Modos de Endereçamento

ISA: Modos de Endereçamento

- Os modos de endereçamento são um aspecto da Arquitetura do conjunto de instruções nos projetos das CPUs.

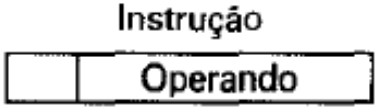
- Os vários modos de endereçamento definidos em uma ISA determinam como as instruções da linguagem de máquina identificam os operandos de cada instrução.

ISA: Modos de Endereçamento

- Modo como os bits do um campo de endereço são interpretados para encontrar o operando:
  - Imediato
  - Direto ou Absoluto
  - Registrador
  - Indireto de registrador
  - Indireto de memória
  - Deslocamento
  - Indexado

ISA: Modos de Endereçamento

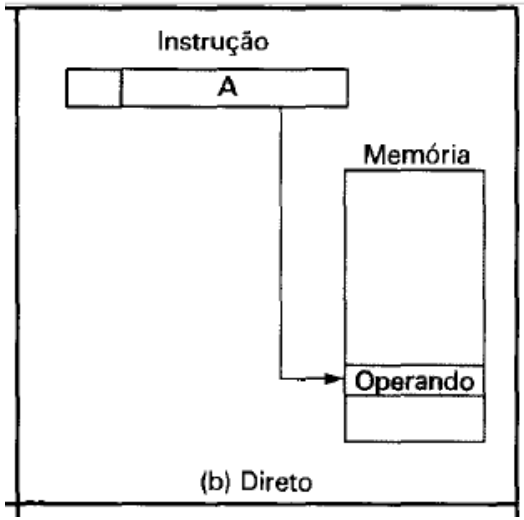
- Endereçamento imediato
  - Forma mais simples de endereçamento é o endereçamento imediato, no qual o valor do operando é especificado diretamente na instrução. Pode ser usado para definir e usar constantes ou para atribuir valores iniciais em variáveis.
  - Vantagem: não requerer qualquer acesso à memória para obter o operando além do acesso para obter a própria instrução, economizando, portanto, um ciclo de cache ou de memória no ciclo de instrução.
  - Desvantagem: tamanho do operando é limitado pelo tamanho do campo de endereço.



Load #3

ISA: Modos de Endereçamento

- Endereçamento direto ou absoluto
  - Forma muito simples de endereçamento, no qual o campo de endereço contém o endereço efetivo do operando
  - Vantagem: requer apenas um acesso à memória e nenhum cálculo especial.
  - Desvantagem: fornece um espaço de endereçamento limitado.

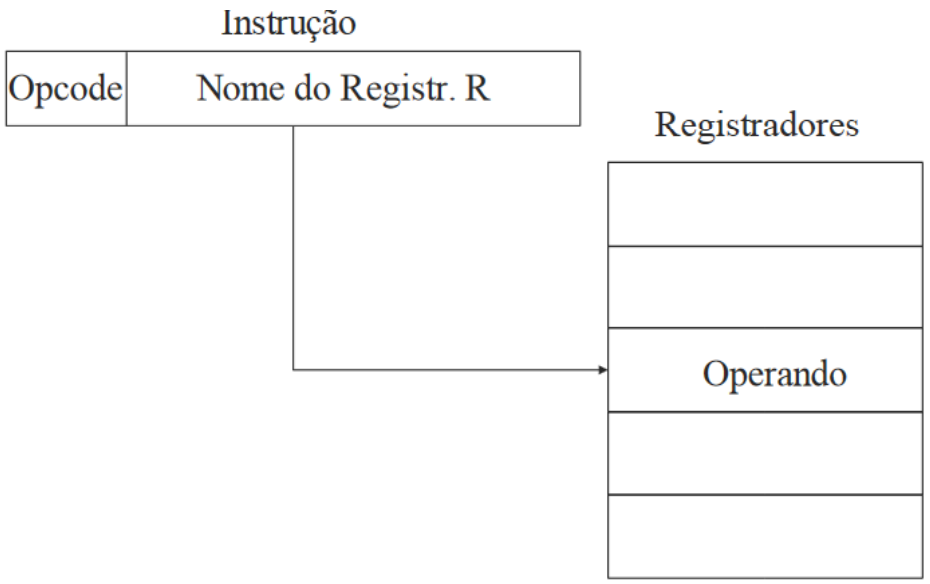


Load (1001)

ISA: Modos de Endereçamento

- Endereçamento por Registrador
  - Um ou mais operandos está em registradores (inclusive o operando de destino)

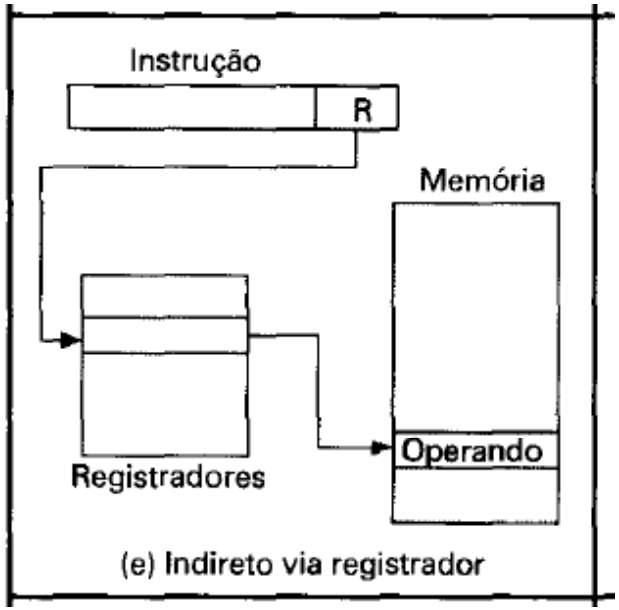
Load R3



ISA: Modos de Endereçamento

- Endereçamento indireto por Registrador
  - O campo de endereço contém o registrador que armazena a localização do operando na memória
  - Endereço intermediário: ponteiro

