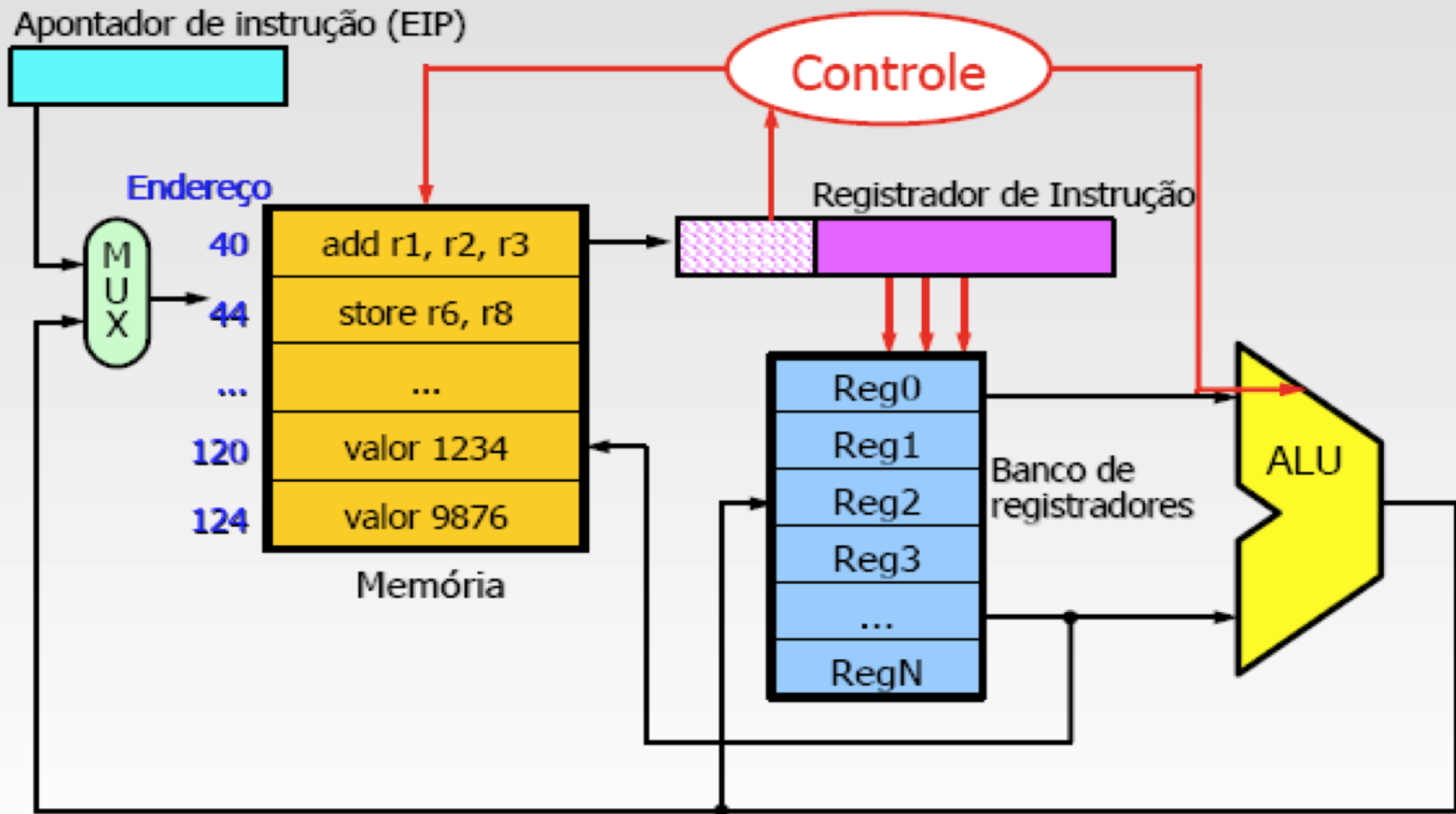


Formatos de Instruções e Modos de Endereçamento

Prof Gustavo Girão

Relembrando - Processador



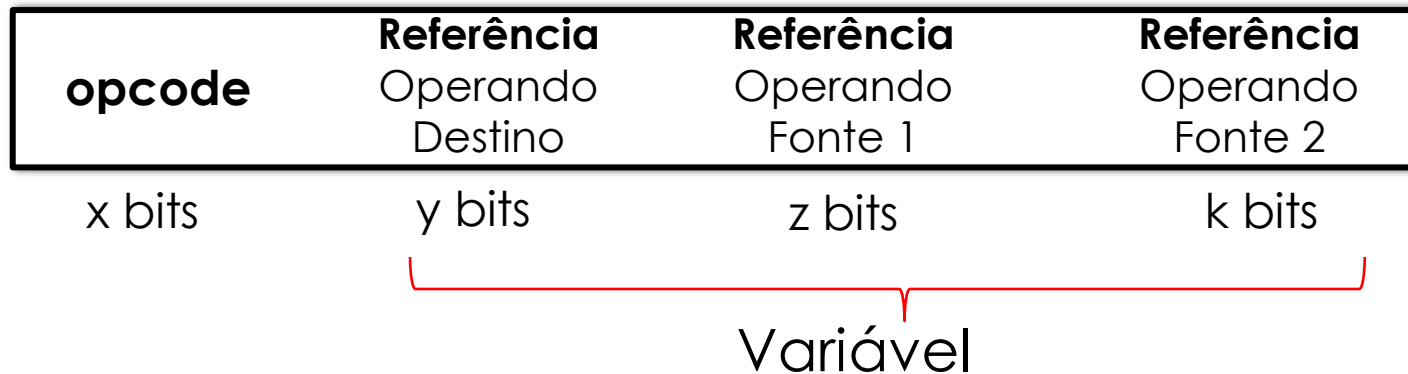
INSTRUÇÕES

Instruções

- Palavras da linguagem do computador
 - São as operações executadas pelo processador
 - Vocabulário está definido no conjunto de instruções
-
- add a, b, c # a = b + c
 - sub d, e, f # d = e - f
 - lw a, 32(b) # a = b(8) = b(32/4)
 # Carrega b(8) da memória e
 coloca no registrador a
-
- Como representar em baixo nível?

