ISA: Comparação entre conjuntos de instruções de processadores contemporâneos (ARM e x86)

ARM: “Advanced RISC Machine”

- **ISA mais popular:**
  - **Computação Embarcada e Computação Móvel**
  - **2011: 9 bilhões de dispositivos**
  - **Recentemente: 2 bilhões por ano**
- **ARM e MIPS têm várias similaridades**
  - **Foram lançados no mesmo ano (1985)**
  - **Seguem a mesma filosofia (RISC)**
    - » **Operações primitivas: instruções/modos simples**
    - » **Máquina load/store**
    - » **Formato de 3 registradores (2 fonte e 1 destino)**
  - **MIPS tem mais registradores (até ARMv7)**
  - **ARM tem mais modos de endereçamento**

Similaridades entre MIPS e ARMv7

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Instruções funcionalmente equivalentes

	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

Diferenças nos modos de endereçamento

Addressing mode	ARM v.4	MIPS
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	—

Diferenças nos modos de endereçamento

Addressing mode	ARM v.4	MIPS
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	—

LDR r0, [r1, #4] ⇔ lw $s0, 4($s1) # s0 = MEM[s1 + 4]

Diferenças nos modos de endereçamento

Addressing mode	ARM v.4	MIPS
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	—

LDR r0, [r1, r2] ⇔ lw $s0, $s1($s2) ⇔
add $at, $s1, $s2; lw $s0, 0($at) # s0 = MEM[s1 + s2]

Diferenças nos modos de endereçamento

Addressing mode	ARM v.4	MIPS
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	—

LDR r0, [r1, r2, LSL#2] ⇔ lw $s0, $s1($s2 <<2) ⇔
sll $at, $s2, 2; add $at, $at, $s1, lw $s0, 0($at) # s0 = MEM[s1 + s2*4]

Diferenças nos modos de endereçamento

Addressing mode	ARM v.4	MIPS
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	—

LDR r0, [r1, #4]! ⇔ (atualizar antes da indexação: update pre-index)
addi $s1, $s1, 4; lw $s0, 0($s1); # s1 = s1 + 4; s0 = MEM[s1]

Diferenças nos modos de endereçamento

Addressing mode	ARM v.4	MIPS
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	—

LDR r0, [r1, #4]! ⇔ (atualizar antes da indexação: update pre-index)
addi $s1, $s1, 4; lw $s0, 0($s1); # s1 = s1 + 4; s0 = MEM[s1]

LDR r0, [r1, r2]! ⇔ (atualizar antes da indexação: update pre-index)
add $s1, $s1, $s2; lw $s0, 0($s1); # s1 = s1 + s2; s0 = MEM[s1]

Diferenças na avaliação de desvios condicionais

- MIPS usa registradores de uso geral
  - slt $t0, $s1, $s2
  - bne $t0, $zero, L
- ARM usa registrador especial
  - Registrador de status
    - » PSW: Program Status Word
  - Códigos de condição: 4 bits de PSW
    - » N: negativo
    - » Z: zero
    - » C: carry
    - » O: overflow

ARM: códigos de condição

Flag	Nome	Condição
C	Carry flag	Ativado se Carry-out (MSB)=1
Z	Zero flag	Ativado se resultado nulo
N	Negative flag	Ativado se resultado negativo
O	Overflow flag	Ativado se transbordo

Generalização: códigos de condição

- ***Flags*** **ativados ou desativados por uma operação**
  - **Comparação (CMP r1, r2) ou aritmética (SUB r1, r2, r3)**
    - » **Algumas instruções aritméticas podem ativar *flags***
- **Diagnóstico a partir do <u>resultado</u> da operação**
  - **CMP a,b → a-b (Z=1 se a=b; N=1 se a < b; N=0 se a≥b)**
  - ***Flags*****: Z, N, O, C (bits de registrador de status)**
- **Diagnóstico utilizado por desvios condicionais**
  - **BEQ, BLT, BGE**
- **Dependências não explícitas no código**
  - **Compilador tem que fazer análises mais complexas ou**
  - **Ser conservador ao reordenar o código**
- **Mas este é o mecanismo usado no ARMv7 e x86**

Diferenças no uso de desvios condicionais

if (a < b) x = x + 1 else x = x – 1 ; ...

Diferenças no uso de desvios condicionais

if (a < b) x = x + 1 else x = x – 1 ; ...

MIPS

```
        slt $t0, $s1, $s2        ; compare a e b
        beq $t0, $zero, else     ; desvie se a ≥ b
        addi $s3, $s3, 1
        j after
else:   addi $s3, $s3, -1
after:  ...
```

(a, b, x) →(s1, s2, s3);

Diferenças no uso de desvios condicionais

if (a < b) x = x + 1 else x = x – 1 ; ...

MIPS

```
        slt $t0, $s1, $s2        ; compare a e b
        beq $t0, $zero, else     ; desvie se a ≥ b
        addi $s3, $s3, 1
        j after
else:   addi $s3, $s3, -1
after:  ...
```

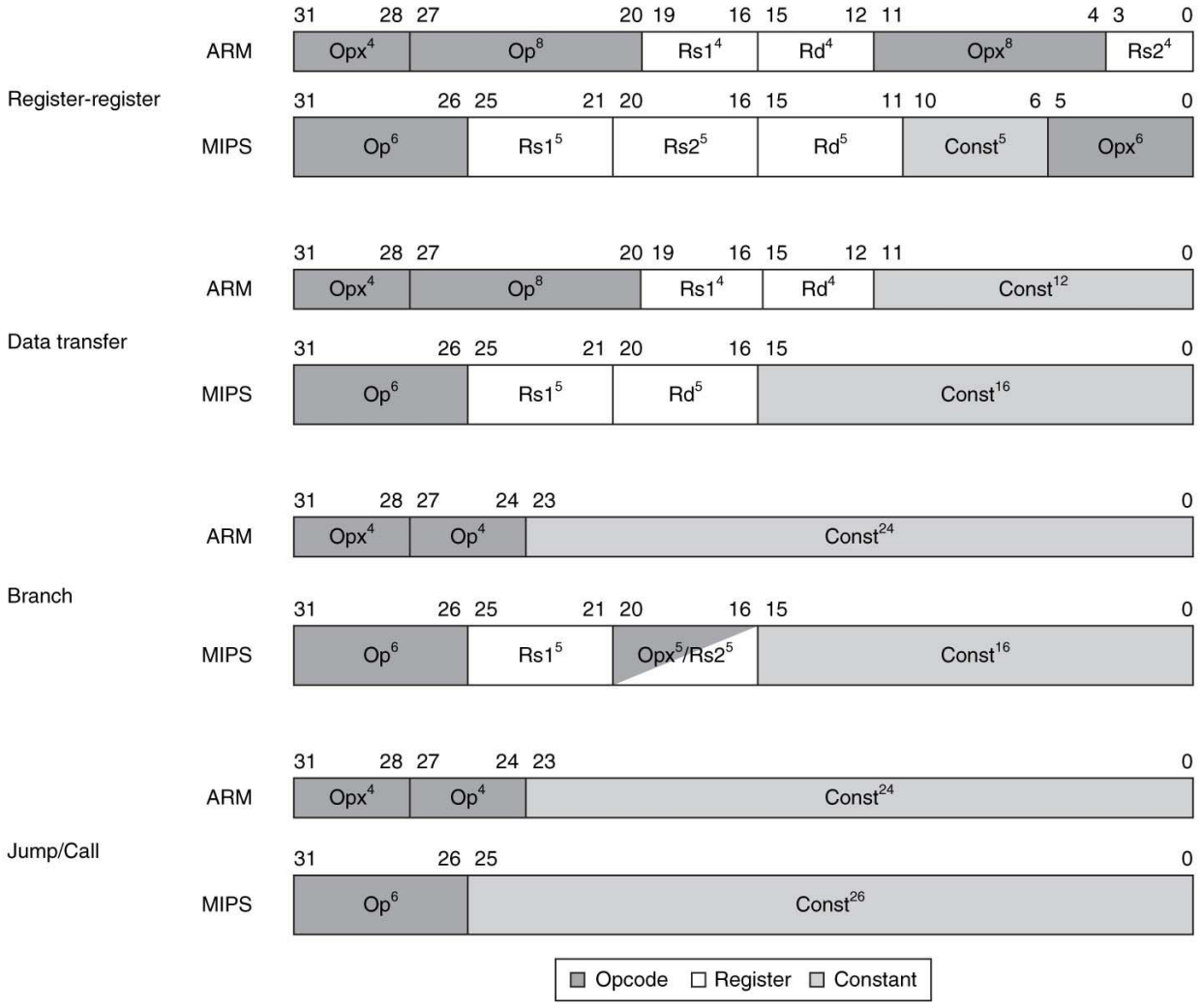
(a, b, x) →(s1, s2, s3);