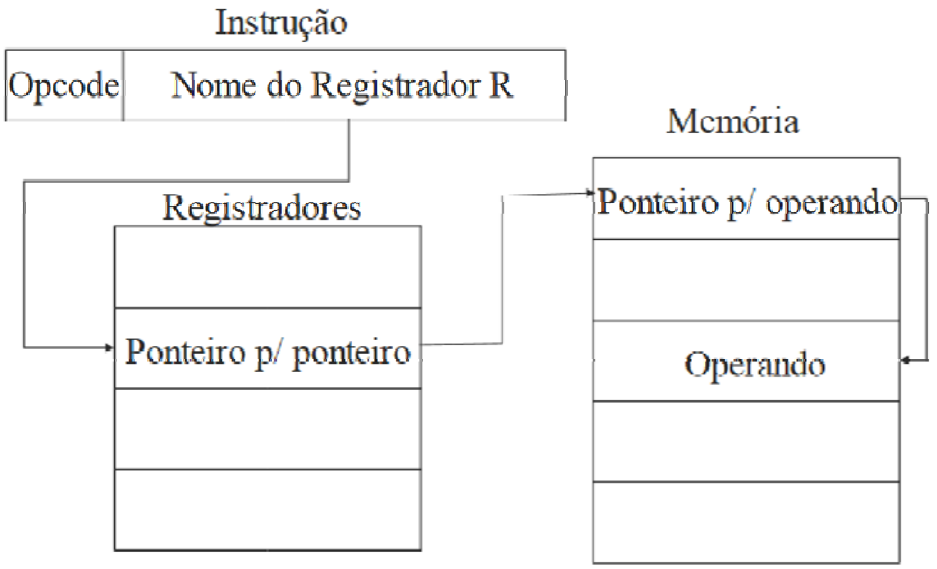
Load (R1)



ISA: Modos de Endereçamento

- Endereçamento indireto de Memória
  - O campo de endereço contém o registrador que armazena a localização do operando na memória
  - Endereços intermediário: ponteiro para ponteiro

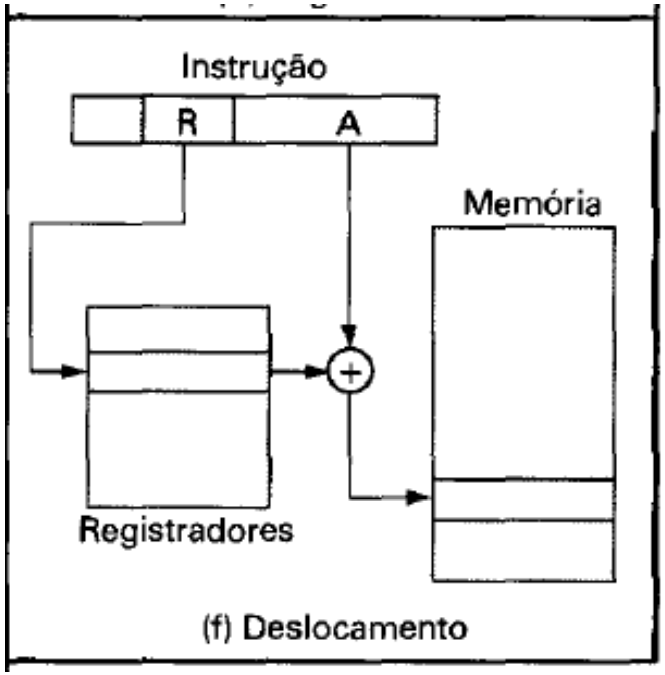
Load @(R3)



ISA: Modos de Endereçamento

- Deslocamento
  - Esta técnica combina as técnicas de endereçamento direto e do endereçamento indireto por registrador. Esta técnica obriga que a instrução tenha dois campos de endereço

Load 100(R1)



ISA: Modos de Endereçamento

- Indexado
  - O Campo do endereço referencia a memória principal e o registrador de referência contém um deslocamento positivo a partir dessa posição de memória
  - Utilizado no acesso aos elementos de um vetor (R1=base, R2=índice)

Load (R1+R2)

ISA: Modos de Endereçamento

Register *(Registrador)*	Add R4, R3	Regs[R4] ← Regs[R4] + Regs[R3]
Immediate *(Imediato/cte)*	Add R4, #3	Regs[R4] ← Regs[R4] + 3
Displacement *(Deslocamento)*	Add R4, 100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]
Register Indirect *(Indireto/Ponteiro)*	Add R4, (R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]
Indexed *(Indexado)*	Add R3, (R1+R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]
Direct *(Direto)*	Add R1, (1001)	Regs[R1] ← Regs[R1] + Mem[1001]
Memory Indirect *(Ponteiro p/ Ponteiro)*	Add R1, @(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]

ISA: Modos de Endereçamento

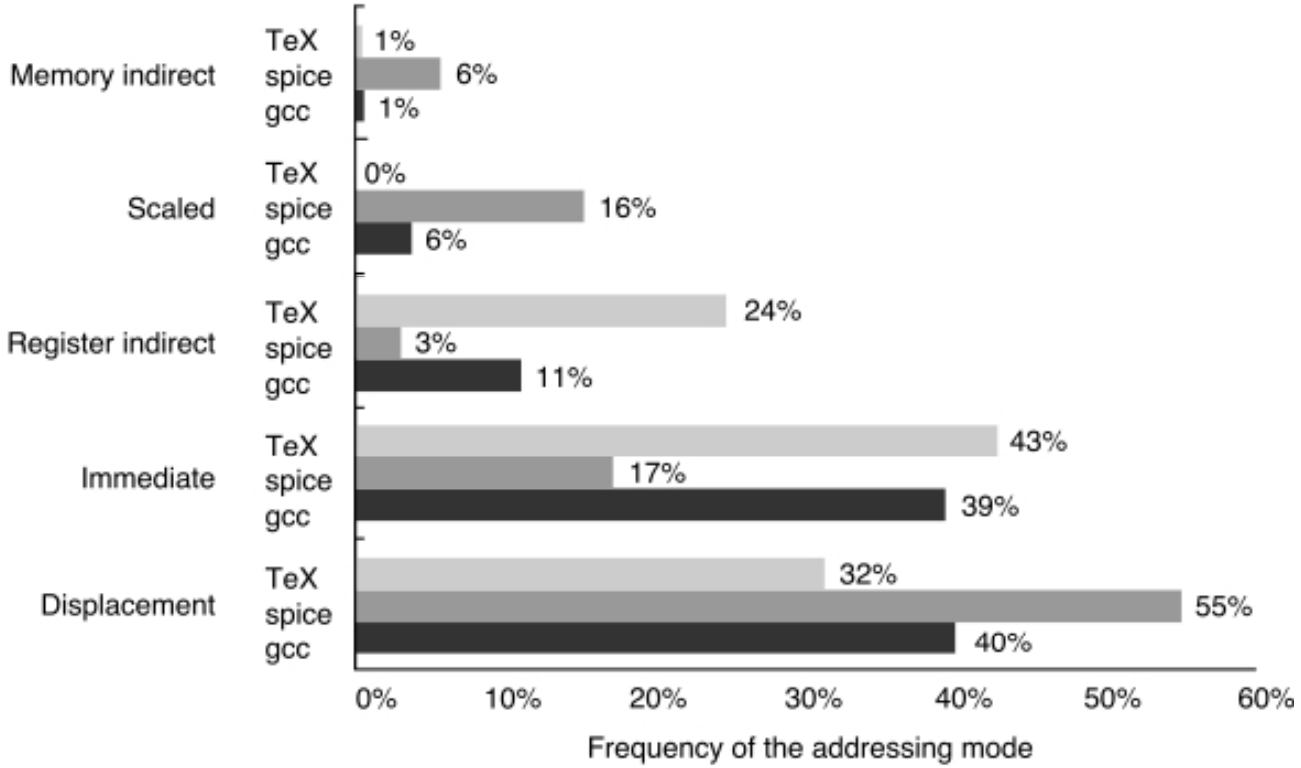
Autoincrement	Add R1, (R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + d (d: size of an array element)
Autodecrement	Add R1, -(R2)	Regs[R2] ← Regs[R2] - d Regs[R1] ← Regs[R1] + Mem[Regs[R2]]
Scaled	Add R1, 100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100+Regs[R2] + Regs[R3] * d]

