ISA instruction set architecture

- ARMv7 32-bit architecture for embedded devices
- ARMv8 64-bit architecture
- x86 Intel architecture

ARMv7

- chips used in phones, tablets
- ARM licenses their IP (intellectual property) to manufacturers
- ARM is a RISC architecture like MIPS
- MIPS has more registers
- ARM has more addressing modes and instruction formats

ARM v. MIPS

	ARM	MIPS
Date announced	1985	1985
Instruction size (bits)	32	32
Address space (size, model)	32 bits, flat	32 bits, flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Integer registers (number, model, size)	15 GPR × 32 bits	31 GPR × 32 bits
I/O	Memory mapped	Memory mapped

	Instruction name	ARM	MIPS
Register-register	Add	add	addu, addiu
	Add (trap if overflow)	adds; swivs	add
	Subtract	sub	subu
	Subtract (trap if overflow)	subs; swivs	sub
	Multiply	mul	mult, multu
	Divide	—	div, divu
	And	and	and
	Or	orr	or
	Xor	eor	xor
	Load high part register	—	lui
	Shift left logical	lsl[1]	sllv, sll
	Shift right logical	lsr[1]	srlv, srl
	Shift right arithmetic	asr[1]	srav, sra
	Compare	cmp, cmn, tst, teq	slt/i,slt/iu
Data transfer	Load byte signed	ldrsb	lb
	Load byte unsigned	ldrb	lbu
	Load halfword signed	ldrsh	lh
	Load halfword unsigned	ldrh	lhu
	Load word	ldr	lw
	Store byte	strb	sb
	Store halfword	strh	sh
	Store word	str	sw
	Read, write special registers	mrs, msr	move
	Atomic Exchange	swp, swpb	ll;sc

Condition codes

- ARMv7 used 4 condition codes that are stored in a program status word:
  - negative, zero, carry, overflow
- these codes are set on any arithmetic/logic instruction

Each ARMv7 instruction begins with a 4-bit field that determines if it acts as a NOP or the instruction, depending on the condition codes

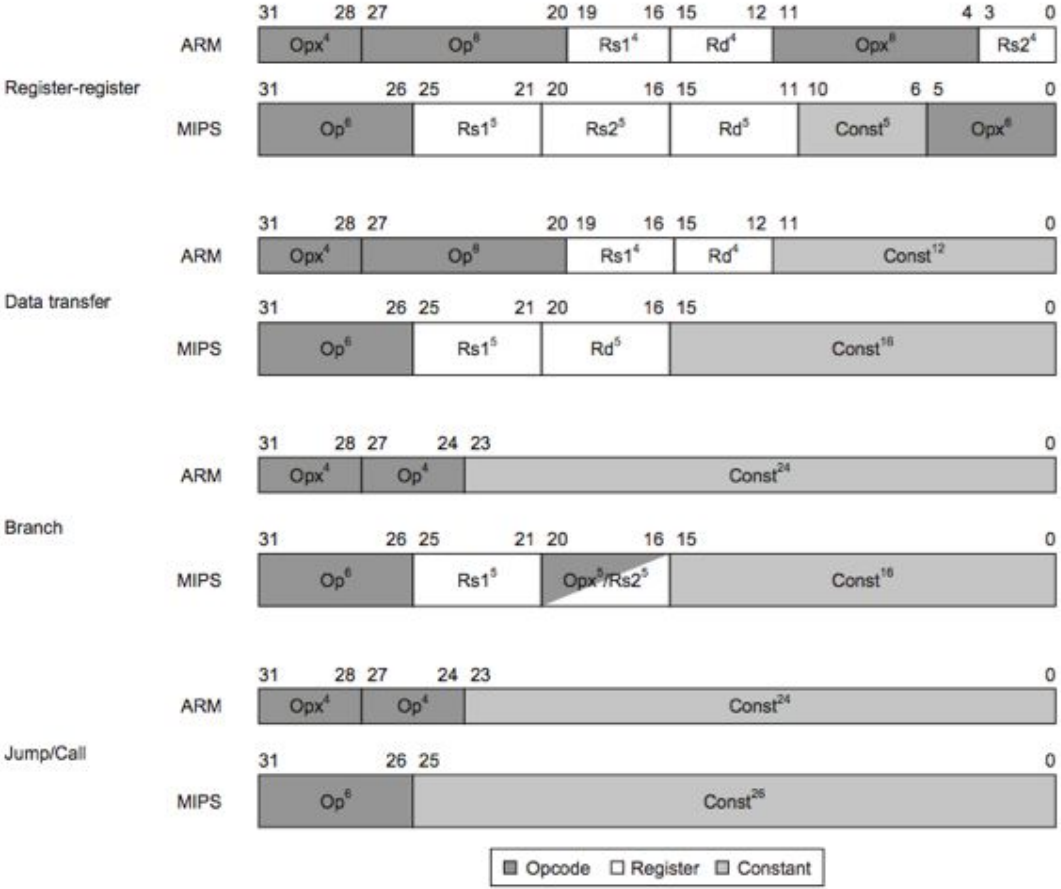


FIGURE 2.34 Instruction formats, ARM and MIPS. The differences result from whether the architecture has 16 or 32 registers.

