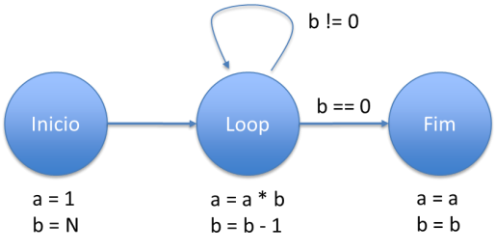
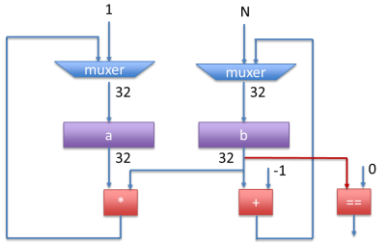
Performance em Sistemas Ciberfísicos Aula – 03

Alison Luis Lando

Arquitetura de computadores

- Problema -> Procedimento (Máquina finita de estados) -> Implementação

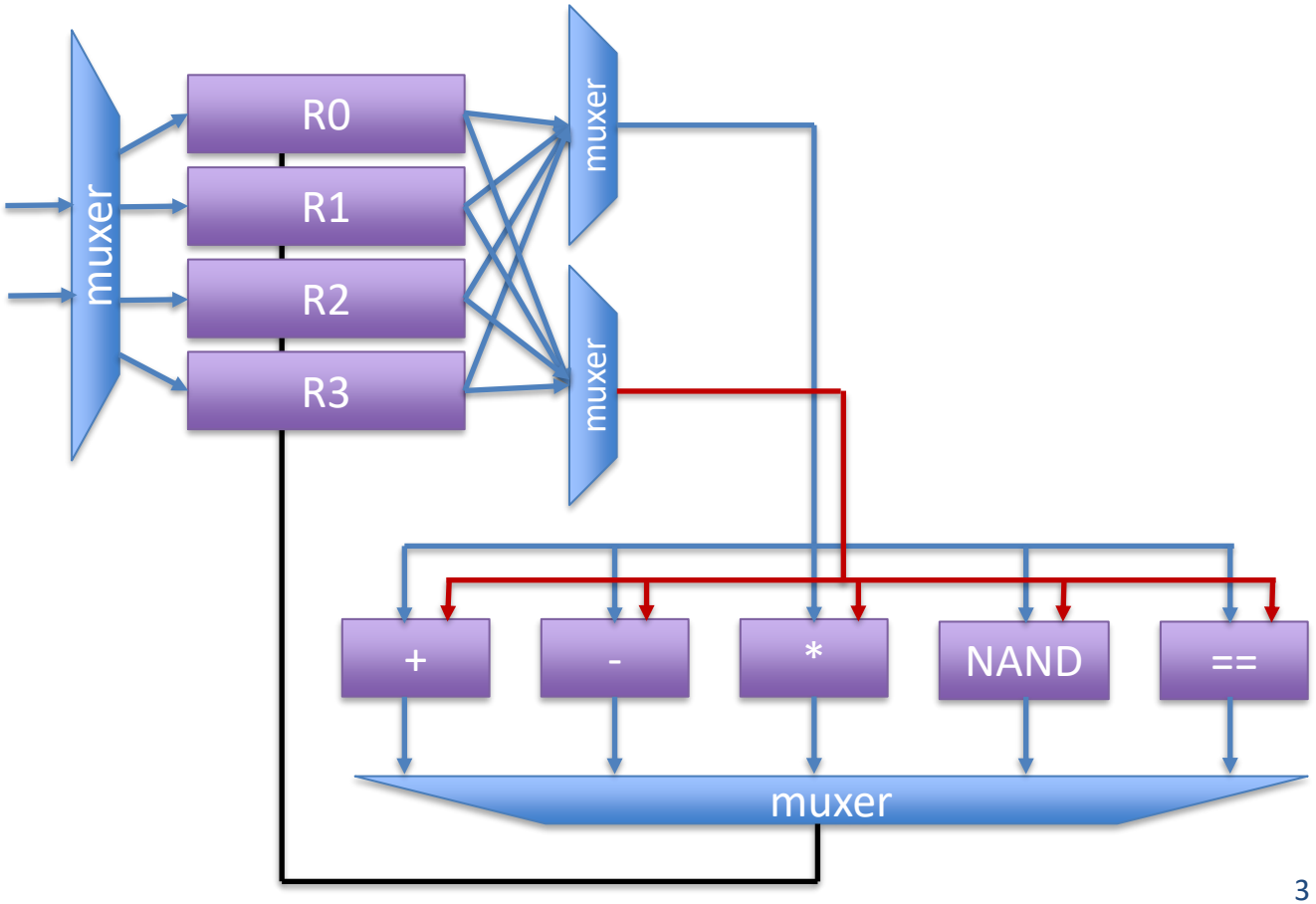




- Hardware de propósito específico!
