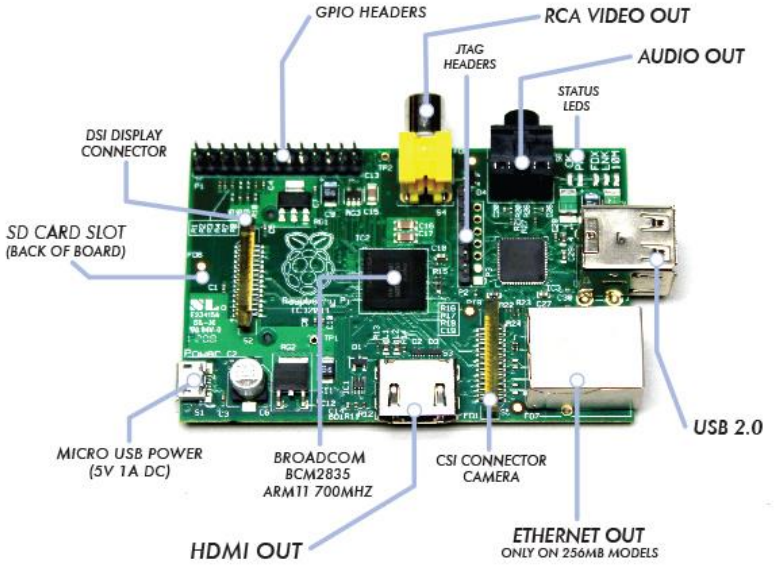
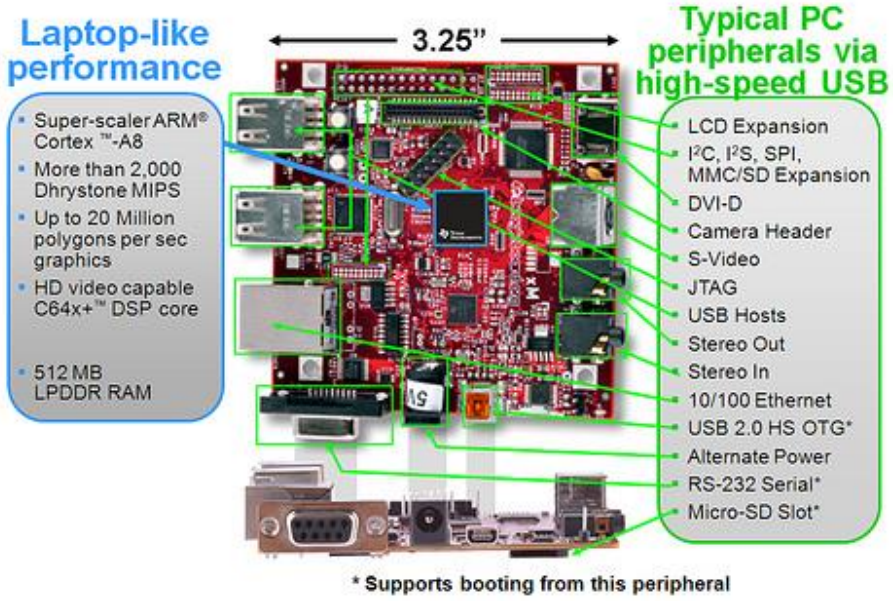
Universidade de Brasília
Departamento de Ciência da Computação

Aula 11
Arquiteturas ARMv7 e x86
32 Bits

Arquitetura ARM v7 (32 bits)

- Arquitetura mais popular para sistemas portáteis (celulares, tablets, sistemas de GPS, etc.)
- Produzida pela ARM Holding (britânica) – Comprada pela NVIDIA em 2020 (US$ 40 bilhões)
- Acorn RISC Machine ⇒ Advanced RISC Machine
- 6.7 bilhões de unidades vendidas em out/nov/dez de 2020 (x86 menos de 500 milhões)
- Baixo consumo (Intel: Atom)
- Baixo custo, customizável, integrável a projetos proprietários
- Usados também no Raspberry Pi, BeagleBoard, BeagleBone, PandaBoard e outros single-board computers,





Semelhanças com RISC-V

	ARM	RISC-V
Date announced	1985	2015
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

FIGURE 2.31 Similarities in RM and MIPS instruction sets.