ARM

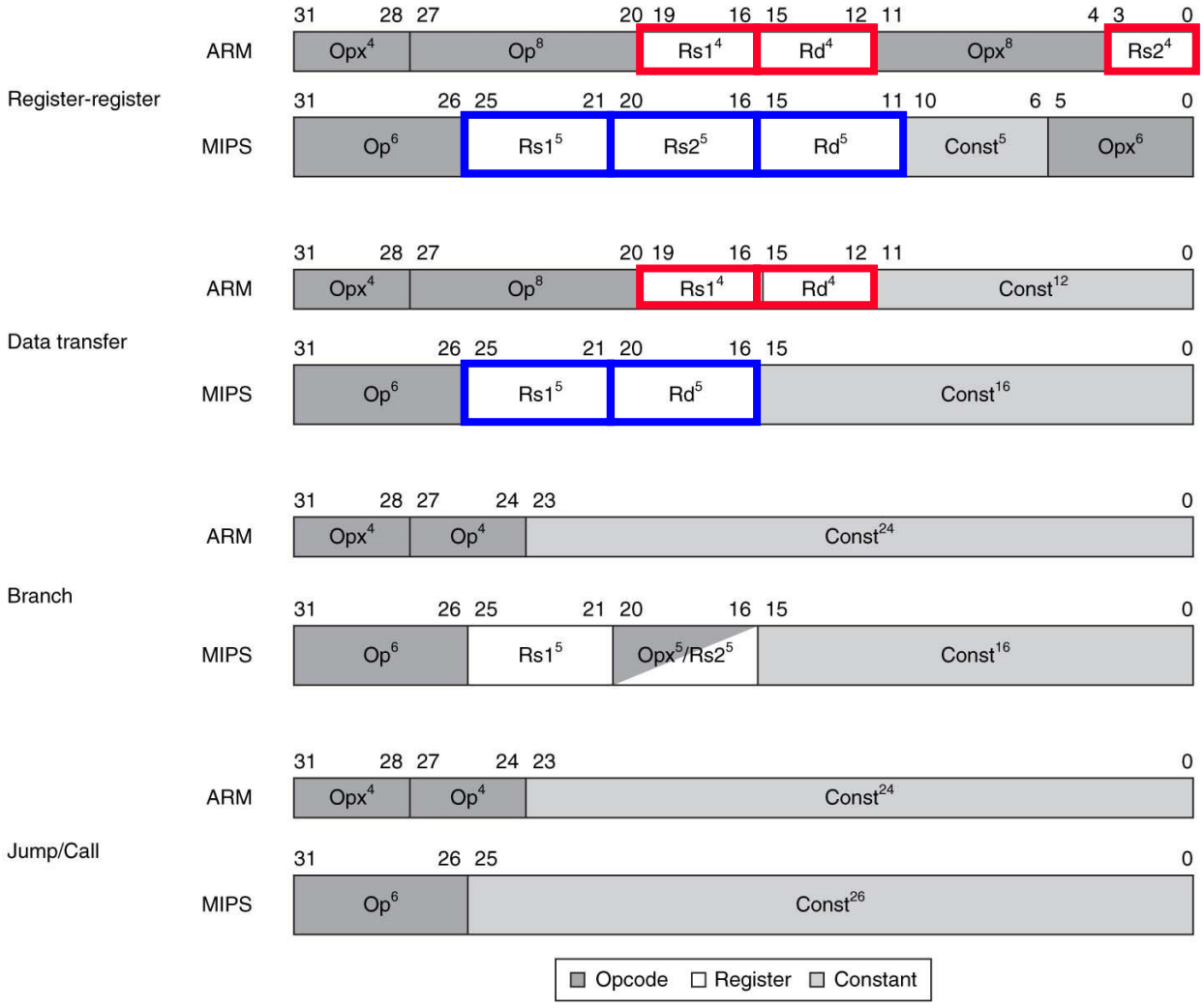
```
        CMP r1, r2               ; compare a e b
        BGE else                 ; desvie se a ≥ b
        ADD r3, r3, #1
        B after
else:   SUB r3, r3, #1
after:  ...
```

(a, b, x) →(r1, r2, r3);

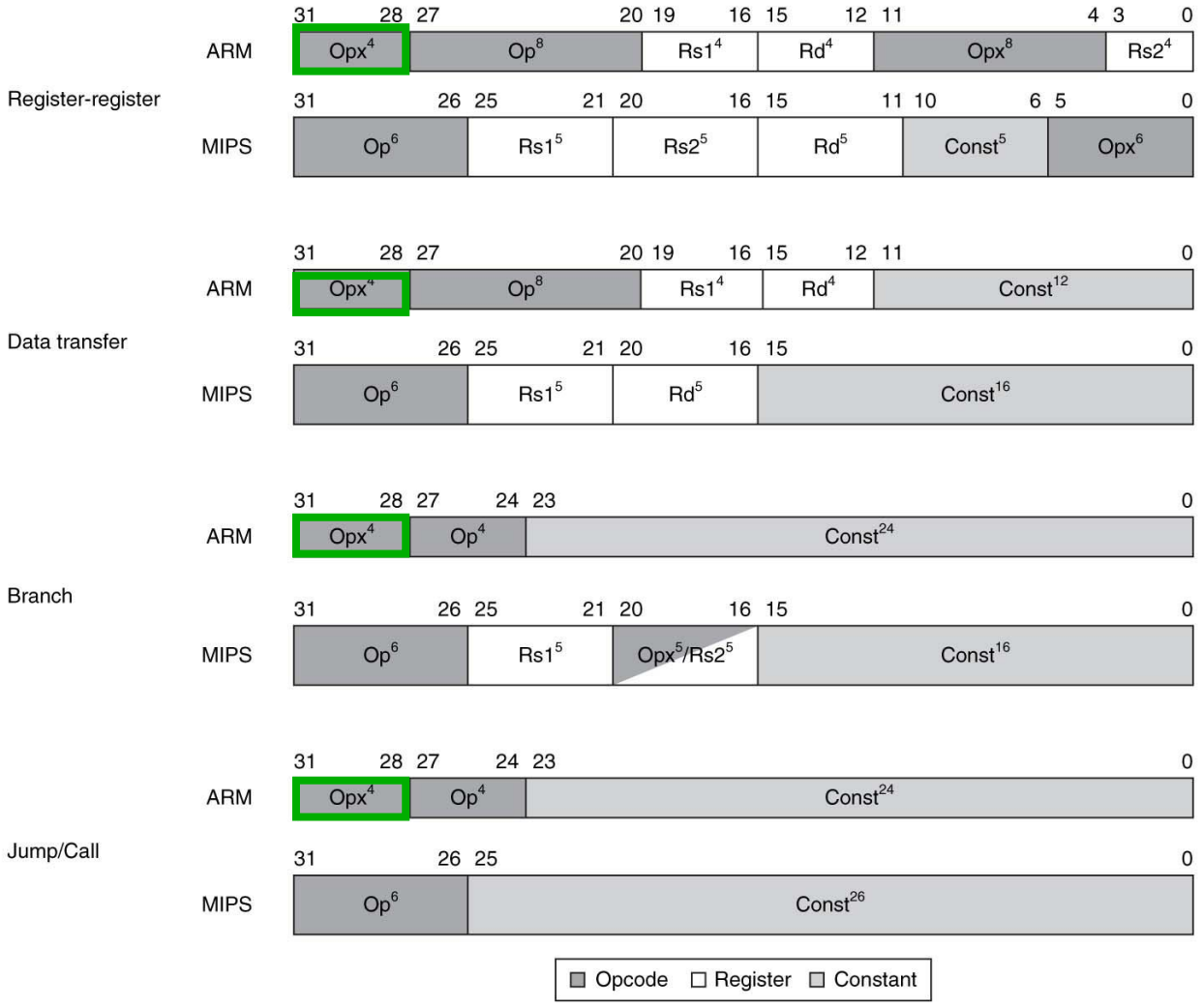
Diferenças nos formatos de instrução



Diferenças nos formatos de instrução



Diferenças nos formatos de instrução



Especificidade do ARM: execução conditional

- **Exemplo: (a, b, x) →(r1, r2, r3);**

  if (a < b) x = x + 1 else x = x – 1 ; ...

Especificidade do ARM: execução conditional

- **Exemplo: (a, b, x) →(r1, r2, r3);**

  if (a < b) x = x + 1 else x = x – 1 ; ...

- **Implementação tradicional:**

```
                CMP r1, r2          ; compare a e b
                BGE else            ; desvie se a ≥ b
                ADD r3, r3, #1
                B after
else:           SUB r3, r3, #1
after:          ...
```

Especificidade do ARM: execução conditional

- **Exemplo:** (a, b, x) →(r1, r2, r3);

  if (a < b) x = x + 1 else x = x – 1 ; ...

