UNIVERSIDADE FEDERAL DE RORAIMA

Conjunto de Instruções: Características e Funções

Prof. Herbert Oliveira Rocha

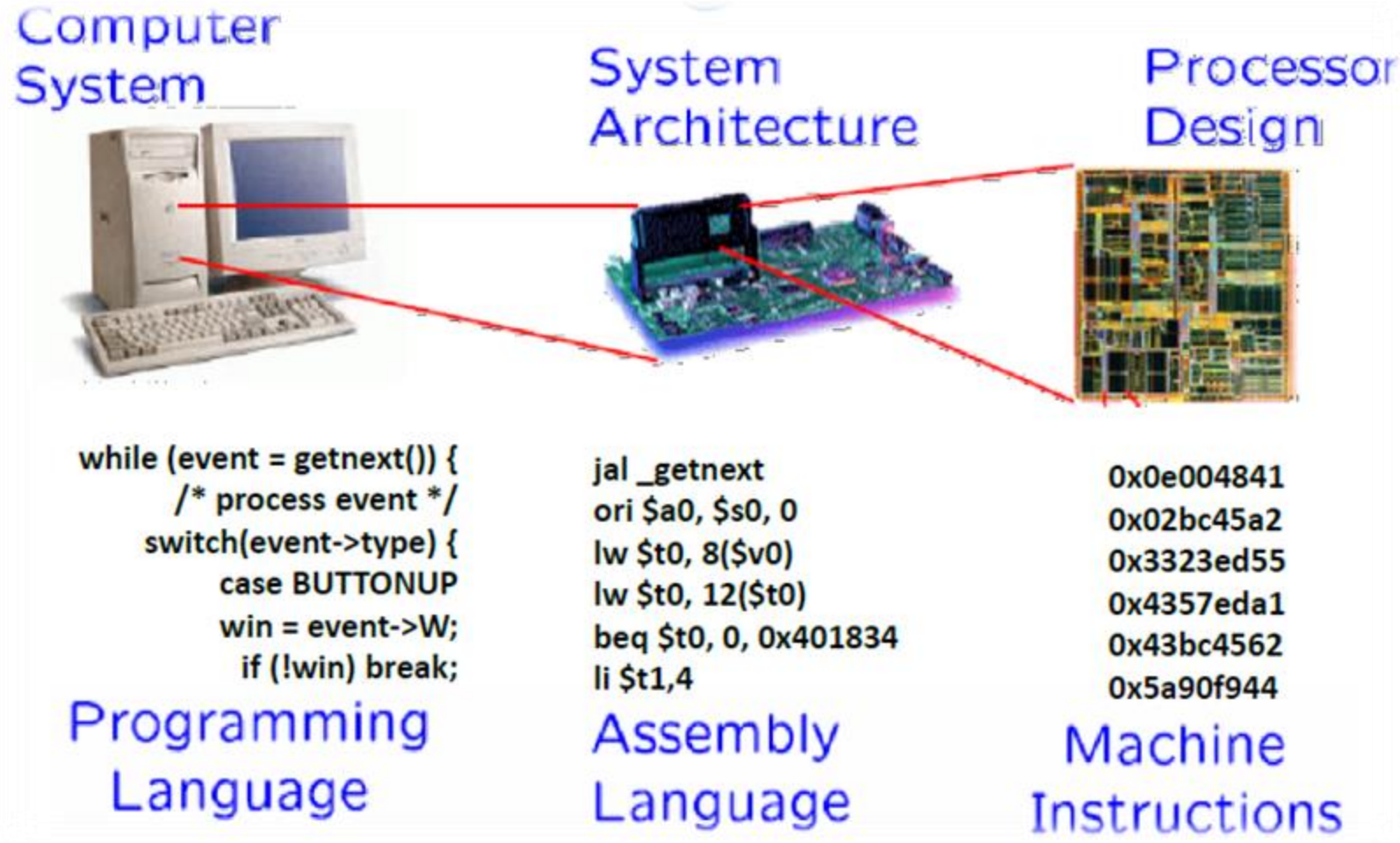
UNIVERSIDADE FEDERAL DE RORAIMA

Conjunto de Instruções: Características e Funções

Baseado nas aulas do Prof. Leandro Galvão e
Prof. Mauricio Figueiredo - UFAM

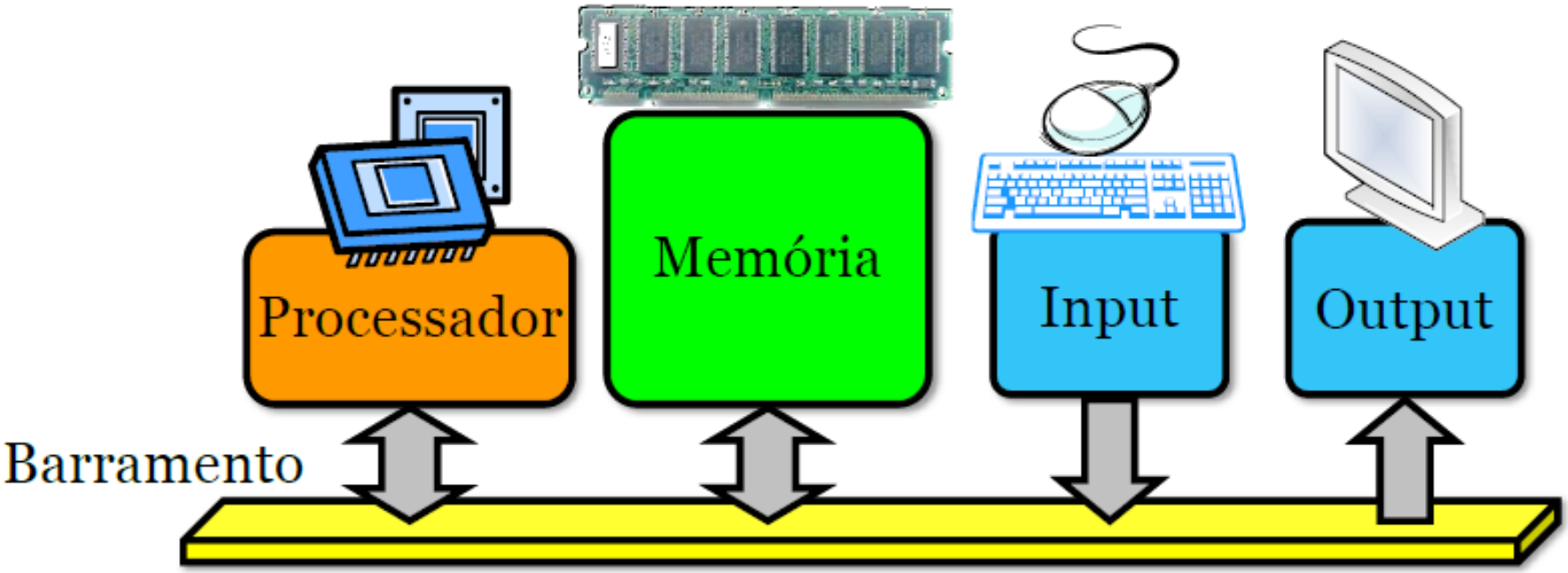
Prof. Herbert Oliveira Rocha

Anatomia de um Computador



Anatomia de um Computador

- Modelo de Von Neumann (1945)
  - Conceito de programa armazenado
  - Separação da Unidade Aritmética e de Controle