ISA: Modos de Endereçamento

- Algumas ISAs incluem todos esses modos (ex: VAX), outras apenas alguns (ex: MIPS)
- Quais são as vantagens/desvantagens?
  - Quanto mais modos de endereçamento, menor o instruction count (IC)
  - Quanto mais modos de endereçamento, mais complexo é o hardware
  - Quanto mais modos de endereçamento, maior o número médio clock cycles necessários para executar uma instrução
- Objetivo: Encontrar um bom ponto de equilíbrio entre hardware mais simples e um número suficiente de modos de endereçamento, de modo que escrever código não seja tão difícil

ISA: Modos de Endereçamento

- Lição: Apenas alguns poucos modos são utilizados c/ frequência



Arquitetura de X-bits ?

- O que significa:
  - Uma arquitetura de 32-bits ?
  - Uma arquitetura de 64-bits ?
- Infelizmente não existe uma definição clara…
  - Pode ser usado p/ descrever que inteiros e endereços de memória são armazenados (codificados) em X bits
  - Pode ser usado p/ descrever uma CPU (ou ALU) que utiliza registradores de X bits
  - Pode ser usado p/ descrever um computador possuindo um barramentos de dados de X bits
- Em geral, quando se diz "uma arquitetura de X-bits" refere-se a um sistema que manipula grupos de X-bits representando valores inteiros (dados e endereços), registradores e barramentos de dados. Porém, existem diversas variações…

Arquitetura de X-bits ?

Processadores Intel e compatíveis

Processador	Barramento CPU-Memória	Registradores
8088	8-bit	16-bit
8086	16-bit	16-bit
286	16-bit	16-bit
386SX	16-bit	32-bit
386DX	32-bit	32-bit
486/AMD-5x86	32-bit	32-bit
Pentium/AMD-K6	64-bit	32-bit
PentiumPro/Celeron	64-bit	32-bit
AMD Duron / Ahtlon/ Athlon XP	64-bit	32-bit
Pentium 4	64-bit	32-bit
IA64	64-bit	64-bit
AMD Athlon 64	64-bit	64-bit

ISA: Tipos de Instruções

ISA: Tipos de Instruções

universal	Aritmética e Lógica	Integer add, subtract, multiply, divide, and, or, ...
universal	Transferência de Dados	load and stores
universal	Control Flow	Branch, jump, procedure (function, method, etc.) call and return, traps
variável	Sistema	Chamadas de primitivas do S.O., Instruções para gerenciar a memória virtual, etc
opcional	Ponto Flutuante	add, multiply, divide, compare
opcional	String	string compare, string search
opcional	Graphics	pixel and vertex operations, compression/decompression operations

ISA: Tipos de Instruções

- Transferência de dados: tipo mais fundamental de instrução de máquina
  - Deve especificar os endereços dos operandos fonte e de destino da operação.
  - Cada endereço pode indicar uma posição de memória, Um registrador ou o topo da pilha.
  - Deve indicar o tamanho dos dados a serem transferidos.
  - Como em todas as instruções com operandos, deve especificar o modo de endereçamento de cada operando

ISA: Tipos de Instruções

- Instruções de sistema
  - São aquelas que apenas podem ser executadas quando o processador está no estado privilegiado ou está executando um programa carregado em uma área especial da memória, que é privilegiada
  - Tipicamente, elas são reservadas para uso pelo sistema operacional

ISA: Tipos de Instruções

- Instruções de controle de fluxo
  - Geralmente a próxima instrução a ser executada é aquela que segue imediatamente, na memória, a instrução corrente
  - No entanto, uma parte considerável das instruções de qualquer programa tem como função alterar a sequência de execução de instruções
  - Nessas instruções, a CPU atualiza o contador de programa com o endereço de alguma outra instrução armazenada na memória

ISA: Tipos de Instruções

- Instruções de controle de fluxo
  - Control Flow Instructions: permitem ao program counter (PC) saltar para uma instrução que não seja a próxima na sequencia (o default)
  - Necessárias por:
    - Executar várias vezes um mesmo conjunto de instruções
    - Executar uma determinada sequência de operações se uma determinada condição é satisfeita
    - Divisão de programas em partes

ISA: Tipos de Instruções

- Instruções de controle de fluxo
  - O PC contém o endereço da instrução sendo executada. Por default , PC =PC + 1 após a execução da instrução corrente. Terminologia:
    - Branch: quando for condicional
    - Jump: quando for incondicional
  - Existem 4 tipos de instruções de controle de fluxo:
    - Conditional branches (desvio condicional)
    - Jumps (desvio/salto)
    - Procedure calls (chamada de procedimento)
    - Procedure returns (retorno de procedimento)

ISA: Tipos de Instruções

- Instruções de controle de fluxo: Desvio
  - Um de seus operandos é o endereço da próxima instrução a ser executada
  - Desvio condicional: o desvio será feito apenas se uma dada condição for satisfeita (o contador de programa é atualizado com o endereço especificado no operando). Caso contrário o PC é incrementado

ISA: Tipos de Instruções

- Instruções de controle de fluxo: Desvio
  - Desvio condicional:

```
            bre r1, r2, L1
            XXX
            yyy
L1:         ZZZ
```

Branch to L1 if contents of R1 = Contents of R2

ISA: Tipos de Instruções

- Instruções de controle de fluxo: Desvio
  - Desvio incondicional:

```
        jump L1
        XXX
        YYY
L1:     ZZZ
```

As instruções XXX e YYY não são executadas

ISA: Tipos de Instruções

- Instruções de controle de fluxo: Chamada de Procedimento
  - Procedimento: um subprograma autocontido, que é incorporado em um programa maior e pode ser invocado (chamado) de qualquer ponto
  - Instrui o processador a executar todo o procedimento e retornar ao ponto de chamada, o que envolve duas instruções básicas:
    - Chamada: desvia a execução da instrução corrente para o início do procedimento;
    - Retorno: provoca o retorno da execução do procedimento para o endereço em que ocorreu a chamada.

