Linguagens de Montagem

DEMAC – Departamento de Estatística
Matemática Aplicada e Computação
UNESP – Rio Claro

Prof. Daniel Carlos Guimarães Pedronette

Aula 2.

Detalhes de Arquitetura e Arquitetura X86

Detalhes de Arquitetura

- Detalhes de Arquitetura:
  - O Assembly está diretamente ligado à linguagem de máquina:
    - Alguns “detalhes” da arquitetura influenciam diretamente o modo de criar um programa em Assembly
- Exemplo:
  - Quantidade de endereços utilizadas pelas instruções providas pela arquitetura

Tipos de Instruções

- Agora temos instruções; não expressões aritméticas
- Binárias:
  - Requerem dois operandos como entrada
  - Exemplos: adição, multiplicação
- Unárias:
  - Requerem um único operando como entrada
  - Exemplo: Not lógico
- Resultados:
  - Maioria das instruções produz um único resultado
  - Mas há exceções: divisão (produz dois resultados: quociente e resto)

Detalhes de Arquitetura

- Detalhes de Arquitetura:
  - O Assembly está diretamente ligado à linguagem de máquina:
    - Alguns “detalhes” da arquitetura influenciam diretamente o modo de criar um programa em Assembly
- Exemplo:
  - Quantidade de Endereços das intruções

Quantidade de Endereços

- **Three-Address Machines**
  - Instruções binárias requerem três endereços:
    - Dois endereços são utilizados para os operandos de entrada
    - Um endereço indica o destino do resultado

Three-Addres Machines

Instruction	Semantics
`add dest,src1,src2`	Adds the two values at `src1` and `src2` and stores the result in `dest`
`sub dest,src1,src2`	Subtracts the second source operand at `src2` from the first at `src1` and stores the result in `dest`
`mult dest,src1,src2`	Multiplies the two values at `src1` and `src2` and stores the result in `dest`

Three-Addres Machines

- Como ficaria uma expressão em C nessas máquinas:

```
A = B + C * D - E + F + A
```

```
mult   T,C,D     ; T = C*D
add    T,T,B     ; T = B + C*D
sub    T,T,E     ; T = B + C*D - E
add    T,T,F     ; T = B + C*D - E + F
add    A,A,T     ; A = B + C*D - E + F + A
```

- Note que em vários casos T (temp) é origem e destino

Quantidade de Endereços

- **Two-Address Machines**
  - Instruções binárias requerem apenas dois endereços:
    - Dois endereços são utilizados para os operandos de entrada/origem
    - Um dos endereços é utilizado como origem e destino

Two-Addres Machines

Instruction		Semantics
`load`	`dest,src`	Copies the value at `src` to `dest`
`add`	`dest,src`	Adds the two values at `src` and `dest` and stores the result in `dest`
`sub`	`dest,src`	Subtracts the second source operand at `src` from the first at `dest` and stores the result in `dest`
`mult`	`dest,src`	Multiplies the two values at `src` and `dest` and stores the result in `dest`

Two-Addres Machines

- Como ficaria uma expressão em C nessas máquinas:

```
A = B + C * D - E + F + A
```

```
load  T,C     ; T = C
mult  T,D     ; T = C*D
add   T,B     ; T = B + C*D
sub   T,E     ; T = B + C*D - E
add   T,F     ; T = B + C*D - E + F
add   A,T     ; A = B + C*D - E + F + A
```

- Note que o operando T é muito comum

Quantidade de Endereços

- **One-Address Machines**
  - Instruções binárias requerem apenas um endereço:
    - Um registrador denominado “acumulador” é utlizado como uma das entradas e como destino
    - O endereço informa a outra origem

Quantidade de Endereços

- **Zero-Address Machines**
  - Nenhum endereço? Onde estão as entradas?
    - Origens devem estar no top-2 da pilha
    - Resultado será armazenado no topo da pilha

Arquitetura Load/Store

- Todos os dados que serão processados devem estar em registradores internos da CPU
- Apenas duas operações interagem diretamente com a memória:
  - LOAD
  - STORE