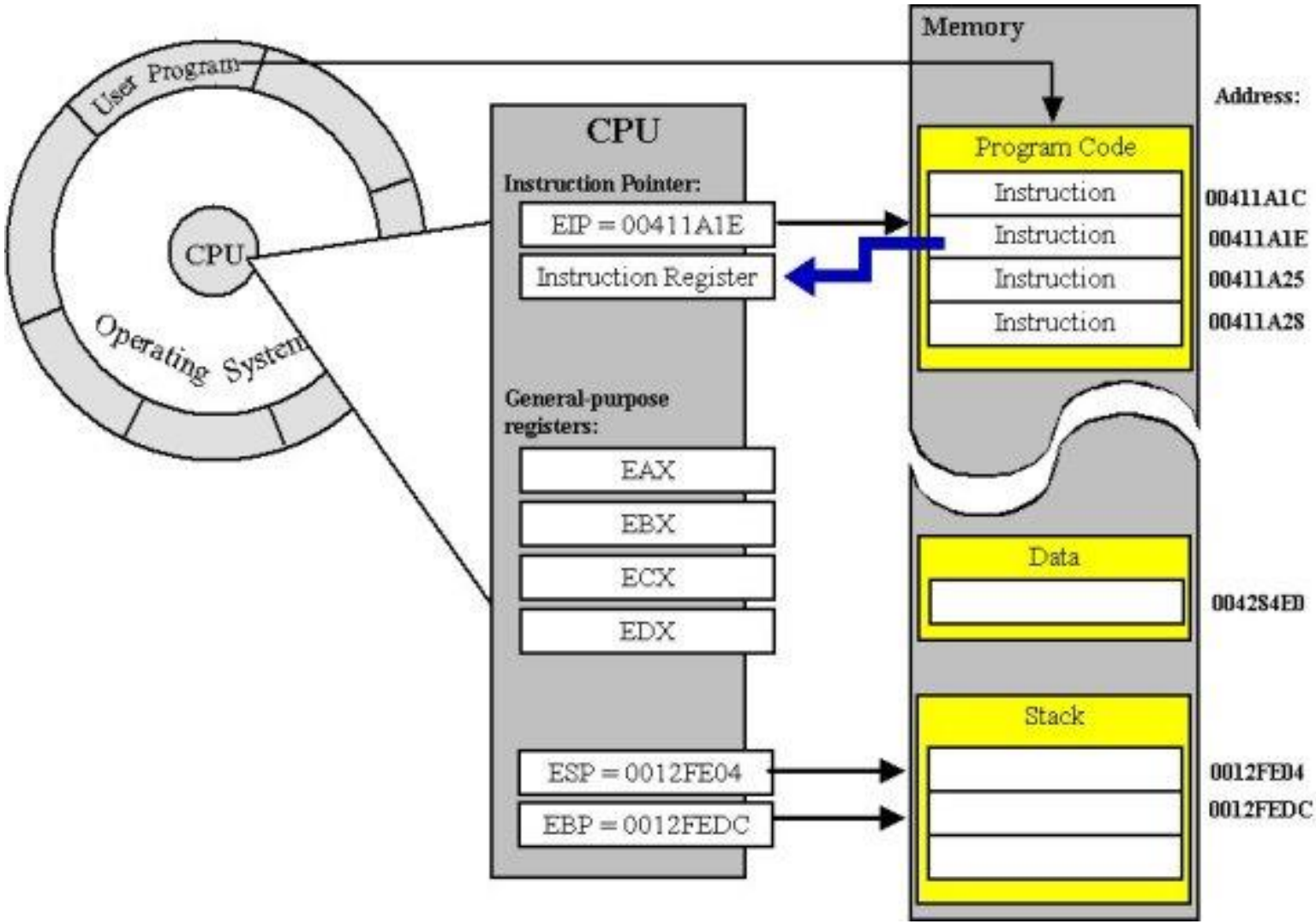


Anatomia de um Computador

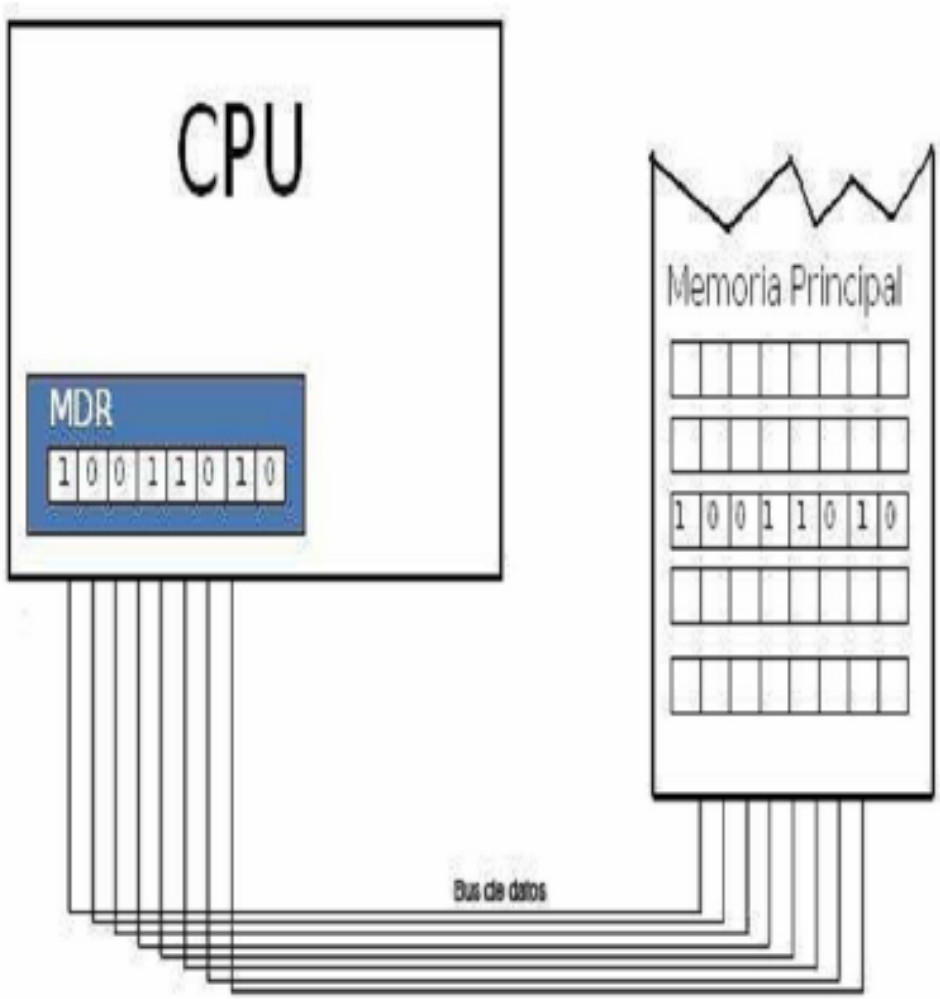
Memória Principal

- A unidade de memória central serve para guardar programas e dados, sob a forma de uma representação binária;
- Cada instrução da máquina é codificada como uma sequência de bits;
- Cada valor de certo tipo é codificado por uma determinada sequência de bits.

Anatomia de um Computador



Anatomia de um Computador



Anatomia de um Computador

Por baixo de seu programa

- Digito binário: computador entende instruções de máquinas, formadas por lotes de bits
- Montador (assembler): converte os programas em linguagem de montagem (assembly) para a linguagem binária
- Linguagens de alto nível: permitem a escrita de programas em uma linguagem mais próxima as notações lógicas e algébricas utilizadas.

Anatomia de um Computador

Unidade Central de Processamento (UCP)

- Trata do controle global das operações e da execução das instruções

- Contém as seguintes unidades internas:
  - Unidade Lógica e Aritimética (ULA): executa as principais operações lógicas e aritméticas do computador
  - Unidade de Controle (UC): busca, decodifica e executa

Anatomia de um Computador

Entrada e Saída

- As unidades periféricas destinam-se a suportar as ações de comunicação da CPU e memória com o exterior

- Também há unidades periféricas destinadas ao armazenamento de dados, que são depois apresentados ao usuário, sob a forma de arquivos, geridos pelos programas do Sistema Operacional