- Como generalizar nossa abordagem para solucionar mais problemas com o mesmo hardware?
  - Precisamos de mais registradores
  - Precisamos de um repertório maior de operações
  - Hardware generalizável de instruções

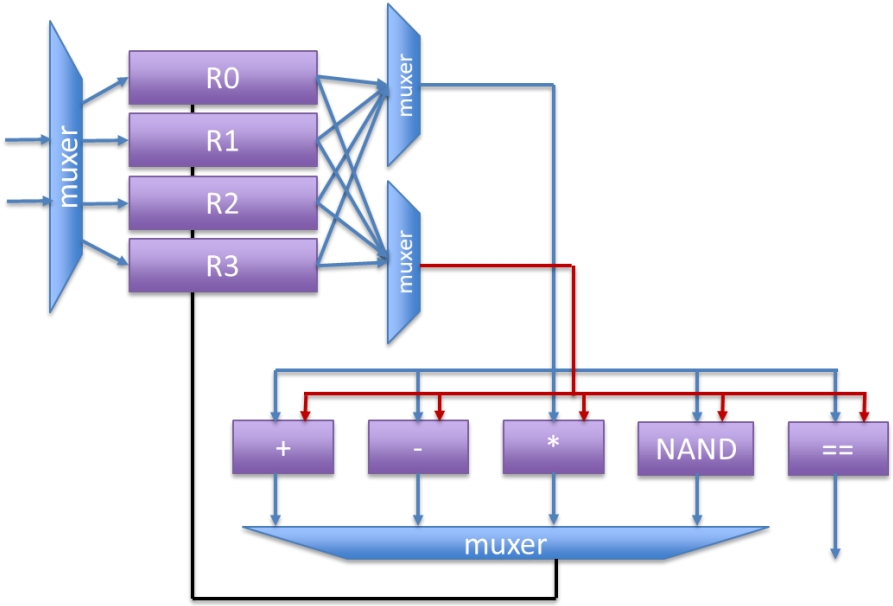
Arquitetura de computadores

- Criar uma máquina finita de estados com operações genéricas e registradores genéricos
  - Divisão, soma, raiz quadrada, …