Representação

- Sequência de bits bem definida



- A **referência** consiste em um endereço de um dado na memória ou valor a ser utilizado no cálculo

Formato da Instrução

opcode	Referência Operando Destino	Referência Operando Fonte 1	Referência Operando Fonte 2
x bits	y bits	z bits	k bits

- Define os campos na instrução
 - Opcode – código da operação
 - Zero ou mais operandos (implícito ou explícito) - ENDEREÇAMENTO
- Conjunto de instruções contém vários formatos de instruções
- Cada operando explícito é referenciado usando um modo de endereçamento
- O formato deve indicar o modo de endereçamento para cada operando

O que considerar numa instrução?

- Tamanho da instrução
- Alocação de bits
- Instrução de tamanho variável

O que considerar numa instrução?

- Tamanho da instrução
 - Tamanho da memória
 - Organização da memória
 - Estrutura do barramento
 - Complexidade e velocidade do processador
- Relação flexibilidade x espaço
 - ✧ Mais opcodes = mais instruções
 - ✧ Mais endereçamento = maior intervalo de memória

O que considerar numa instrução?

- Alocação de bits
 - Divisão da quantidade de bits entre **opcodes** e **operandos**
 - Considerar
 - ✧ Número de modos de endereçamento
 - ✧ Número de operandos
 - ✧ Registrador x memória
 - ✧ Intervalo de endereços
 - ✧ etc

O que considerar numa instrução?