ARMv8 de 64 bits é muito mais similar ao RV64I e MIPS64

	Instruction name	ARM	RISC-V
Register-register	Add	add	add; addi
	Add (trap if overflow)	adds; swivs	-
	Subtract	sub	sub
	Subtract (trap if overflow)	subs; swivs	-
	Multiply	mul	mul; mulh/u/su
	Divide	—	div, divu
	And	and	and
	Or	orr	or
	Xor	eor	xor
	Load high part register	—	lui
	Shift left logical	lsl[1]	sll, slli
	Shift right logical	lsr[1]	srl, srli
	Shift right arithmetic	asr[1]	sra, srai
	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	csr/*
	Atomic Exchange	swp, swpb	amo/*

Diferenças:
Modos de endereçamento

Addressing mode	ARM	RISC-V
Register operand	X	X
Immediate operand	X	X
Register + offset (displacement or based)	X	X
Register + register (indexed)	X	—
Register + scaled register (scaled)	X	—
Register + offset and update register	X	—
Register + register and update register	X	—
Autoincrement, autodecrement	X	—
PC-relative data	X	—

FIGURE 2.33 Summary of data addressing modes. ARM has separate register indirect and register + offset addressing modes, rather than just putting 0 in the offset of the latter mode. To get greater addressing range, ARM shifts the offset left 1 or 2 bits if the data size is halfword or word.

Diferenças: Modos da CPU

- RISC-V:
  - Machine mode: modo privilegiado, *bare metal*, sem restrições
  - Supervised mode: modo privilegiado, usado pelo Kernel do Sistema
  - User mode: modo não privilegiado
- ARM:
  - User mode: único modo não privilegiado
  - FIQ mode: FIQ Fast interrupt (prioridade em relação às demais IRQs)
  - IRQ mode: IRQ Normal interrupt.
  - Supervisor Call mode: SVC instruction is executed. (similar ecall)
  - Abort mode: prefetch abort or data abort exception occurs.
  - Undefined mode: undefined instruction exception occurs.
  - System mode : executing an instruction that writes to the mode bits of the CPSR.
  - Monitor mode : A monitor mode to support TrustZone extension
  - Hyp mode : A non-secure hypervisor mode

Diferenças:
Banco de Registradores

Sem registrador hardwired Zero.
R0 a R7: iguais para qualquer modo

R13 e R14: cada modo possui
estes individuais (banked)
geralmente similares ao sp e ra

R8-R12: modo FIQ possui próprios

R15 : Registrador PC

CPSR: Current Program Status Reg

SPSR: Saved Program Status Reg
Cópia do CPSR quando muda o modo

Registers across CPU modes

usr	sys	svc	abt	und	irq	fiq
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Diferenças:
Registrador de Status: CPSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	IT		J	DNM				GE				IT						E	A	I	F	T	M				

M (bits 0–4) is the processor mode bits.
T (bit 5) is the Thumb state bit. (modo de codificação com 16 bits)
F (bit 6) is the FIQ disable bit.
I (bit 7) is the IRQ disable bit.
A (bit 8) is the imprecise data abort disable bit.
E (bit 9) is the data endianness bit.
IT (bits 10–15 and 25–26) is the if-then state bits.
GE (bits 16–19) is the greater-than-or-equal-to bits.
DNM (bits 20–23) is the do not modify bits.
J (bit 24) is the Java state bit (Jazelle). (RV32J está previsto)
Q (bit 27) is the sticky overflow bit. (add e sub com saturação)
V (bit 28) is the overflow bit.
C (bit 29) is the carry/borrow/extend bit.
Z (bit 30) is the zero bit.
N (bit 31) is the negative/less than bit.

Diferenças: Instruções com condição

A maior parte das instruções da ISA ARMv7 são condicionais

Isto é, possuem 4 bits (cond) que determinam se a instrução deve ser executada ou não dependendo dos bits ZNCV do registrador CPSR.