- **Implementação tradicional:**

```
                CMP r1, r2          ; compare a e b
                BGE else            ; desvie se a ≥ b
                ADD r3, r3, #1
                B after
else:           SUB r3, r3, #1
after:          ...
```

- **Implementação com instruções condicionais:**

```
                CMP r1, r2          ; compare a e b
                ADDLT  r3, r3, #1
                SUBGE r3, r3, #1
```

Histórico das ISAs x86

- **1978: Intel 8086**
  - **Extensão do 8080 (8 bits)**
    - **Registradores e endereçamento: 16 bits**
  - **Sem registradores de propósitos gerais**
- **1980: Intel 8087**
  - **Co-processador para ponto flutuante (PF)**
  - **Extensão: 60 instruções de PF**
    - **Operandos mantidos em pilha**

Histórico das ISAs x86

- **1982: Intel 80286**
  - **Extensão do 8086**
    - **Registradores: 16 bits**
    - **Endereçamento: 24 bits**
  - **Umas poucas instruções adicionadas**
- **1985: Intel 80386**
  - **Extensão do 80286**
    - **Registradores e endereçamento: 32 bits**
  - **Novas instruções e modos**
    - **8 registradores de uso geral (32 bits)**

Histórico das ISAs x86

- **1989-95: 80486, Pentium e Pentium Pro**
  - **Implementações para desempenho**
  - **Só quatro instruções novas**
- **1997: MMX: Multi Media Extension**
  - **8 registradores de 64 bits: mm0 a mm7**
  - **57 novas instruções;**
    - » **Datapath: 64 bits divisível em fatias (8, 16, 32)**
    - » **Operações idênticas em múltiplas fatias**
    - » paddb mmi, mmi; paddw mmi, mmi; paddd mmi, mmi
    - » **Exemplo de aplicação: fatia = pixel**

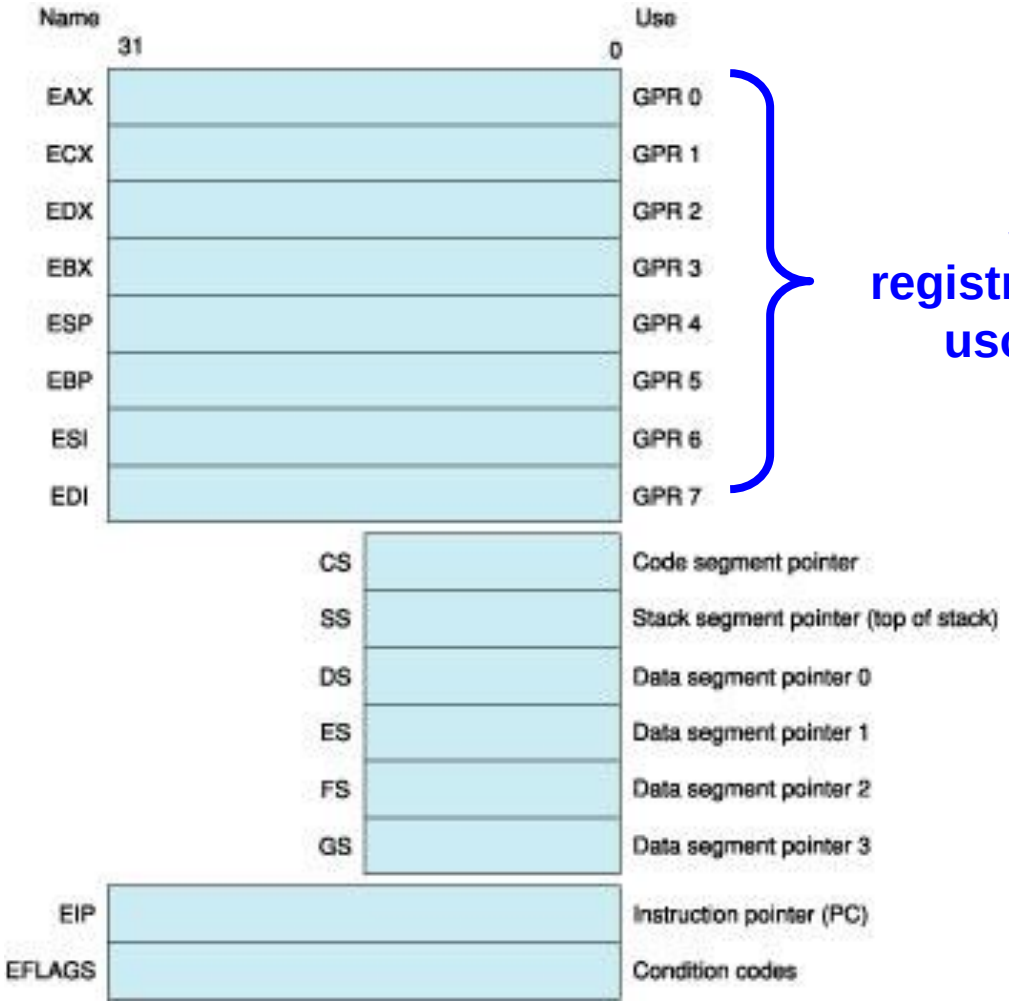
Histórico das ISAs x86

- **1997: Pentium II**
  - **Nenhuma nova instrução**
- **1999: SSE - Pentium III**
  - **Streaming SIMD Extensions**
    - **70 novas instruções**
    - **8 registradores extra de 128 bits**
    - **Suporte para FP em precisão simples**
    - **4 operações PF 32 bits em paralelo**

Histórico das ISAs x86

- **2001: Extensão SSE 2 (Pentium 4)**
  - **Suporte p/ aritmética em precisão dupla**
  - **2 operações PF de 64 bits em paralelo**
  - **144 novas instruções**
  - **Impacto em ponto flutuante**
    - » **Alternativa às instruções PF baseadas em pilha**
    - » **Operandos em registradores SSE**

x86-32: registradores



Só 8 registradores de uso geral!

x86-32: operandos

- **Um operando fonte é também destino**
  - **ADD R1, R2 # R1 = R1 + R2**

Consequência de destino = fonte1

- **Se ambos fontes precisam ser preservados**
  - **Um deles precisa ser salvo em temporário**
- **Exemplo 1:**
- **a = a + b; (a, b) → ($s1, $s2)**

MIPS	x86-32(*)
add $s1, $s1, $s2	add $s1, $s2

- **a = b + c; (a, b, c) → ($s1, $s2, $s3)**

MIPS	x86-32
add $s1, $s2, $s3	move $s1, $s2
	add $s1, $s3

Quero preservar também o valor de $s2!

(*) representado esquematicamente em linguagem de montagem com sintaxe similar à do MIPS

x86-32: operandos

- **Um operando pode residir em memória**
  - **x86-32 não é máquina load/store**

Fonte/Destino	Fonte	Exemplos(*)
registrador	registrador	add $s1 , $s2
registrador	imediato	addi $s1 , K
registrador	memória	add $s1 , ($s2)
memória	registrador	add ($s1) , $s2
memória	imediato	addi ($s1) , K

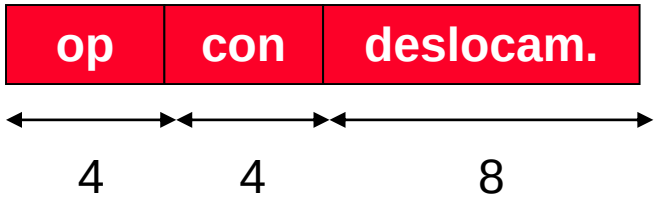
(*) representados esquematicamente em linguagem de montagem com sintaxe similar à do MIPS

x86-32: amostra de instruções

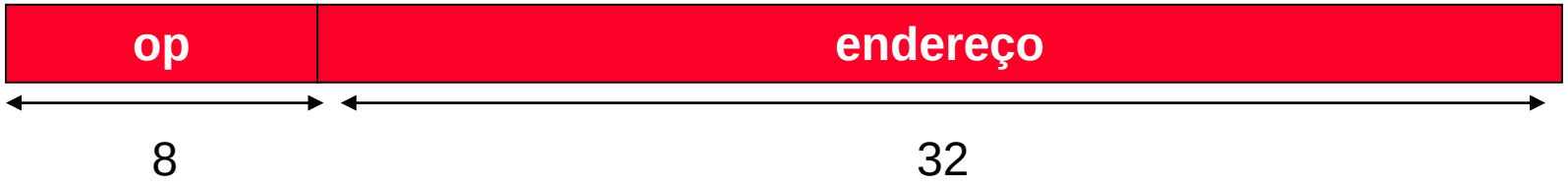
- **JE name**
  - **If (CC ⇒ 'equal') PC = PC + name;**
  - **Similar a beq no MIPS**
- **JMP name**
  - **PC = name**
  - **Similar a j no MIPS**
- **CALL name**
  - **SP = SP - 4; M[SP] = PC + 5; PC = name**
  - **Similar a jal no MIPS**
- **PUSH e POP**
  - **Pseudo-instruções no MIPS**

x86-32: formatos de instrução

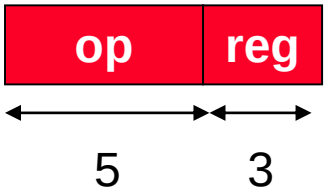
JE



CALL



PUSH



Formato variável:

compactação de código x paralelismo no pipeline

x86-32: Códigos de condição

Flag	Nome	Condição
C	Carry flag	Ativado se Carry-out (MSB)=1
Z	Zero flag	Ativado se resultado nulo
S	Signal flag	Ativado se resultado negativo
O	Overflow flag	Ativado se transbordo

