Arquitetura de Computadores
ACH2055
Aula 09 – CISC $\times$ RISC

Norton Trevisan Roman
(norton@usp.br)

22 de novembro de 2019

CISC × RISC

Sic erat in principio

- No início, códigos eram escritos diretamente em assembly
- Existiam poucas linguagens de alto nível
  - Não havia grandes preocupações em adequar o *hardware* a elas

Fonte: `https://comic.browserling.com/32`

CISC × RISC

Sic erat in principio

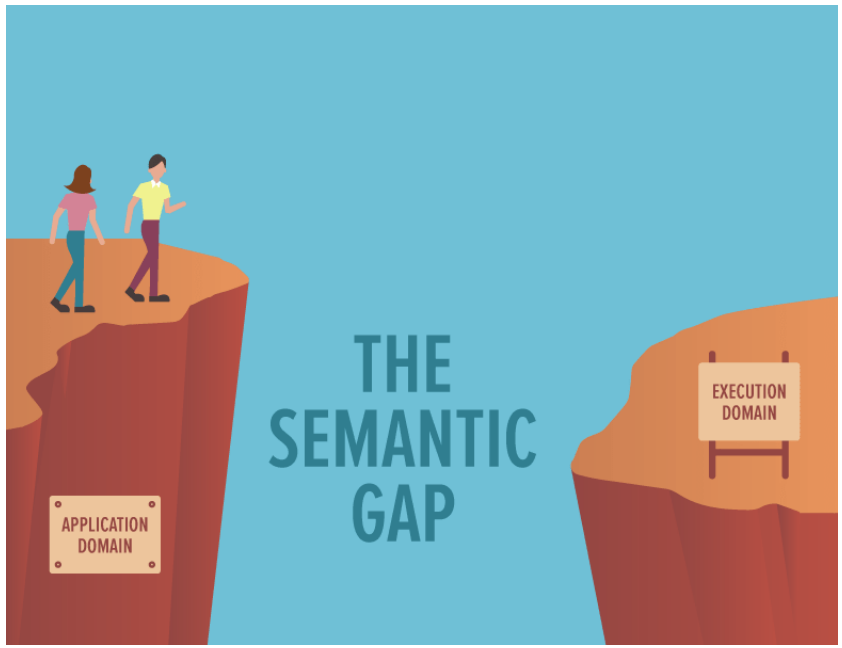
- Com o tempo, contudo, linguagens mais poderosas, e de mais alto nível surgiram
  - Permitindo ao programador codificar os algoritmos de forma mais concisa

Fonte: `https://medium.com/the-andela-way/the-rise-of-modern-programming-languages-c923a2b914fc`

CISC × RISC

Sic erat in principio

- O que fez surgir um problema conhecido como **disparidade semântica** (*semantic gap*)
  - A diferença entre as operações fornecidas por essas linguagens e as oferecidas pela arquitetura

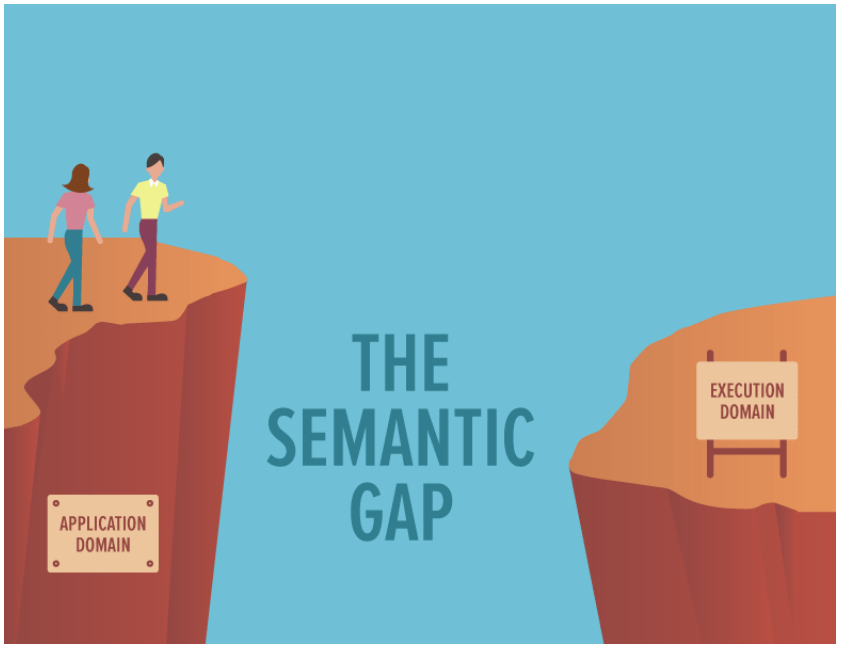


Fonte: [2]

