# Princípios de Funcionamento de Computadores II

Prof. Gustavo Girão

# Roteiro

- Por trás de um programa
  - Etapas necessárias para a tradução em linguagem de máquina e execução do programa
- Elementos de Processamento
  - Microprocessador x Microcontrolador
  - Outros elementos de processamento
    - ASIC, ASIP, FPGA

# Por trás do Programa

# Necessidade da tradução

- O computador apenas compreende sinais elétricos - ligado/desligado (0V/5V)
- Esses sinais foram traduzidos para uma linguagem simbólica com 0 e 1
- A partir dessa tradução, a base binária passou a ser usada pelos computadores
- Muito difícil de ser compreendida e manipulada por humanos
  - Palavras são mais fáceis de lembrar do que sequências de zeros e uns

# Necessidade da tradução

- Surgiu então o *Assembly* ou linguagem de montagem

```
L4:
        mult $t0, $t1
        mflo $s0
        addi $s0, $s0, 1
        bne $s0, $zero, L4
```

**O que esse trecho de programa faz?**

# Necessidade da tradução

```
L4:
        mult $t0, $t1
        mflo $s0
        addi $s0, $s0, 1
        bne $s0, $zero, L4
```

Multiplica o conteúdo dos registradores $t0 e $t1 e coloca o resultado nos registradores Hi e Lo

Copia o conteúdo do registrador Lo no registrador $s0

Soma 1 ao conteúdo do registrador $s0 e armazena em $s0

Compara se o valor do registrador $s0 é diferente do valor do registrador $zero. Se sim, volta para a linha de código com o *label* L4

# Necessidade de tradução

- Programar em assembly ainda era difícil
- Programadores precisavam que os computadores fossem capazes de falar línguas ainda mais parecidas com linguagens humanas
- A partir daí, surgiram as linguagens de programação
  - Cada uma, seguindo um conjunto de regras sintáticas e semânticas usadas para definir os programas

# Traduzindo um Programa para ser executado

- Computadores são construídos considerando dois princípios básicos
  1. Instruções são representadas como números
  2. Programas são armazenados em memória para ser lidos ou escritos, como números

- Esses princípios compõem o conceito de programa armazenado (*stored-program*)

- Existem 4 passos principais para transformar um programa implementado em linguagem de programação para instruções que serão executadas pelo processador

- Alguns sistemas combinam esses passos para reduzir o tempo de tradução

# Traduzindo um Pr

Compatibilidade binária
Por que?

- Computadores sã
básicos
  1. instruções são representadas como números
  2. Programas são armazenados em memória para ser lidos ou escritos, como números

- Esses princípios compõem o conceito de programa armazenado (*stored-program*)

- Existem 4 passos principais para transformar um programa implementado em linguagem de programação para instruções que serão executadas pelo processador

- Alguns sistemas combinam esses passos para reduzir o tempo de tradução

# Passos de tradução de um programa

- Compilador
  - Compilador transforma um programa implementado em linguagem de programação em um programa em linguagem *assembly*
  - O Assembly é dependente do conjunto de instruções do processador que executará o programa

- Assembler
  - Traduz um programa assembly em um código de máquina
    - Assembly ainda é uma linguagem intermediária, mais próxima da máquina, porém, intermediária
  - Transforma um programa assembly em um arquivo objeto (*object file*): instruções em linguagem de máquina + dados + informações para posicionar as instruções na memória

# Passos de tradução de um programa

- Co
  - li
    linguag
  - O Assembly é e do conjunto de instruções do processad ecutará o programa

- Assem er
  - Traduz um programa assembly em um código de máquina
    - ✧ Assembly ainda é uma linguagem intermediária, mais próxima da máquina, porém, intermediária
  - Transforma um programa assembly em um arquivo objeto (*object file*): instruções em linguagem de máquina + dados + informações para posicionar as instruções na memória

- Pseudo instruções:
(Assembly-MIPS) move, bge, ble, etc
- Referência a registradores: $zero
- Uso de diferentes bases numéricas

# Passos de tradução de um programa

- Compi
  - 
  - O As
    processador

- Assembler
  - Traduz  ograma assembly em um código de máquina
    - A bly ainda é uma linguagem intermediária, mais próxima  máquina, porém, intermediária
  - Transforma um programa assembly em um arquivo objeto (*object file*): instruções em linguagem de máquina + dados + informações para posicionar as instruções na memória

Programas UNIX
- *Object file header*
- *Text segment*
- *Static data segment*
- *Realocation information*
- *Symbol table*
- *Debbuging information*

# Passos de tradução de um programa

- Linker ou Link Editor
  - Necessidade de evitar as etapas de compilação e *assembling* **sempre** no programa todo
  - Assim, cada procedimento é compilado e *assembled* **independentemente**
  - O Linker é responsável por pegar todos os procedimentos independentes traduzidos para linguagem de máquina e **juntá-los**.
    1. Posiciona os módulos de código e dados simbolicamente na memória
    2. Determina o endereço dos *labels* de instruções e dados
    3. Realiza todas as referências internas e externas
  - Gera um código executável (mesmo formato do arquivo objeto, sem referências não resolvidas)

# Passos de tradução de um programa

- Linker ou Link Edito
  - Necessidade de e
    **sempre** no programe
  - Assim, cada procedime
    compilado e *assembled*
    **independentemente**
  - O Linker é responsável por pegar todos os procedimentos independentes traduzidos para linguagem de máquina e **juntá-los**.
    1. Posiciona os módulos de código e dados simbolicamente na memória
    2. Determina o endereço dos *labels* de instruções e dados
    3. Realiza todas as referências internas e externas
  - Gera um código executável (mesmo formato do arquivo objeto, sem referências não resolvidas)

Exemplo: procedimentos para ler, escrever são sempre os mesmos...