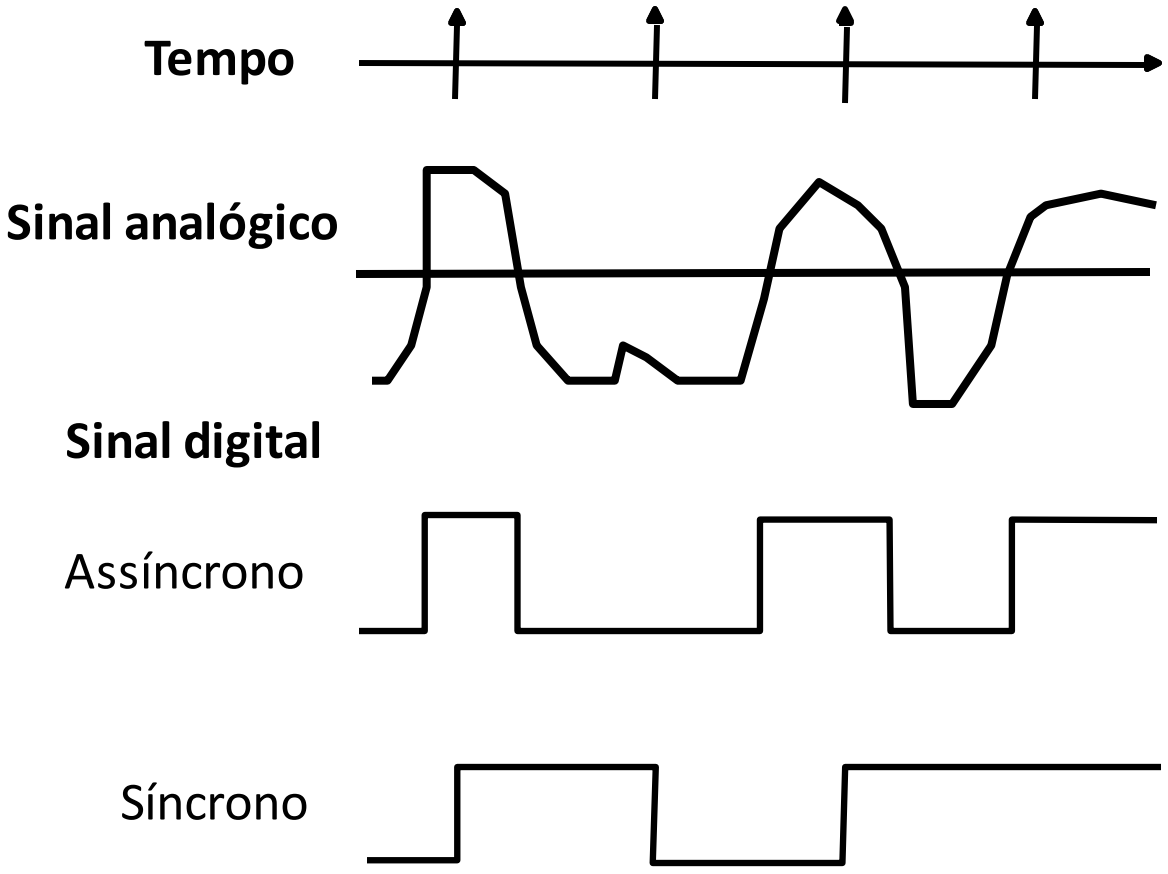
Anatomia de um Computador

Processadores

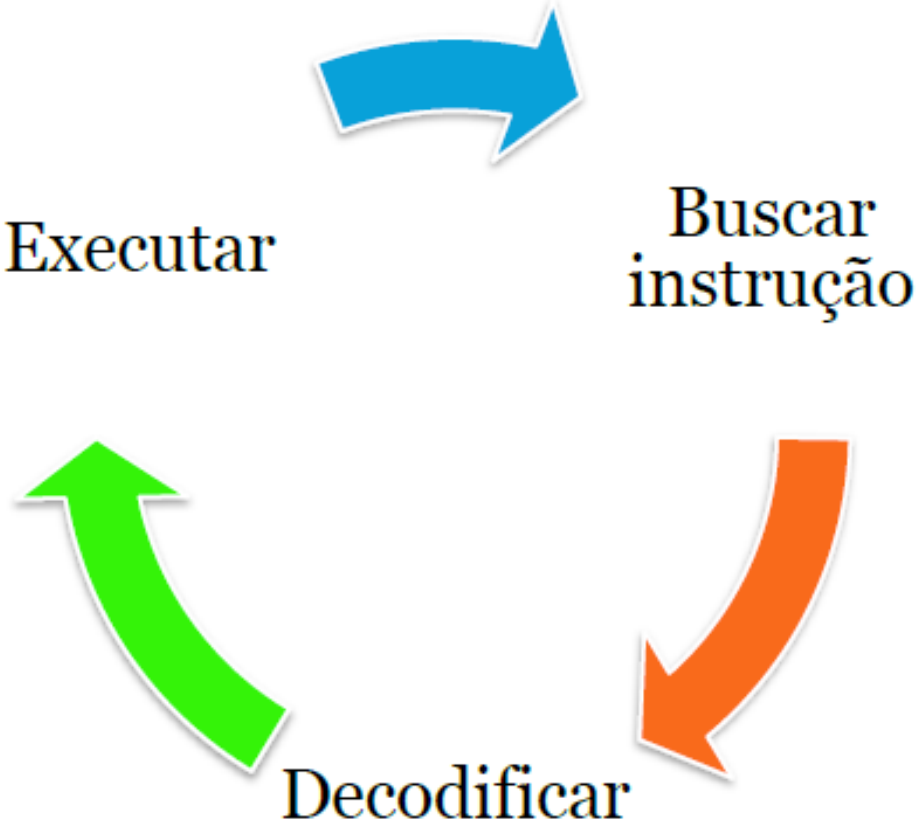
- Seu funcionamento é coordenado pelos programas
- Executa cálculos muito simples de forma bastante rápida
- Trabalha diretamente com a memória principal
- Utiliza o ciclo busca-execução regulado pelo clock (relógio)

Anatomia de um Computador

Exemplos de Sinais ao longo do Tempo



Ciclo Básico de Execução



RISC × CISC

CISC (Complex Instruction Set Computer)

- Visa facilitar a construção dos compiladores, assim, programas complexos são compilados em programas de máquina mais curtos

RISC (Reduced Instruction Set Computer)

- Parte do pressuposto de que um conjunto simples de instruções vai resultar numa Unidade de Controle simples, barata e rápida.