CISC × RISC

Sic erat in principio

- Os projetistas de hardware buscaram reduzir então essa disparidade
  - Criando grandes conjuntos de instruções
  - Com dúzias de modos de endereçamento
  - E vários comandos de alto nível implementados em hardware



Fonte: [2]

CISC

- Nasceu assim a abordagem **CISC**
  - *Complex Instruction Set Computers*

CISC

- Nasceu assim a abordagem **CISC**
  - *Complex Instruction Set Computers*

Objetivos

- Simplificar a construção de compiladores
  - Se as instruções de máquina parecerem com os comandos da linguagem de alto nível, a tarefa de tradução de uma para outra é simplificada
- Fornecer suporte a linguagens de alto nível melhores e mais sofisticadas

CISC

Objetivos (cont.)

- Melhorar a eficiência, pela geração de programas menores e mais rápidos
  - Programas menores ocupam menos memória
  - São mais rápidos porque precisam buscar menos instruções na memória, além de ocuparem menos páginas, evitando assim *page faults*

CISC

Objetivos (cont.)

- Melhorar a eficiência, pela geração de programas menores e mais rápidos
  - Programas menores ocupam menos memória
  - São mais rápidos porque precisam buscar menos instruções na memória, além de ocuparem menos páginas, evitando assim *page faults*
  - Atinge esses objetivos implementando sequências complexas de operações em microcódigo

CISC

Objetivos (cont.)

- Melhorar a eficiência, pela geração de programas menores e mais rápidos
  - Programas menores ocupam menos memória
  - São mais rápidos porque precisam buscar menos instruções na memória, além de ocuparem menos páginas, evitando assim *page faults*
  - Atinge esses objetivos implementando sequências complexas de operações em microcódigo
  - Microcódigo?

CISC

Microcódigo

- Conjunto de instruções elementares em um modelo CISC
  - Reside em memória separada (Ex: ROM)
  - Serve como uma camada de tradução entre as instruções de máquina e o circuito

CISC

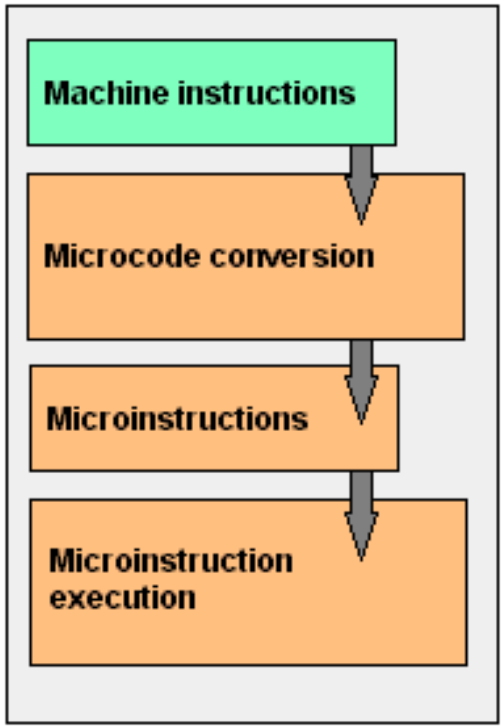
Microcódigo

- Conjunto de instruções elementares em um modelo CISC
  - Reside em memória separada (Ex: ROM)
  - Serve como uma camada de tradução entre as instruções de máquina e o circuito
- Permite a criação de instruções sem implementação direta no circuito
  - Seu microcódigo é que possui tal implementação
  - Uma única instrução no microcódigo é chamada de **microinstrução**

CISC

Microcódigo

- Passos de um programa em CISC:
  - O código fonte é traduzido em instruções de máquina (pelo compilador ou assembler)
  - Em tempo de execução, essas instruções são convertidas em microinstruções
  - Essas microinstruções, por sua vez, interagem com o circuito



Fonte: [4]

CISC

Mas nem tudo é um mar de rosas...