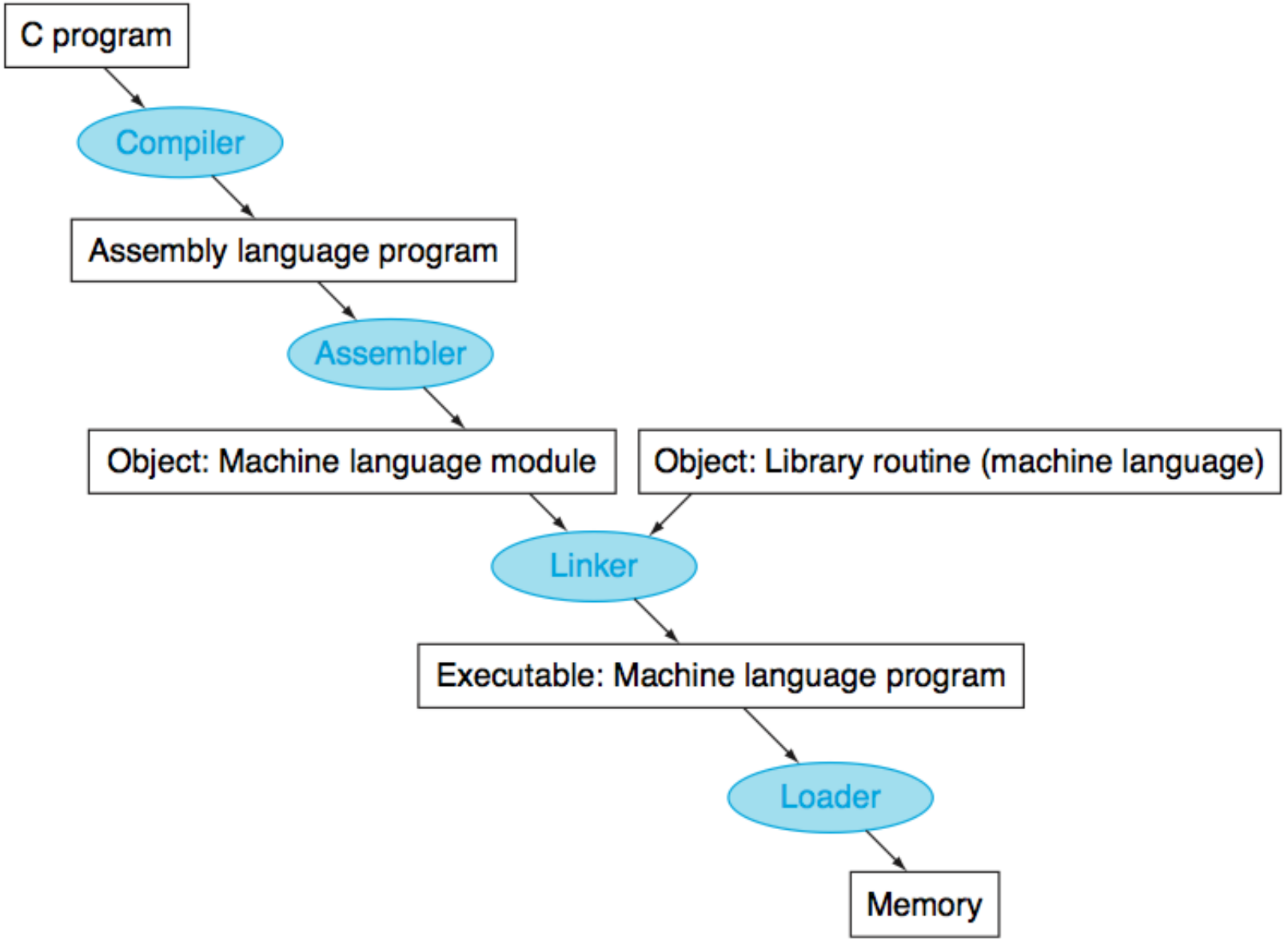
# Passos de t

**Se os procedimentos são compilados e *assembled* separadamente, não é possível fazer a conexão entre os diferentes procedimentos na etapa de assembly**

- Linker ou Link Editor
  - Necessidade de evitar as e compilação e *assembling* **sempre** no programa todo
  - Assim, cada procedimento é compilado e *assembled* **independentemente**
  - O Linker é responsável por pegar todos os procedimentos independentes traduzidos para linguagem de máquina e **juntá-los**.
    1. Posiciona os módulos de código e dados simbolicamente na memória
    2. Determina o endereço dos *labels* de instruções e dados
    3. Realiza todas as referências internas e externas
  - Gera um código executável (mesmo formato do arquivo objeto, sem referências não resolvidas)