ISA: Tipos de Instruções

- Instruções de controle de fluxo: Chamada de Procedimento
  - Considerando que CALL X, em linguagem máquina, representa uma chamada ao procedimento de endereço X e o endereço de retorno é armazenado em registrador:

RN <- PC + DELTA

PC <- X

ISA: Instruções Executadas Frequentemente

- Fonte dos dados: **SPEC Benchmark on 80x86:**

Load	22%
Conditional branch (desvio)	20%
Compare	16%
Store	12%
Add	8%
And	6%
Sub	5%
Move register-register	4%
Call	1%
Return	1%
Other	4%

Observação: 10 instruções simples correspondem a 96% de todas as instruções

Lição 1: Projeto a CPU de modo que elas executem eficientemente ("make the common case fast")

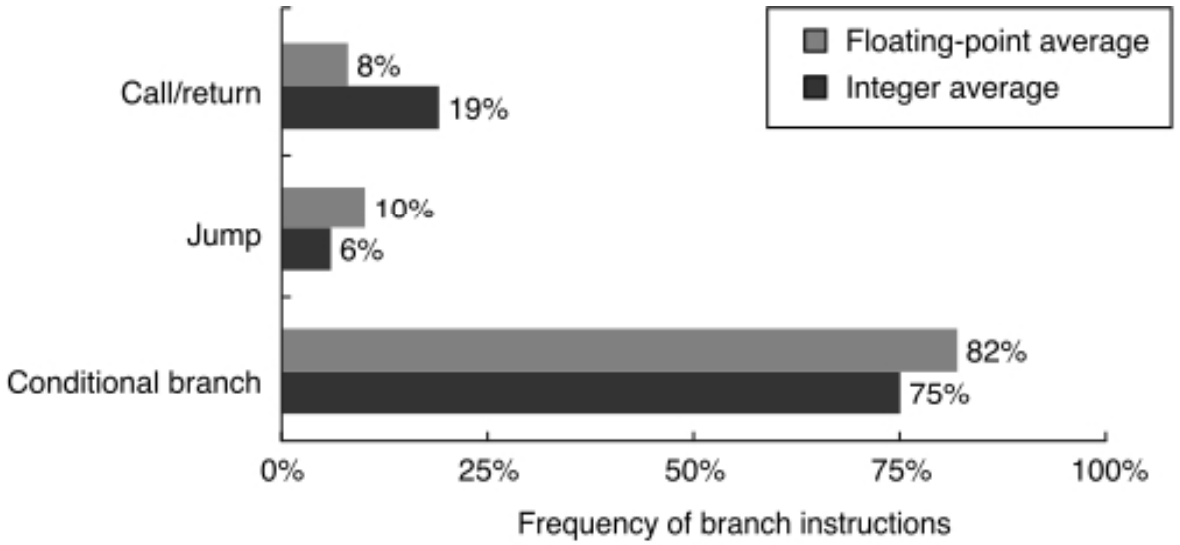
Lição 2: É duvidoso se vale a pena implementar muitas outras instruções sofisticadas.

Esta são as motivações da mudança de paradigma de CISC para RISC (ver slides mais adiante)

ISA: Instruções de Fluxo de Controle

- Conditional branches são muito mais frequentes que outras instruções de fluxo de controle

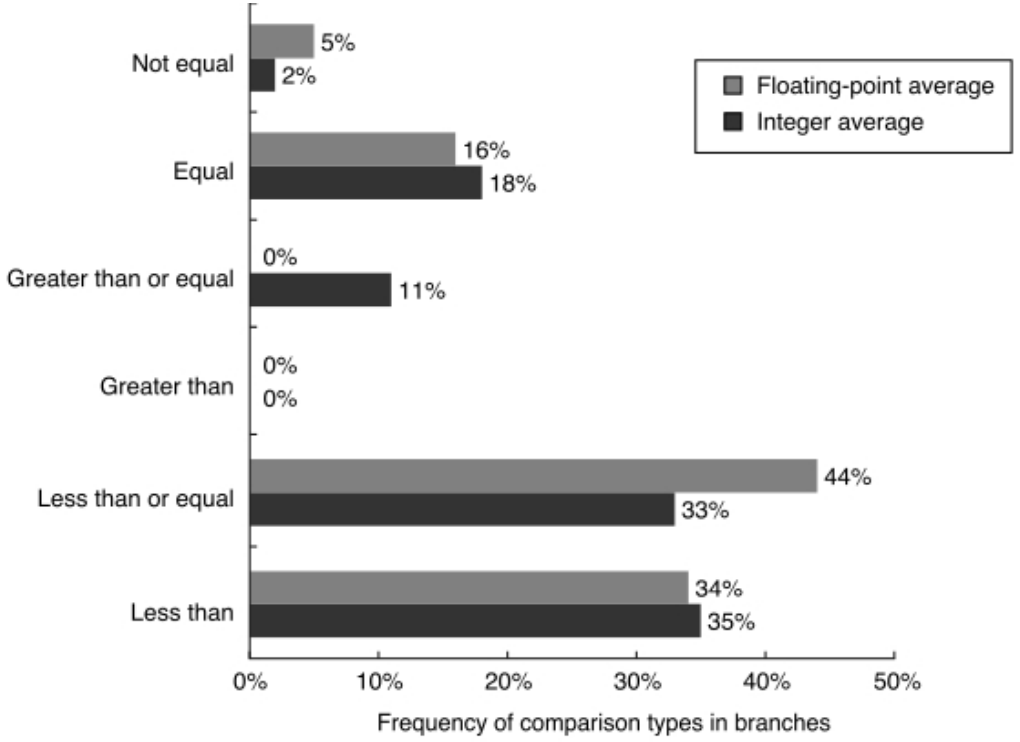
(SPEC benchmark)



**Útil p/ determinar quais instruções serão implementadas na arquitetura, ou de maneira mais eficiente.*

ISA: Conditional Branches

**Útil p/ determinar quais instruções serão implementadas na arquitetura, ou de maneira mais eficiente.*



Codificando a ISA

- O formato das instruções define a forma como os campos são distribuídos

- Para cada possível instrução da ISA:
  - Escolha uma sequencia única de bits para representá-la (chamado **opcode, ou código da operação**)
  - Represente os operandos também como sequencia de bits. Cada operando explícito tem de ser referenciado utilizando um dos métodos descritos anteriormente (endereçamento)
  - Construa blocos de hardware para decodificar e executar cadas uma das instruções (ou seja, a CPU).