x86-32 : modos de endereçamento

- **Absoluto:**
  add $s1, (1001) **# $s1 = $s1 + MEM[1001]**
- **Indireto:**
  add $s1, ($s2) **# $s1 = $s1 + MEM[$s2]**
- **Indexado:**
  add $s1, ($s2+$s3) **# $s1 = $s1 + MEM[$s2+$s3]**
- **Base-deslocamento:**
  add $s1, K($s2) **# $s1 = $s1 + MEM[K + $s2]**
- **Base-deslocamento indexado:**
  add $s1, K($s2+$s3) **# $s1 = $s1 + MEM[K+$s2+$s3]**
- **Base indexado ampliado:**
  add $s1, ($s2) [$s3] **# $s1 = $s1 + MEM[$s2 + 2^n * $s3],**
  # com n = 0, 1, 2, 3
- **Base-deslocamento indexado ampliado:**
  add $s1, K($s2) [$s3] **# $s1 = $s1 + MEM[K+$s2 + 2^n * $s3],**
  # com n = 0, 1, 2, 3
- **Entre 7 e 11 modos de endereçamento!**
  - **Dependendo de como se conte**

x86-32 : modos de endereçamento

- **Absoluto: 20%**
  add $s1, (1001) **# $s1 = $s1 + MEM[1001]**
- **Indireto: 13%**
  add $s1, ($s2) **# $s1 = $s1 + MEM[$s2]**
- **Indexado:**
  add $s1, ($s2+$s3) **# $s1 = $s1 + MEM[$s2+$s3]**
- **Base-deslocamento: 40%**
  add $s1, K($s2) **# $s1 = $s1 + MEM[K + $s2]**
- **Base-deslocamento indexado:**
  add $s1, K($s2+$s3) **# $s1 = $s1 + MEM[K+$s2+$s3]**
- **Base indexado ampliado: 22%**
  add $s1, ($s2) [$s3] **# $s1 = $s1 + MEM[$s2 + 2^n * $s3],**
  # com n = 0, 1, 2, 3
- **Base-deslocamento indexado ampliado:**
  add $s1, K($s2) [$s3] **# $s1 = $s1 + MEM[K+$s2 + 2^n * $s3],**
  # com n = 0, 1, 2, 3
- **53% dos casos são "cobertos" com o modo base+deslocamento**

Fonte: Patterson and Hennessy, "Computer Architecture: A Quantitative Approach", MKP, 2nd edition, p.81, 1996.
Condições do experimento: média de 5 benchmarks SPECint92 (compress, espresso, eqntott, gcc, li)

x86-32 : as 10 mais frequentes

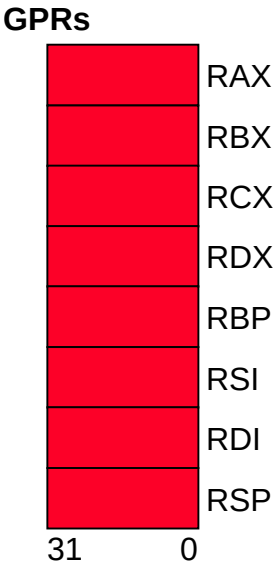
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		96%

Fonte: Patterson and Hennessy, “Computer Architecture: A Quantitative Approach”, MKP, 2nd edition, p.81, 1996.
Condições do experimento: média de 5 benchmarks SPECint92 (compress, espresso, eqntott, gcc, li)

Evolução: x86-32 → x86-64

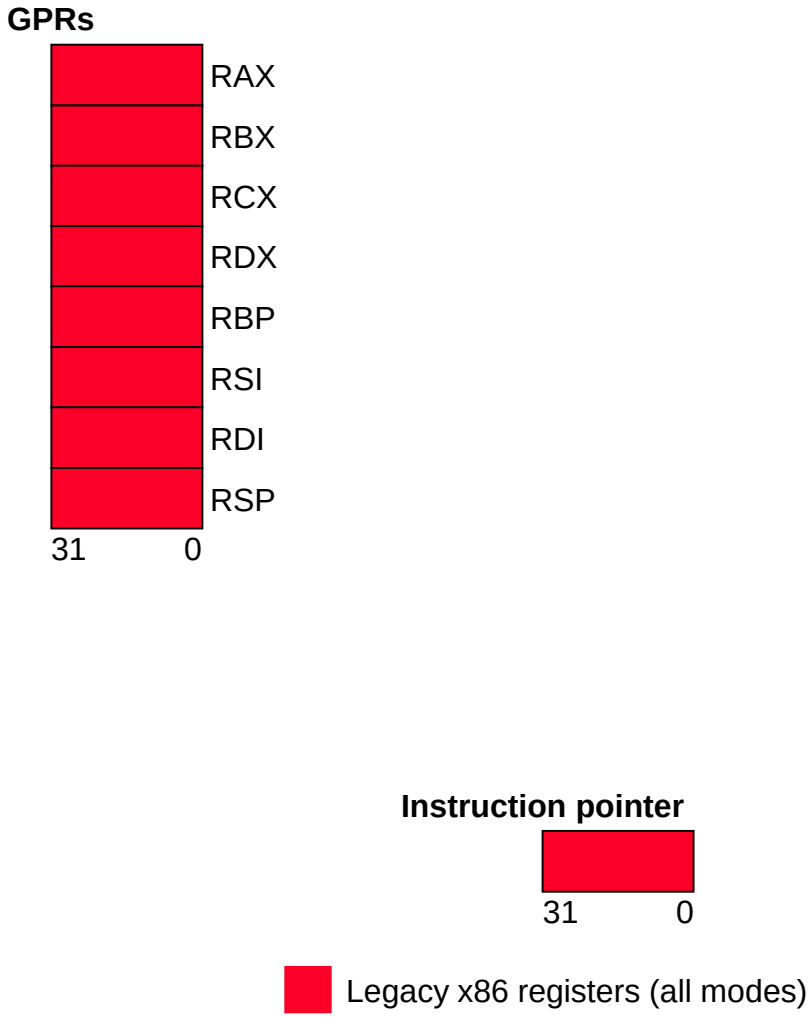
- **2003: AMD 64**
  - **Extensão da x86-32: x86-64**
    - **Endereçamento: 64 bits**
    - **No. de registradores inteiros (64 bits): 16**
    - **No. de registradores SSE (128 bits): 16**

Extensão: x86-32 → x86-64

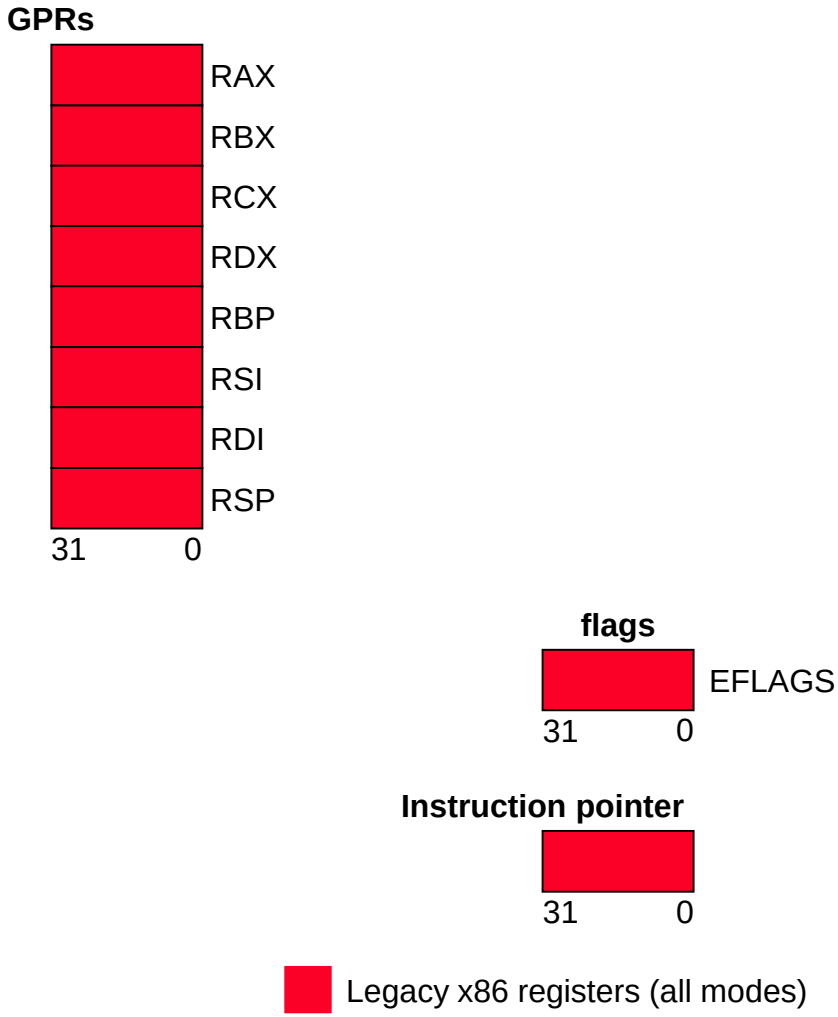


Legacy x86 registers (all modes)