Name	Definition	ARM	MIPS
Load immediate	Rd = Imm	mov	addi $0,
Not	Rd = ~(Rs1)	mvn	nor $0,
Move	Rd = Rs1	mov	or $0,
Rotate right	Rd = Rs i >> i $Rd_{0...i-1} = Rs_{31-i...31}$	ror	
And not	Rd = Rs1 & ~(Rs2)	bic	
Reverse subtract	Rd = Rs2 – Rs1	rsb, rsc	
Support for multiword integer add	CarryOut, Rd = Rd + Rs1 + OldCarryOut	adcs	—
Support for multiword integer sub	CarryOut, Rd = Rd – Rs1 + OldCarryOut	sbcs	—

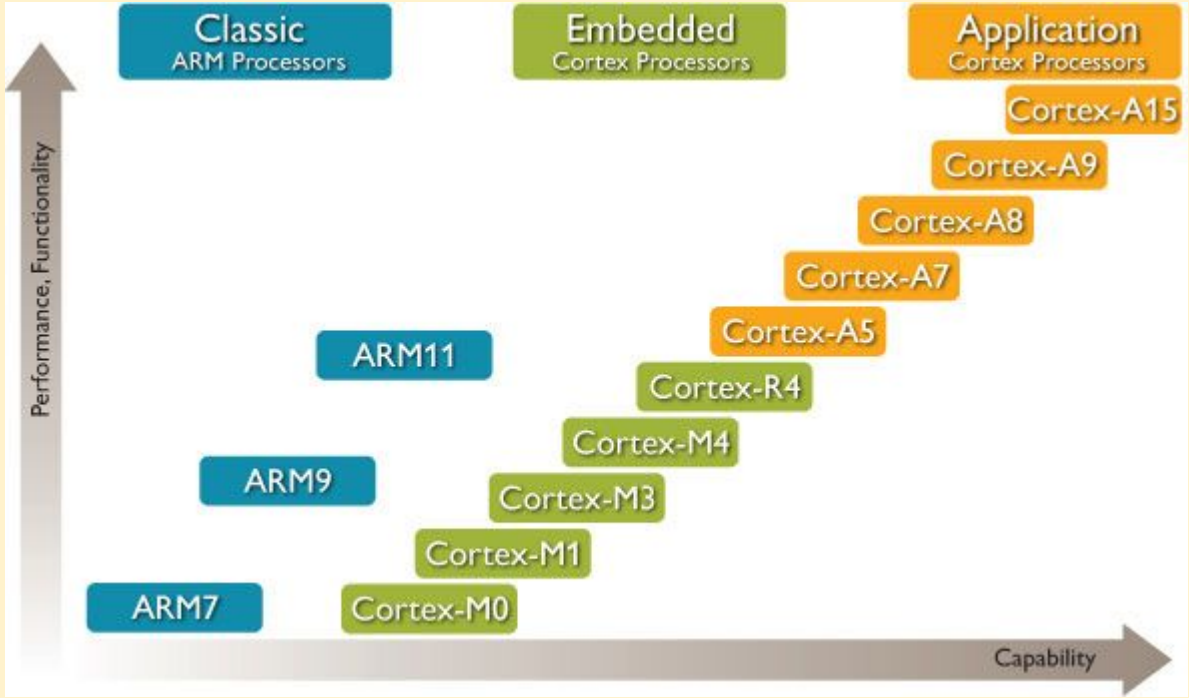
FIGURE 2.35 ARM arithmetic/logical instructions not found in MIPS.