Codificando a ISA

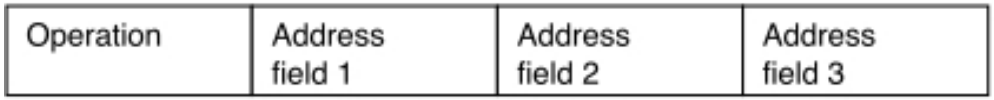
- Formato (codificação) de instruções fixo ou variável ?

- Três forças conflitantes:
  - #1: Desejo de se ter o maior número possível de registradores e modos de endereçamento → codificação longa
  - #2:Desejo de reduzir o tamanho do programa executável → codificação curta
  - #3: Desejo de se ter instruções que são fáceis de decodificar, com tamanho sendo múltiplo de bytes, e não arbitrário.

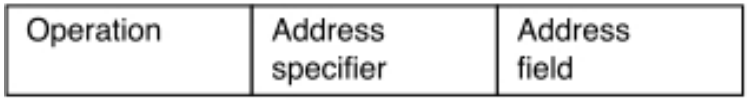
  - Na prática, as soluções mais populare são:
  - Formato variável (forças #1 and #2)
  - Formato fixo (força #3)
  - Formato híbrido (tentativa de encontrar um bom balanceamento)

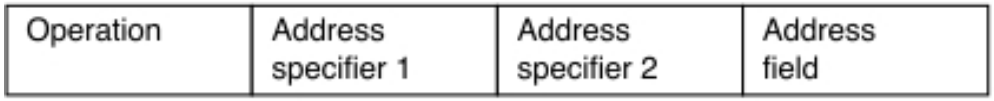
Codificando a ISA – Tamanho das Instruções

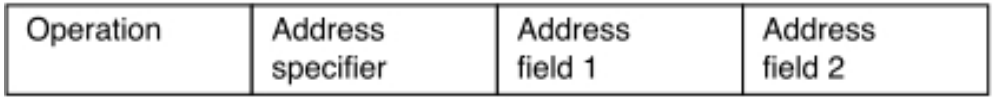
(a) Variable (e.g., VAX, Intel 80x86)



(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)







(c) Hybrid (e.g., IBM 360/70, MIPS16, Thumb, TI TMS320C54x)

Codificação de Instruções MIPS

MIPS:

- Tamanho de Instrução:
  - Fixo – 32 bits
- Codificação:
  - Variável – 3 tipos
- Detalhes: Próxima Aula →

CISC x RISC

CISC vs. RISC

- A redução do custo do hardware e a crescente utilização dos sistemas computacionais tornaram o custo do software maior no ciclo de vida de de um sistema do que o do hardware.

- Solução: desenvolvimento de linguagens de programação de alto nível cada vez mais poderosas e complexas.
  - Permitem expressar algoritmos de maneira mais concisa, abstraindo detalhes de máquina, e oferecem suporte à programação estruturada e ao projeto orientado a objetos.

CISC vs. RISC

- Problema: a grande a distância semântica entre as operações disponíveis em linguagens de alto nível e as operações disponibilizadas pelo hardware
  - Ineficiência na execução de programas, tamanho excessivo dos programas e grande complexidade dos compiladores.
- Solução: desenvolvimento de arquiteturas que diminuíssem a distância entre instruções de linguagens de alto nível e instruções de máquina
  - Grandes conjuntos de instruções, dúzias de modos de endereçamento e implementação de diversos comandos de linguagens de alto nível no hardware da máquina.

CISC vs. RISC

- Questionamento: muitas das instruções complexas são de fato necessárias nos programas “típicos”?

- Estudos realizados mostram os comandos mais utilizados pelos programas. Eles se concentram em alguns poucos comandos, em geral bastante simples

Carga	Huck, 1983 – Pascal Científico	Knuth, 1971 FORTRAN	Patterson e Sequin, 1982 - Pascal	Patterson e Sequin, 1982 - C	Tanenbaum, 1978 – Sistema SAL
Assignment	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	-	3	-
Outras	-	7	6	1	6

CISC vs. RISC

- Esses estudos sobre o comportamento da execução de programas em linguagens de alto nível orientaram o projeto de um novo tipo de arquitetura para processadores: o computador com um conjunto reduzido de instruções (Reduced Instruction set computer - RISC)

CISC vs. RISC

- **CISC: Complex Instruction Set Computer**
  - Cada instrução é “complexa” → faz várias operações.
    - Ex: lê um dado da memória, faz uma operação aritmética, armazena o resultado na memória, tudo isso c/ uma única instrução.
  - A ideia é aproximar o hardware das instruções de linguagens de alto nível.
  - Estratégia interessante na era “pré-compiladores”.

CISC vs. RISC

- **RISC: Reduced Instruction Set Computer**
  - Surgiu após os sistemas CISC, motivado por vários fatos:
    - A maioria dos programas usa apenas uma pequena fração das instruções disponíveis na ISA;
    - A construção de hardware para executar instruções complexas (e pouco utilizadas) resultava em hardware lento também para executar as instruções simples (e frequentes).
    - Instrições complexas são mais dificeis de decodifcar
    - Instrições complexas necessitam de vários ciclos de clock para executar
    - Com os avanços em VLSI, implementar registradores se tornou mais barato, viabilizando o projeto mais eficientes.
    - Compiladores funcionam melhor quando as instruções de máquinas são mais simples.
    - Popularização no uso da técnica de pipelining para a construção de CPUs.

CISC vs. RISC

- **Filosofia chave diferenciando RISC de CISC:**
  - Não há microprograma para interpretar as instruções. Existe um conjunto reduzido de instruções simples RISC parecidas com as microinstruções da arquitetura CISC.
  - Em RISC os operandos são registradores idênticos (em número razoável)
  - Use operações load e store para fazer a comunicação memória <-> registradores, usando modos de endereçamento simples.
  - O código resultante é uma série mais longa de instruções simples.
  - Obs: Muitos preferem o termo "arquitetura load-store " para se referir a RISC