# Passos de tradução de um programa

- Loader
  - Carrega o arquivo objeto na memória principal para prepará-lo para a execução
  - Loader no UNIX
    - **Lê o cabeçalho** do arquivo executável e determina o tamanho do segmento de textos e de dados
    - **Aloca um espaço de endereçamento** grande o suficiente para caber os segmentos
    - **Copia as instruções e os dados** do arquivo executável para a memória
    - **Copia os parâmetros (se existirem)** para o programa principal na pilha
    - **Inicializa os registradores e posiciona ponteiros** em posições de memória
    - **Salta para a rotina de início e chama a rotina principal do programa**. Quando a rotina principal retorna, a rotina de início termina o programa com uma chamada de sistema *exit*


C program
Compiler
Assembly language program
Assembler
Object: Machine language module
Object: Library routine (machine language)
Linker
Executable: Machine language program
Loader
Memory

# Elementos de Processamento

Microcontrolador x Microprocessador

# O QUE ELES TÊM EM COMUM?

# Propósito Geral – GPP (*General Purpose Processor*)

- Processador projetado para uma variedade de tarefas
- Projeto cuidadoso com alto custo de projeto
- Apesar disso, possui baixo custo por unidade por dividir o custo de projeto em uma grande quantidade de unidades
- Programado exclusivamente via SW

# Propósito Geral – GPP (*General Purpose Processor*)

- Vantagem
  - Flexibilidade
- Desvantagem
  - Desempenho
  - Dissipação de potência/calor
  - Custo
  - Área

# Operações

- Do caminho de dados (Parte operativa)
  - Load => operações de ULA => store
- Do controle
  - Quebra da instrução em sub-instruções
  - IF => ID => EX => MEM => WB

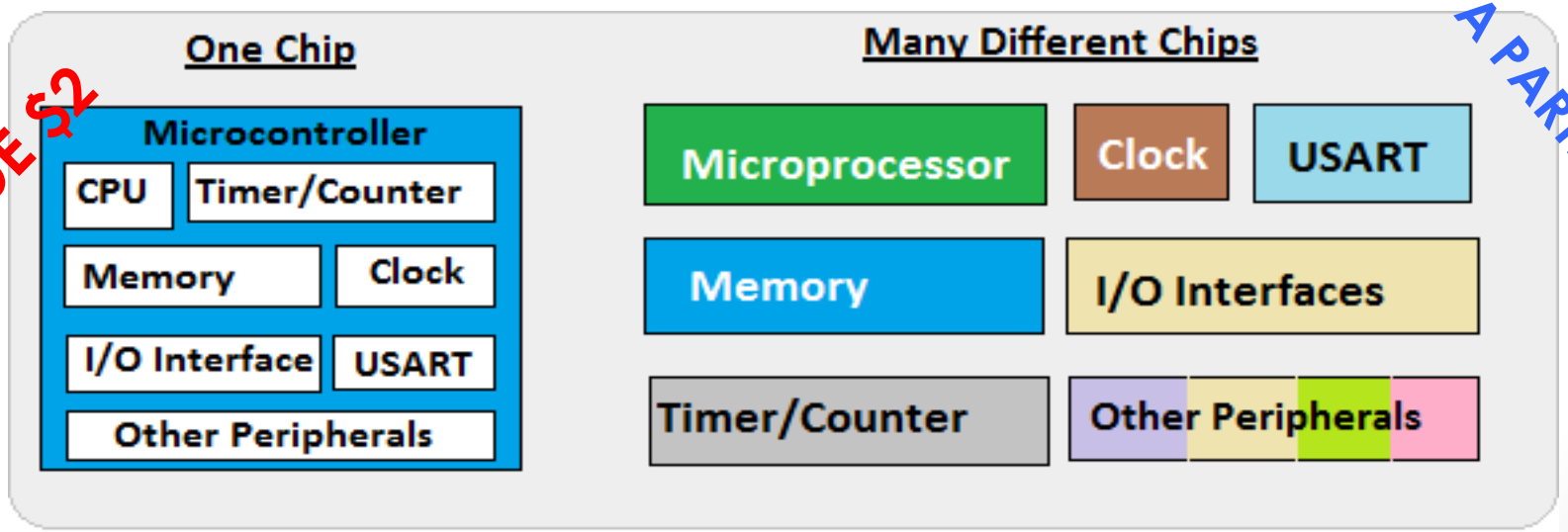
Microcontrolador x Microprocessador

# NO QUE ELES DIFEREM?

# Diferenças

- Microcontrolador
  - Computador em um chip
    - CPU
    - E/S
    - Memória
    - Clock
    - Periféricos
  - Único (poucos) programa

- Microprocessador
  - "Somente" CPU no chip
  - Componentes externos
    - Memória
    - Clock
    - E/S
    - Periféricos
  - Vários programas




http://www.avr-tutorials.com/general/comparison-between-microcontroller-and-microprocessor

# ASIC, ASIP, E FPGA

- ASIC: APPLICATION SPECIFIC INTEGRATED CIRCUIT (***Circuito Integrado de Aplicação Específica***)

- ASIP: APPLICATION SPECIFIC INSTRUCTION SET PROCESSOR (***Processador com Conjunto de Instruções para Aplicação Específica***)

- FPGA: FIELD PROGRAMMABLE GATE ARRAYS (***Arranjos de portas programáveis por campo***)