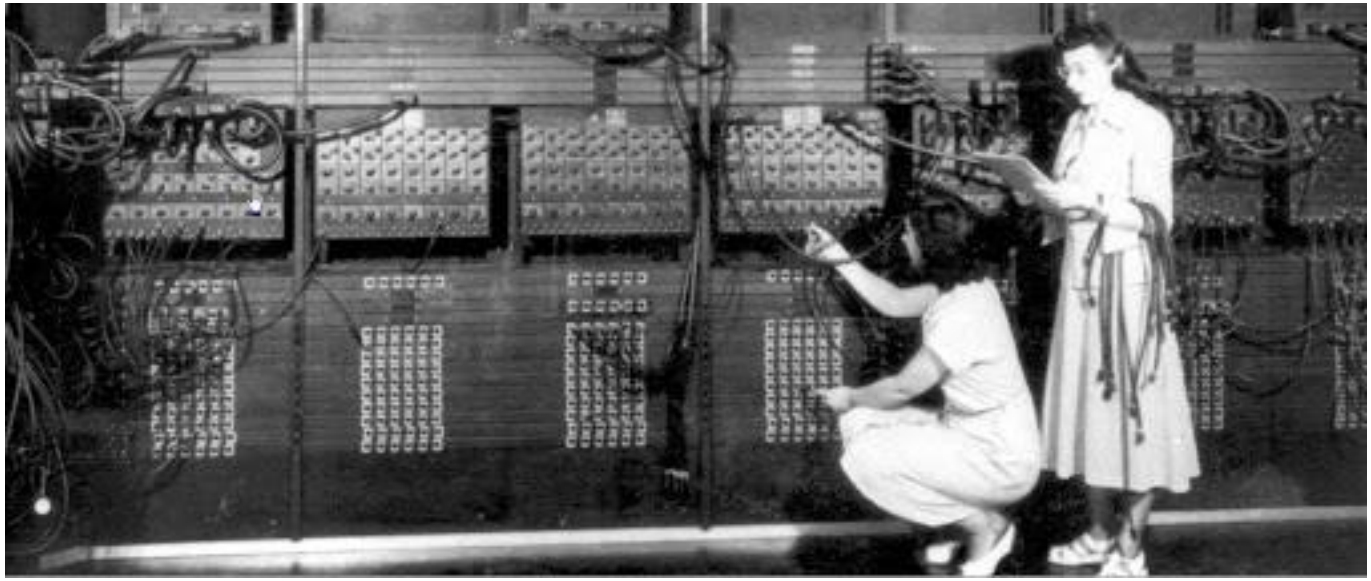
Arquitetura de computadores

- Criar uma máquina finita de estados com operações genéricas e registradores genéricos
  - Divisão, soma, raiz quadrada, …
  - Apenas operações com 4 registradores
- Criamos um hardware programável!



Arquitetura de computadores

- Primeiros computadores digitais eram programados deste modo!
  - ENIAC (1943)
    - Primeiro computador digital de propósito geral
    - Programado através de switches
    - Reprogramação levava ~3 semanas



Arquitetura de computadores

- Primeiro desafio era "como criar um computador de propósito geral eficiente?"
  - Diversas abordagens foram propostas
- Modelo von Neumann – 1943
  - Modelo que quase todos computadores modernos são baseados
  - Componentes

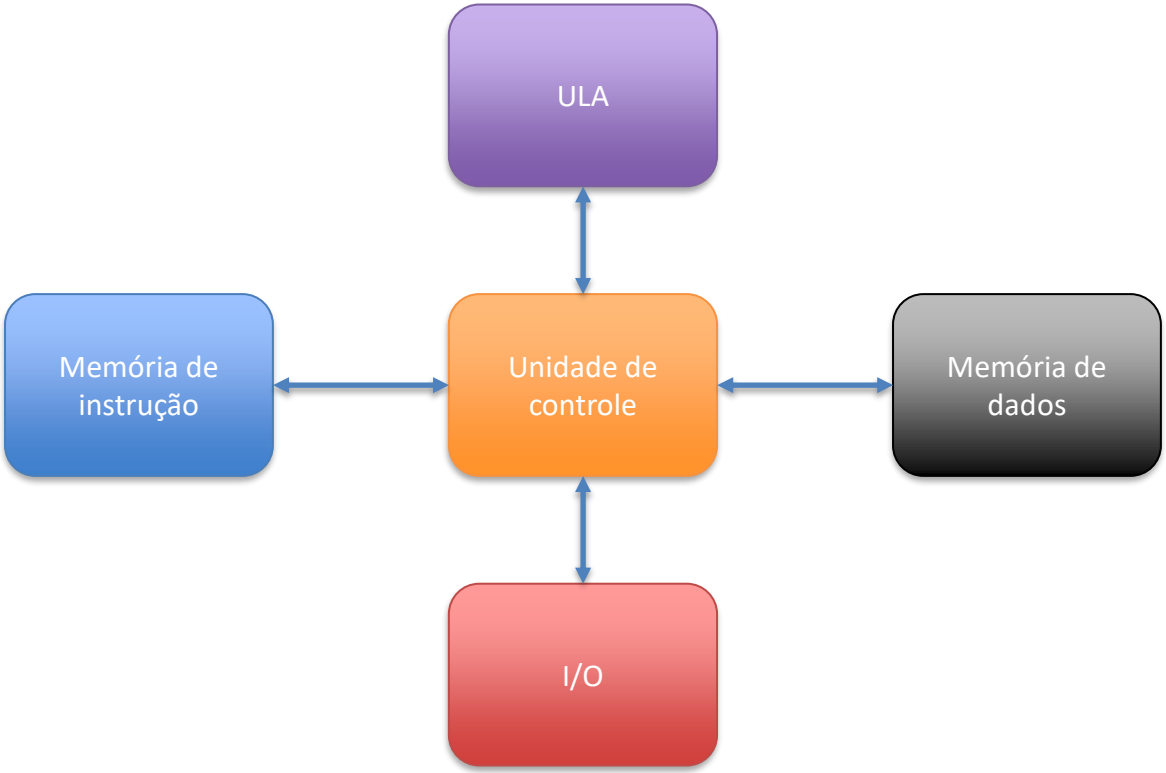
Matriz de Bits

Efetua operações sobre valores de memória

Comunica com o mundo externo

Arquitetura de computadores

- Outro modelo foi proposto denominado Arquitetura Harvard



- Computadores atuais são baseados no modelo von Neumann!