- Instruções de tamanho variável

- Vantagens

- ✧ Maior repertório de opcodes
 - ✧ Endereçamento flexível



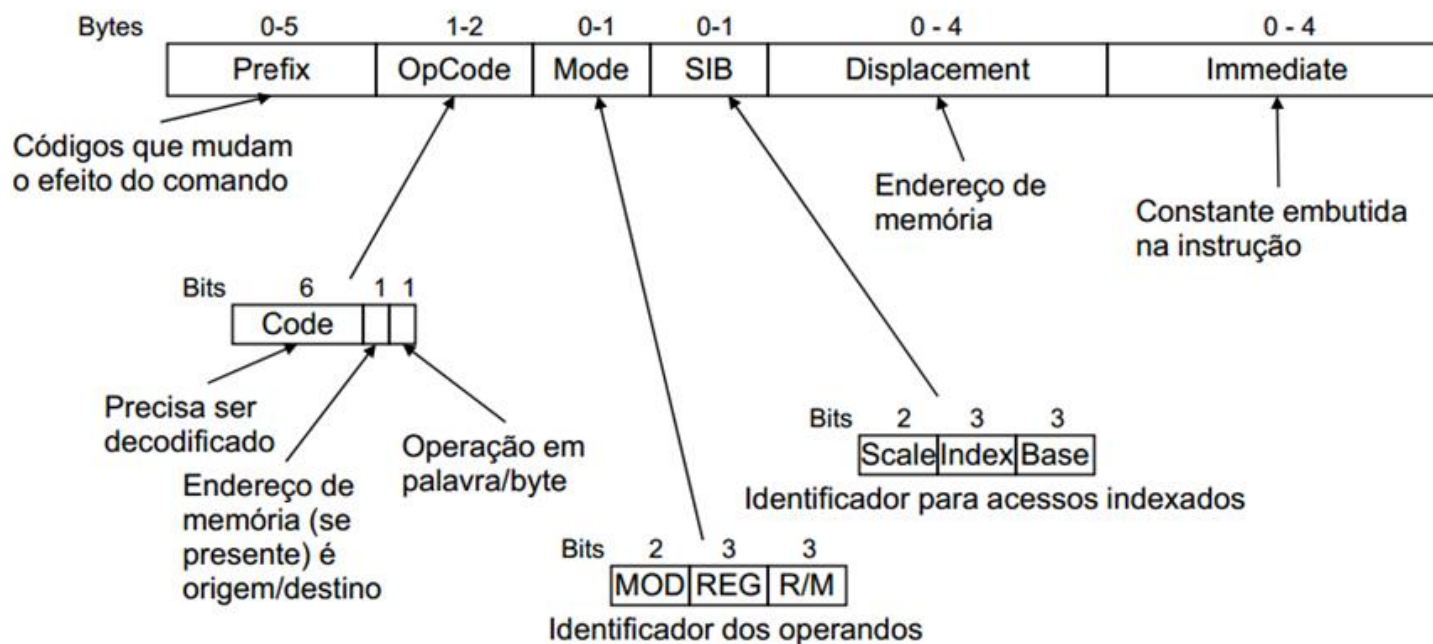
- Desvantagens

- ✧ Aumento da complexidade do processador



Formato de Instrução - Pentium 4

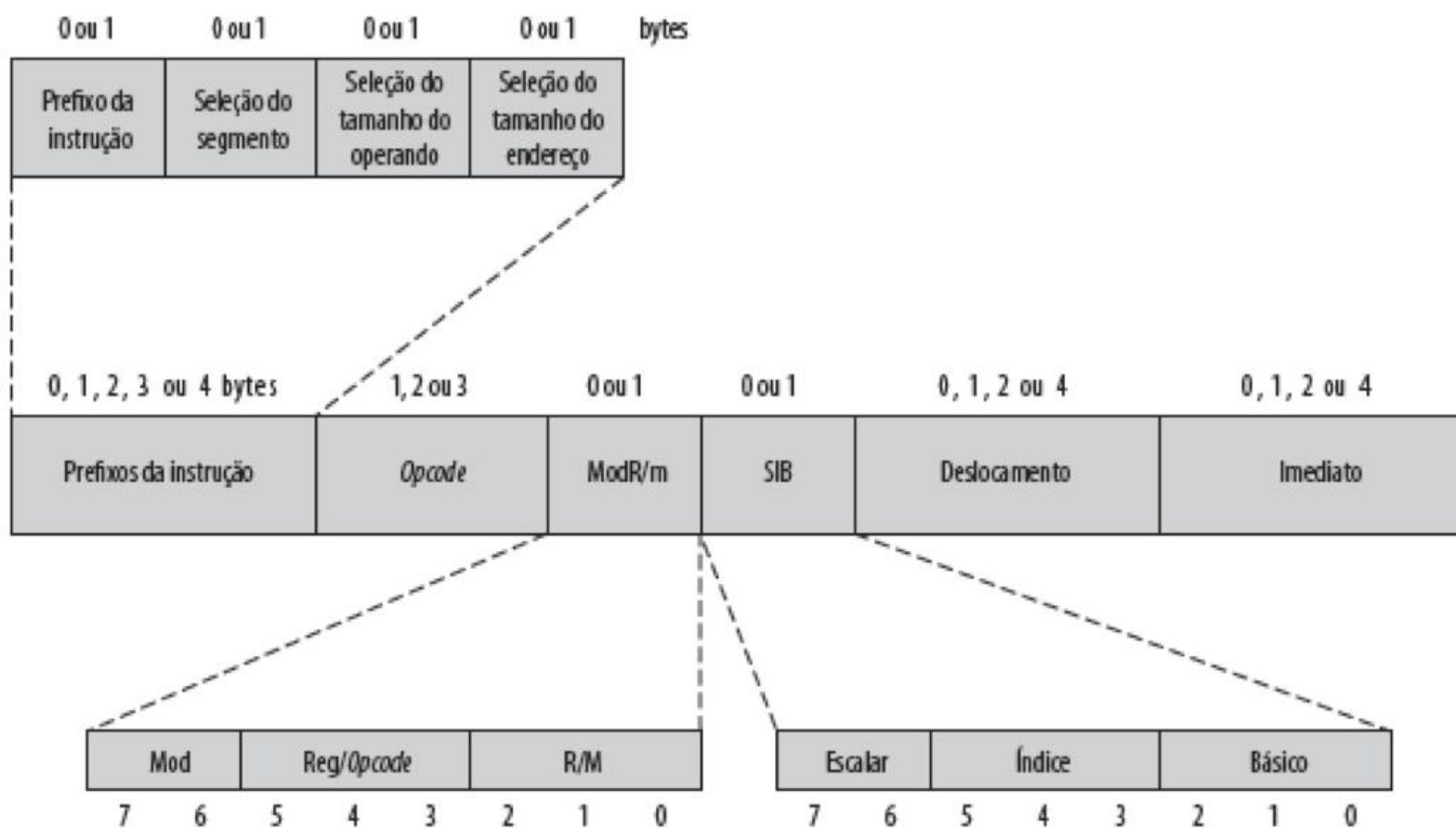
- Muito variável – difícil decodificação



Fonte: <http://homepages.dcc.ufmg.br/~jussara/swb/slides/cap5.pdf>

Formato de Instrução – x86

- Menos variação – decodificação menos complexa



Formato de Instrução - MIPS

- Tamanho fixo
 - OP = 6 bits (fixo): 64 possíveis instruções diferentes

Formato R (registrador)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Formato I (imediato)

op	rs	rt	imediato
6 bits	5 bits	5 bits	16 bits

Formato J (jump)

op	imediato
6 bits	26 bits

Fonte: Patterson & Hennessy, "Organização e Projeto de Computadores: A interface HARDWARE/SOFTWARE," 3a Edição

Modos de endereçamento

Modos de Endereçamento

- Direto
- Indireto
- Imediato
- Registrador
- Indireto de registrador
- Entre outros

Endereçamento Direto

- O campo de endereçamento contém o **endereço** efetivo do operando na MEMÓRIA

opcode	END_OP1 = 001	END_OP2 = 010	END_OP3 = 111
--------	----------------------	----------------------	----------------------

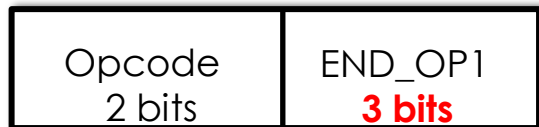
MEMÓRIA PRINCIPAL

END	VALOR
000	23456767
001	45
010	92
011	456789
100	TWEFGSDH
101	0
110	JHSDFSKD
111	12

Endereçamento Indireto

- Se o **tamanho do campo de endereço** for menor que o **tamanho do endereço de memória**?
- Exemplo:

Instrução



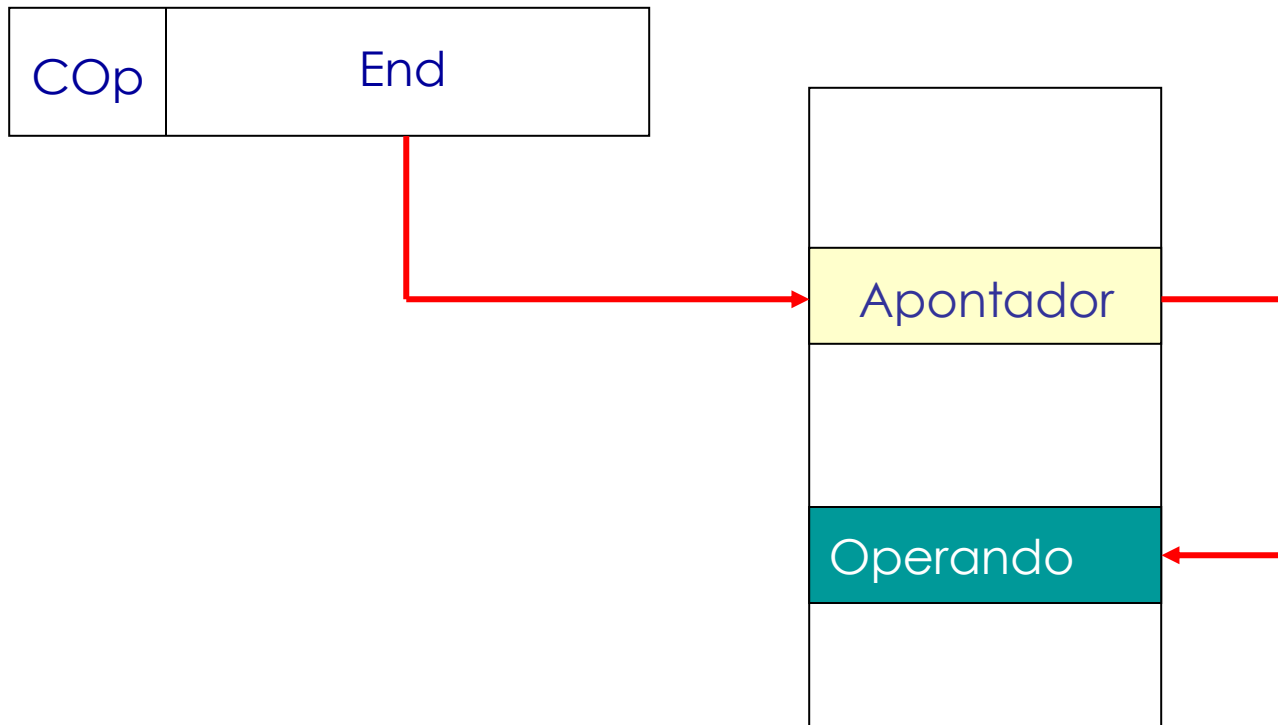
Memória



Como endereçar a posição de memória 1111 utilizando 3 bits?

Endereçamento Indireto

Instrução



Memória

Endereçamento Indireto

- O campo de endereçamento se refere a um endereço da memória

opcode	END_OP1 = 00001	END_OP2 = 00010	END_OP3 = 00111
--------	-----------------	-----------------	-----------------

MEMÓRIA PRINCIPAL

END	VALOR
00000	23456767
00001	<u>01001</u>
00010	<u>01010</u>
00011	456789
00100	TWEFGSDH
00101	0
00110	JHSDFSKD
00111	<u>01111</u>

MEMÓRIA PRINCIPAL (cont)

END	VALOR
01000	
<u>01001</u>	<u>30</u>
<u>01010</u>	<u>9</u>
01011	
01100	
01101	
01110	
<u>01111</u>	<u>0</u>

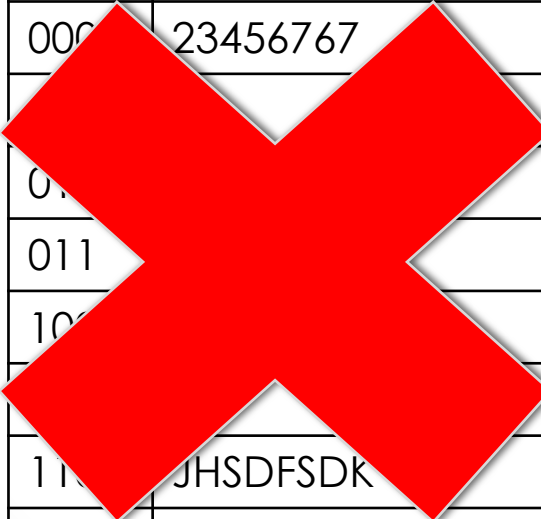
Endereçamento Imediato

- O campo de endereçamento contém o valor do operando

opcode	45	92	
--------	----	----	--

MEMÓRIA PRINCIPAL

END	VALOR
000	23456767
001	
010	
011	
100	
101	
110	JHSDFSKD
111	



Memória não é acessada

Endereçamento Direto por Registrador

- Semelhante ao Endereçamento direto, exceto que endereço se refere a um registrador

opcode	END_OP1 = 001	END_OP2 = 010	END_OP3 = 111
--------	---------------	---------------	---------------

BANCO DE REGISTRADORES

END	VALOR
000	23456767
001	45
010	92
011	456789
100	TWEFGSDH
101	0
110	JHSDFSKD
111	12


Quais as vantagens?

Endereçamento Direto por Registrador

- Semelhante ao Endereçamento direto, exceto que endereço se refere a um registrador

opcode	END_OP1 = 001	END_OP2 = 010	END_OP3 = 111
--------	---------------	---------------	---------------

BANCO DE REGISTRADORES



END	VALOR
000	23456767
001	45
010	92
011	456789
100	TWEFGSDH
101	0
110	JHSDFSKD
111	12

- Pequeno campo de endereço = + registradores
- Nenhuma referência de memória


Por que não usar somente registradores?