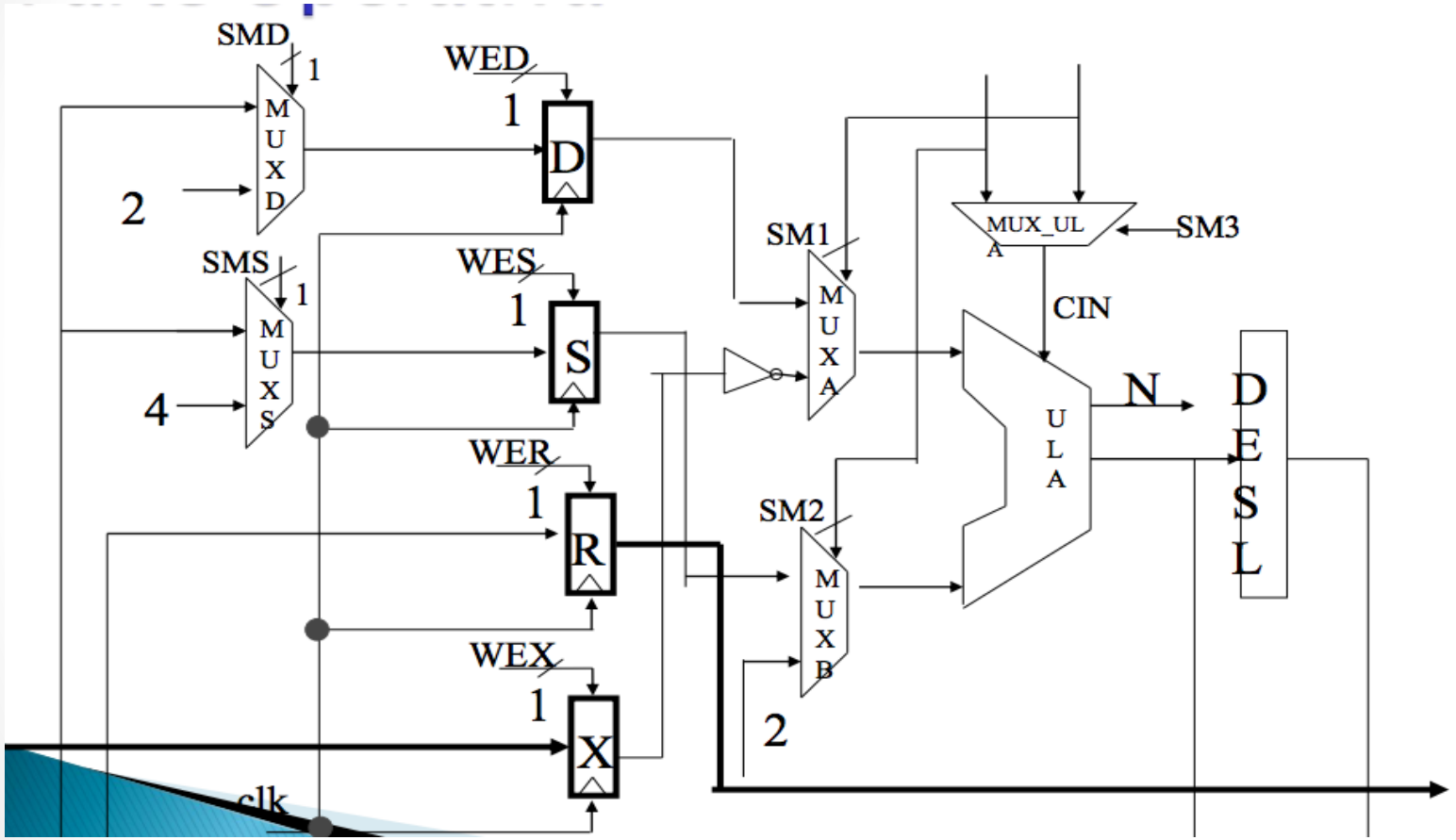
# *Application Specific Integrated Circuit* ASIC

- Circuito integrado projetado especificamente para uma aplicação ou propósito especial
- Exemplo: MP3 player, Módulo raiz quadrada
- Vantagem
  - Desempenho
  - Dissipação de potência
- Desvantagem
  - Flexibilidade
  - Extensibilidade