Table A8-1 Condition codes

cond	Mnemonic extension	Meaning (integer)	Meaning (floating-point) [a]	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS [b]	Carry set	Greater than, equal, or unordered	C == 1
0011	CC [c]	Carry clear	Less than	C == 0
0100	MI	Minus, negative	Less than	N == 1
0101	PL	Plus, positive or zero	Greater than, equal, or unordered	N == 0
0110	VS	Overflow	Unordered	V == 1
0111	VC	No overflow	Not unordered	V == 0
1000	HI	Unsigned higher	Greater than, or unordered	C == 1 and Z == 0
1001	LS	Unsigned lower or same	Less than or equal	C == 0 or Z == 1
1010	GE	Signed greater than or equal	Greater than or equal	N == V
1011	LT	Signed less than	Less than, or unordered	N != V
1100	GT	Signed greater than	Greater than	Z == 0 and N == V
1101	LE	Signed less than or equal	Less than, equal, or unordered	Z == 1 or N != V
1110	None (AL) [d]	Always (unconditional)	Always (unconditional)	Any

Diferenças: Instruções com condição

-Gera códigos bem mais densos.

Ex.: cálculo do MDC(i,j)

```
int mdc(int i, int j)
{
   while (i != j)
      if (i > j)
         i = i - j;
      else
         j = j - i;
   return i;
 }
```

Obs.: cmp r0, r1 é implementado como r0-r1

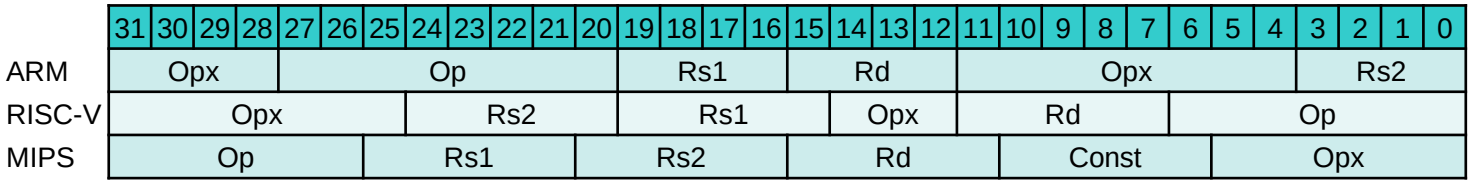
```
mdc: cmp r0, r1
     bxne lr
     subgt r0, r0, r1
     suble r1, r1, r0
     b mdc
```

```
mdc:  bne a0, a1, .L5
.L2:  mv a0, a1
      jalr zero, ra, 0x00
.L3:  sub a1, a1, a0
.L4:  beq a1, a0, .L2
.L5:  bge a1, a0, .L3
      sub a0, a0, a1
      jal zero, .L4
```

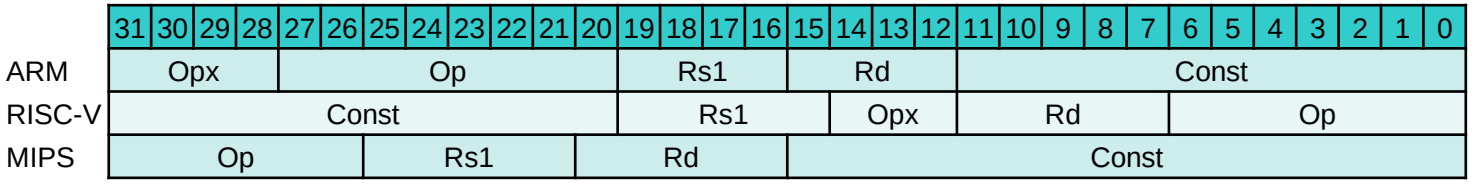
Assim, por exemplo, um salto condicional é um salto incondicional com condição 🤔