Endereçamento Direto por Registrador

- Semelhante ao Endereçamento direto, exceto que endereço se refere a um registrador

opcode	END_OP1 = 001	END_OP2 = 010	END_OP3 = 111
--------	---------------	---------------	---------------

BANCO DE REGISTRADORES



END	VALOR
000	23456767
001	45
010	92
011	456789
100	TWEFGSDH
101	0
110	JHSDFSKD
111	12

- Pequeno campo de endereço = + registradores
- Nenhuma referência de memória

Por que não usar somente registradores?

- Número limitado de registradores
- Custo de registrador é elevado

Endereçamento Indireto por Registrador

- Semelhante ao Endereçamento direto, exceto que o endereço se refere a um registrador

opcode	END_OP1 = 00001	END_OP2 = 00010	END_OP3 = 00111
--------	-----------------	-----------------	-----------------

BANCO DE REGISTRADORES

END	VALOR
00000	23456767
00001	01001
00010	01010
00011	456789
00100	TWEFGSDH
00101	0
00110	JHSDFSKD
00111	01111

BANCO DE REGISTRADORES (cont)

END	VALOR
01000	
01001	30
01010	9
01011	
01100	
01101	
01110	
01111	0

Tipos de processadores baseado
no conjunto de instruções

CISC X RISC

CISC x RISC

- Não são tecnologias: são estratégias de projeto de CPUs
- Evoluíram a partir de um conjunto de condições tecnológicas que existiram em um dado momento
- A comparação precisa levar em conta a situação da tecnologia no momento da criação das abordagens

CISC

Tecnologia nos anos 70-80

1) Memória Principal

- Magnética
 - ✧ Cara
 - ✧ Lenta
- RAM
 - ✧ Cara
- Armazenamento secundário
 - Muito lento
- Solução: código compacto para caber em memórias pequenas

Tecnologia nos anos 70-80

2) Compiladores

- Simples
- Compilação lenta
- Sem otimização
- Para otimização e rapidez: necessário programação assembly manual

Tecnologia nos anos 70-80

3) Hardware

- Baixa densidade de transistores
- Solução: distribuir as unidades funcionais CISC entre vários chips
 - ✧ Problema de comunicação

CISC – Complex Instruction Set Computer

- Início dos anos 70
 - Compiladores fracos e memória lenta e cara levou a uma crise no software

CISC – Complex Instruction Set Computer

- Início dos anos 70
 - Compiladores fracos e memória lenta e cara levou a uma crise no software
 - Com o hardware mais barato, a solução seria migrar a complexidade do software para o hardware

CISC – Complex Instruction Set Computer

- Início dos anos 70
 - Compiladores fracos e memória lenta e cara levou a uma crise no software
 - Com o hardware mais barato, a **solução seria migrar a complexidade do software para o hardware**

FILOSOFIA POR TRÁS DO CISC

CISC – Complex Instruction Set Computer

Se uma instrução complexa escrita numa linguagem de alto nível fosse traduzida em, exactamente, uma instrução assembly, então:

- Compiladores mais fáceis de escrever
 - ✧ Reduzir tempo e esforço para programadores
 - ✧ Reduzir custos
- Código mais compacto
 - ✧ Poupar memória
 - ✧ Reduzir custo global do hardware
- Detecção e correção de erros
 - ✧ Baixar custos de desenvolvimento de sw e manutenção

Como CISC melhora desempenho

- Transferir complexidade do sw para o hw
- Equação de desempenho de um processador:

$$\frac{\text{Tempo}}{\text{Programa}} = \left[\left(\frac{\text{Instruções}}{\text{Programa}} \right) \times \left(\frac{\text{Ciclos}}{\text{Instrução}} \right) \times \left(\frac{\text{Tempo}}{\text{Ciclo}} \right) \right]$$

Como CISC melhora desempenho

- Transferir complexidade do sw para o hw
- Equação de desempenho de um processador:

$$\boxed{\frac{\textit{Tempo}}{\textit{Programa}}} = \left[\left(\frac{\textit{Instruções}}{\textit{Programa}} \right) \times \left(\frac{\textit{Ciclos}}{\textit{Instrução}} \right) \times \left(\frac{\textit{Tempo}}{\textit{Ciclo}} \right) \right]$$

Como CISC melhora desempenho

- Transferir complexidade do sw para o hw
- Equação de desempenho de um processador:

$$\frac{\text{Tempo}}{\text{Programa}} = \left[\left(\frac{\text{Instruções}}{\text{Programa}} \right), \left(\frac{\text{Ciclos}}{\text{Instrução}} \right) \times \left(\frac{\text{Tempo}}{\text{Ciclo}} \right) \right]$$