# Exemplo – Raiz Quadrada

```
d <- 2;
s <- 4;
Fazer
d <- d + 2;
r <- d / 2;
s <- s + d + 1;
Enquanto (s <= x);
```

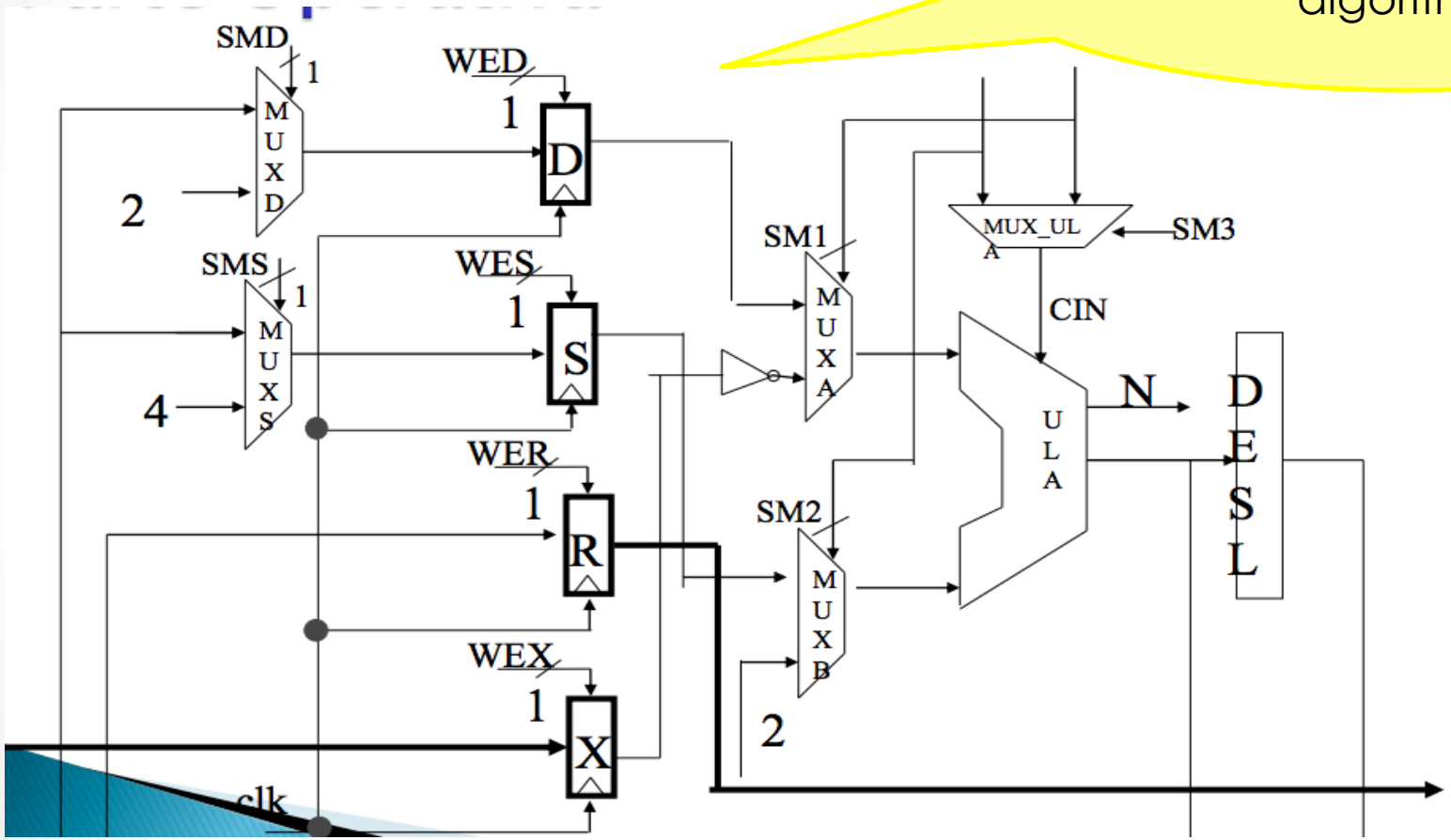
# Exemplo – Raiz Quadrada




http://www2.ufersa.edu.br/portal/view/uploads/setores/145/arquivos/arq/trabalhos/vhdl_epoca.pdf

# Exemplo – Raiz Quadra

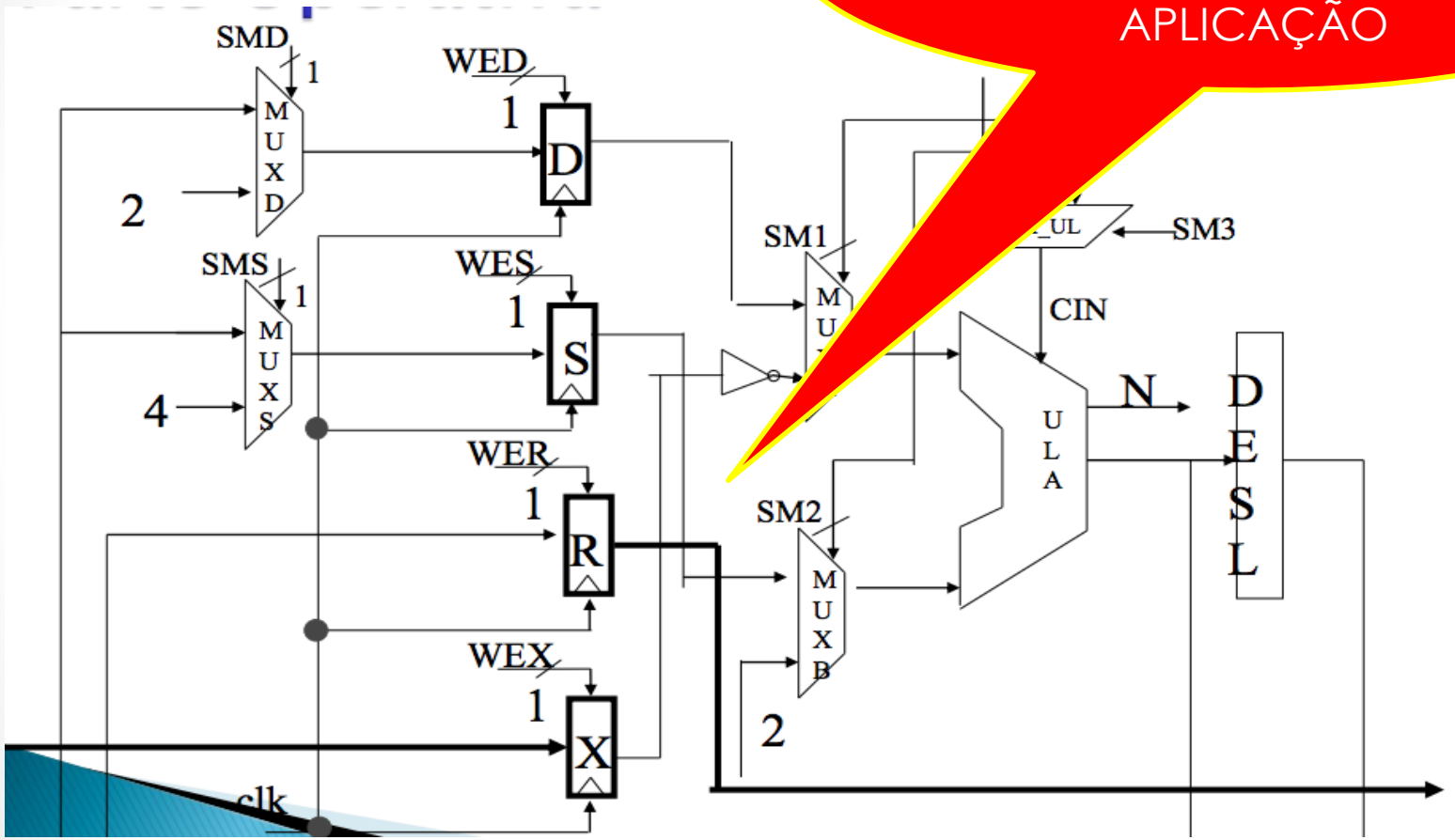
Hardware projetado para executar somente esse algoritmo.