Codificação das Instruções

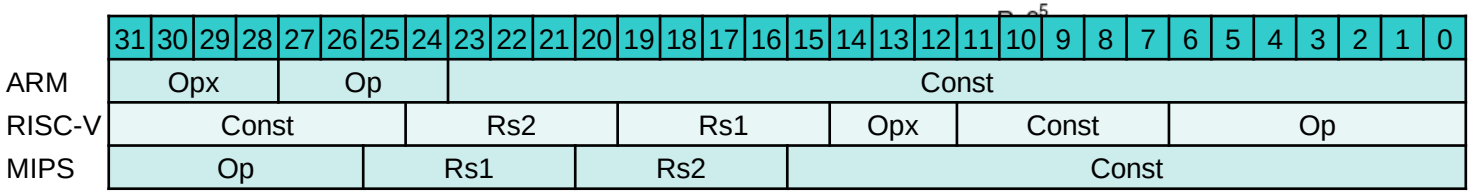
Register-Register



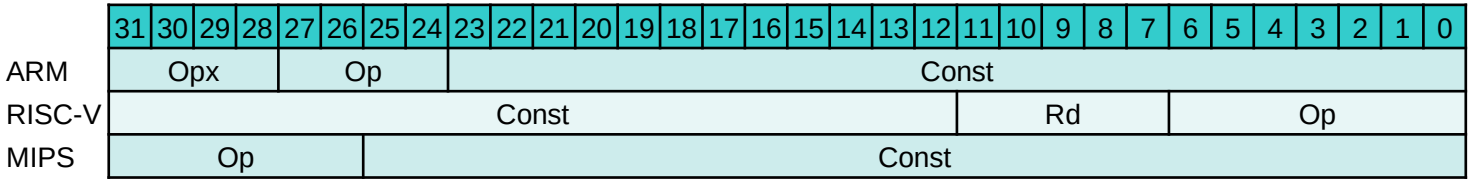
Data Transfer



Branch

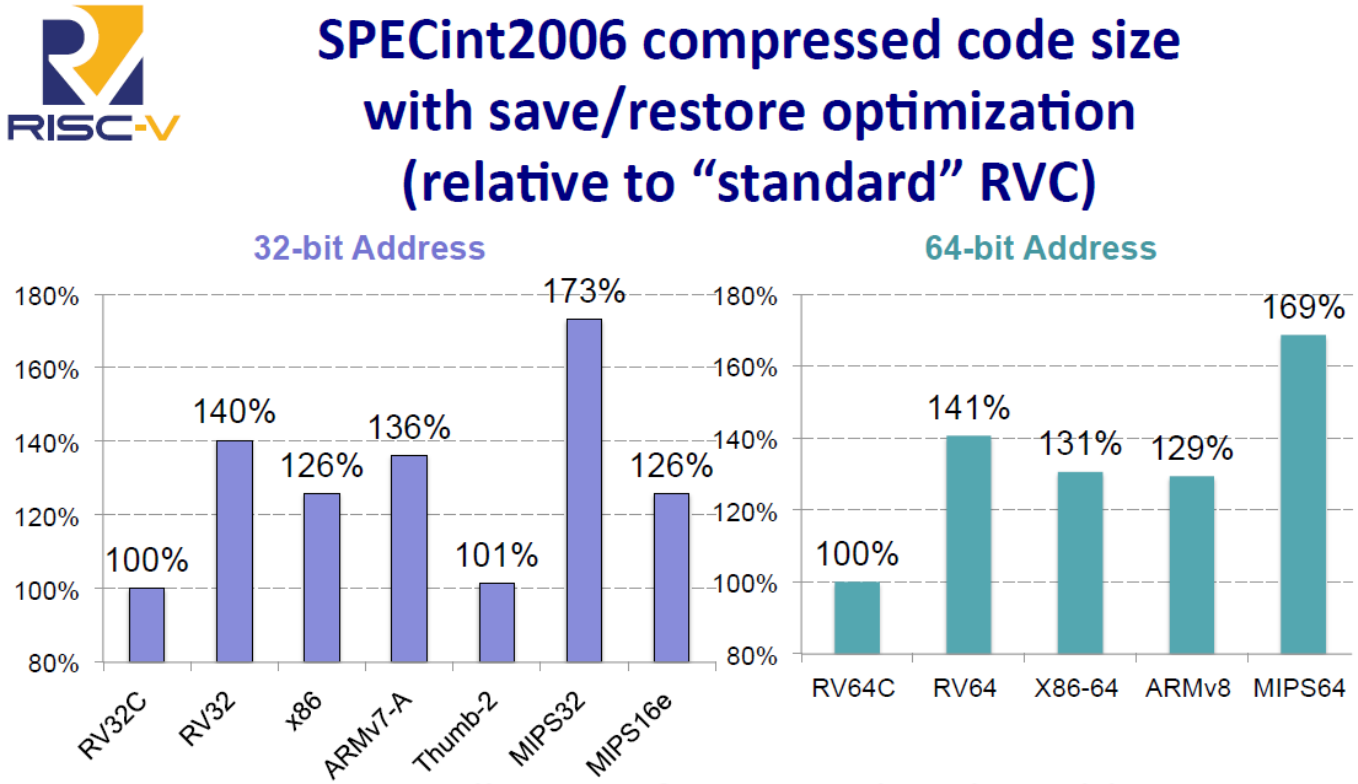


Jump/Call



ISA Compacta – 16bits

Assim como o RISC-V e o MIPS, o ARM possui ainda o modo Thumb, onde as instruções são simplificadas e codificadas em 16 bits (código mais compacto ainda!)



- RISC-V now smallest ISA for 32- and 64-bit addresses
- All results with same GCC compiler and options

Ps.:Comparação tendenciosa?!

Afinal ARM v7 é RISC ou CISC?

Embora seja projetado com a filosofia RISC (Load/Store), o ARM possui diversas complexidades, modos de endereçamento e... instruções complexas!

Ex.:

```
STMFD sp!,{r0-r12, lr} ; stack all registers
........               ; and the return address
........
LDMFD sp!,{r0-r12, pc}  ; load all the registers
                        ; and return automatically
```

The Intel x86 ISA

- Evolução e Retrocompatibilidade
  - 8080 (1974): 8-bit microprocessor
    - Accumulator, plus 3 index-register pairs
  - 8086 (1978): 16-bit extension to 8080
    - Complex instruction set (CISC)
  - 8087 (1980): floating-point coprocessor
    - Adds 60 FP instructions and register stack
  - 80286 (1982): 24-bit addresses, MMU
    - Segmented memory mapping and protection
  - 80386 (1985): 32-bit extension (now IA-32)
    - Additional addressing modes and operations
    - Paged memory mapping as well as segments

The Intel x86 ISA

- i486 (1989): pipelined, on-chip caches and FPU
  - Compatible competitors: AMD, Cyrix, …
  - Added only 4 new instructions
- Pentium (1993): superscalar, 64-bit datapath
  - Later versions added 57 MMX (Multi-Media eXtension) instructions
  - The infamous FDIV bug
- Pentium Pro (1995), Pentium II (1997)
  - New microarchitecture
  - No new instructions
- Pentium III (1999)
  - Added 70 new SSE (Streaming SIMD Extensions) instructions and associated 128 bits registers
- Pentium 4 (2001)
  - New microarchitecture
  - Added 144 new SSE2 instructions