ARMv8

- 2007 design began; released 2013
- much closer to MIPS than ARMv7 because:
  - got rid of the 4-bit conditional execution field
  - has 32 general-purpose registers
  - includes a divide instruction

ARM

- Current reports (June 2020) indicate Apple will use 12-core 5nm ARM processors in 2021 Mac instead of Intel chips
- ARM chips currently used in iPhones, iPads, etc. (currently based on the A14 chip)
- reason: they can design their own custom chips
- reason: ARM chips are more power efficient
- potential problems: software compatibility

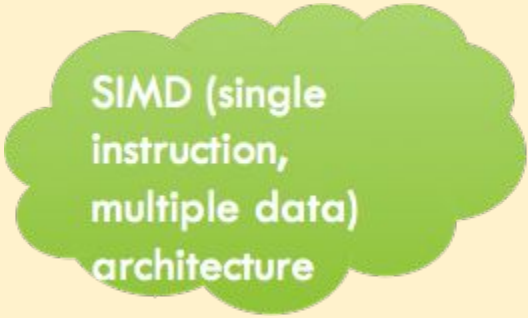

Classic
ARM Processors
Embedded
Cortex Processors
Application
Cortex Processors
Cortex-A15
Cortex-A9
Cortex-A8
Cortex-A7
Cortex-A5
ARM11
Cortex-R4
Cortex-M4
ARM9
Cortex-M3
Cortex-M1
ARM7
Cortex-M0
Performance, Functionality
Capability

x86 evolution

- MIPS and ARM were developed by small groups, starting in 1985
- x86 is the product of many groups over decades

➢ 1978 - Intel 8086 16-bit CPU, an extension of the 8-bit 8080
➢ 1980 - Intel 8087 floating point coprocessor
➢ 1982 - Intel 80286 24-bit processor
➢ 1985 - Intel 80386 32-bit, paging support
➢ 1989 - 1995 - 80486 and Pentium improved performance and allowed multiprocessing

x86 evolution



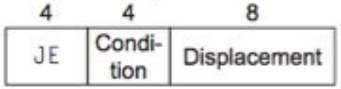
- 1997 - Pentium and Pentium Pro added MMX (multimedia extension) instructions
- 1999 - added SSE (streaming SIMD extensions) for faster video processing
- 2003 - AMD widened all architectures to 64-bit and Intel followed soon
- 2006 - more SSE instructions and support for virtual machines
- 2011 - Advanced Vector Extensions

80386

- only 8 general-purpose registers
- x86 arithmetic/logic instructions must have one operand that is both a source and destination
- one of the operands can be in memory
- conditional branches use condition codes or flags like ARMv7

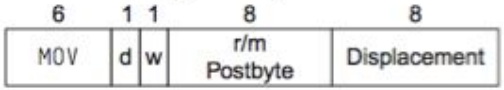
FIGURE 2.36 The 80386 register set. Starting with the 80386, the top eight registers were extended to 32 bits and could also be used as general-purpose registers.

a. JE EIP + displacement

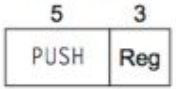


b. CALL

c. MOV EBX, [EDI + 45]



d. PUSH ESI



e. ADD EAX, #6765

f. TEST EDX, #42

FIGURE 2.41 Typical x86 instruction formats. Figure 2.42 shows the encoding of the postbyte.

Instruction	Meaning
Control	**Conditional and unconditional branches**
jnz, jz	Jump if condition to EIP + 8-bit offset; JNE (forJNZ), JE (for JZ) are alternative names
jmp	Unconditional jump—8-bit or 16-bit offset
call	Subroutine call—16-bit offset; return address pushed onto stack
ret	Pops return address from stack and jumps to it
loop	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠ 0
Data transfer	**Move data between registers or between register and memory**
move	Move between two registers or between register and memory
push, pop	Push source operand on stack; pop operand from stack top to a register
les	Load ES and one of the GPRs from memory
Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
add, sub	Add source to destination; subtract source from destination; register-memory format
cmp	Compare source and destination; register-memory format
shl, shr, rcr	Shift left; shift logical right; rotate right with carry condition code as fill
cbw	Convert byte in eight rightmost bits of EAX to 16-bit word in right of EAX
test	Logical AND of source and destination sets condition codes
inc, dec	Increment destination, decrement destination
or, xor	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
movs	Copies from string source to destination by incrementing ESI and EDI; may be repeated
lods	Loads a byte, word, or doubleword of a string into the EAX register

FIGURE 2.40 Some typical operations on the x86. Many operations use register-memory format, where either the source or the destination may be memory and the other may be a register or immediate operand.