CISC



Redução

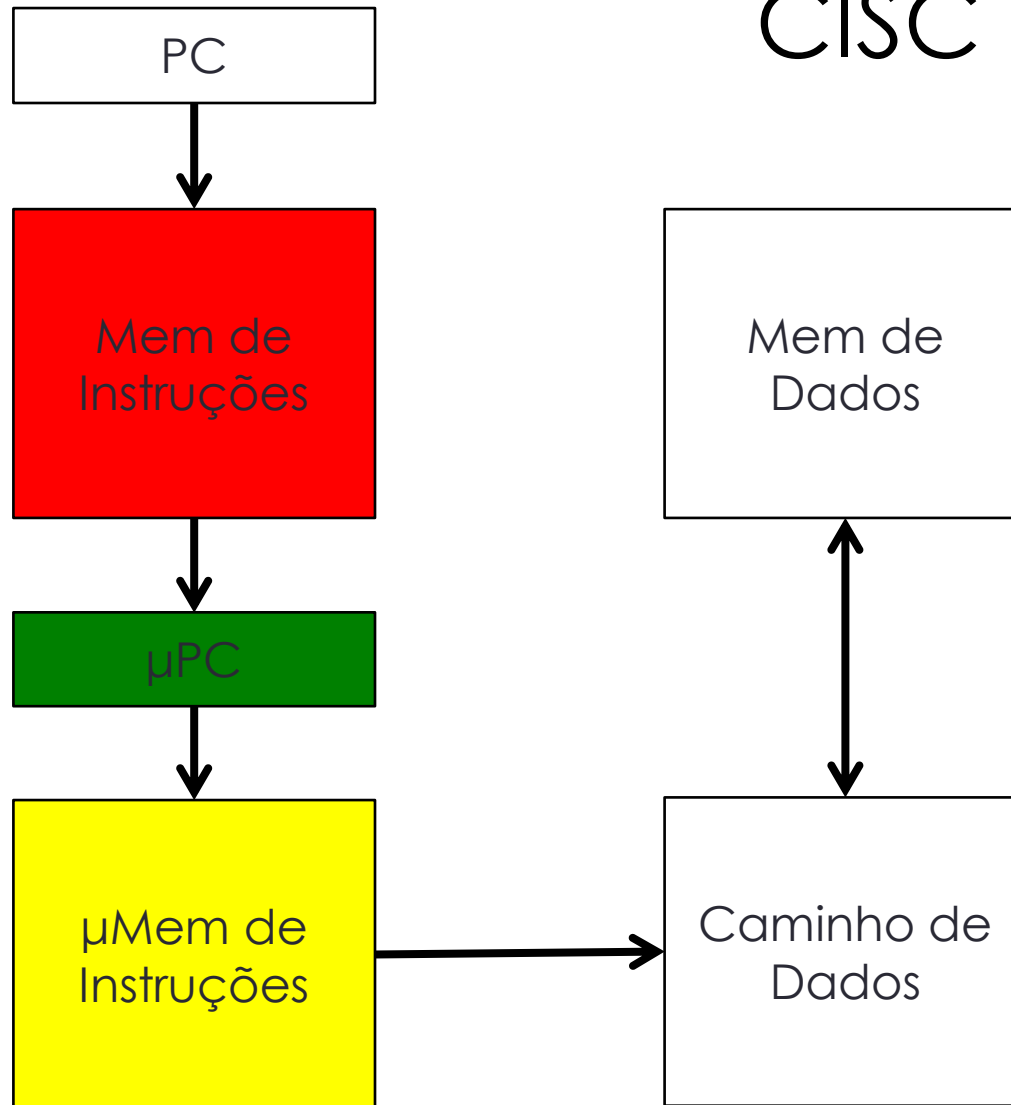
Como CISC melhora desempenho

- Reduzir o número de instruções que a máquina executa para completar uma determinada tarefa, permite reduzir o tempo que ela necessita para completar essa mesma tarefa, aumentando, assim, o seu desempenho.
- Reduzir o tamanho dos programas:
 - Menor quantidade de memória para armazenar o código
 - Diminuição do tempo de execução, pois havia menos linhas de código para executar.
- Exemplo: copiar string; conversão para BCD, etc

Como CISC Funciona

- **Microprogramação**
- “Quase como ter um mini-processador dentro do processador”
- A unidade de execução executa micro-instruções
 - Instrução normal carregada na memória
 - Decodificada
 - Entregue ao **processador de microcódigo**
 - Processador executa sub-rotina de microcódigo
 - Essa sub-rotina diz o que cada **unidade funcional** irá **executar**

CISC



Microprogramação

- Inicialmente o microcódigo era lento, porém armazenado na ROM
- A ROM era 10x mais rápida do que a memória magnética
- Com evolução, microcódigo mais rápido
 - Fazia cada vez mais sentido transferir funcionalidades do software para o hardware.
 - Os conjuntos de instruções cresceram rapidamente e o número médio de instruções por programa decresceu.

Problema da Microprogramação

- Demanda:
 - Microcódigos altamente otimizados e eficientes
 - Bastante compactos para manter os custos de memória
- Situação real:
 - Microprogramas grandes
 - Difícil de testar e detectar e corrigir erros.
- Resultado:
 - Microcódigo incluído nas máquinas que vinham para o mercado tinham erros e tinham que ser corrigido inúmeras vezes
- Foram estas dificuldades de implementação do microcódigo que levaram a que os investigadores **questionassem** se a implementação de **todas** estas instruções **complexas** e elaboradas em microcódigo seria, realmente, **o melhor caminho**

RISC

RISC – Reduced Instruction Set Computer

- Solução ideal: todo o hw em um único chip
- Problema: limite de área
- Solução proposta: deixar algumas funcionalidades de fora
 - Como: descobrir funcionalidades mais frequentemente utilizadas e otimizar essas funcionalidades
 - Resultado: fazer mais rápido as tarefas mais comuns

