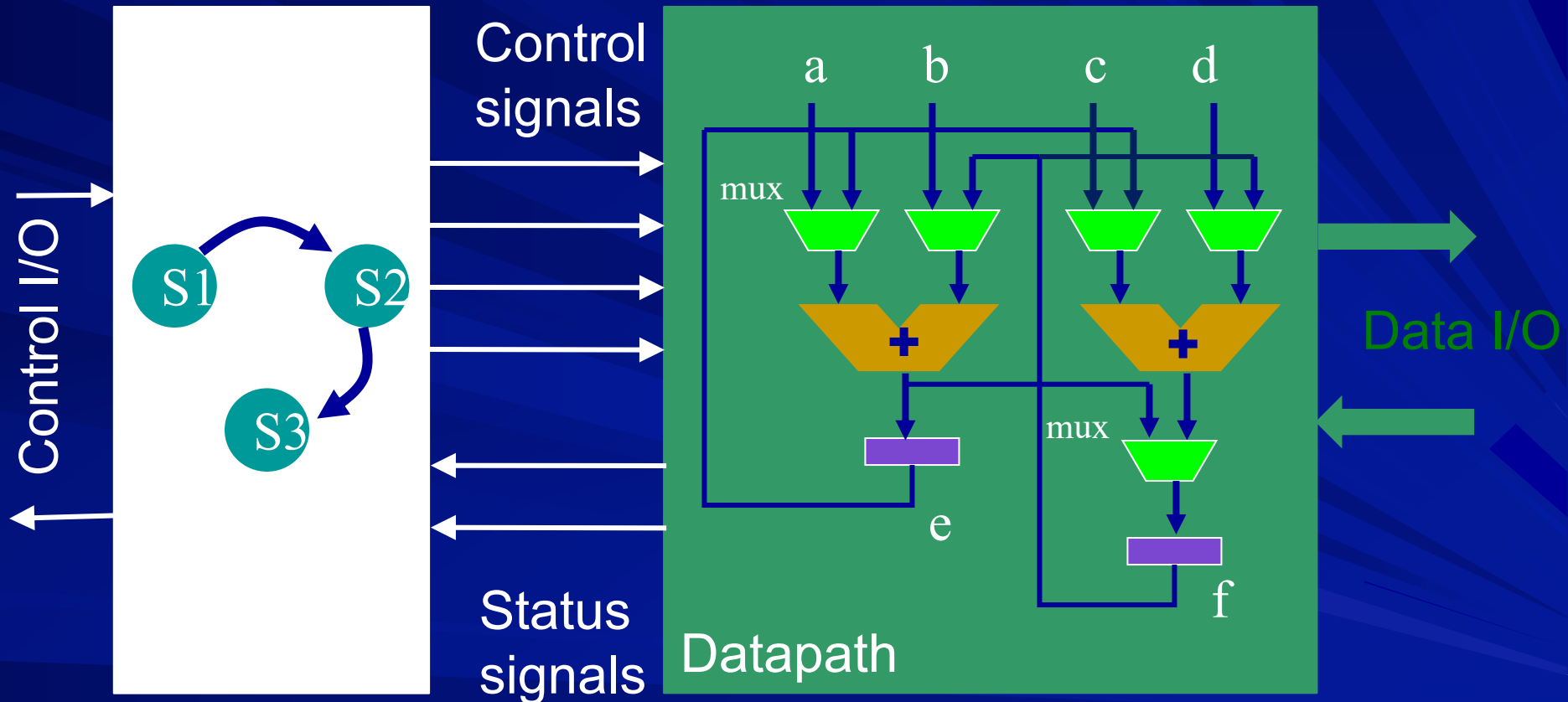
Motivations for complex processors

- To provide complex instruction to do common tasks, typically required by high level language programs
- Since memory was expensive and slower than processor, executing fewer instructions to do a given task was better
- New generations created by adding new instructions, rather than fresh designs