Arquitetura Load/Store

Instruction		Semantics
`load`	`Rd,addr`	Loads the `Rd` register with the value at address `addr`
`store`	`addr,Rs`	Stores the value in `Rs` register at address `addr`
`add`	`Rd,Rs1,Rs2`	Adds the two values in `Rs1` and `Rs2` registers and places the result in `Rd` register
`sub`	`Rd,Rs1,Rs2`	Subtracts the value in `Rs2` from that in `Rs1` and places the result in `Rd` register
`mult`	`Rd,Rs1,Rs2`	Multiplies the two values in `Rs1` and `Rs2` and places the result in `Rd` register

Arquitetura Load/Store

```
A = B + C * D - E + F + A
```

```
load   R1,B       ; load B
load   R2,C       ; load C
load   R3,D       ; load D
load   R4,E       ; load E
load   R5,F       ; load F
load   R6,A       ; load A
mult   R2,R2,R3   ; R2 = C*D
add    R2,R2,R1   ; R2 = B + C*D
sub    R2,R2,R4   ; R2 = B + C*D - E
add    R2,R2,R5   ; R2 = B + C*D - E + F
add    R2,R2,R6   ; R2 = B + C*D - E + F + A
store  A,R2       ; store the result in A
```

CISC x RISC

- CISC:
  - Complex Instruction Set Computer
  - Instruções complexas
    - Em muitas situações, código assembly reduzido
- RISC:
  - Reduced Instruction Set Computer
  - Instruções muito simples
    - Podem ser executadas rapidamente

CISC x RISC

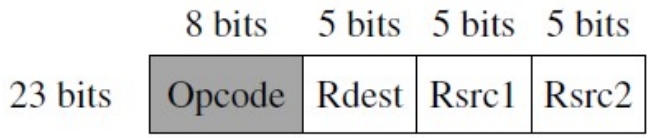
- CISC:
  - X86 – Pentium:
    - Two-Address, Acesso Memória
- RISC:
  - MIPS, PowerPC
    - Three-Address, Arquitetura Load/Store
- Atualmente:
  - Distinção CISC x RISC está tênue, máquinas “híbridas”

CISC x RISC

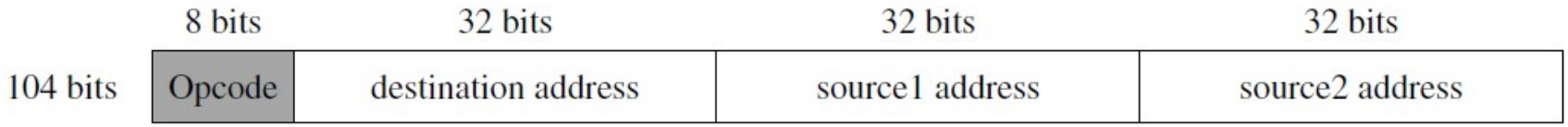
- CISC:
  - X86 – Pentium:
    - Two-Address, Acesso Memória,
- RISC:
  - MIPS, PowerPC
    - Three-Address, Arquitetura Load/Store
- Atualmente:
  - Distinção CISC x RISC está tênue, máquinas “híbridas”

Registradores

- CISC:
  - Poucos registradores (Pentium: 10)
- RISC:
  - Maior número de registradores (MIPS: 32)
- Comparação das instruções:



Register format

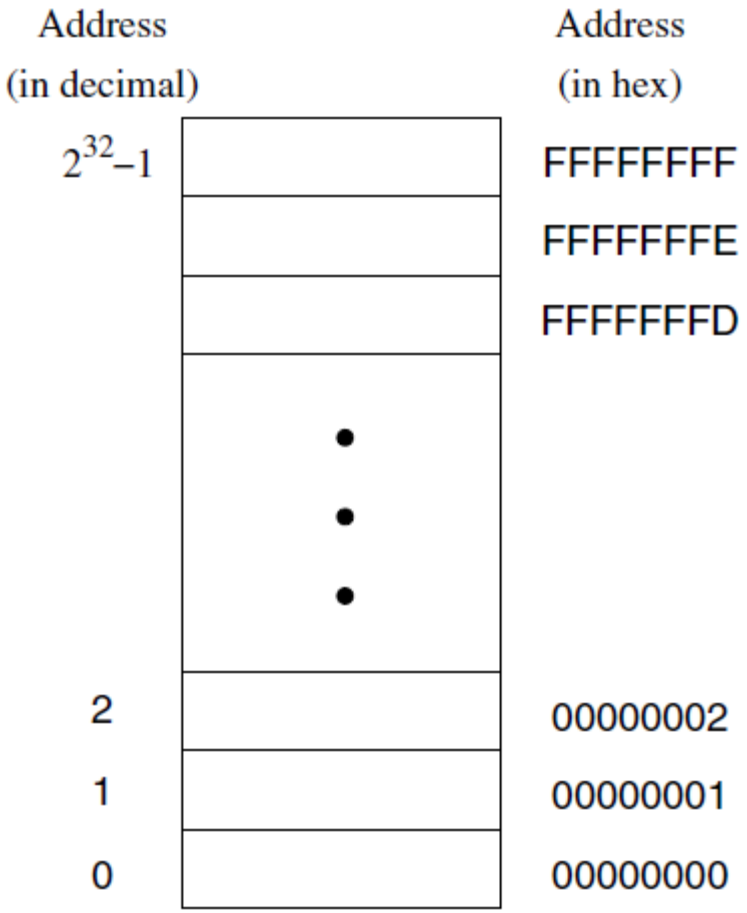


Memory format

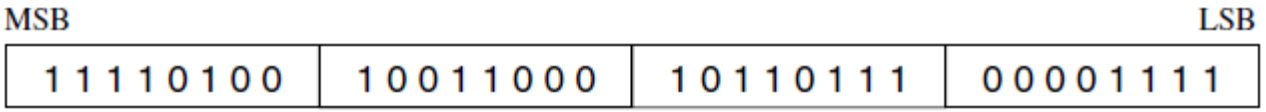
Memória

- Memória é basicamente uma estrutura de armazenamento binária:
  - É “endereçavel” por bytes, para qualquer operação de leitura ou escrita
  - Como é realizado o endereçamento

Memória

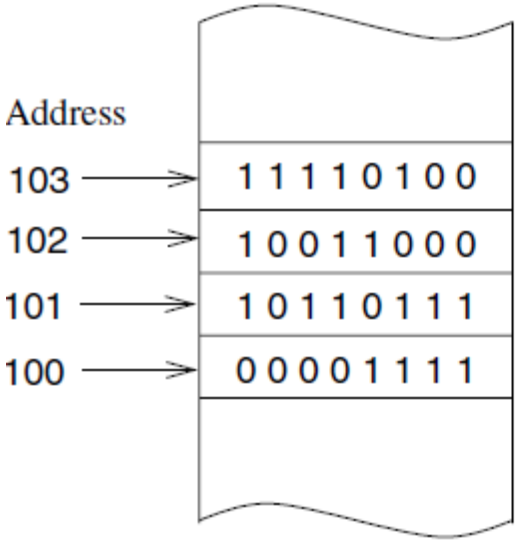

Address
(in decimal)
Address
(in hex)
$2^{32}-1$
FFFFFFFF
FFFFFFFE
FFFFFFFD
2
00000002
1
00000001
0
00000000