- Instruções complexas são de difícil aplicação
  - O compilador precisa encontrar os casos que casam exatamente com a sequência de alto nível
  - A otimização do código gerado (para minimizar seu tamanho, reduzir o número de instruções e melhorar o uso da *pipeline*) é mais difícil com um conjunto de instruções complexo

CISC

Mas nem tudo é um mar de rosas...

- Instruções complexas são de difícil aplicação
  - O compilador precisa encontrar os casos que casam exatamente com a sequência de alto nível
  - A otimização do código gerado (para minimizar seu tamanho, reduzir o número de instruções e melhorar o uso da *pipeline*) é mais difícil com um conjunto de instruções complexo
- Por conterem mais instruções, são necessários *opcodes* maiores
  - Gerando assim instruções mais longas

CISC

Mas nem tudo é um mar de rosas...

- Então programas em CISC podem até conter menos instruções
  - Mas as instruções são maiores, ocupando mais memória
  - Não é garantido que os programas serão menores realmente

CISC

Mas nem tudo é um mar de rosas...

- Então programas em CISC podem até conter menos instruções
  - Mas as instruções são maiores, ocupando mais memória
  - Não é garantido que os programas serão menores realmente
- A existência de um conjunto de instruções maior aumenta a complexidade da unidade de controle
  - Aumentando o tempo de execução das instruções mais simples, que deveriam ser mais rápidas

CISC

Mas nem tudo é um mar de rosas...

- Até faz sentido que operações complexas executem mais rápido como uma única instrução
  - Mas para isso elas precisam ser usadas de fato

CISC

Mas nem tudo é um mar de rosas...

- Até faz sentido que operações complexas executem mais rápido como uma única instrução
  - Mas para isso elas precisam ser usadas de fato
  - Vários estudos apontam para a predominância de comandos de atribuição (especialmente do tipo `A` $\leftarrow$ `B`), e condicionais (*if*s e laços)
  - Além da maioria das referências serem feitas a variáveis escalares simples e locais (dentro de sub-rotinas)

CISC

Mas nem tudo é um mar de rosas...

- Até faz sentido que operações complexas executem mais rápido como uma única instrução
  - Mas para isso elas precisam ser usadas de fato
  - Vários estudos apontam para a predominância de comandos de atribuição (especialmente do tipo `A` $\leftarrow$ `B`), e condicionais (*if*s e laços)
  - Além da maioria das referências serem feitas a variáveis escalares simples e locais (dentro de sub-rotinas)
  - Então esse ganho é realmente observado, se as instruções usadas são as relativamente simples?

CISC × RISC

RISC

- Esse cenário fez com que uma nova abordagem fosse proposta
  - Projetar uma arquitetura que dê suporte a linguagens mais simples, em vez de mais complexas

CISC × RISC

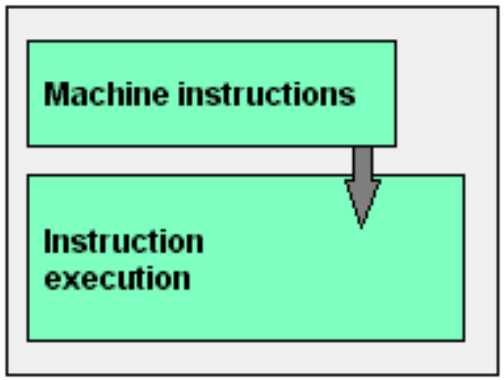
RISC

- Esse cenário fez com que uma nova abordagem fosse proposta
  - Projetar uma arquitetura que dê suporte a linguagens mais simples, em vez de mais complexas
- Nasce assim a arquitetura **RISC**
  - *Reduced Instruction Set Computer*

RISC

Características

- Uma instrução por ciclo de máquina
  - Um ciclo de máquina é o tempo que leva para ler 2 operandos de registradores, executar uma operação na ALU, e armazenar o resultado em um registrador (um estágio da *pipeline*)
  - Com isso, não há necessidade de microcódigo