RISC × CISC

CISC	RISC
• Grande variedade de instruções • Instruções de tamanho variado • Poucos registradores • Operações em memória • Utiliza microcódigo	• Número reduzido de instruções • Instruções de mesmo tamanho • Muitos registradores • Operações somente entre registradores • Instruções executadas diretamente em HW

http://www.gruponetcampos.com.br/2011/03/arquitetura-cisc-e-risc-qual-diferenca

UNIVERSIDADE FEDERAL DE RORAIMA

Instruções: Características e Funções

Baseado nas aulas do Prof. Leandro Galvão e
Prof. Mauricio Figueiredo - UFAM

Prof. Herbert Oliveira Rocha

Introdução

- Instruções
  - Operações (ou comandos) dados ao hardware
  - São as palavras da **linguagem de máquina** (ling. de baixo nível)

- Objetivos
  - Como humanos passam instruções aos computadores?
  - Como computadores as interpretam e executam?
  - Conjuntos de instruções típicas?

- Linguagens de máquina são semelhantes
  - HW seguem os mesmo princípios fundamentais

Introdução

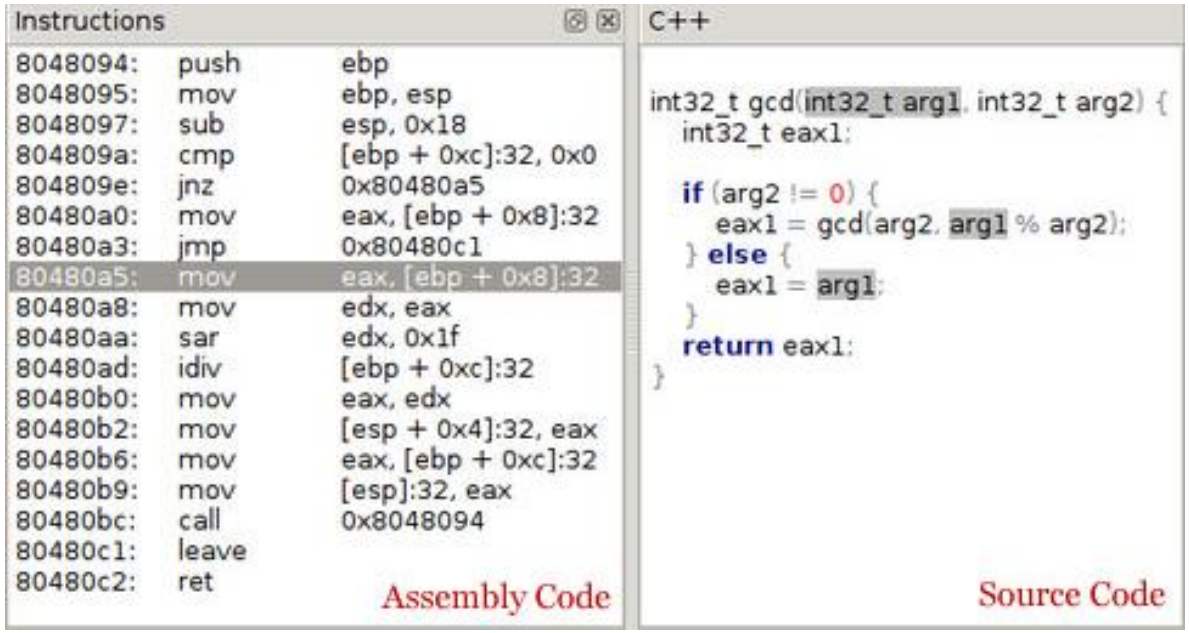
- Muitas situações exigem o conhecimento de linguagem de máquina
    - Ex. Sistemas embarcados, Sistemas Operacionais, Drivers de diferentes periféricos etc.

- Quem conhece como o HW funciona e como interagimos com ele em baixo nível tem facilidade com qualquer outra linguagem de alto nível
    - É o papel do engenheiro ou cientista da computação!!

Introdução

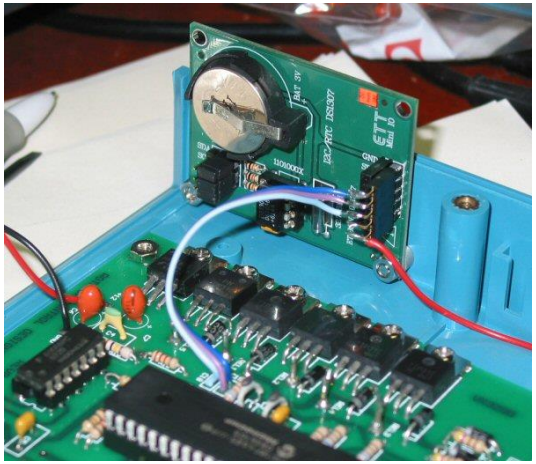
- O programa elaborado pelo programador (o código-fonte, composto de instruções complexas) precisa ser "traduzido" em pequenas operações elementares executáveis pelo hardware

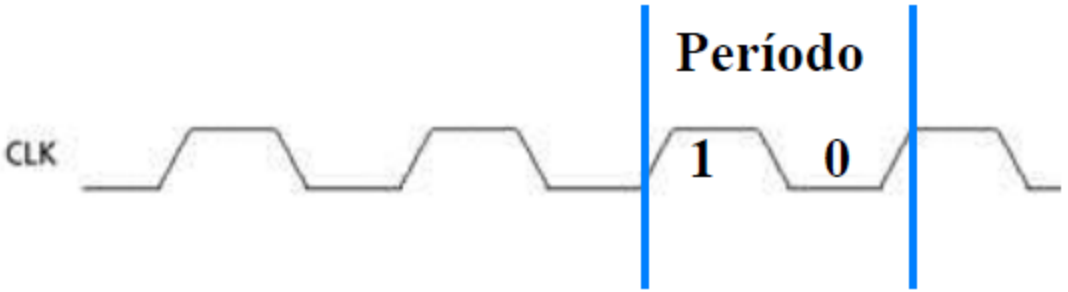
- Cada uma das instruções tem um código binário associado, que é o código da operação