Memória - Endereçamento

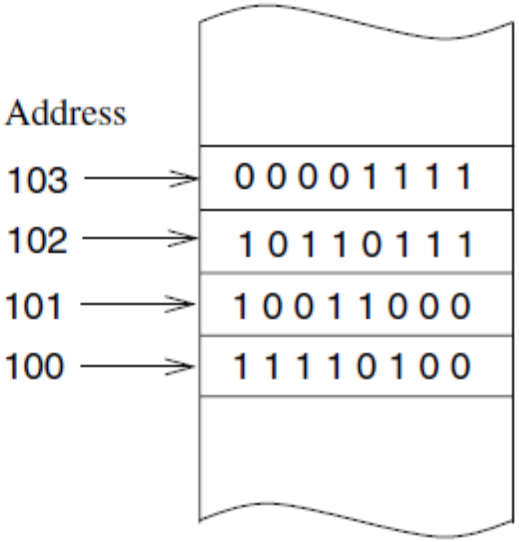


(a) 32-bit data

Pentium



(b) Little-endian byte ordering



(c) Big-endian byte ordering

MIPS

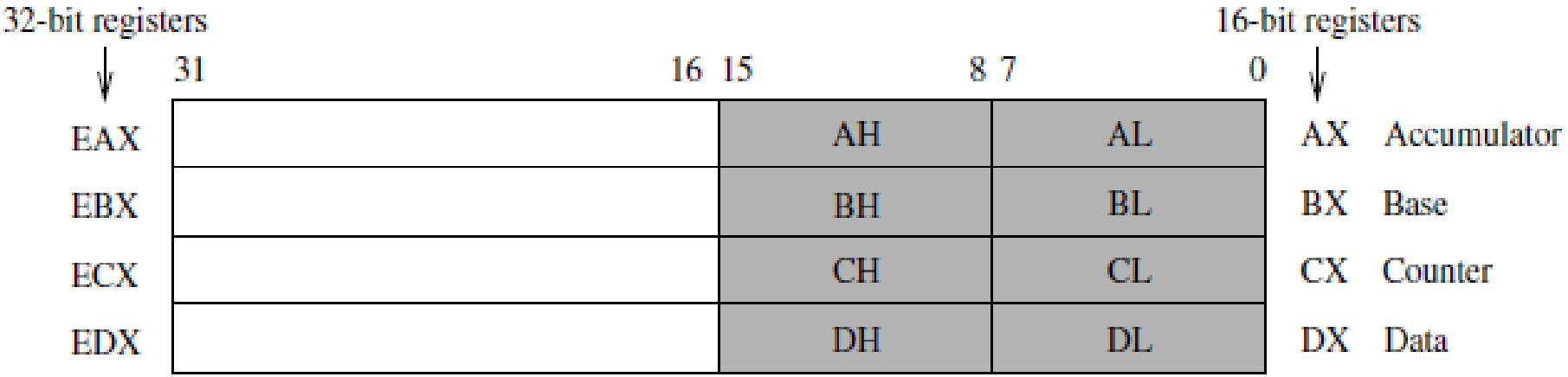
Arquitetura X86 - Pentium

- Registradores:
  - 10 32 bits; 6 16 bits
- Classificados em:
  - Dados
  - Ponteiros
  - Controle

Registradores de Dados

- Em geral utilizados para:
  - Armazenar dados de operações
  - Lógicas
  - Artiméticas
  - Outras
- Registradores:
  - 4 32-bit (EAX, EBX, ECX, EDX)
  - 4 16-bit (AX, BX, CX, DX)
  - 8 8-bit (AH, AL, BH, BL, CH, CL, DH, DL)
-

Registradores de Dados



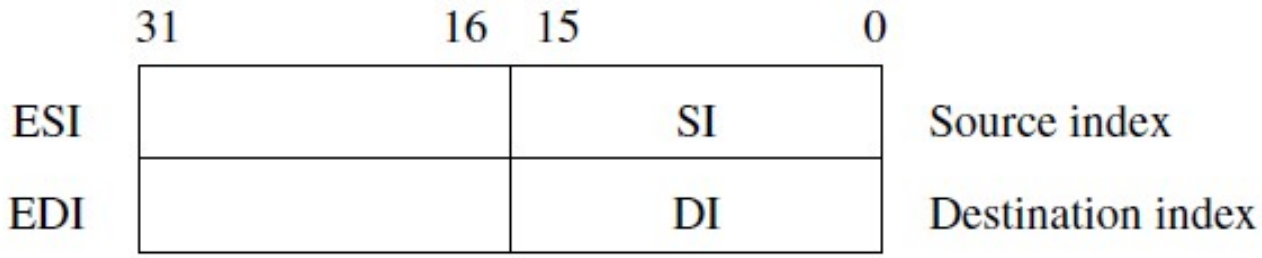
Registradores de Ponteiros/Índices

- Dois Registradores de Índices (ESI, EDI)
  - 16- or 32-bit registers
  - Utilizados em instruções com strings
  - Source (SI) and destination (DI)
  - Podem ser usados para propósito geral