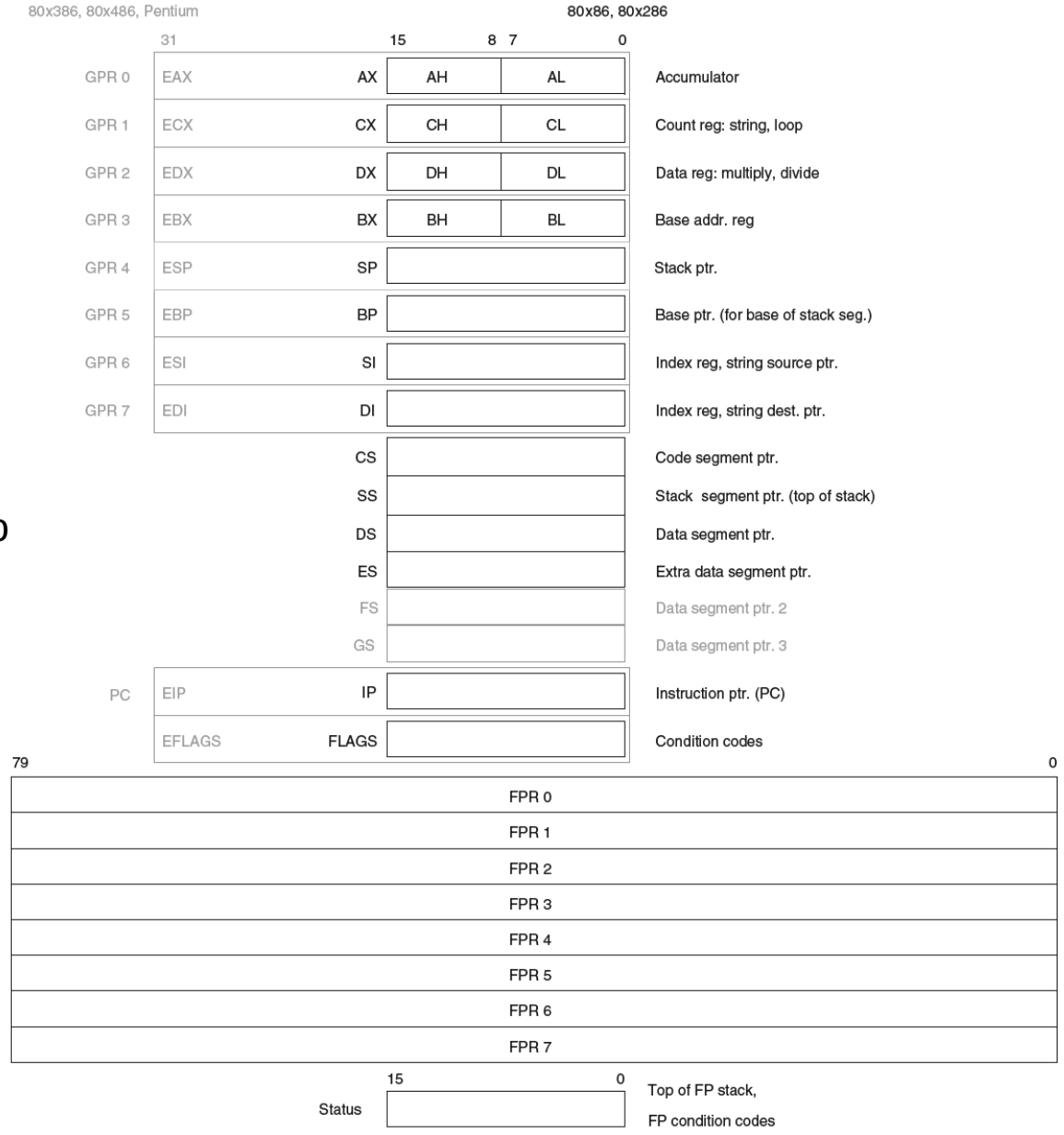
The Intel x86 ISA

- AMD64 (2003): extended architecture to 64 bits
- EM64T – Extended Memory 64 Technology (2004)
  - AMD64 adopted by Intel (with refinements)
  - Added 13 new SSE3 instructions (complex operations)
- Intel Core (2006)
  - Added 54 new SSE4 instructions, virtual machine support
- AMD64 (2007): 170 new SSE5 instructions
  - Intel declined to follow, instead…
- Intel i7
  - Advanced Vector Extension - AVX (2008)
  - Longer SSE registers (256 bits), redefined 250 instructions and added more 128 new instructions
  - Introduced 47 new 3 operands instructions (like MIPS and RISC-V)
- Intel i7 terceira geração
  - AVX2 e FMA3 e 4 (2011 e 2013)
  - Longer SSE registers, more 24 instructions

Banco de registradores básico

Incremento histórico:
8085 : 8 bits
8086 : 16 bits
80386 : 32 bits
Opteron : 64 bits