Arquitetura de computadores

- Modelo von Neumann – 1943
  - Modelo que quase todos computadores modernos são baseados
  - Componentes

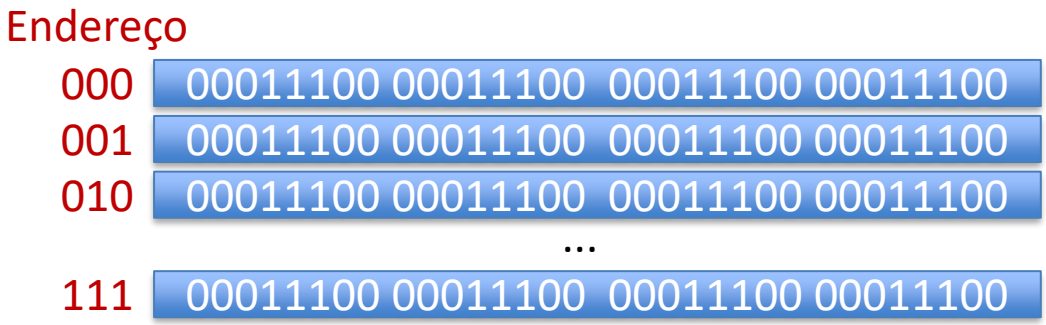
Matriz de Bits

Efetua operações sobre valores de memória

Comunica com o mundo externo

Arquitetura de computadores

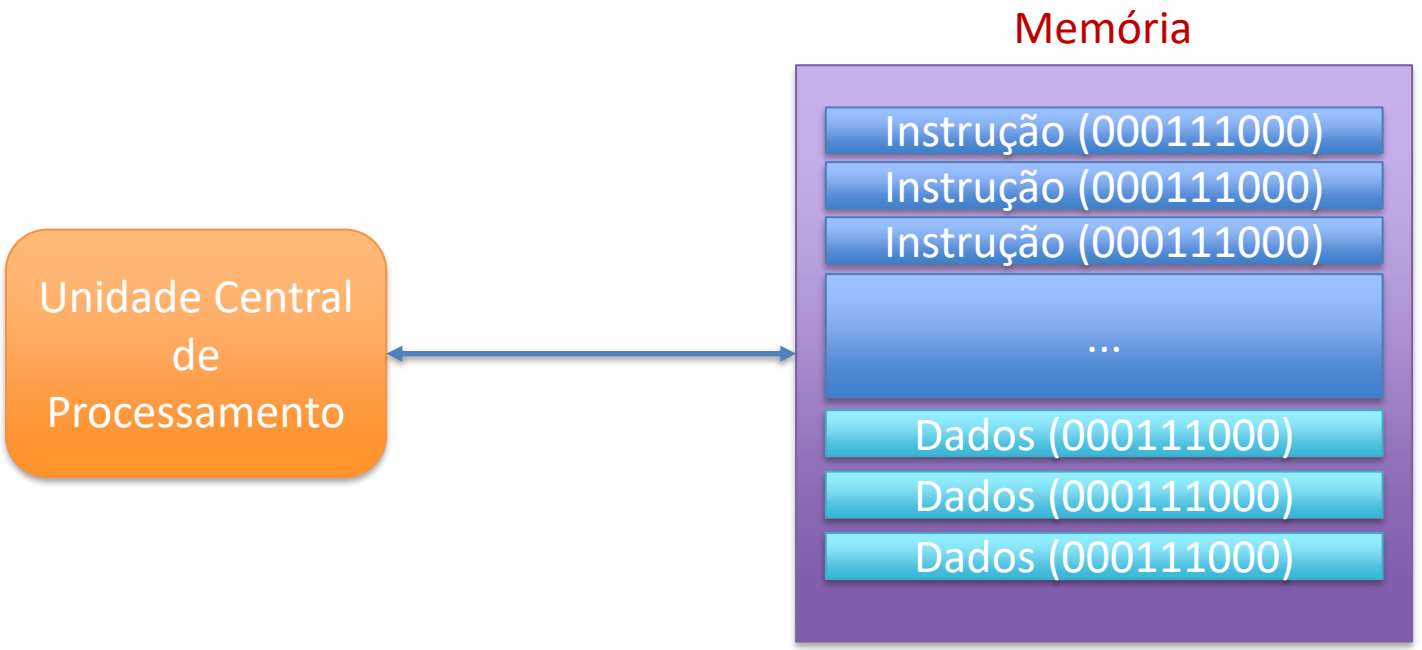
- Memória principal
  - *Random Access Memory (RAM)*
  - Matriz de bits estruturados em W palavras de N bits cada
  - Exemplo: W = 8 (k = 3), N = 32 bits