Relógio - Clock

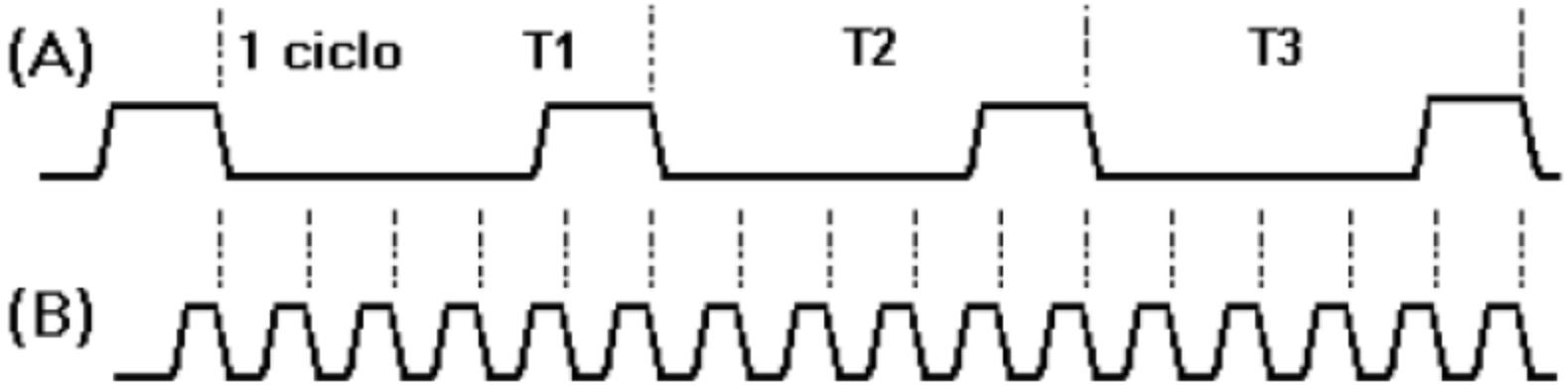
- Dispositivo gerador de pulsos cuja duração é chamada de ciclo
- A unidade de medida usual para a frequência dos relógios de UCP é o Hertz (Hz), que significa 1 ciclo por segundo





Relógio - Clock

- Como podemos ver pelo exemplo a seguir, o processador com o clock ilustrado em (B) teria um tempo de ciclo cinco vezes menor que o (A) e portanto teria (teoricamente) condições de fazer cinco vezes mais operações no mesmo tempo.



Estratégias de Implementação de Processadores

CISC - Complex Instruction Set Computer

- Exemplo: **PC, Macintosh**; um conjunto de instruções maior e mais complexo, implicando num processador mais complexo, com ciclo de processamento mais lento

RISC - Reduced Instruction Set Computer

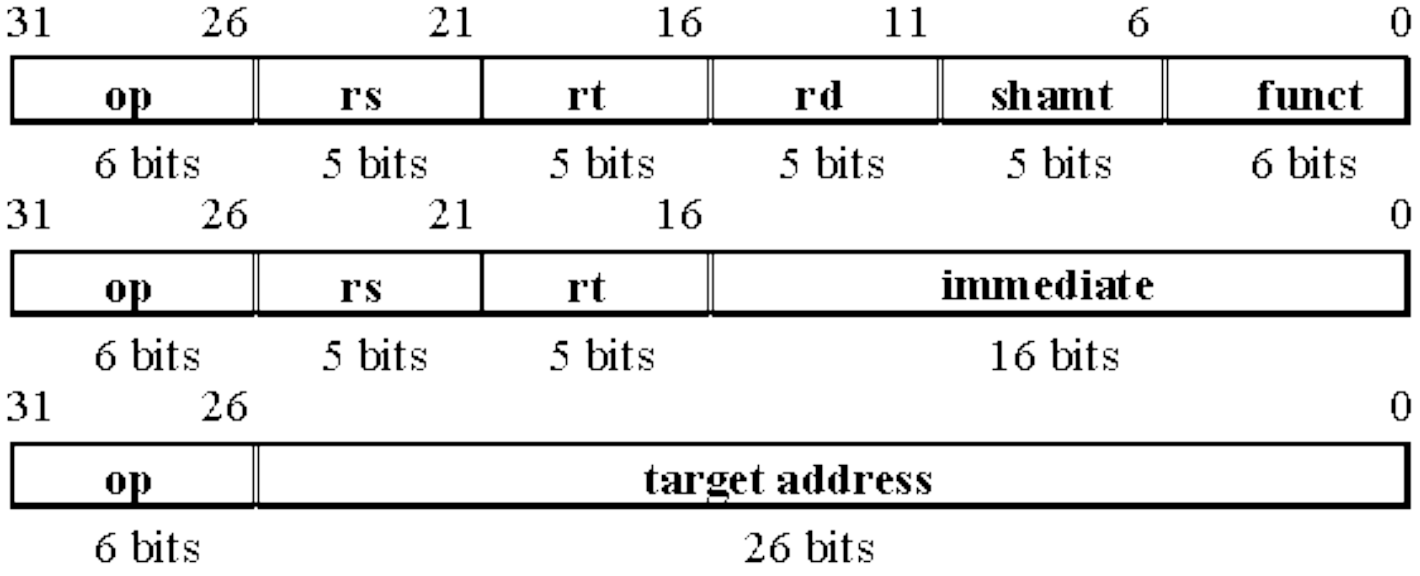
- Exemplo: **Power PC, Alpha, MIPS, Sparc**; um conjunto de instruções menor e mais simples, implicando num processador mais simples, com ciclo de processamento rápido