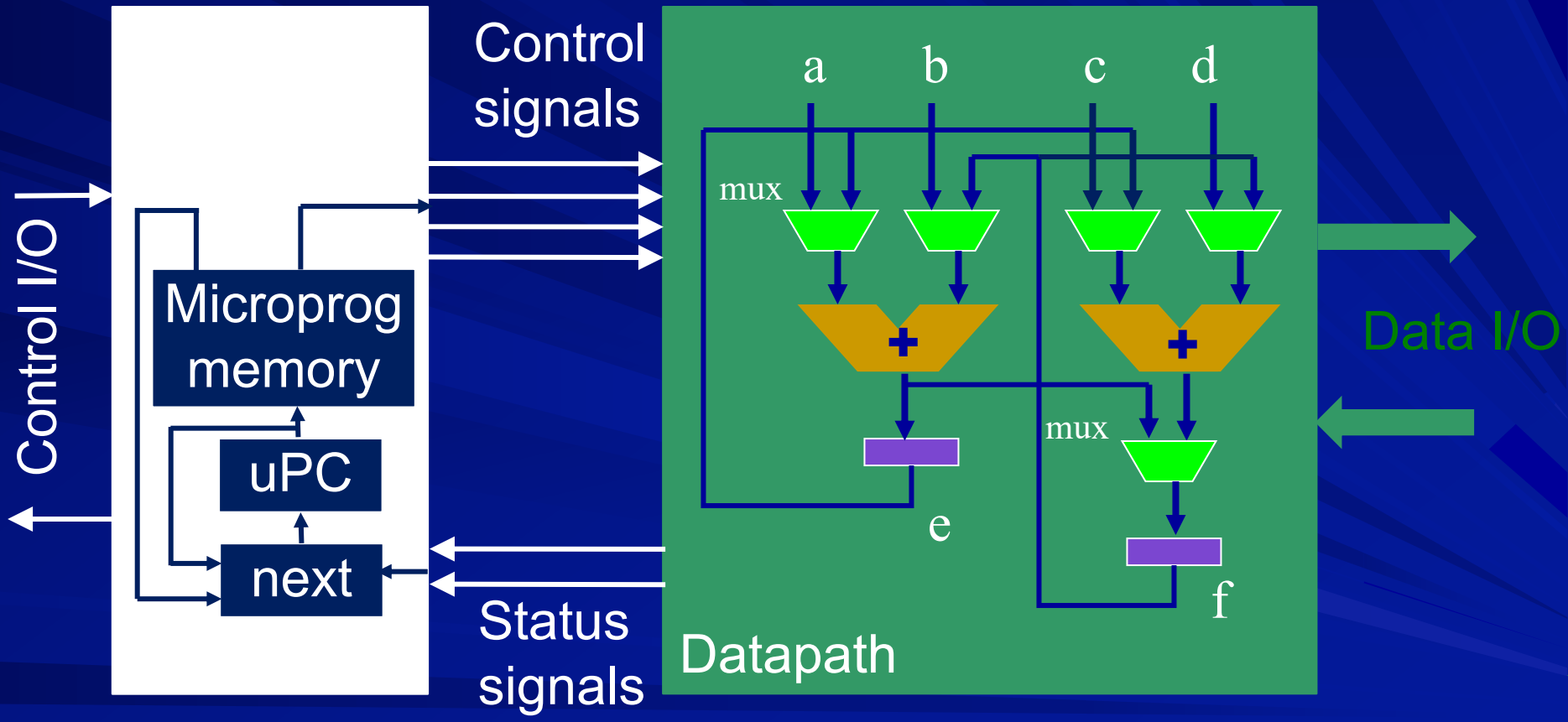
What facilitated complex designs?

- Processors were typically implemented using microprogrammed control
- Adding instructions was made easy by adding microcode (control store size 256x56 in PD11, 5120x96 in VAX 11/780)
- Simple processors can be easily implemented using FSM based control

FSM based control



Microprogrammed control



Consequences of CISC approach

- Only a few instructions used by compilers
- Sequence of simpler instructions may be faster than complex instructions
- Advantage of higher code density diminished with advances in memory technology
- Benefit for HLL features questionable
- Lengthened design times, increased design errors

RISC and VLSI technology

- Easier to fit in a chip
- Short design time suitable for rapidly changing technology
- Efficient implementation
- Area saved usable for cache, pipelining etc

RISC characteristics

- Uniform instruction size, limited formats
- Simple set of operations and addressing modes
- Register based architecture (R-R)
- 3 address instructions for arithmetic/logic operations
- Separate load – store operations

RISC examples

- MIPS at Stanford and Berkeley RISC
  - MIPS used by NEC, Nintendo, Silicon Graphics, Sony
- SUN's SPARC
  - Sun, Texas Instr, Toshiba, Fujitsu, Cypress, Tatung
- PowerPC: developed by IBM+Motorola+Apple in 1993
  - Based on POWER architecture of IBM (System/6000)
  - Used in Macintosh, embedded systems
- HP's PA-RISC
- DEC's Alpha
- ARM, Hitachi SuperH, Mitsubishi M32R (embedded)
- CDC 6600 (1960's)

CISC examples

- Main frames
- VAX-11/780 from Digital Equipment Corporation (DEC) - 1977
  - upward compatible with mini computer PDP-11
- Motorola 680x0
  - Apple Macintosh and other desk top computers
- Intel 80x86
  - Personal computers, servers