http://www2.ufersa.edu.br/portal/view/uploads/setores/145/arquivos/arq/trabalhos/vhdl_epoca.pdf

# Exemplo – Raiz Qua

NÃO PRECISA SER PROGRAMADO POIS NÃO EXECUTARÁ OUTRA APLICAÇÃO




http://www2.ufersa.edu.br/portal/view/uploads/setores/145/arquivos/arq/trabalhos/vhdl_epoca.pdf

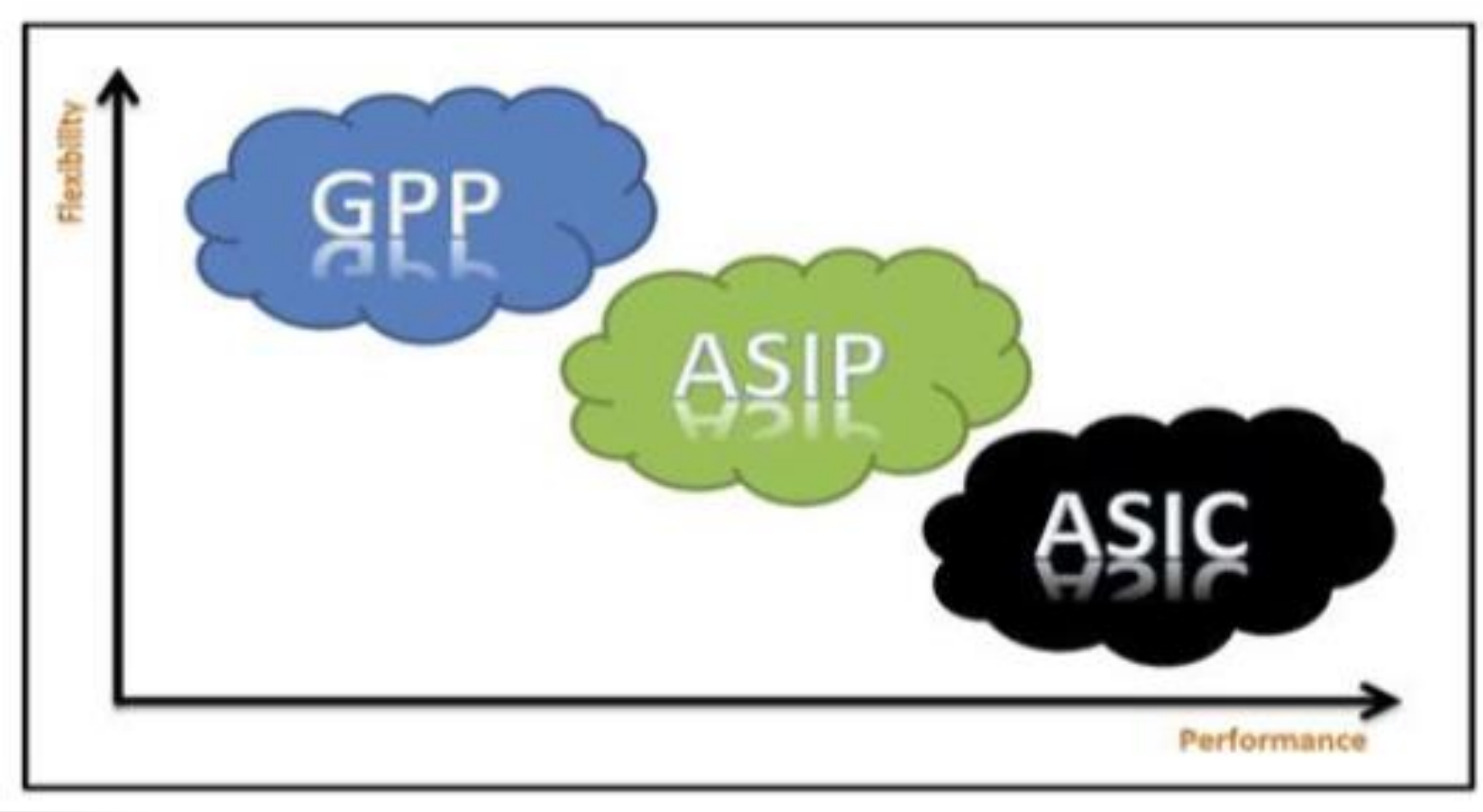
# *Application Specific Instruction Set Processor* ASIP