Estudo do MIPs

- Processador MIPs
  - Comum na década de 80 em plataformas Nintendo, NEC, Sony, SG etc.
  - Simples, sendo bastante didática
  - Muito semelhante aos RISC modernos
- Simulador SPIM
  - http://spimsimulator.sourceforge.net
- O conjunto de instruções do MIPs e seu uso se dá de forma análoga a outras arquiteturas

Estudo do MIPs

- No MIPs
  - São 32 registradores de uso geral de 32 bits cada.
  - Nomenclatura $s0 - $s31 (variáveis) ou $t0 - $t31 (temporários)
  - Porque só 32? Porque mais registradores aumentam a complexidade do processador e afetam desempenho



Estudo do MIPs

- Operações fundamentais em qualquer arquitetura
- No MIPs:

Linguagem de montagem do MIPS

Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	add	add a , b , c	a = b + c	Sempre três operandos
	subtract	sub a , b , c	a = b − c	Sempre três operandos

Estudo do MIPs

- Em linguagem de alto nível:

```
f = (g + h) - (i + j);
```

- Em assembly (ling. de montagem) do MIPs:
  - Assumindo variáveis nos regs. $s0 a $s4
  - Uso de $t0 e $t1 como regs. temporários

```
add   $t0,$s1,$s2   # registrador $t0 contém g + h
add   $t1,$s3,$s4   # registrador $t1 contém i + j
sub   $s0,$t0,$t1   # f recebe $t0 – $t1, que é (g + h) – (i + j)
```

Transferência de Dados

- Programas podem usar estruturas de dados complexas e muito extensas.
  - Há poucos registradores para operações
  - Logo, muito comum o uso de instruções de transferência de dados entre a memória e os registradores
  - No MIPs: Load (lw) e Store (sw)

Transferência de dados	load word	`lw $s1, 100 ($s2)`
	store word	`sw $s1, 100($s2)`

`$s1 = Memória[$s2 + 100]`	Dados transferidos da memória para registradores
`Memória[$s2 + 100] = $s1`	Dados transferidos de registradores para a memória

Break

Memória

Armazena instruções e dados durante a execução de um programa

• A memória principal pode ser vista como um “array” de bytes, cada um com seu endereço (“índice” do array), iniciando com 0

  – dados ocupam um número de bytes que depende do seu tipo

  – instruções tem um número variável de bytes

• Registradores da CPU armazenam dados e endereços

  – o tamanho (em bits) dos registradores limita o número de posições de memória endereçáveis.

• tamanho máximo da memória endereçável é 2^{32}

Break

Representação da informação

- Computadores armazenam e processam informações representadas por “sinais” de dois valores (estados)
  - “binary digits” ou “bits“

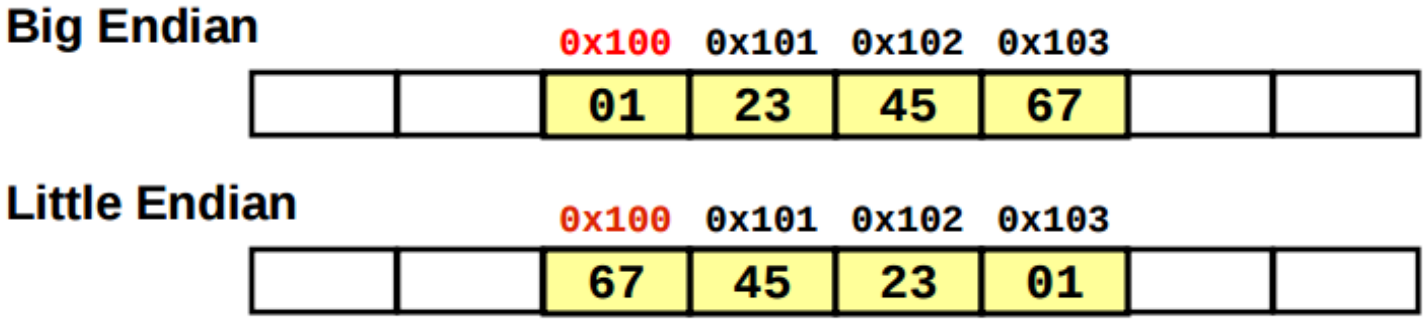
- Agrupando conjuntos de bits podemos representar valores numéricos
  - sequência de bits: representação em notação posicional, base 2
  - representação de valores numéricos podem utilizar diferentes tamanhos (número de bytes)

Ordenação de Bytes

- Big Endian (computadores Sun, Mac – não Intel)
  - byte mais significativo com endereço mais baixo

- Little Endian (Intel)
  - byte mais significativo com endereço mais alto

- Exemplo



Break

Limitações de Representação

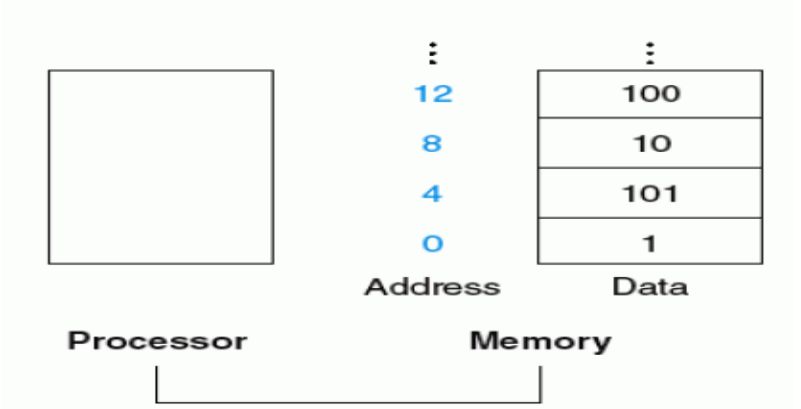
- Representação de valores numéricos podem utilizar diferentes tamanhos (número de bytes)