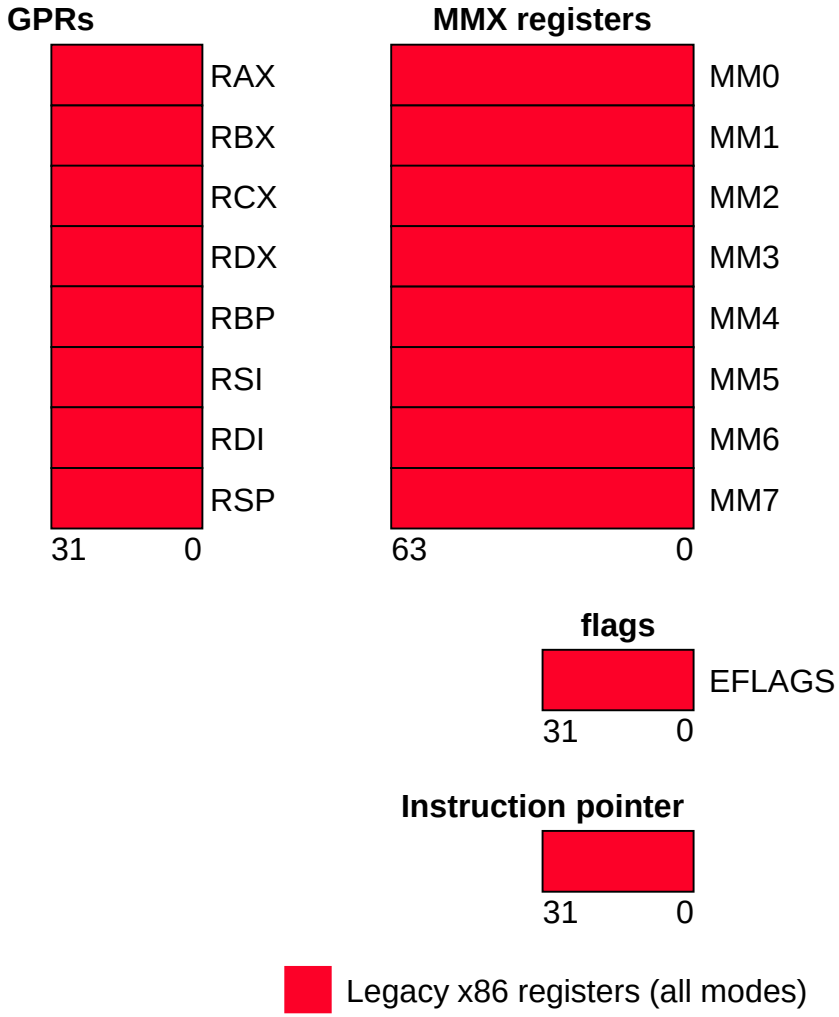
Extensão: x86-32 → x86-64



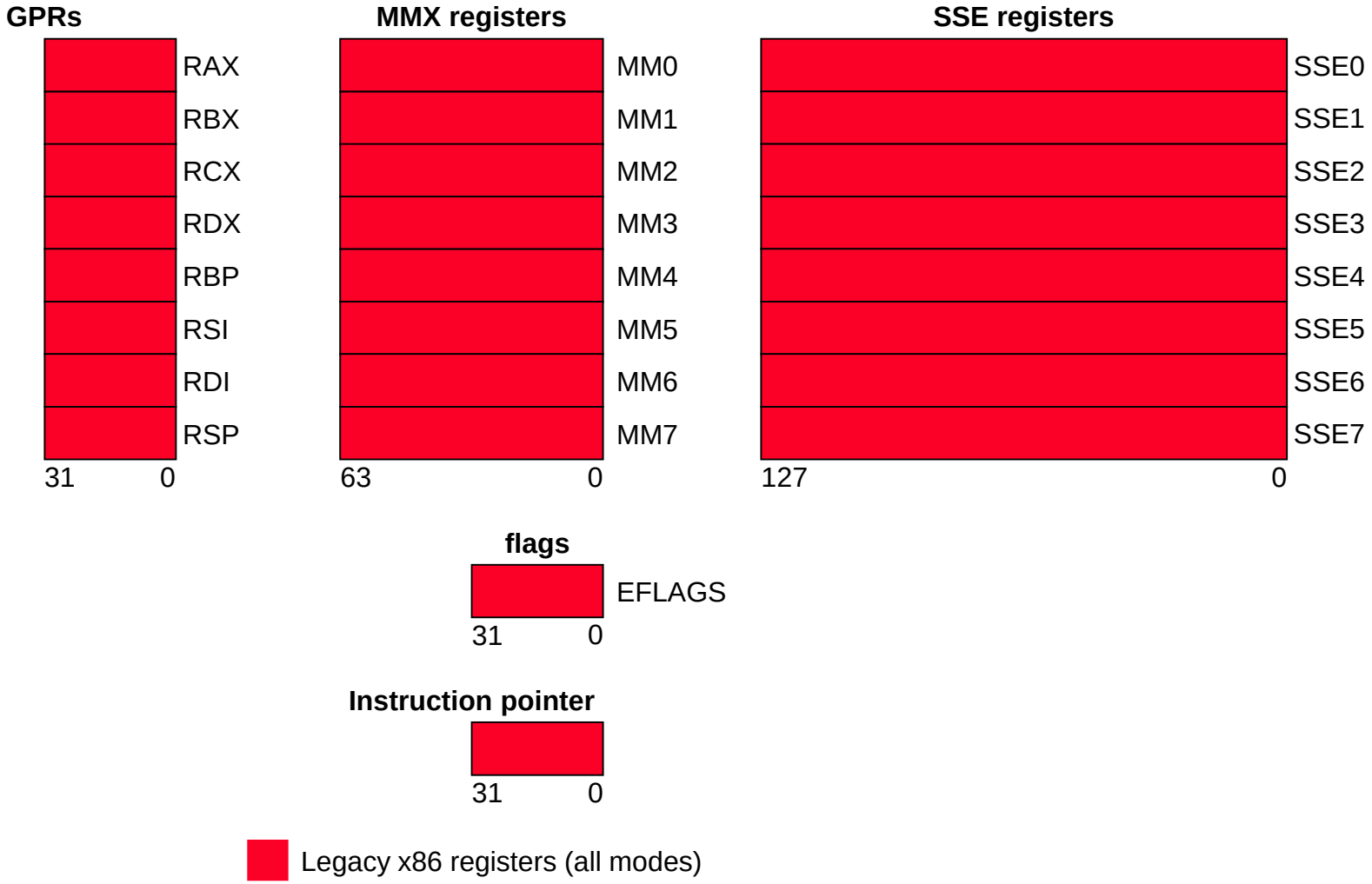
Extensão: x86-32 → x86-64



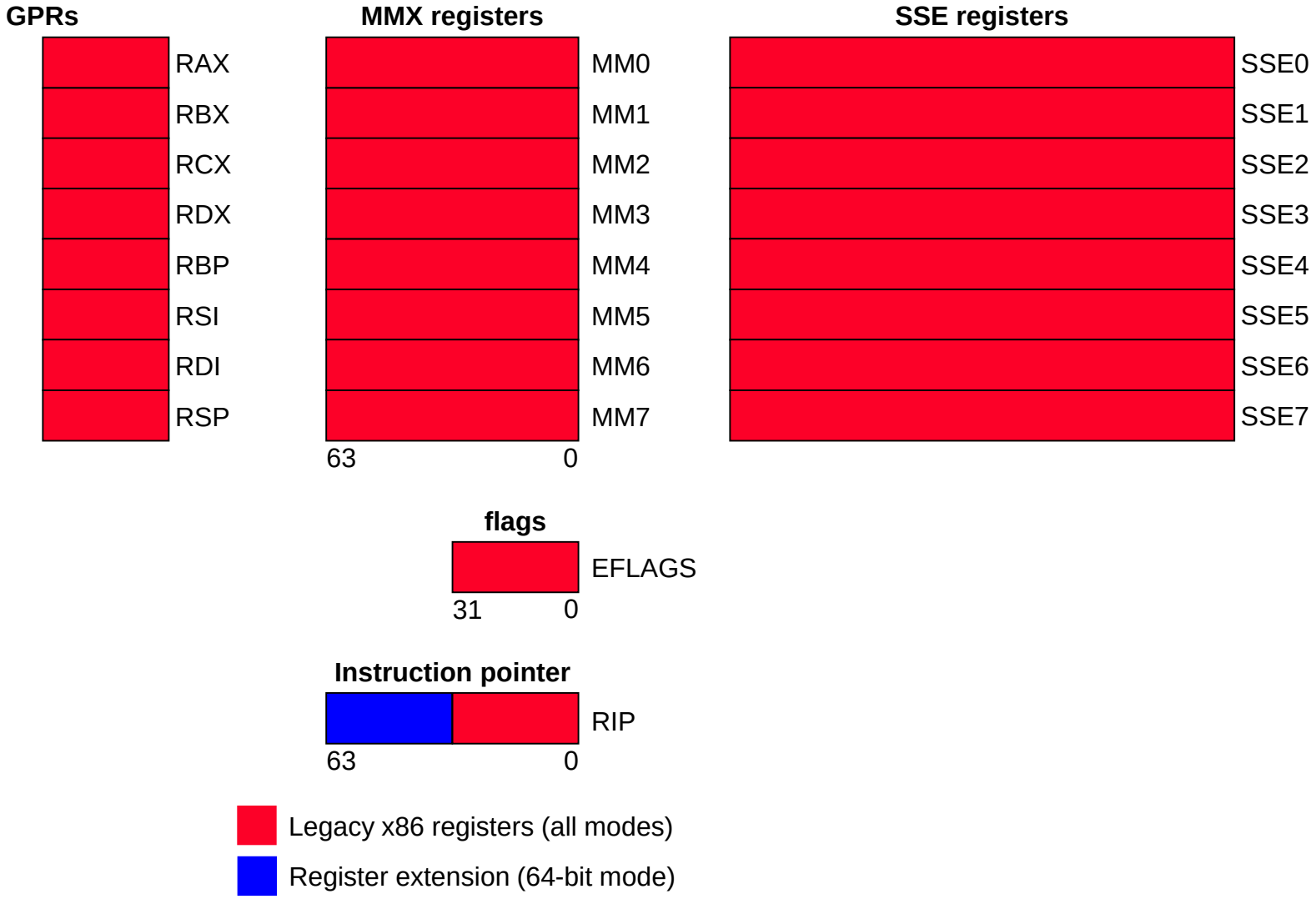
Extensão: x86-32 → x86-64



Extensão: x86-32 → x86-64

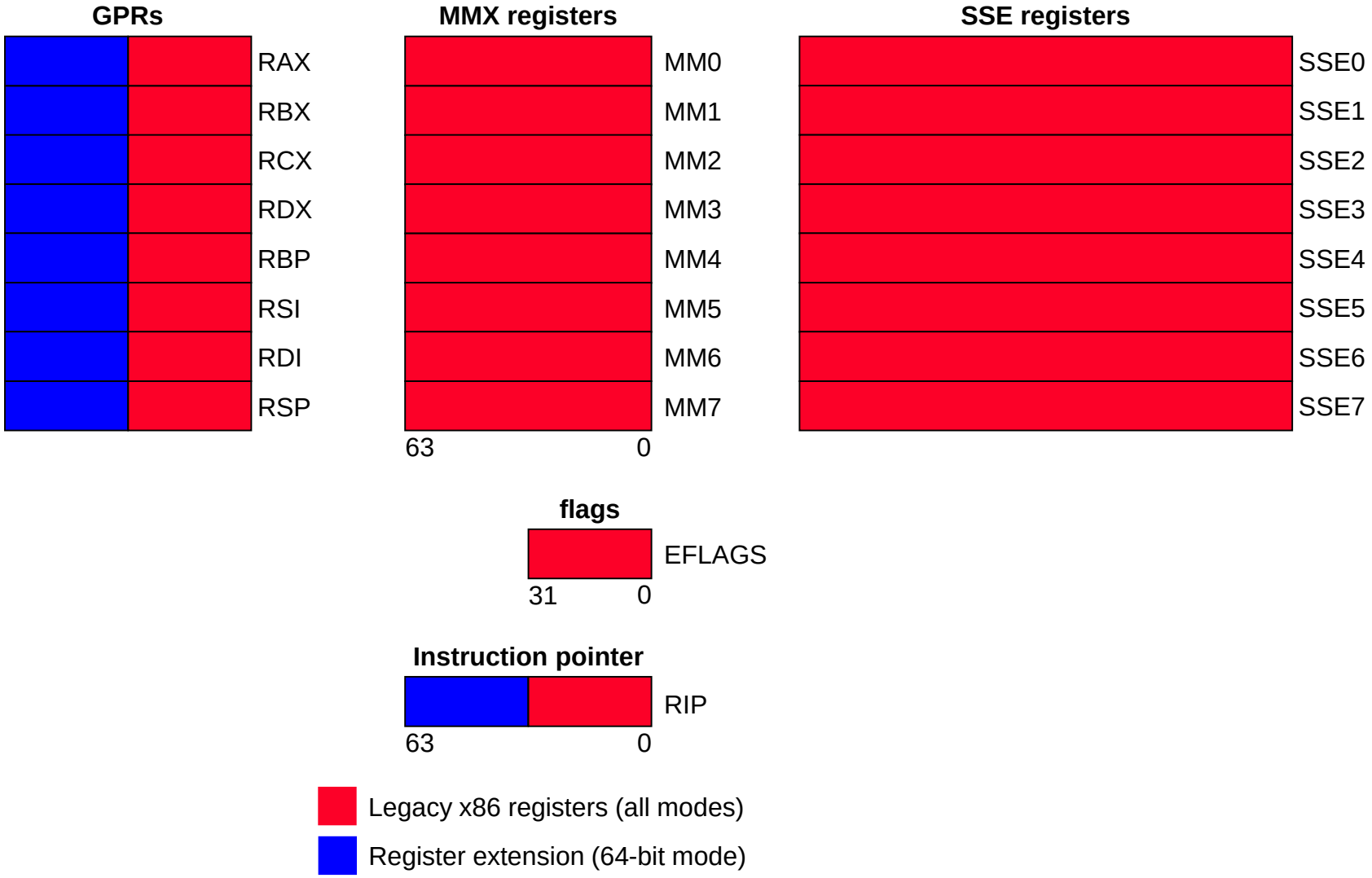


Extensão: x86-32 → x86-64



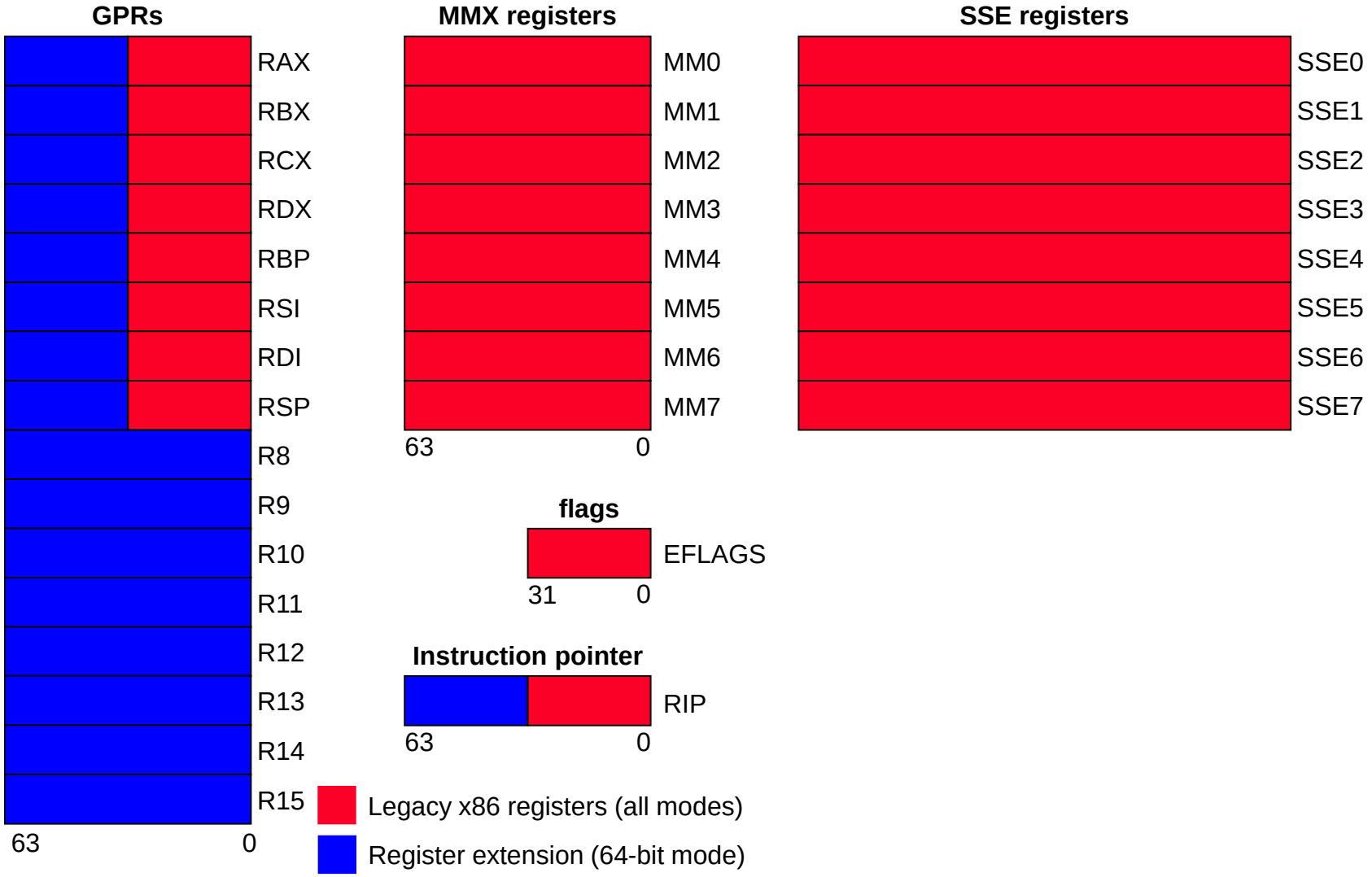
Fonte: AMD x86-64 White Paper

Extensão: x86-32 → x86-64



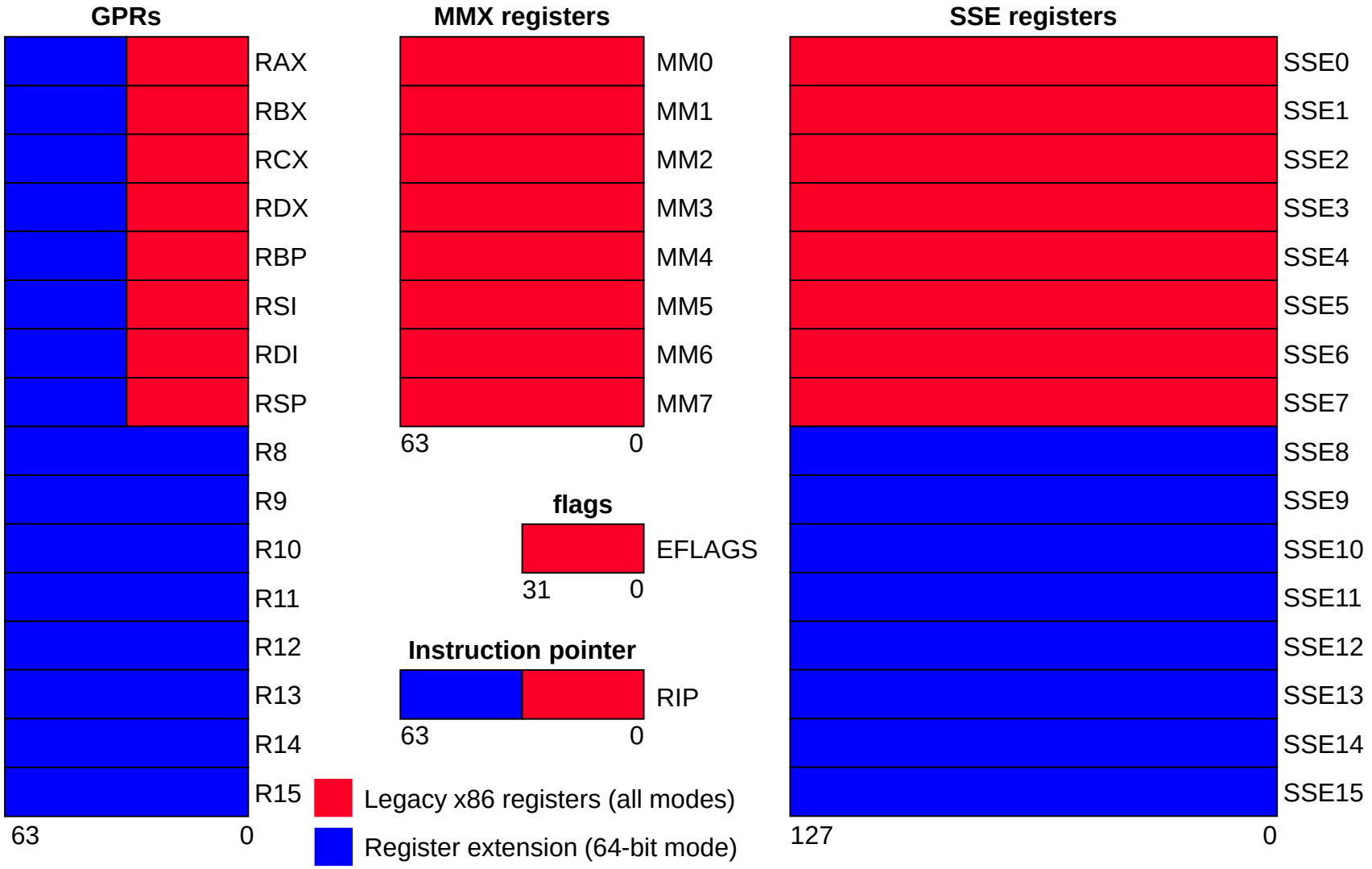
Fonte: AMD x86-64 White Paper

Extensão: x86-32 → x86-64



Fonte: AMD x86-64 White Paper

Extensão: x86-32 → x86-64



Fonte: AMD x86-64 White Paper

Evolução: x86-32 → x86-64

- 2004: EM64T (Intel 64)
  - Extended Memory 64 Technology
    - » Adaptação da Intel para o AMD 64 (1 instrução a mais)
  - Extensão SSE3: 13 novas instruções
    - » Aritmética complexa, operações gráficas, codificação de vídeo, conversão PF, etc.