Fonte: [4]

RISC

Características

- Operações registrador-registrador
  - Apenas operações de `load` e `store` acessam a memória
  - Isso simplifica o conjunto de instruções e a unidade de controle

RISC

Características

- Operações registrador-registrador
  - Apenas operações de `load` e `store` acessam a memória
  - Isso simplifica o conjunto de instruções e a unidade de controle
- Grande número de registradores
  - De modo a otimizar a referência a operadores

RISC

Características

- Operações registrador-registrador
  - Apenas operações de `load` e `store` acessam a memória
  - Isso simplifica o conjunto de instruções e a unidade de controle
- Grande número de registradores
  - De modo a otimizar a referência a operadores
- Modos simples de endereçamento
  - Enfatizam as referências a registradores em vez de memória, exigindo assim menos bits para endereçamento

RISC

Características

- Formatos simples de instrução
  - Apenas uns poucos formatos são usados (em MIPS, tipo-R, tipo-I e tipo-J)
  - O tamanho das instruções é fixo, facilitando assim sua busca
  - As instruções são alinhadas aos limites das palavras, evitando que elas ultrapassem as bordas das páginas
  - Campos em instruções (especialmente o *opcode*) são fixos, fazendo com que a decodificação e acesso a registradores possa ocorrer ao mesmo tempo e simplificando a unidade de controle

RISC

Vantagens

- Aproveita-se do fato de que a maioria das instruções geradas por computadores são simples

RISC

Vantagens

- Aproveita-se do fato de que a maioria das instruções geradas por computadores são simples
- Pela natureza simples e regular do RISC, esquemas eficientes de *pipeline* podem ser empregados
  - Há pouca variação no tempo de execução das instruções, permitindo a adequação da *pipeline* a isso

RISC

Vantagens

- Aproveita-se do fato de que a maioria das instruções geradas por computadores são simples
- Pela natureza simples e regular do RISC, esquemas eficientes de *pipeline* podem ser empregados
  - Há pouca variação no tempo de execução das instruções, permitindo a adequação da *pipeline* a isso
- Permite a aplicação de *delayed branch*
  - Quando rearranjamos as instruções para evitar bolhas

CISC $\times$ RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

`mult` e_1`,` e_2

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

`mult` é traduzida em microcódigo antes de rodar

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

No caso de CISC, não há reaproveitamento de registradores

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

Até porque não sabemos em quais registradores estão os operandos

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

Ou então porque sempre os mesmos são usados, e não há como garantir que o valor continuará lá

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

Então se um dos operandos for usado em outro cálculo, terá de ser recarregado

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

- Ex: `x = x * y`
  - Com `x` e `y` nos endereços de memória e_1 e e_2, respectivamente

CISC

```
mult e1, e2
```

No caso de RISC, basta reutilizar o registrador

RISC

```
lw $t1, 0(e1)
lw $t2, 0(e2)
mul $t3, $t1, $t2
sw $t3, 0(e1)
```

CISC × RISC

Comparação

CISC	RISC
Inclui instruções com múltiplos ciclos	Instruções de ciclo único
Memória-para-memória	Registrador-para-registrador
`load` e `store` incorporadas às instruções	`load` e `store` como instruções independentes
Códigos pequenos	Códigos grandes
CPI alta	CPI baixa
Precisa de memória para armazenar instruções complexas	Precisa de mais registradores