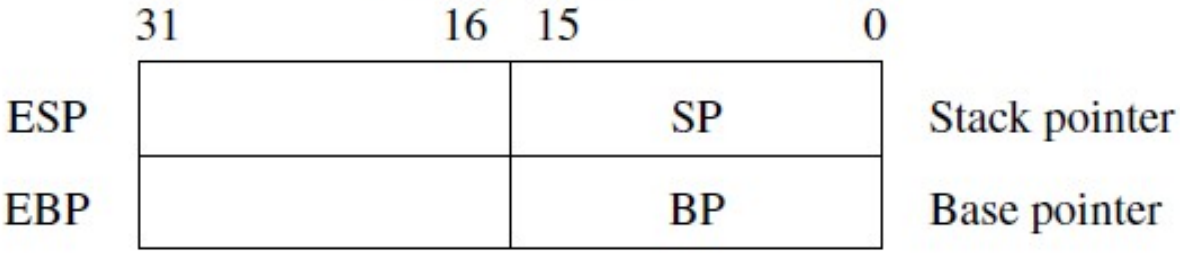
- Dois registradores de Ponteiros (ESP, EBP)
  - 16- or 32-bit registers
  - Usados exclusivamente para manter a pilha

Registradores de Ponteiros/Índices

Index registers



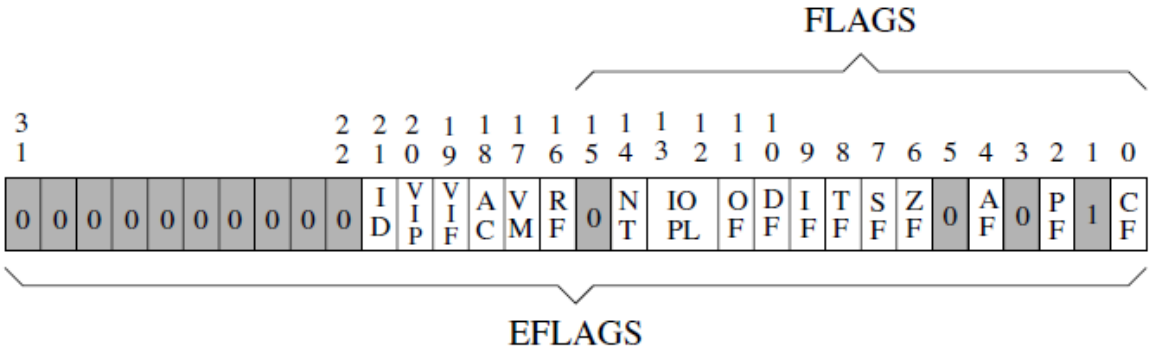
Pointer registers



Registradores de Controle

- Dois Registradores de 32 bits
  - Instruction Pointer (EIP)
    - “Program Counter”
  - Flags (EFLAGS)
    - Status flags: status da última operação lógico aritmética
    - Direction flag: direção na cópia de dados
    - System flags: controle de interrupções

Registradores de Controle



Status flags

CF = Carry flag
PF = Parity flag
AF = Auxiliary carry flag
ZF = Zero flag
SF = Sign flag
OF = Overflow flag

Control flag

DF = Direction flag

System flags

TF = Trap flag
IF = Interrupt flag
IOPL = I/O privilege level
NT = Nested task
RF = Resume flag
VM = Virtual 8086 mode
AC = Alignment check
VIF = Virtual interrupt flag
VIP = Virtual interrupt pending
ID = ID flag