MIPS, PowerPC and SPARC

- MIPS - only 3 very simple formats
  - No flags, slt instruction
  - lui instruction
  - Opcode implies addressing mode, eg add / addi
- PowerPC
  - condition register, predication
  - link register for subroutines and loops
  - count register - counter update and branch
- SPARC: **S**calable **P**rocessor **ARC**hitecture
  - register windows, condition flags

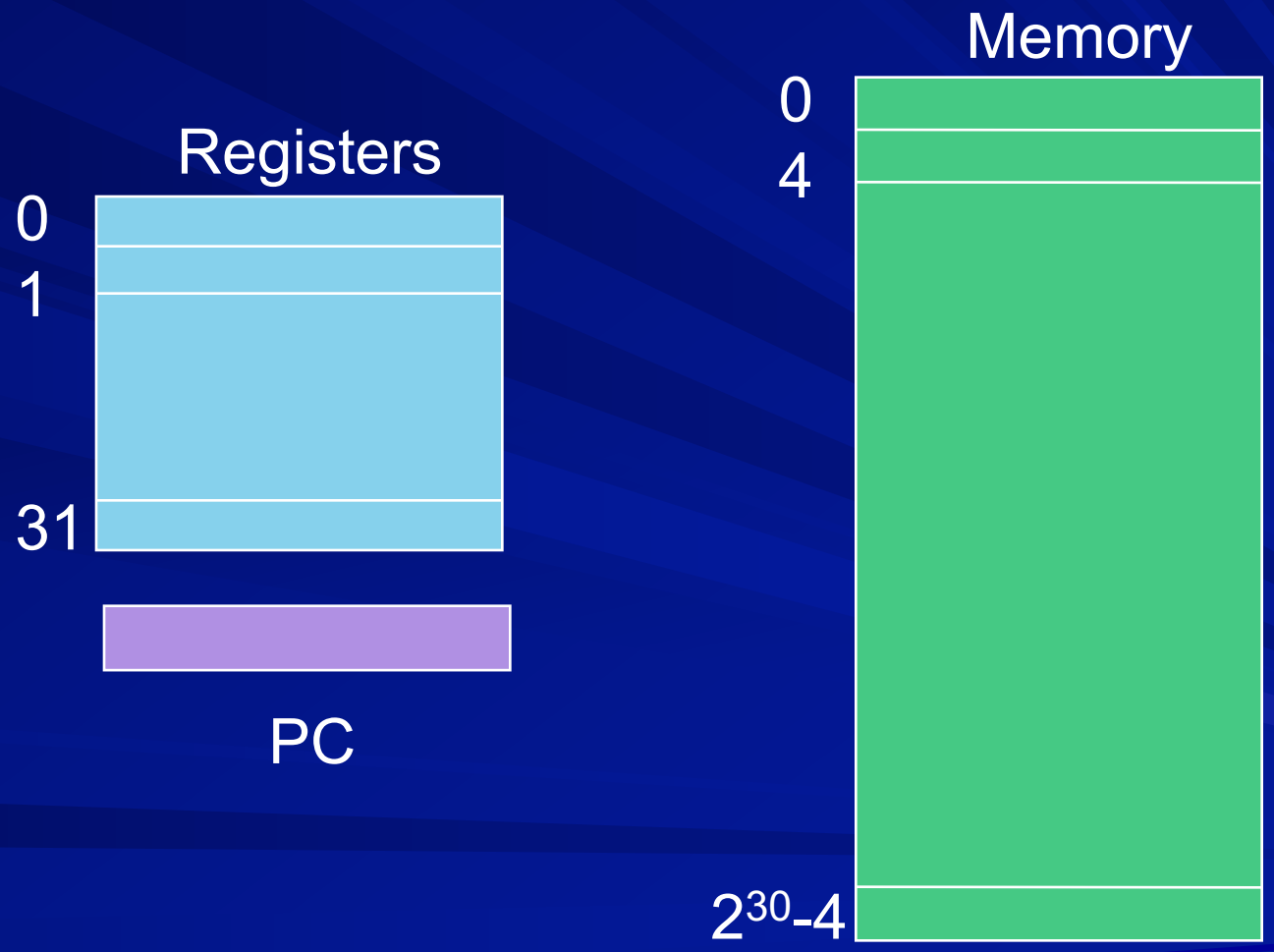
Advanced RISC Machine

- Available as a core that can be embedded in a chip.
- Supports implementations across a wide spectrum of performance points.
- Billions of devices have shipped, dominant architecture across many market segments.
- The architectural simplicity has led to very small implementations allowing devices with very low power consumption.