RISC – Reduced Instruction Set Computer

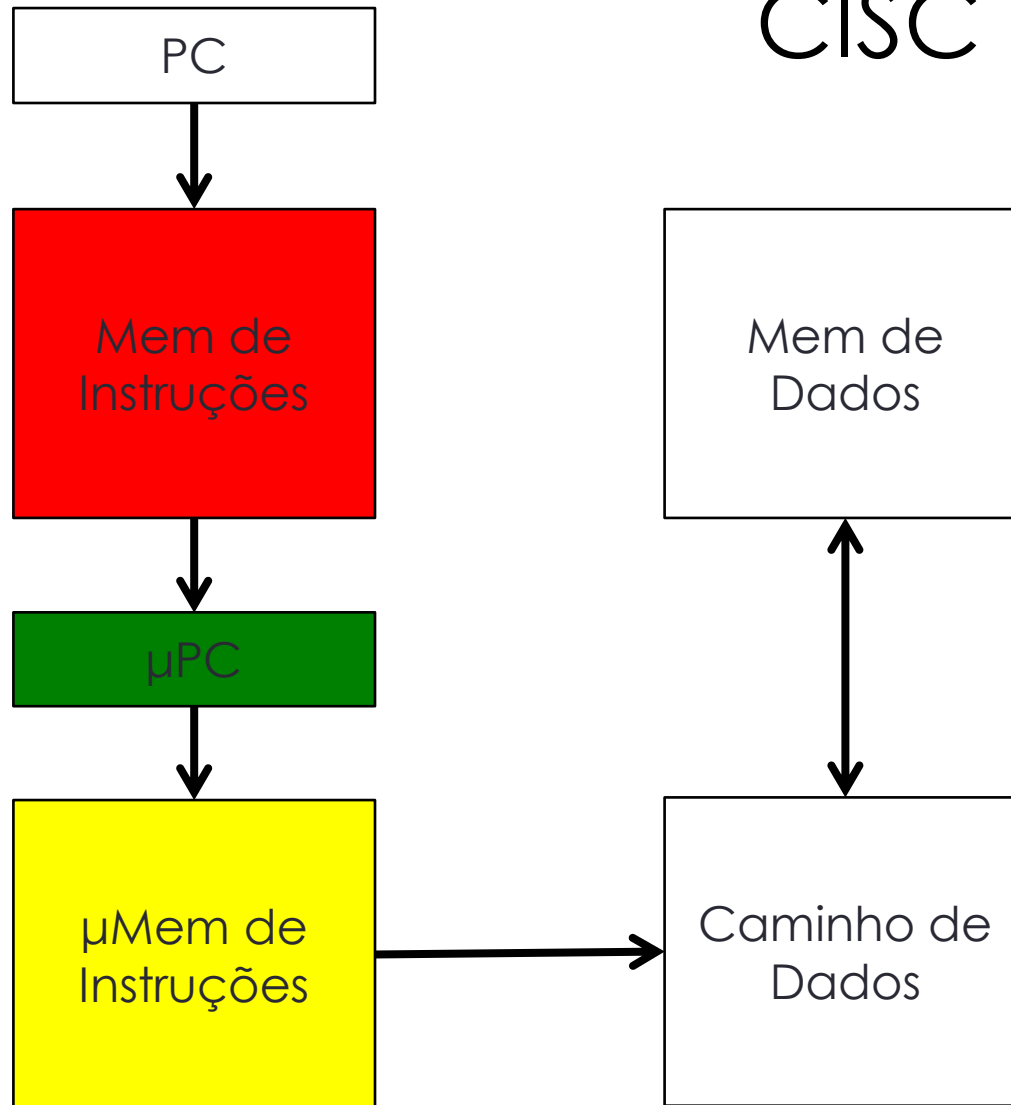
- Solução ideal: todo o hw em um único chip
- Problema: limite de área
- Solução proposta: deixar algumas funcionalidades de fora
 - Como: descobrir funcionalidades mais frequentemente utilizadas e otimizar essas funcionalidades
 - Resultado: fazer mais rápido as tarefas mais comuns