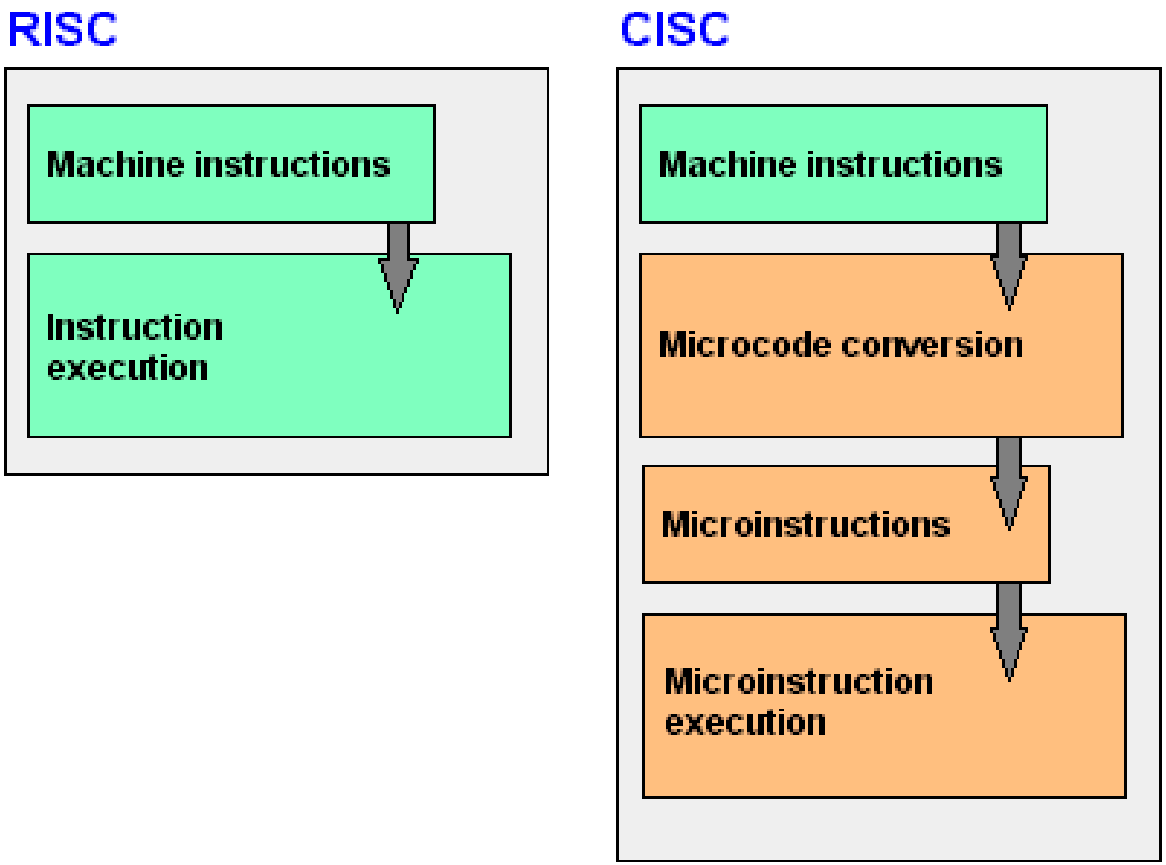
CISC × RISC

Comparação

CISC	RISC
Ou restringem interrupções aos limites das instruções, ou definem pontos específicos em que podem ocorrer, com algum mecanismo para reinício da instrução	Responde melhor a interrupções, pois estas são checadas entre operações mais elementares
i386, i486 (família Intel x86)	SUN Sparc, MIPS R2000 - R5000

CISC × RISC

Comparação



Fonte: [4]

CISC $\times$ RISC

Comparação

$$t_P = n_P \times CPI_P \times t_c$$

CISC	**RISC**
Reduz o número de instruções (n_P) de um programa, sacrificando sua CPI	Reduz a CPI do programa, às custas de seu número de instruções

CISC × RISC

E qual é a melhor?

- Difícil medir
  - Não há 2 máquinas RISC e CISC comparáveis em termos de custo, tecnologia, complexidade, suporte de S.O. ou sofisticação do compilador
  - Não há *benchmark* definitivo, e o desempenho depende do programa
  - É difícil separar efeitos devidos ao *hardware* dos devidos ao compilador

CISC × RISC

A situação hoje

- Muito da controvérsia já desapareceu, dando lugar a uma convergência de tecnologias
  - Com o aumento da densidade dos chips e velocidade do *hardware*, sistemas RISC ficaram mais complexos
  - Ao mesmo tempo, arquiteturas CISC começaram a dar mais ênfase ao projeto da *pipeline* e ao uso de registradores gerais
    - Fornecendo instruções registrador-para-registrador, além das memória-para-memória