Modes, Extensions

- Thumb mode
  - 16 bit instructions
  - Slower but saves power
- Jezelle extension
  - Java byte code execution
- NEON
  - SIMD extension
- VFP
  - Floating point

MIPS ISA - storage



MIPS ISA – addressing modes`

Purpose

- Operand sources
- Result destinations
- Jump targets

Addressing modes

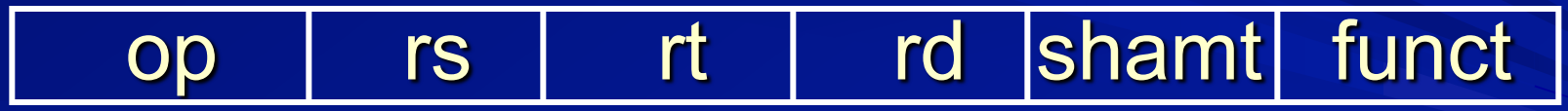
- Immediate
- Register
- Base/index
- PC relative
- (pseudo) Direct
- Register indirect

MIPS ISA features - encoding

- addi, lui, beq, bne, lw, sw I - format

- j, jal J - format

- add, slt, jr R - format



PowerPC Architecture

- Developed by IBM, Motorola and Apple in 1993
- Based on POWER architecture of IBM (System/6000)
- 32 bit as well as 64 bit architectures
- Used in Macintosh, embedded systems