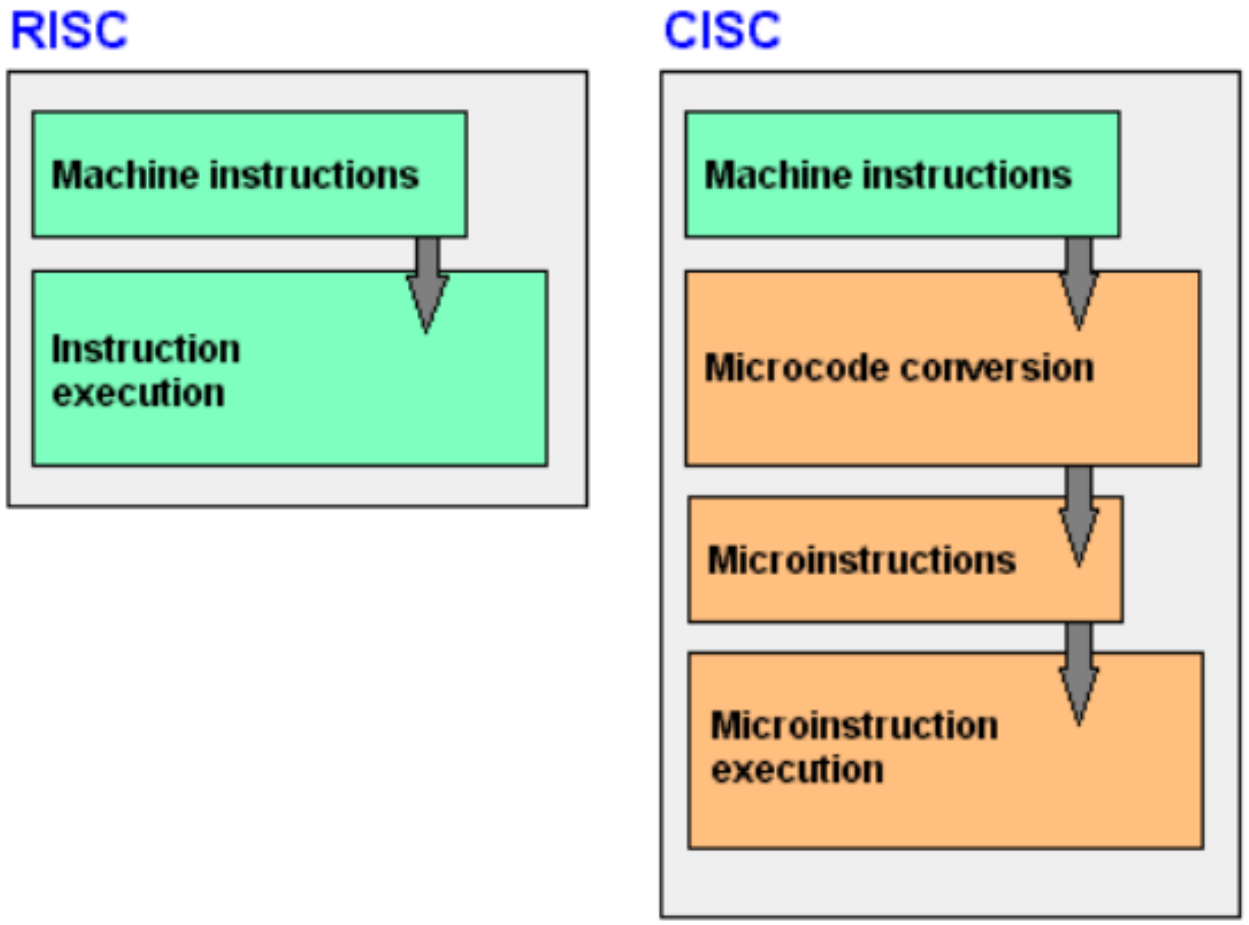
CISC vs. RISC

- O termo “CISC” surgiu após as arquiteturas RISC serem desenvolvidas, apenas p/ diferenciar de RISC

- Note que RISC não é uma boa opção quando o tamanho do código dos programas executáveis deve ser o menor possível
  - Não é problema para desktops e servidores
  - Problema sério p/ sistemas embarcados

CISC vs. RISC

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

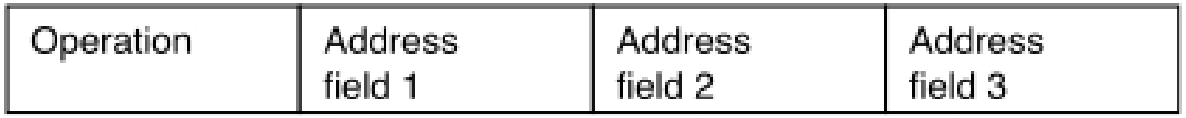


CISC vs. RISC

- Máquinas CISC usam codificação variável

(a) Variable (e.g., VAX, Intel 80x86)

- Máquinas RISC usam codificação fixa:



(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)

CISC vs. RISC

	CISC			RISC	
Modelo	IBM 370/168	VAX 11/780	Intel 80486	Sparc	MIPS R4000
Ano desenv.	1973	1978	1989	1987	1991
Num. instruções	208	203	235	69	94
Tam. instrução (bytes)	2 a 6	2 a 57	1 a 11	4	4
Modos de endereçam.	4	22	11	1	1
Num. de reg. prop. geral	16	16	8	40 a 520	32
Tam. mem. Controle (kbits)	420	480	246	-	-
Tam. cache (kbytes)	64	64	8	32	128

ISA x86

ISA x86

- **x86 história antiga:**
  - 1978: Intel cria o 8086 processor (16-bit)
    - accumulator machine
  - 1980: Intel 8087
    - Pretty much a stack architecture
  - 1982: 80286 (24-bit)
    - Novas instruções
    - backward compatible (in a special mode) with the 8086
  - 1985: 80386 (32-bit – IA-32)
    - Novos modos de endereçamento e instruções
    - backward compatible c/ 8086
- A preocupação c/ “backward compatibility”, devido a base de software existente limitou o avanço a passos incrementais, sem verdadeiras mudanças na arquitetura