CISC × RISC

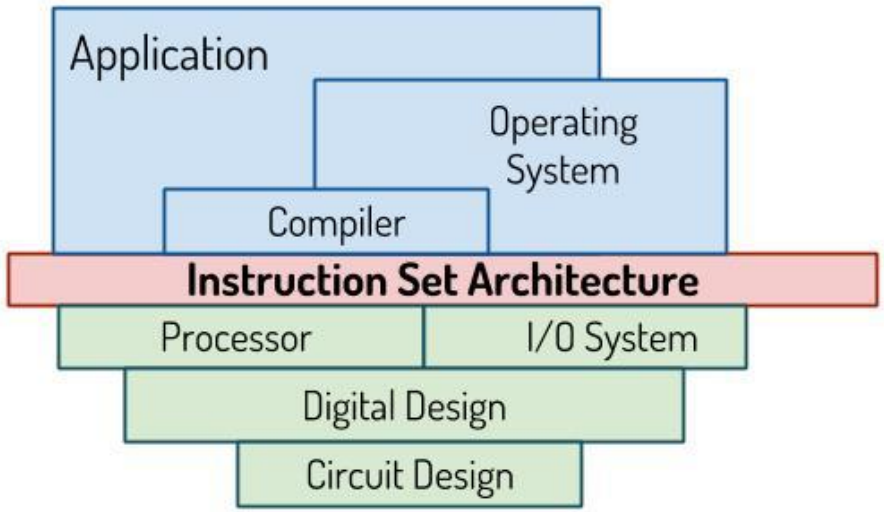
A situação hoje

- Como resultado, RISCs recentes, como o PowerPC, não são puramente RISC
  - E CISCs recentes como Pentium II e posteriores, incorporam algumas características RISC

CISC × RISC

ISA e Microarquitetura

- CISC e RISC são tipos de **ISA**
  - *Instruction Set Architecture*
  - O conjunto de instruções, conforme apresentado ao programador
  - Ex: MIPS, x86

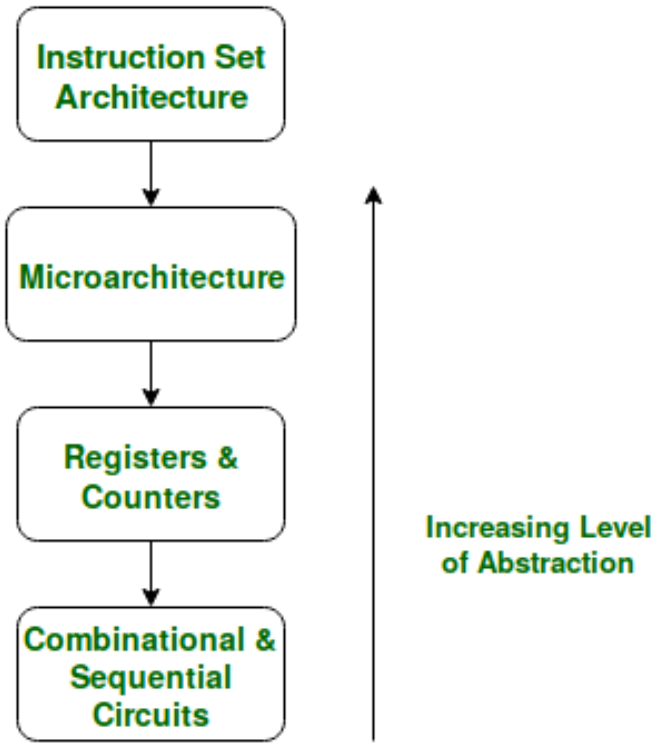


Fonte: `http://accs.magnumdev.webfactional.com/shakti-an-open-source-processor-ecosystem/3/`

CISC × RISC

ISA e Microarquitetura

- A implementação desse conjunto no circuito é conhecida como **microarquitetura**
  - Uma mesma ISA pode ser implementada por diferentes microarquiteturas