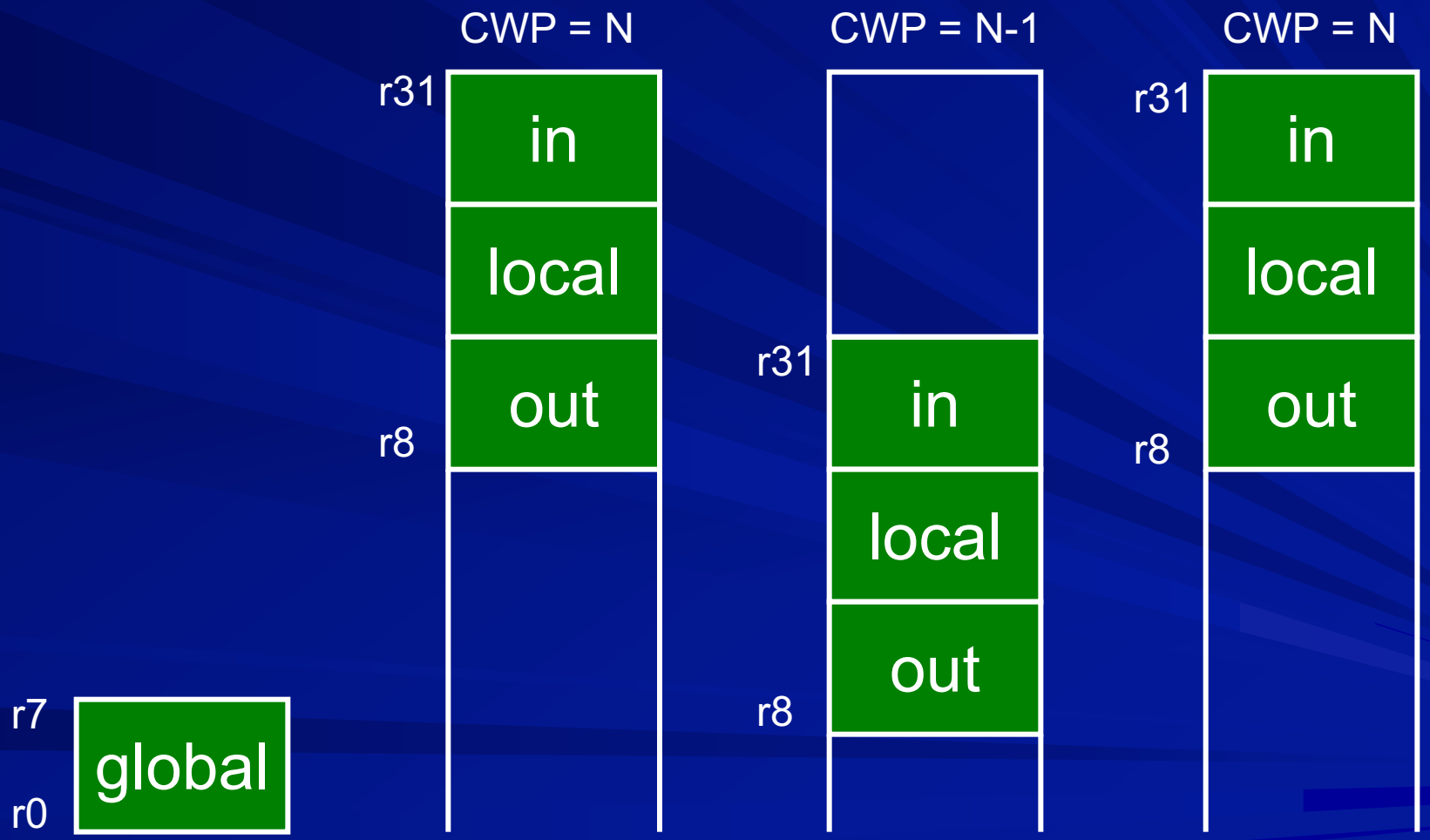
PowerPC Registers

- 32 general purpose registers
- Condition register: set by arithmetic/logical instructions, tested by conditional branches
- Link register: stores return addresses for procedure calls and loops
- Count register: iteration count for a loop

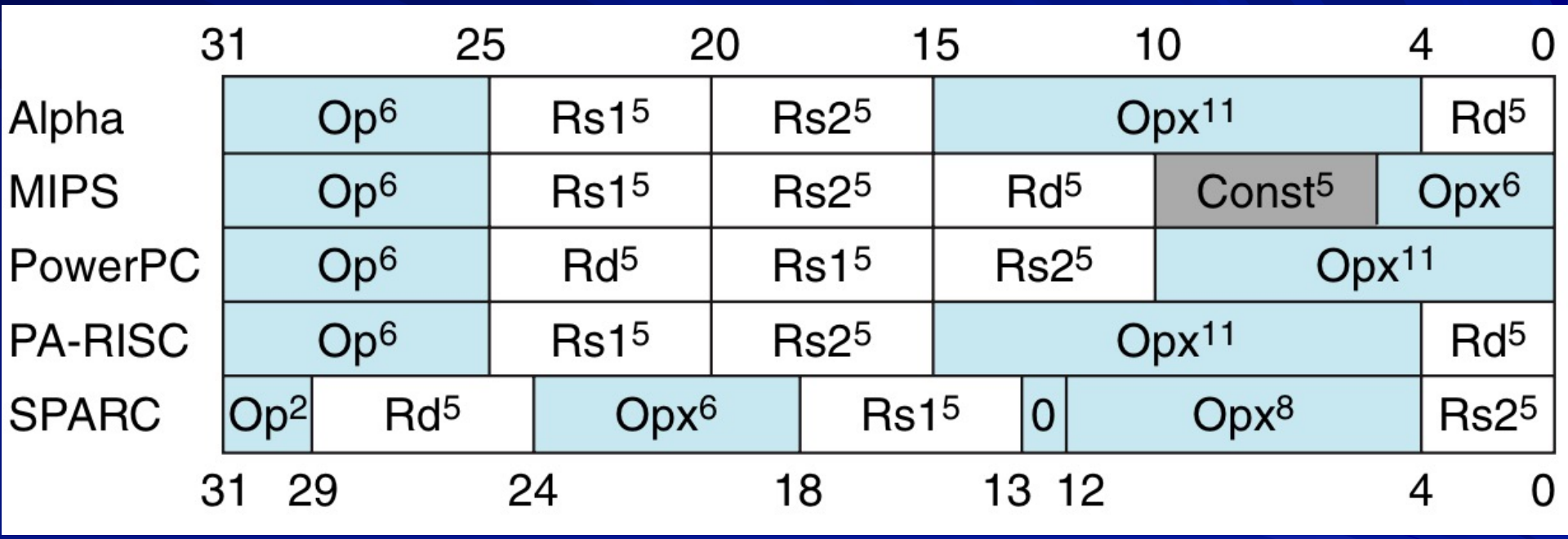
SPARC Architecture

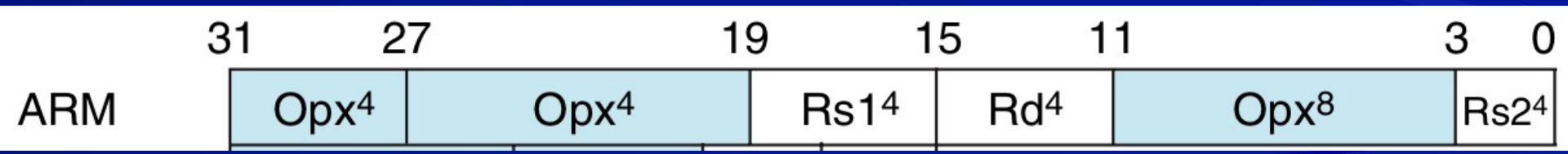
- **S**calable **P**rocessor **ARC**hitecture
- Scalable: the design allows for forward compatibility of programs
- Has been ‘scaled’ to 64 bit processors
- Implemented by Sun, Texas Instruments, Toshiba, Fujitsu, Cypress, Tatung etc.