  - Intel 64 ≅ x86-64 ≠ IA-64
- 2006: Intel anuncia extensão SSE4 (54 instruções)
- 2007: AMD anuncia extensão SSE5 (170 instruções)
  - 46 delas usam o formato de 3 operandos similar ao MIPS
- 2011: Intel Advanced Vector Extension (AVX)
  - Registradores SSE: 128 para 256 bits
  - 250 instruções redefinidas
  - 128 novas instruções

Evolução: x86-32 → x86-64

- **2004: EM64T (Intel 64)**
  - **Extended Memory 64 Technology**
    - » **Adaptação da Intel para o AMD 64 (1 instrução a mais)**
  - **Extensão SSE3: 13 novas instruções**
    - » **Aritmética complexa, operações gráficas, codificação de vídeo, conversão PF, etc.**
  - **Intel 64 ≅ x86-64 ≠ IA-64**
- **2006: Intel anuncia extensão SSE4 (54 instruções)**
- **2007: AMD anuncia extensão SSE5 (170 instruções)**
  - **46 delas usam o formato de 3 operandos similar ao MIPS**
- **2011: Intel Advanced Vector Extension (AVX)**
  - **Registradores SSE: 128 para 256 bits**
  - **250 instruções redefinidas**
  - **128 novas instruções**

(SSE2)

```
addsd %xmm1, %xmm2
addpd %xmm1, %xmm2
```

Evolução: x86-32 → x86-64

- **2004: EM64T (Intel 64)**
  - **Extended Memory 64 Technology**
    - » **Adaptação da Intel para o AMD 64 (1 instrução a mais)**
  - **Extensão SSE3: 13 novas instruções**
    - » **Aritmética complexa, operações gráficas, codificação de vídeo, conversão PF, etc.**
  - **Intel 64 ≅ x86-64 ≠ IA-64**
- **2006: Intel anuncia extensão SSE4 (54 instruções)**
- **2007: AMD anuncia extensão SSE5 (170 instruções)**
  - **46 delas usam o formato de 3 operandos similar ao MIPS**
- **2011: Intel Advanced Vector Extension (AVX)**
  - **Registradores SSE: 128 para 256 bits**
  - **250 instruções redefinidas**
  - **128 novas instruções**