- Projeto do hardware mais dedicado do que um GPP
- Arquitetura **programada por SW** projetada para executar algumas tarefas mais eficientemente
- Conjunto de instruções específico a um domínio de aplicações

# *Application Specific Instruction Set Processor*
# ASIP

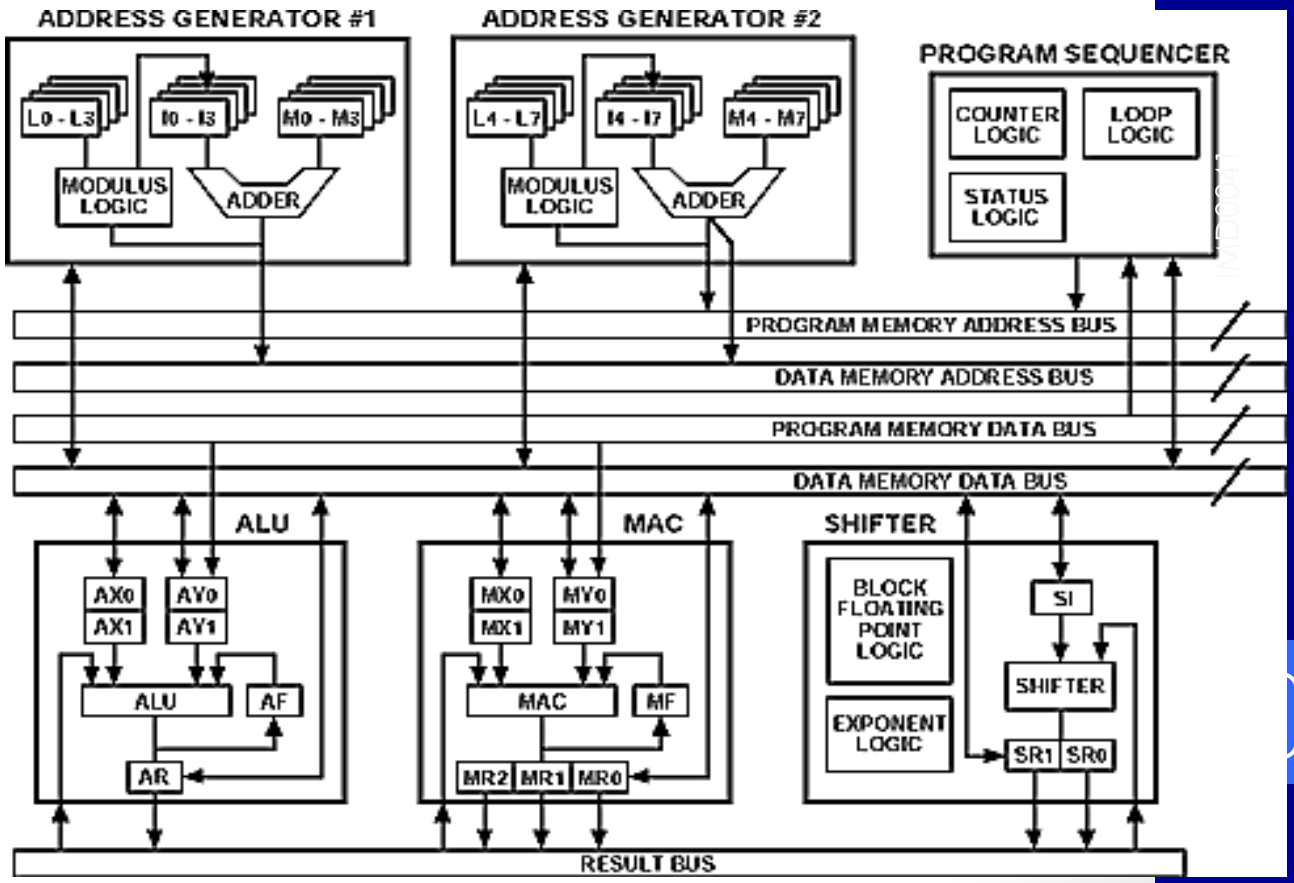
- Meio termo entre GPP e ASIC
- Um pouco de flexibilidade
- Melhor desempenho que GPP