Obs.: Itanium VLWI
128 registers
Compilador mais complexo



Modos de endereçamento básicos

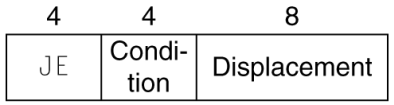
- Somente dois operandos por instrução

Source/dest operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

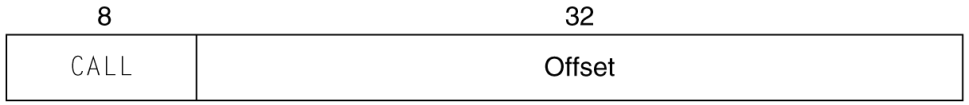
- Modos de Endereçamento da Memória
  - Address in register
  - Address = R_{base} + displacement
  - Address = $R_{base} + 2^{scale} \times R_{index}$ (scale = 0, 1, 2, or 3)
  - Address = $R_{base} + 2^{scale} \times R_{index}$ + displacement

Exemplo de codificação das Instruções x86

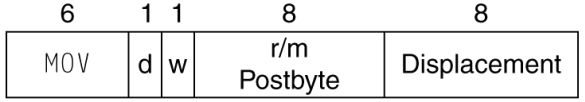
a. JE EIP + displacement



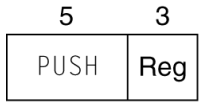
b. CALL



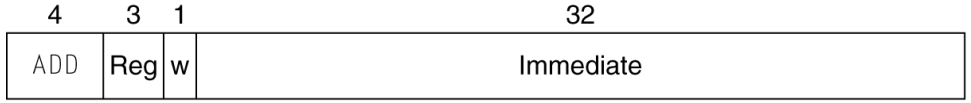
c. MOV EBX, [EDI + 45]



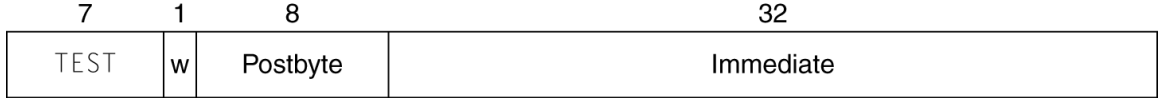
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



- Codificação de tamanho variável de acordo com a instrução
  - Postfix bytes especificam o modo de endereçamento
  - Prefix bytes modificam a operação
  - Próxima instrução não é PC+4! Precisa ser calculado!
- Tamanho dos operandos, repetição das instruções, travamentos, …, definidos pela instrução.

Ex.: MDC(i,j)

```
int mdc(int i, int j)
{
   while (i != j)
      if (i > j)
         i = i - j;
      else
         j = j - i;
return i;
 }
```

```
mdc:    movl %ecx, %eax
        cmpl %edx, %ecx
        je .L2
.L5:    cmpl %edx, %eax
        jle .L3
        subl %edx, %eax
        jmp .L4
.L3:    subl %eax, %edx
.L4:    cmpl %eax, %edx
        jne .L5
.L2:    ret
```

```
mdc:  bne a0, a1, .L5
.L2:  mv a0, a1
      jalr zero, ra, 0x00
.L3:  sub a1, a1, a0
.L4:  beq a1, a0, .L2
.L5:  bge a1, a0, .L3
      sub a0, a0, a1
      jal zero, .L4
```