- Permite escrever e ler palavras
  - Dado um endereço retorna uma palavra
  - Dado um endereço e palavra, atualiza a palavra no endereço

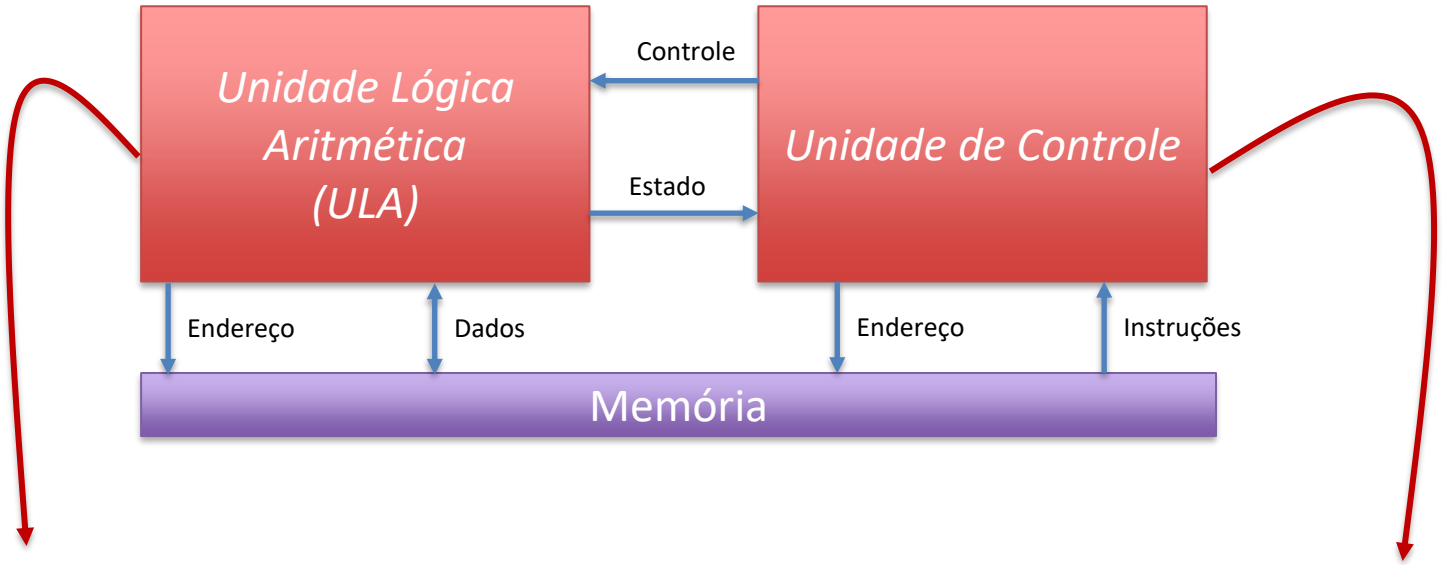
Arquitetura de computadores

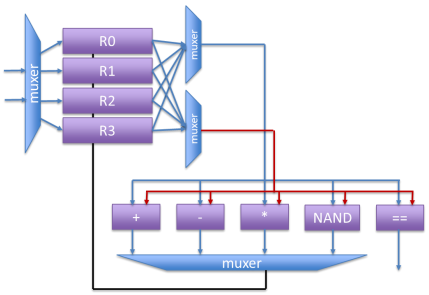
- Idéia central
  - Crie um programa como uma sequência de instruções codificadas
  - Memória armazena dados e instruções
  - CPU coleta, interpreta e executa instruções sucessivas do programa