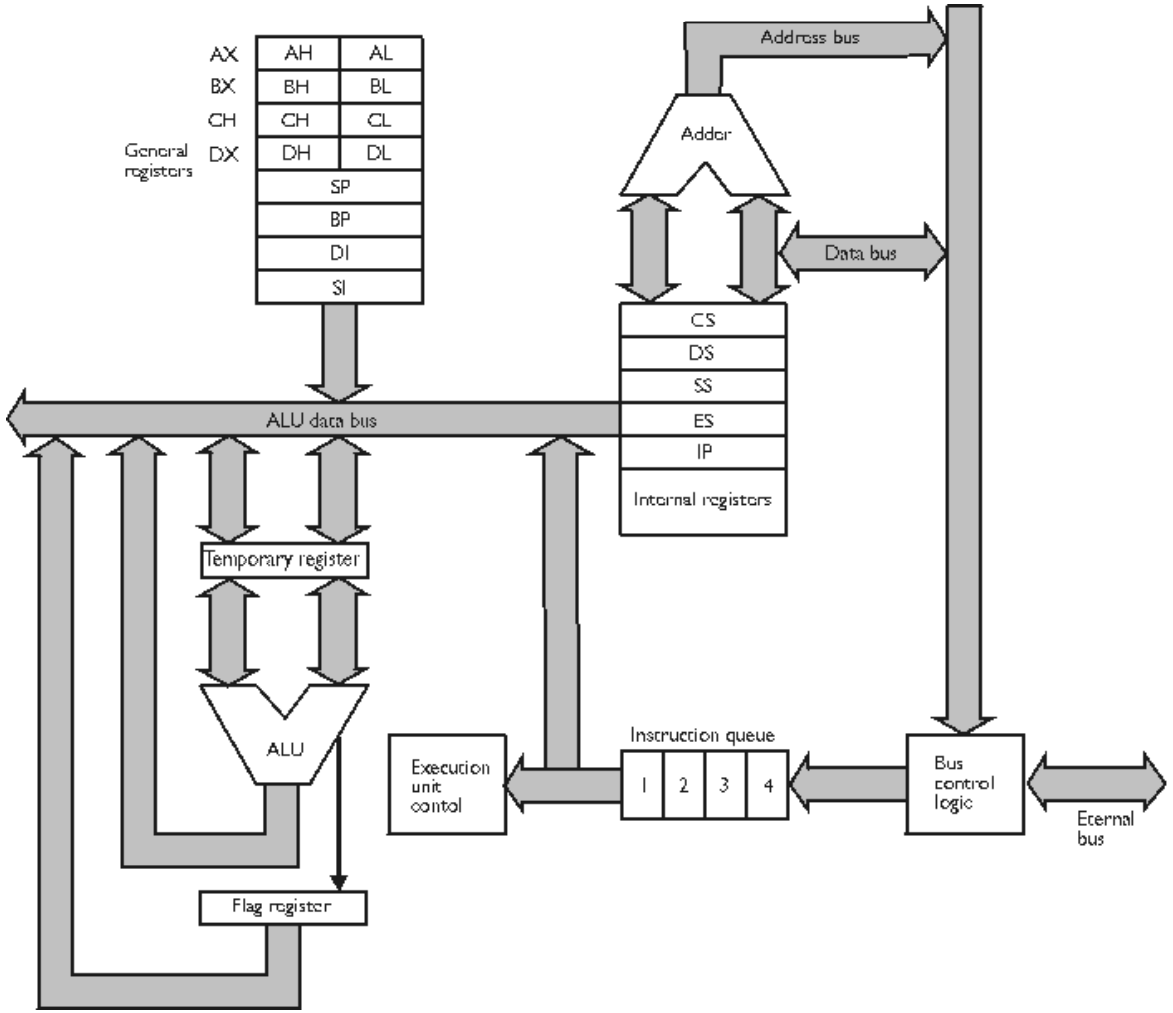
ISA x86

- **x86 história moderna:**
  - 1989: 80486, 1992: Pentium, 1995: P6
    - Buscou aumentar desempenho, poucas adições à ISA
  - 1997: MMX Extension
    - 57 novas instruções operando em paralelo em tipos de dados "estreitos" (8-bit, 16-bit, 32-bit)
      - Usadas em multi-media
  - 1999: SEE Extension (Pentium III)
    - 70 novas instruções
      - 4 32-bit floating-point operações em paralelo
      - Novas instruções p/ "cache prefetch"
  - 2001: SSE2 Extension
    - 144 novas instruções
    - Basicamente MMX e SSE, mas p/ números de 64-bits em ponto flutuante

ISA x86

- **Conclusões:**
  - A ISA x86 ISA não é “ortogonal”, ou seja, existem muitos casos especiais e exceções.
  - É difícil determinar o registrador e modo de endereçamento correto (ou mais eficiente) para uma dada tarefa.
  - Esses problemas tem origem na compatibilidade forçada com o processador
  - 8086, e sua “evolução” improvisada
- **Mas como x86 é tão bem sucedida ?**
  - Processadores Intel de 16-bits apareceram antes no mercado de PCs,
  - Os processadores x86 atuais decodificam instruções x86 em instruções menores (chamadas “micro-ops”), as quais são executadas por uma arquitetura RISC
  - Objetivo atingido: Bom desempenho devido a RISC, mas ainda expondo a ISA usual ao programador (assembly ou compilador)

ISA x86



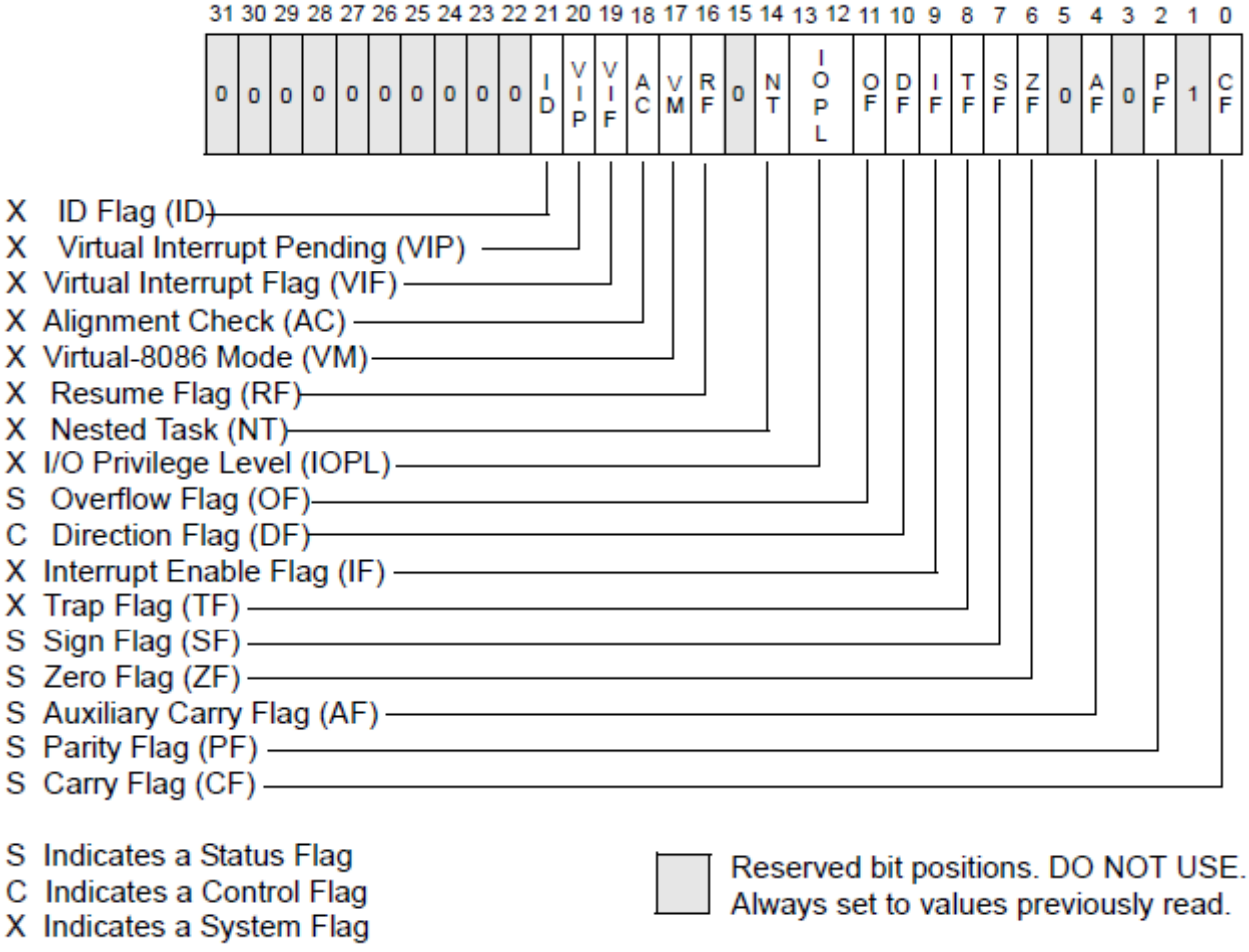
STALLINGS, W. Arquitetura e Organização de Computadores, 8ed. Pearson Education, 2009.