SPARC Registers

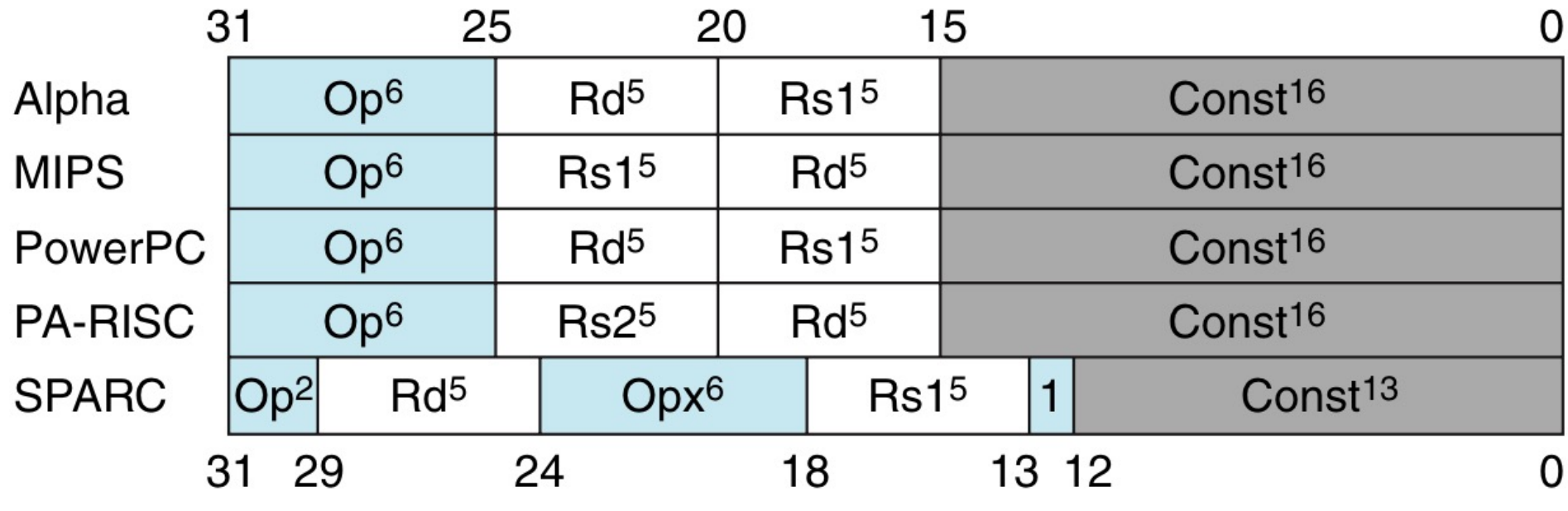


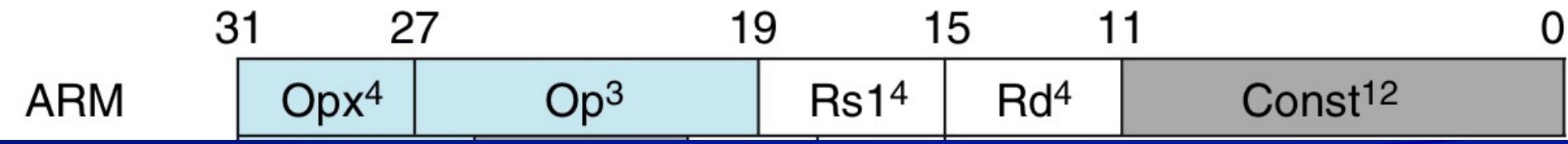
Data Processing Instructions



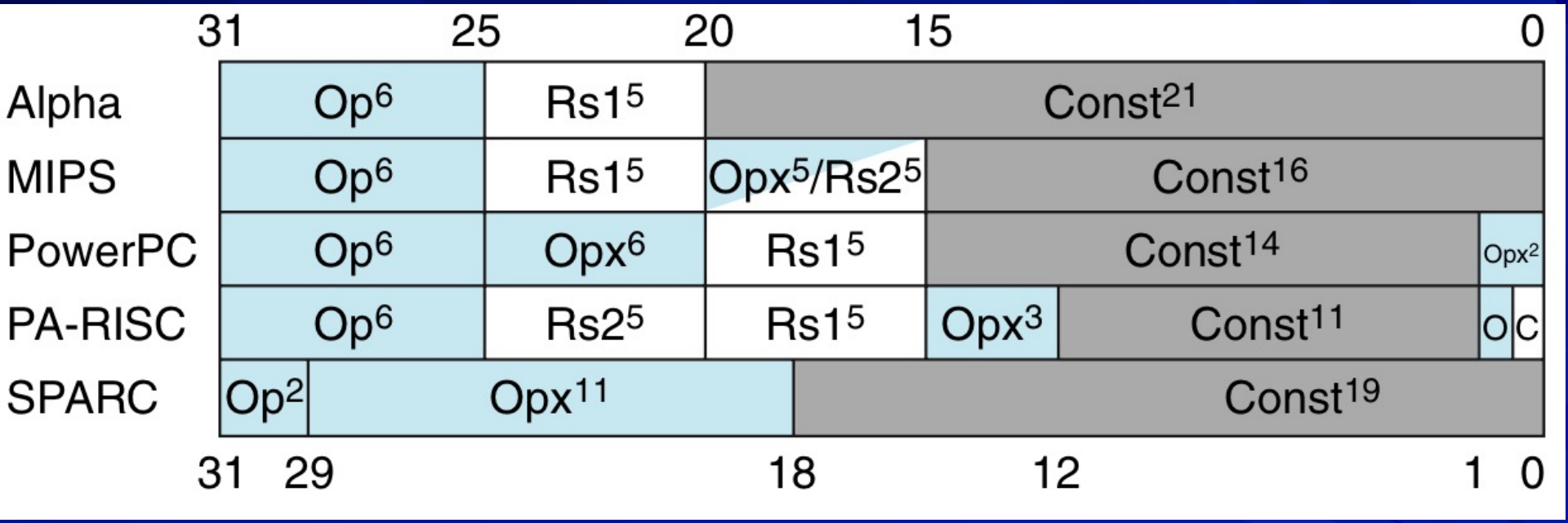


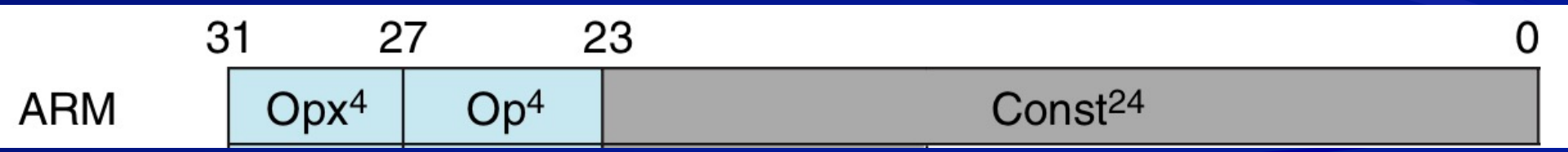
Immediate, Load/store



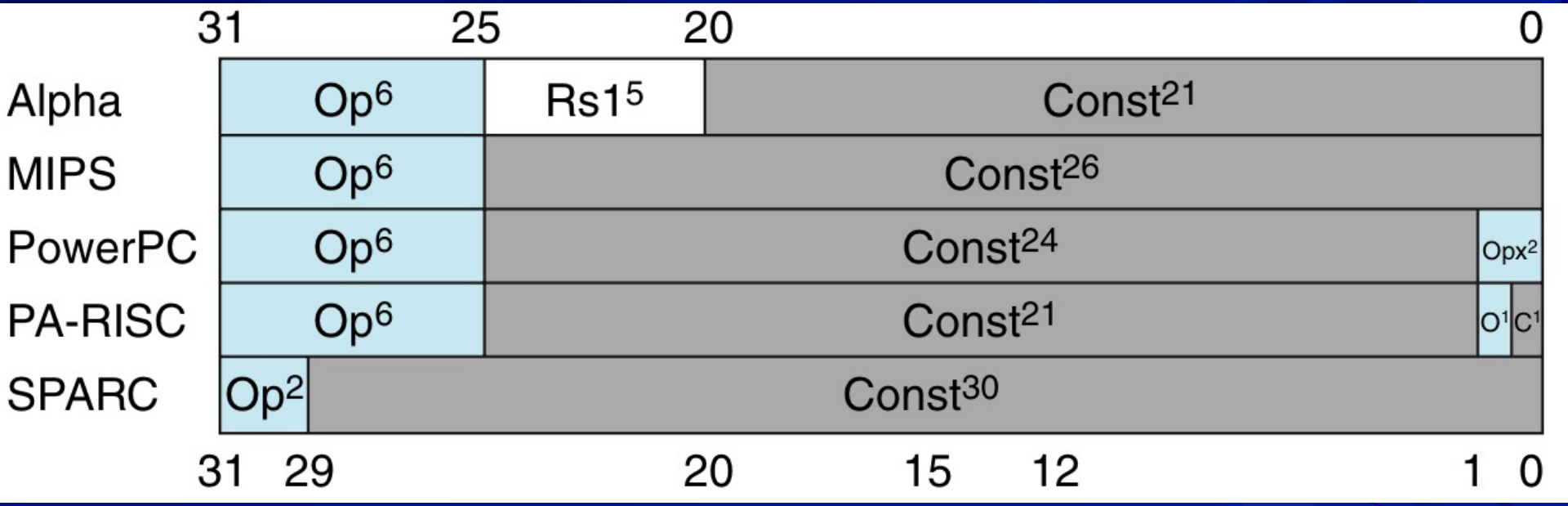


Conditional Branch





Unconditional jump / call



VAX Architecture

- Objective: minimize code size, make assembly language easy

  instructions from 1 to 54 bytes long!
- 32-bit words and addresses
- virtual memory
- 16 GPRs (r15 = PC, r14 = SP), CCs
- extremely orthogonal and memory-memory
- decode as byte stream - variable in length
- opcode: operation, #operands, operand types

VAX data types and addr modes

- Data types
  - 8, 16, 32, 64, 128
  - char string - 8 bits/char
  - decimal - 4 bits/digit
- Addressing modes
  - literal 6 bits. immediates 8, 16, 32 bits
  - register, register deferred
  - 8, 16, 32 bit displacements [deferred]
  - indexed (scaled)
  - autoincrement, autodecrement [deferred]

VAX Operations

- data transfer including string move
- arith/logical (2 and 3 operands)
- control (branch, jump, etc)
- function calls save state
- bit manipulation