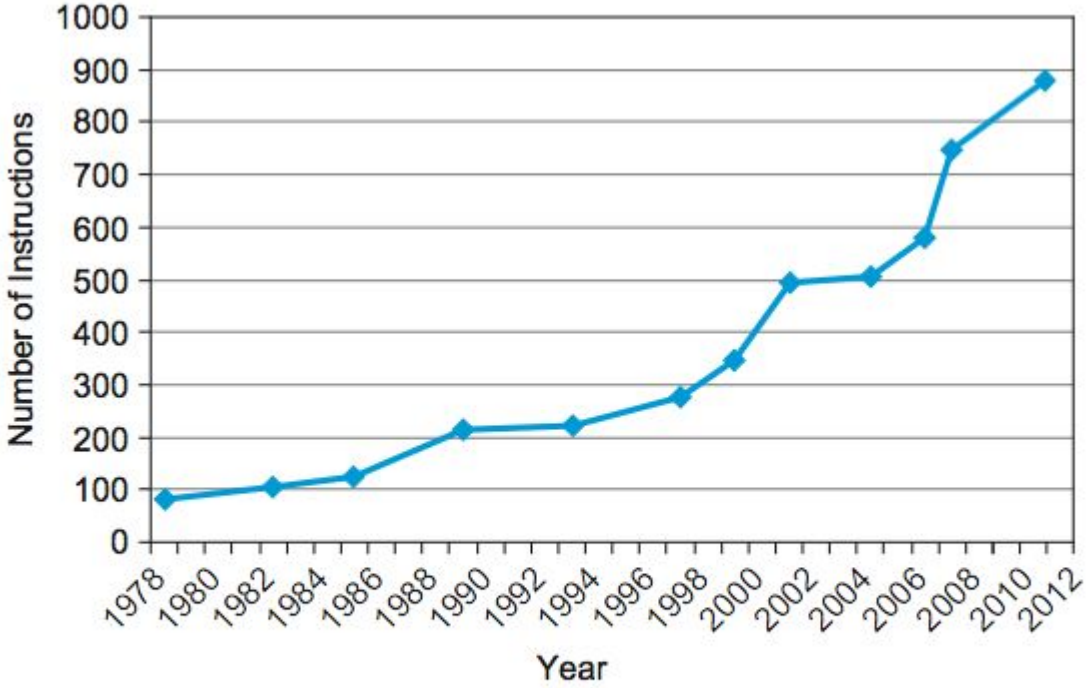
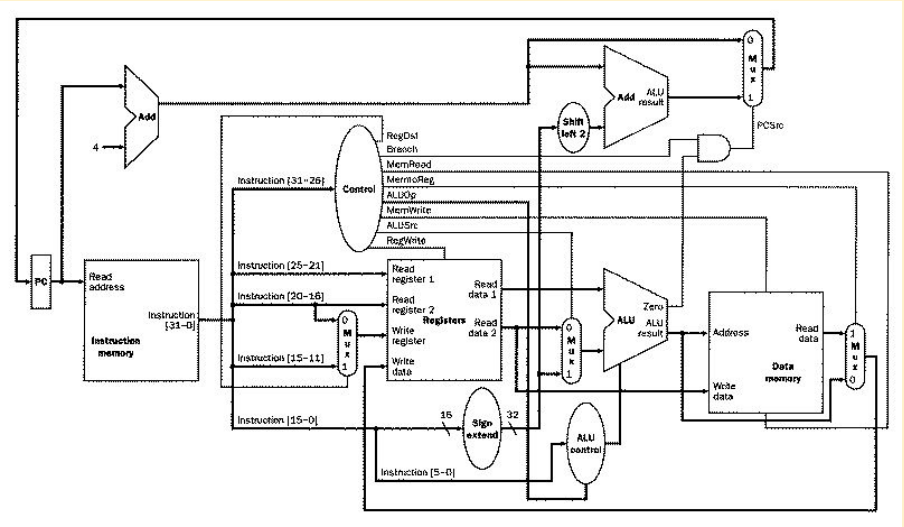


FIGURE 2.43 Growth of x86 instruction set over time. While there is clear technical value to some of these extensions, this rapid change also increases the difficulty for other companies to try to build compatible processors.

```
# Hello World in x86
# run like this: gcc -c hello.s && ld hello.o && ./a.out


                .global _start

                .text
_start:
                mov     $1, %rax                # syscall 1 for write
                mov     $1, %rdi                # file 1 is console output
                mov     $message, %rsi          # string address
                mov     $13, %rdx               # number of bytes
                syscall

                mov     $60, %rax               # syscall 60 for exit
                xor     %rdi, %rdi              # return 0
                syscall

message:
                .ascii  "Hello world!\n"
```

Comparing architectures

- designing CPU and ISA go hand-in-hand
- choices: how many and what types of instructions and how fast they can execute



ISA shoot-out

- Talk at Berkeley by a grad student of David Patterson: https://www.reddit.com/r/RISCV/comments/7ikq6q/isa_shootout_a_comparison_of_risc_v_arm_and_x86/
- Technical paper here: https://arxiv.org/pdf/1607.02318.pdf
- This is extra information for those who want to learn more.