(SSE2)
addsd %xmm1, **%xmm2**
addpd %xmm1, **%xmm2**

(AVX)
vaddsd %xmm1, %xmm2, %xmm3
vaddpd %ymm1, %ymm2, %ymm3

Evolução: x86-32 → x86-64

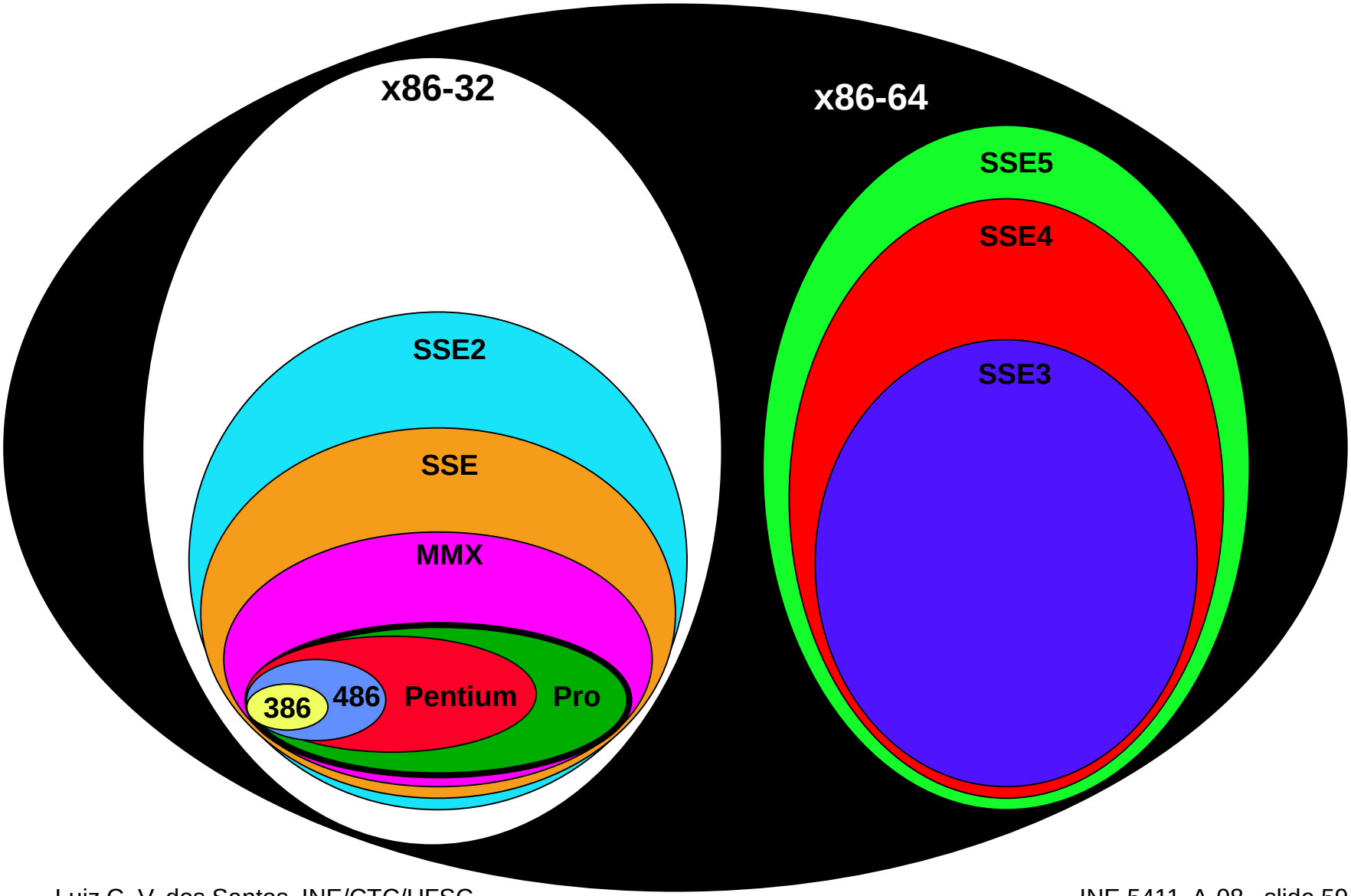
- **2004: EM64T (Intel 64)**
  - **Extended Memory 64 Technology**
    - » **Adaptação da Intel para o AMD 64 (1 instrução a mais)**
  - **Extensão SSE3: 13 novas instruções**
    - » **Aritmética complexa, operações gráficas, codificação de vídeo, conversão PF, etc.**
  - **Intel 64 ≅ x86-64 ≠ IA-64**
- **2006: Intel anuncia extensão SSE4 (54 instruções)**
- **2007: AMD anuncia extensão SSE5 (170 instruções)**
  - **46 delas usam o formato de 3 operandos similar ao MIPS**
- **2011: Intel Advanced Vector Extension (AVX)**
  - **Registradores SSE: 128 para 256 bits**
  - **250 instruções redefinidas**
  - **128 novas instruções**

(subword parallelism)

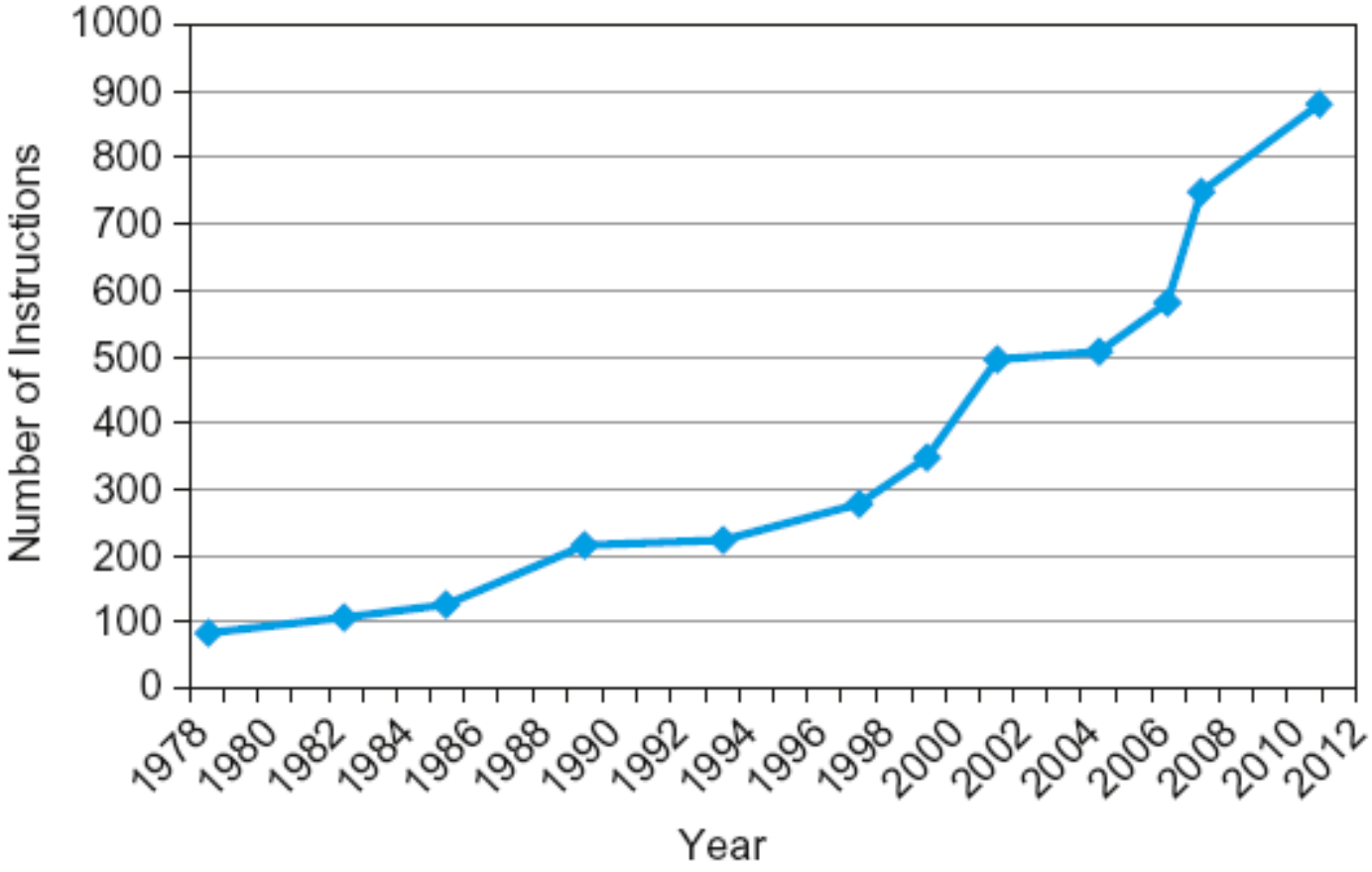
(SSE2)
addsd %xmm1, **%xmm2**
addpd %xmm1, **%xmm2**

(AVX)
vaddsd %xmm1, %xmm2, %xmm3
vaddpd %ymm1, %ymm2, %ymm3

x86-32 e x86-64



A evolução da ISA x86



x86: conclusão

- **Escolha do 8086 para o IBM PC = domínio da Intel**
- **ISA adaptada ao longo de 40 anos**
- **ISA mais popular para PCs**
  - **350 milhões por ano (contra 2 a 9 bilhões para o ARM)**
- **Na era Pós-PC:**
  - **x86-64 (Atom) não competitivo em *PMDs***
  - **Mas x86-64 (Sandy Bridge) predomina *na Cloud***
    - » **Embora, em 2014, AMD tenha lançado servidor com ARMv8**
- **Compiladores evitam instruções lentas da ISA**
  - **Evitando gerar instruções complexas inteiras**
  - **Usando SSE para ponto flutuante e multimídia**

ARM: conclusão

- **ISA mais popular**
  - **Computação Embarcada e Computação Móvel**
  - **2 a 9 bilhões por ano**
- **Na era Pós-PC:**
  - **ARMv7 (32 bits) predomina em *PMDs***
    - » **Exceto os construídos com Apple A7 (iphone 5S, iPad Air, etc.)**
  - **ARMv8 (64 bits) começa a competir com x86-64 na *Cloud***
    - » **AMD lançou servidor com ARMv8 (Opteron A1100)**

ARM: conclusão

- **Ao mudar para 64 bits, ARM não fez uma simples extensão, mas uma reforma profunda na ISA**
  - **Número de registradores expandido de 16 para 32**
  - **PC não mais endereçável como de propósito gerais**
  - **Registrador r0 agora contém constante zero**
  - **Incluída instrução de divisão inteira**
  - **Incluídos desvios similares a beq e bne**
  - **Imediato (12 bits) agora representa diretamente a constante**
  - **Modos de endereçamento funcionam para todos os tamanhos de dados**
  - **Descartadas instruções load/store de múltiplas palavras**
  - **Descartado campo de execução condicional (predicado)**
- **ARMv8 muito parecido com MIPS!**