- floating point - add, sub, mul, div, polyf
- crc (cyclic redundancy check),
- insque (insert in Q)

VAX instruction format example

Instruction length: 1 to 54 bytes

addl3 R1, 737(R2), #456

- byte 1: addl3
- byte 2: mode, R1
- byte 3: mode, R2
- byte 4,5: 737
- byte 6: mode
- byte 7-10: 456

VAX instruction with 6 operands

addp6 op1, op2, op3, op4, op5, op6

⇒add two packed decimal numbers

op1, op2: length and start addr of number1

op3, op4: length and start addr of number2

op5, op6: length and start addr of sum

Intel x86 history

Grown from 4 bit ⇒ 8 bit ⇒ 16 bit ⇒ 32 bit ⇒ 64 bit

- 1978: 8086 16 bit architecture
- 1980: 8087 floating point coprocessor
- 1982: 80286 24 bit addr space, more instr
- 1985: 80386 32 bits, new addressing modes
- 1989-1995: 80486, Pentium, -- Pro, performance
- 1997: MMX, Pentium II
- 1999: Pentium III (Streaming SIMD Extension)
- 2001: SSE2, double precision FP
- 2003: AMD64
- 2004: SSE3, 2006: SSE4, 2007: SSE5(AMD) . . .

Intel x86 features

- Complexity:
  - Instructions (~900) from 1 to 17 bytes long
  - one operand must act as both a src and dst
  - one operand can come from memory
  - complex addressing modes, e.g., “base + scaled index with 8 - 32 bit displacement”