- Representação de inteiros não negativos (sem sinal)
  - com 1 byte (8 bits) podemos representar inteiros de 0 a 255 (2^8-1)
  - com 2 bytes (16 bits), inteiros de 0 a 65535 (2^{16}-1)
  - com 4 bytes (32 bits), inteiros de 0 a 4294967295 (2^{32}-1)

- Essa limitação vale também para endereços
  - com 2 bytes (16 bits) só podemos endereçar 64KB de memória
  - com 4 bytes (32 bits) podemos endereçar **4GB de memória**

Estudo do MIPs - Restrição de Alinhamento

- Programas costumam armazenar dados em Arrays
- A memória é endereçada em bytes (8 bits) na maioria das arquiteturas
  - Como manipular dados em 32 bits (palavra), como no MIPs?
  - **Palavras sequenciais em múltiplos de 4 bytes!**
  - É um ajuste necessário na maioria das arquiteturas
    - Se deve à evolução gradual das arquiteturas (8, 16, 32 e agora 64 bits)
- Em um array, para acessar a 8o. palavra da memória:
  - Índice 8 de um array de itens de 32 bits – Ex. A[8]
  - Posição da memória = 4x8 = 32



Estudo do MIPs - Exemplo

- Em ling. De alto nível:

Suponha que a variável h esteja associada ao registrador $s2 e que o endereço-base do array A esteja armazenado em $s3. Qual o código de montagem do MIPS para o comando de atribuição seguinte, escrito em C?

```
A[12] = h + A[8];
```


- Em assembly do MIPs:

```
lw    $t0,32 ($s3)    # Registrador temporário $t0 recebe A[8]
add   $t0,$s2, $t0    # Registrador temporário $t0 recebe h + A[8]
 sw   $t0,48($s3)   # h + A[8] é armazenado de volta na memória em A[12]
```

Estudo do MIPs – Exemplo 2

- Array indexado

A seguir, apresentamos um exemplo de um array indexado por uma variável.

```
g = h + A[i];
```

Suponha que `A` é um array de 100 elementos cujo endereço-base está no registrador `$s3` e que o compilador associa as variáveis `g`, `h` e `i` aos registradores `$s1`, `$s2` e `$s4`. Qual o código gerado para o MIPS, correspondente ao comando C acima?

Estudo do MIPs - Solução

- Índice do Array i, na memória : 4x$4

```
add     $t1,$s4,$s4    # Registrador temporário $t1 recebe 2 * i
add     $t1,$t1,$t1    # Registrador temporário $t1 recebe 4 * i
```

- Endereço de A[i]

```
add  $t1,$t1,$s3   # Registrador temporário $t1 recebe o endereço de A[i] (4 * i + $s3)
```

- Carregando A[i] no reg. Temporário

```
lw    $t0,0($t1)    # Registrador temporário $t0 recebe A[i]
```

- Soma

```
add   $s1,$s2,$t0   # g recebe h + A[i]
```

Instruções de Desvio

- Programas necessitam de desvios e repetições em sua execução conforme a verificação de alguma condição
  - **Diferença básica entre computador e calculadora**
- No MIPs (beq - branch if equal, bne – branch if not equal)
  - Se reg1 == reg2, pula se igual, p/ a instrução com label L1

```
beq registrador1, registrador2, L1
```

  - Se reg1 <> reg2, pula se não igual

```
bne registrador1, registrador2, L1
```

- No MIPs, salto incondicional (jump)

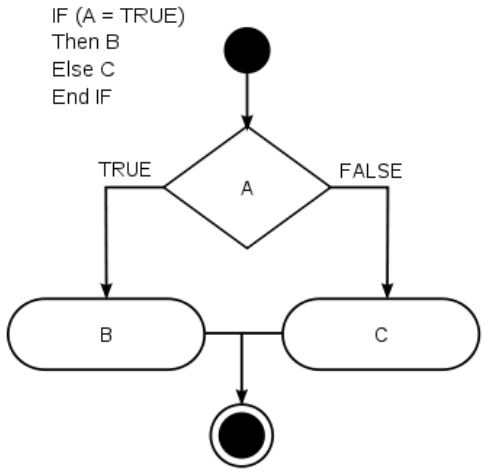
```
j L1 # jump to L1
jr $reg # jump p/ endereço em $reg
```

- Label: Endereço da memória onde se encontra a instrução
  - Montador resolve endereços de labels
  - Linguagens de alto nível normalmente abstraem labels

Estudo do MIPs

- Em alto nível:

```
        if (i == j) go to L1;
        f = g + h;
L1:     f = f - i;
```



- f, g, h, i e j em $s0 a $s4

```
beq    $s3,$s4,L1     # desvia para L1 se i for igual a j
add  $s0,$s1,$s2   # f = g + h (instrução não executada se i for igual a j)
L1: sub $s0,$s0,$s3   # f = f - i (sempre executado)
```

Estudo do MIPs

- Com as mesmas variáveis e regs. Anteriores

```
if (i == j) f = g + h; else f = g - h;
```

- Melhor verificar a condição negativa primeiro, que onde há necessidade de desvio

```
bne   $s3,$s4,Else  # desvia para Else se i ≠ j
add  $s0,$s1,$s2    # f = g + h (salta essa instrução se i ≠ j)
j Exit   # desvia para Exit
Else:   sub $s0, $s1, $s2 # f = g - h (salta essa instrução se i = j)
Exit:
```