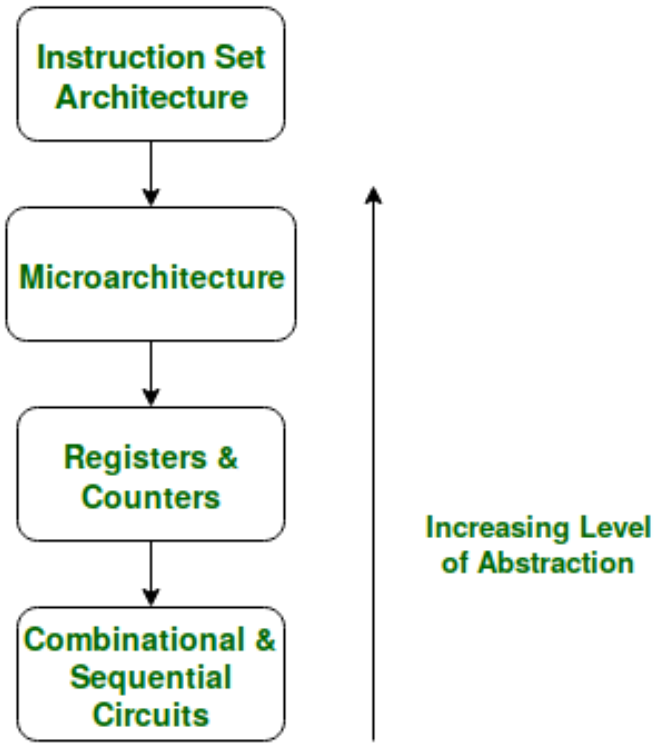


Fonte: [6]

CISC × RISC

ISA e Microarquitetura

- A arquitetura de um computador é então a combinação de sua microarquitetura e sua ISA

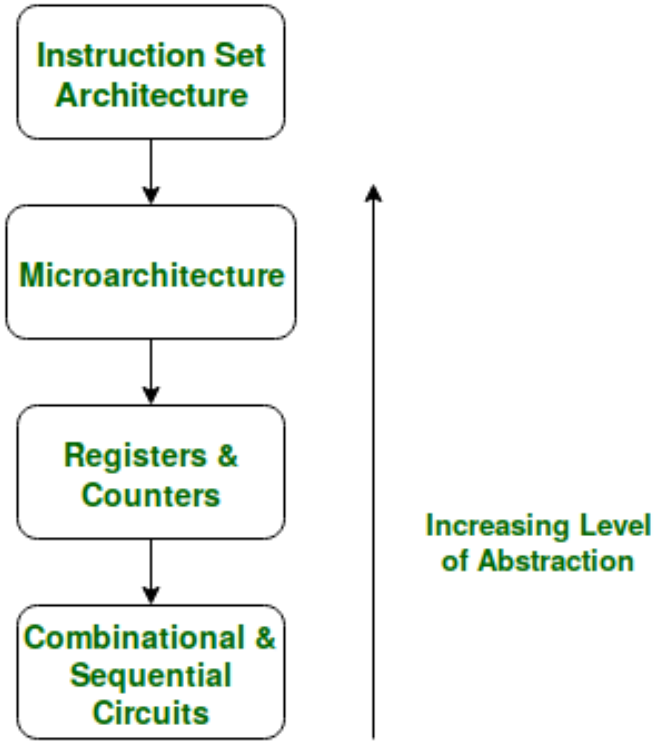


Fonte: [6]

CISC × RISC

ISA e Microarquitetura

- A separação ISA/Microarquitetura faz com que software escrito para uma ISA possa rodar em todas suas implementações
  - Ex: Core 2 Duo e Athlon possuem diferentes microarquiteturas, embora implementem praticamente a mesma versão do x86



Fonte: [6]

Referências

1. Stallings, W (2010): Computer Organization and Architecture: Designing for Performance. Prentice Hall. 8ª ed.
2. https://fractalide.com/blog/2018-11-23-the-semantic-gap.html
3. https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/
4. PC Magazine Encyclopedia
   - https://www.pcmag.com/encyclopedia/term/46918/microcode
   - https://www.pcmag.com/encyclopedia/term/50548/risc
   - https://www.pcmag.com/encyclopedia/term/46940/microinstruction

Referências

5. Wikipedia
   - https://en.wikipedia.org/wiki/Microarchitecture
   - https://en.wikipedia.org/wiki/Instruction_set_architecture
6. https://www.geeksforgeeks.org/microarchitecture-and-instruction-set-architecture/