  - Permitted instruction - address mode combinations irregular (lots of special cases, hard to learn!)
  - Effect by each instruction on condition codes is somewhat complex, irregular

Intel x86 registers

Size	Accumulator	Counter	Data	Base	Stack pointer	Base pointer	Source index	Dest index
64	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
32	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
16	AX	CX	DX	BX	SP	BP	SI	DI
8	AH AL	CH CL	DH DL	BH BL				

Other registers in Intel x86

- 16 bit segment registers:
  CS, SS, DS, ES, FS, GS
- Flags (condition codes) 32 bit
- Instruction pointer (PC) 32 bit

Intel x86 : peculiar feature

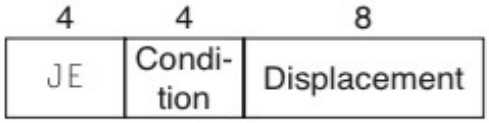
Upto 3 “prefixes” for an instruction

- override default data size
- override default segment register
- lock bus for a semaphore
- repeat the following instruction
- override default address size

Instruction format

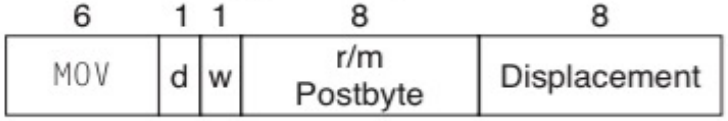
Some Instruction Formats

a. JE EIP + displacement

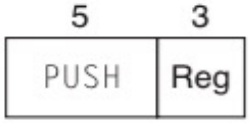


b. CALL

c. MOV EBX, [EDI + 45]



d. PUSH ESI



e. ADD EAX, #6765

f. TEST EDX, #42

x86 instruction listings - Wikipedia

Thank you