ISA x86

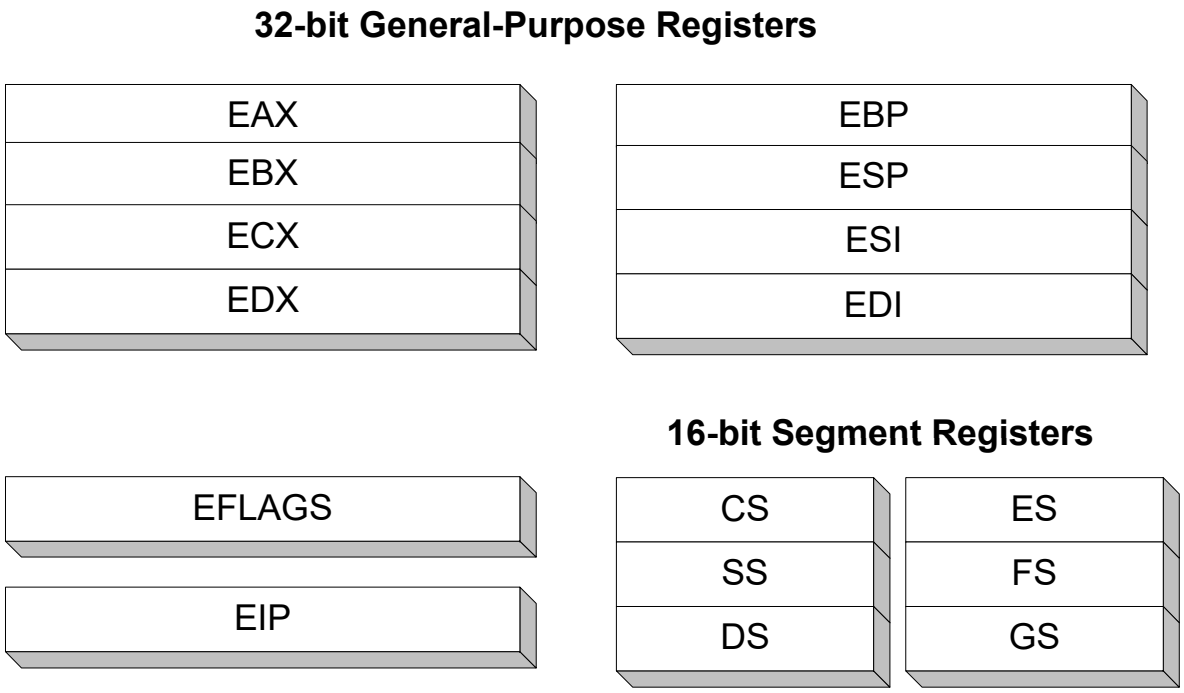
- Registradores do 8086
  - AX: acumulador para operandos e resultados
  - BX: ponteiro para dados no segmento DS
  - CX: contador para operações com string e loops
  - DX: ponteiro para E/S
  - SI: ponteiro para origem em operações com strings
  - DI: ponteiro para destino em operações com strings
  - SP: ponteiro de pilha, no segmento SS
  - BP: ponteiro base de dados na pilha, no segmento SS
  - IP: ponteiro de instrução no segmento CS
  - CS: segmento de código
  - DS: segmento de dados
  - ES: segmento extra
  - SS: segmento de pilha

ISA x86

- Registrador de flags

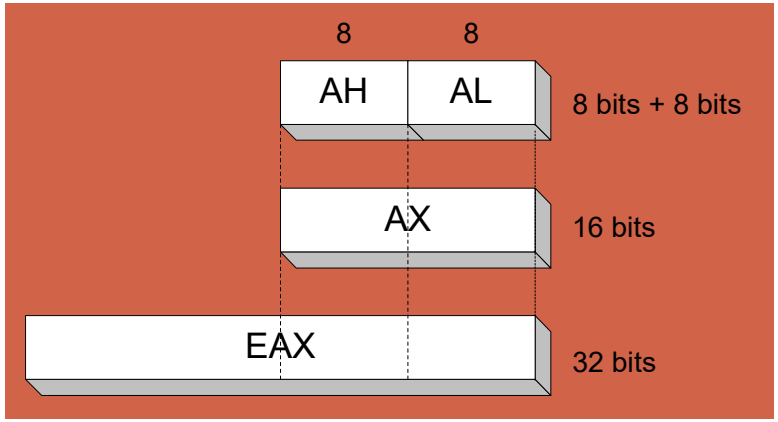


ISA x86



ISA x86

- Use 8-bit name, 16-bit name, or 32-bit name
- Applies to EAX, EBX, ECX, and EDX



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

ISA x86

- Modes of Operation
  - Protected mode
    - native mode (Windows, Linux)
  - Real-address mode
    - native MS-DOS
  - System management mode
    - power management, system security, diagnostics

- Virtual-8086 mode
  - hybrid of Protected
  - each program has its own 8086 computer

ISA x86

- Protected Mode
  - 4 GB addressable RAM
    - (00000000 to FFFFFFFFh)
  - Each program assigned a memory partition which is protected from other programs
  - Designed for multitasking
  - Supported by Linux & MS-Windows

ISA x86

- A tabela do endereço abaixo mostrando a evolução cronológica dos principais recursos tecnológicos acrescentados às várias famílias de processadores implementando a arquitetura x86:

- http://en.wikipedia.org/wiki/X86