FILOSOFIA POR TRÁS DO RISC

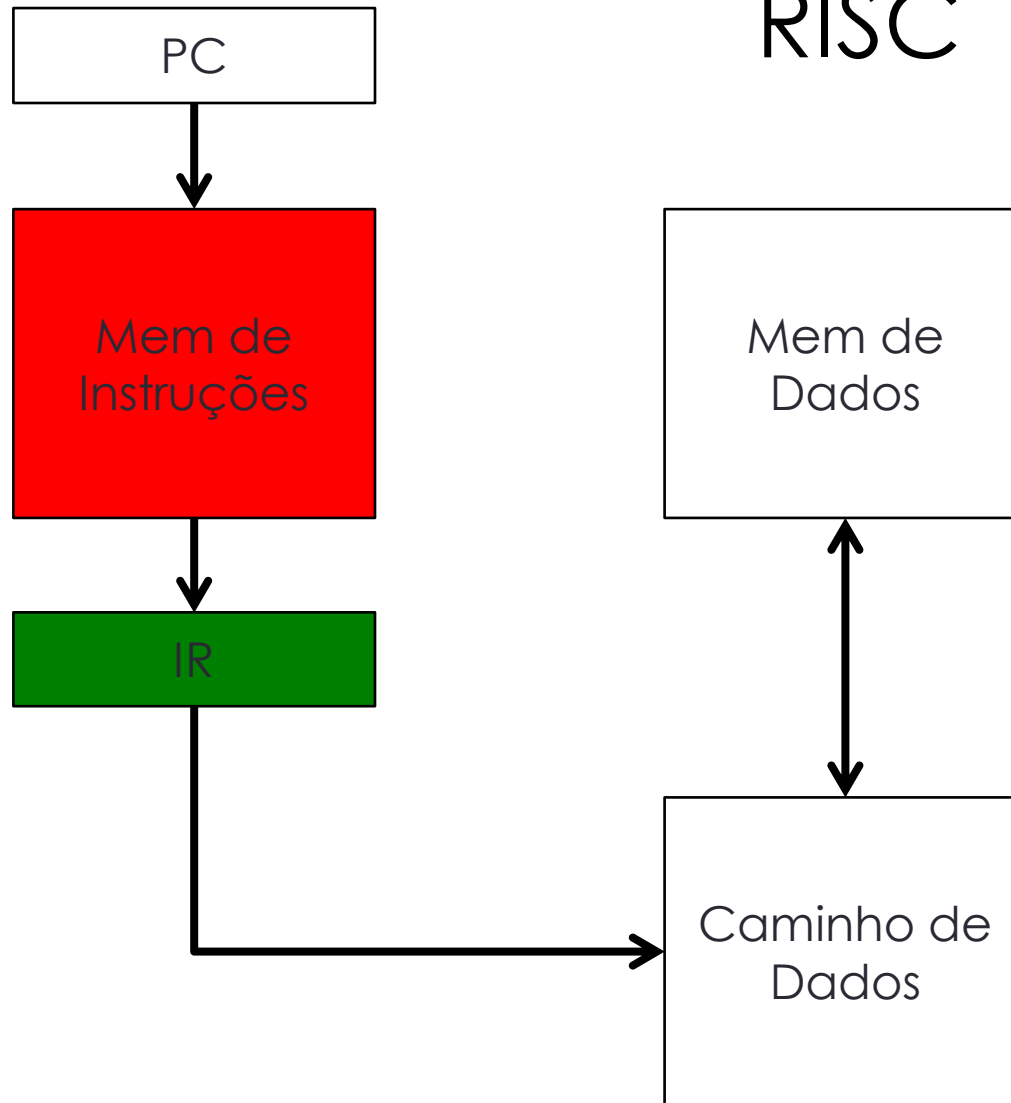
RISC – Reduced Instruction Set Computer

- Diminuir complexidade do HW e passar para o SW
- Motivação
 - Memória ficando mais barata
 - Compiladores ficando mais eficientes
 - Suporte para as linguagens de alto nível poderia ser mais eficiente em sw

CISC



RISC



ARGUMENTO RISC

Mesmo que uma máquina RISC precisasse de 4 ou 5 instruções para fazer o que uma máquina CISC faria com apenas 1 instrução, se a instrução RISC fosse 10 vezes mais rápida (só hardware) a máquina RISC venceria.

RISC – Reduced Instruction Set Computer

- Otimizações do HW
 - Excluir microcódigo
 - Excluir grupo de instruções complexas (pouco utilizadas)
 - Reduzir número e tamanho das instruções
 - Reduzir modos de endereçamento
 - Aumentar número de registradores
- **Maioria do trabalho** era feito por um **pequeno conjunto** de instruções

Como RISC melhora desempenho

- Instrução menor = menos ciclos
- Redução de complexidade = **pipeline**

$$\frac{\text{Tempo}}{\text{Programa}} = \left[\left(\frac{\text{Instruções}}{\text{Programa}} \right) \times \left(\frac{\text{Ciclos}}{\text{Instrução}} \right) \times \left(\frac{\text{Tempo}}{\text{Ciclo}} \right) \right]$$

Como RISC melhora desempenho

- Instrução menor = menos ciclos
- Redução de complexidade = **pipeline**

$$\frac{\textit{Tempo}}{\textit{Programa}} = \left[\left(\frac{\textit{Instruções}}{\textit{Programa}} \right) \times \left(\frac{\textit{Ciclos}}{\textit{Instrução}} \right) \times \left(\frac{\textit{Tempo}}{\textit{Ciclo}} \right) \right]$$

RISC



Redução

Princípios de projetos RISC

1. Todas as instruções são diretamente executadas por hardware (sem microcódigo)
2. Maximizar a taxa à qual as instruções são executadas (concorrência)
3. As instruções precisam ser facilmente decodificadas (formato e modos de ender.)
4. Somente duas instruções acessam à memória: LOAD (carrega/ler da memória) e STORE (armazena/escreve na memória)
5. Disponibilidade de um grande número de registradores

Exemplos de arquiteturas RISC

- Sistemas RISC comerciais
 - MIPS [1]
 - ✧ Poucas instruções
 - ✧ Instruções simples
 - ✧ Muitas baseadas em registrador
 - PowerPC – IBM/Motorola/Apple [2]
 - ✧ Baseado em uma ISA conhecida como POWER
 - Removidas instruções excessivas de condicionais
 - Removida acesso à memória com precisão quádrupla
 - Removidas instruções de string

CISC versus RISC

- POR QUE ENTÃO A ESTRATÉGIA RISC NÃO SUPLANTOU A CISC?

CISC versus RISC

- POR QUE ENTÃO A ESTRATÉGIA RISC NÃO SUPLANTOU A CISC?
 - Problemas de compatibilidade com máquinas antigas com software já desenvolvido
 - Aparecimento de soluções híbridas: por exemplo, a INTEL usa RISC para instruções de uso mais frequente (núcleo RISC) e interpretação para instruções mais complexas e de uso menos frequente (núcleo CISC)

CISC

- Sistemas CISC comerciais
 - PDP-11 (Digital Equipment Corporation) – 1970 a 1990
 - VAX (Digital Equipment Corporation) – 1970
 - Motorola 680x0 (Atari, Sega Mega Drive, Neo Geo,)
 - x86 (Intel)
 - AMD



Bibliografia

- STALLINGS, William

Arquitetura e organização de computadores. 8. ed.
São Paulo: Pearson, 2010. 624 p. ISBN: 9788576055648

Capítulo 11 – Conjuntos de instruções: modos e formatos de endereçamento

- SOUZA, Antonio Carlos

RISC x CISC – Pipeline. Curso de Análise e Desenvolvimento de Sistemas Arquitetura de Computadores. IFBA – Instituto Federal de Educação em Ciência e Tecnologia da Bahia

- Luís Filipe Silva e Vítor José Marques Antunes
 - **Comparação entre as arquitecturas de processadores RISC e CISC**

<http://www.inf.unioeste.br/~guilherme/oac/Risc-Cisc.pdf>

Bibliografia

- PATTERSON, D.A. & HENNESSY, J. L.
Organização e Projeto de Computadores -
A Interface Hardware/Software. 3ª ed. Campus, 2005.
Capítulo 2