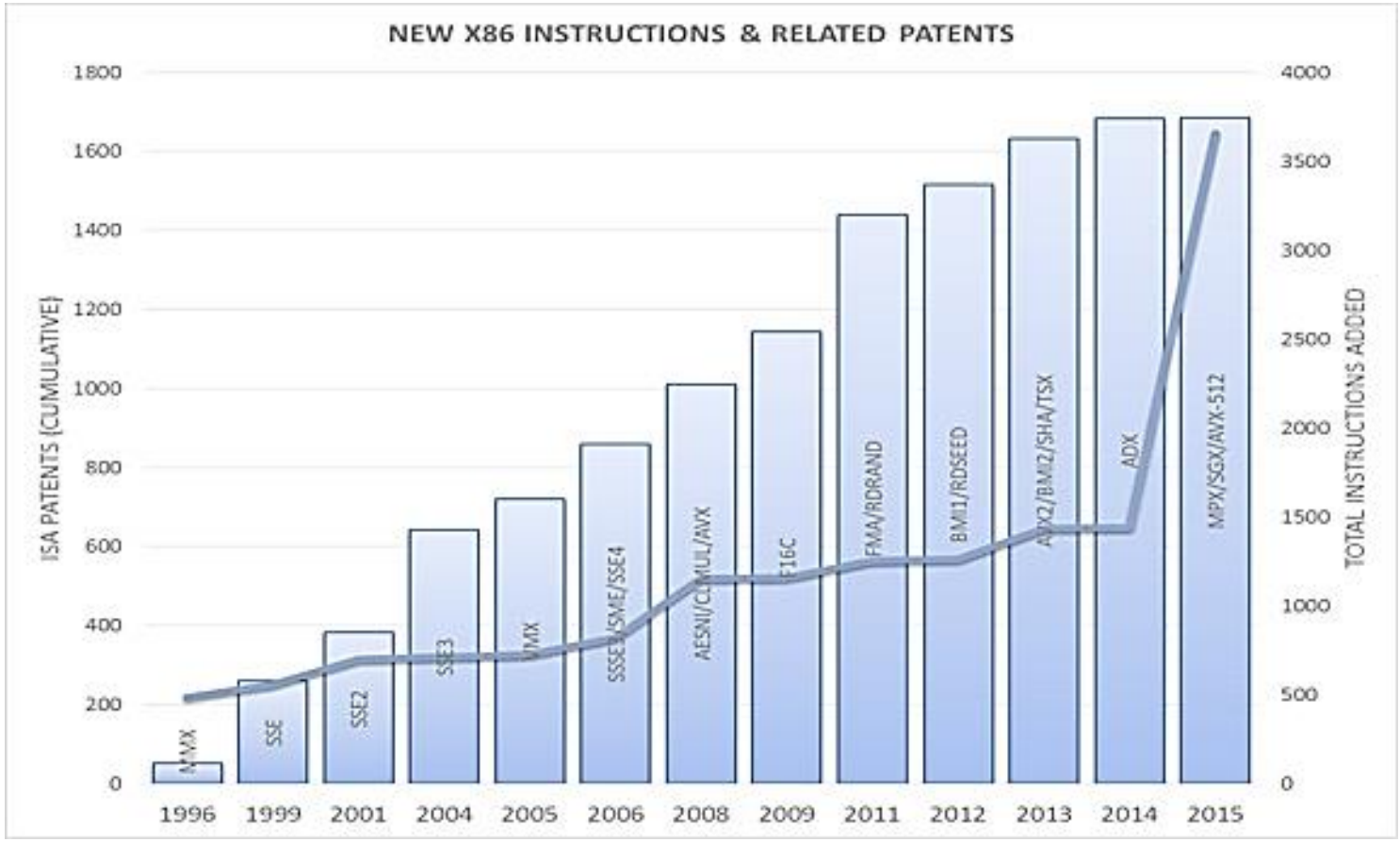
Implentando a arquitetura IA-32

- O conjunto de instruções complexo dificulta muito a implementação
  - O hardware traduz as instruções da ISA em microoperações (microinstruções)
    - Instruções Simples: 1–1
    - Instruções Complexas: 1–várias
  - Microengine (processador de instruções) similar a um RISC
- Esta metodologia será apresentada no projeto Multiciclo
- “Desempenho comparável aos RISC” (?!?!? Tendenciosa?)
- Cabe aos compiladores tirar proveito da complexidade!

Falácias

- Instruções Poderosas ⇒ Melhor Desempenho
  - São requeridas poucas instruções
  - MAS, instruções complexas são difíceis de implementar
    - Pode tornar mais lentas todas as instruções, até as mais simples!
- Uso da Linguagem Assembly ⇒ Melhor Desempenho
  - Os compiladores modernos são melhores para lidar com os modernos e complexos processadores
  - Quanto mais linhas de código ⇒ maior probabilidade de erros e menor produtividade!

- Retrocompatibilidade ⇒ Conjunto de instruções não muda
  - MAS, são acrescentadas novas instruções!



https://newsroom.intel.com/editorials/x86-approaching-40-still-going-strong/

Conclusão

In 1994, AMD's 80x86 architect, Mike Johnson,
famously quipped,
“The x86 really isn't all that complex,
it just doesn't make a lot of sense"