- Fatores
  - Desempenho
  - Custo
  - Potência
  - Simplificação do projeto


Flexibility
GPP
ASIP
ASIC
Performance

# Exemplo - DSPs

- Realizam cálculos matemáticos complexos, garantindo tempo real
- Devem realizar várias operações aritméticas em um ciclo
  - Multiplica/acumula
  - Soma; subtração
  - AND/OR
  - Deslocamentos bit a bit



# Exemplo – DSPs/Fluxo de programa

- Unidade de ponto flutuante integrada diretamente no caminho de dados

- Unidade MACs paralelas
  - MAC, usado em operações com matrizes, convolução em filtros, produtos e polinomiais

# *Field Programmable Gate Arrays*
# FPGAs

- Também chamadas de **ARQUITETURAS RECOFIGURÁVEIS**
- Em uma arquitetura reconfigurável, é possível **mudar** o **comportamento** do **hardware** através da mudança da função lógica que a arquitetura realiza
- Paradigma completamente diferente de uma Porta Lógica de um Circuito Lógico (DISCIPLINA CIRCUITOS LÓGICOS)

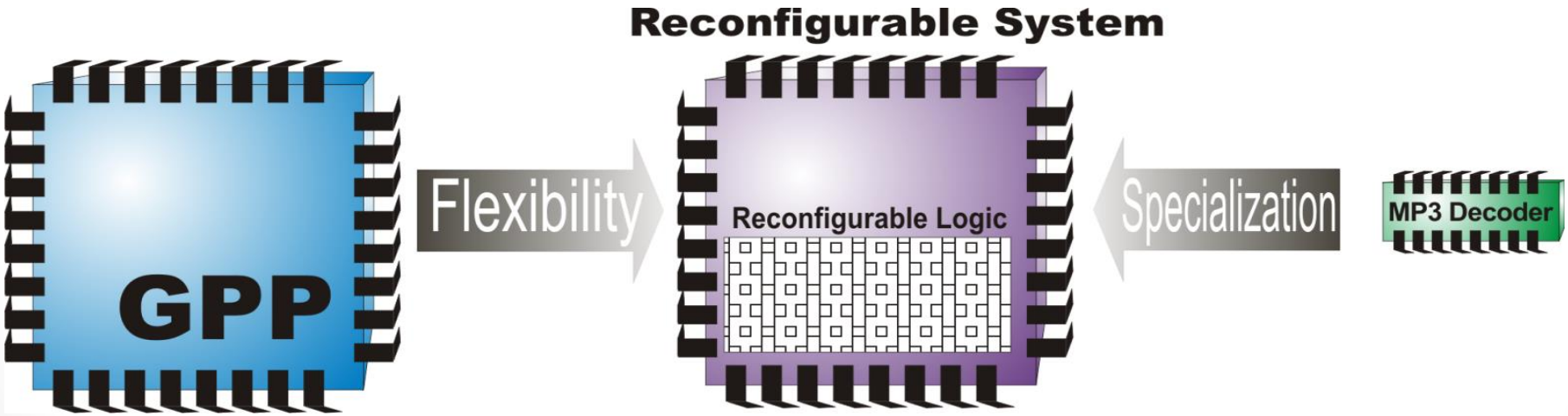
# *Field Programmable Gate Arrays*
# FPGAs

- Também chamadas de **ARQUITETURAS RECOFIGURÁVEIS**
- Em uma arquitetura reconfigurável, é possível **mudar** o **comportamento** do **hardware** através da mudança da função lógica que a arquitetura realiza
- Paradigma ... ma Porta Lógica ... UITOS LÓGIC...

Em outras palavras:
RECONFIGURAR O HARDWARE
PARA DESEMPENHAR UMA
FUNÇÃO DIFERENTE

# Sistemas Reconfiguráveis

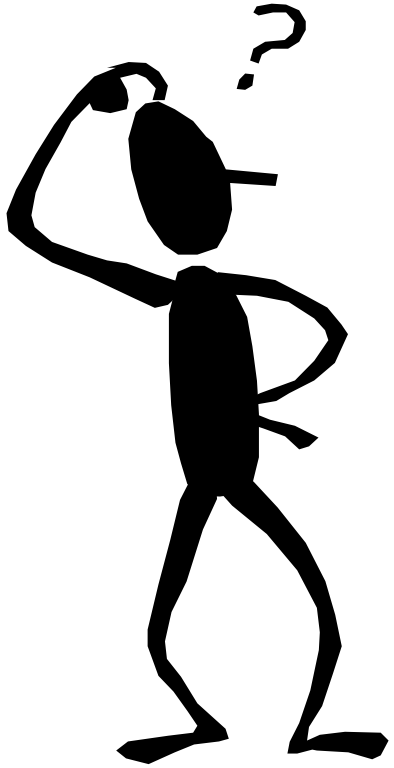
- Especializado X Propósito geral
  - ASIC X GPP



# Arquiteturas Reconfiguráveis

- Vantagens
  - Maior flexibilidade (em comparação a ASICs)
  - Melhor desempenho (em comparação aos GPPs)

- Desvantagens
  - Tempo de configuração
  - Memória de configuração
  - Potência alta para realizar a configuração



# Para saber mais ...

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

- STALLINGS, William. Arquitetura e organização de computadores. 8. ed. São Paulo: Pearson, 2010. **Capítulo 2**

# Próxima aula

- Panorama das Tecnologias Atuais