Arquitetura de computadores

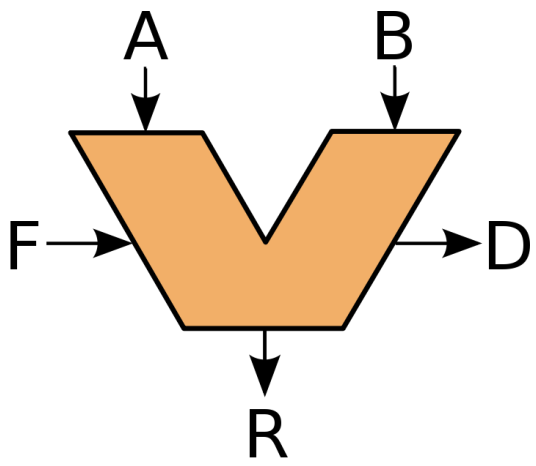
- Arquitetura típica de um computador von Neumann





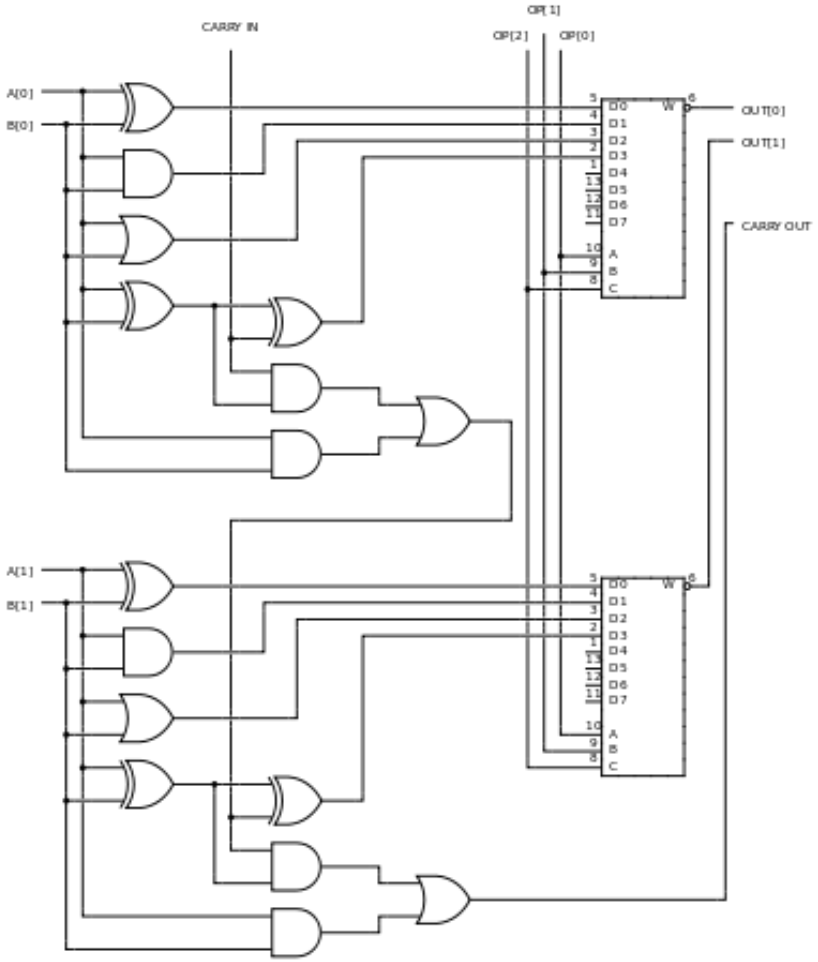
- Instruções codificadas como código binário
- Contador de programa (PC): endereço da instrução a ser executada
- Lógica para traduzir a instrução para sinais a ULA

Unidade lógica e aritmética



Um símbolo esquemático típico para uma ULA, onde "A" e "B" são operandos, "R" é a saída, "F" é a entrada da unidade de controle e "D" é a saída de status

Operações simples
Muitas ULAs podem realizar as seguintes operações:
•operações aritméticas com inteiros;
•operações lógicas bit a bit AND, NOT, OR, XOR;
•operações de deslocamento de *bits* (deslocamento, rotação por um número específico de bits para esquerda ou direita, com ou sem sinal); deslocamentos podem ser interpretados como multiplicações ou divisões por 2



Uma simples ULA de 2-bit que faz AND, OR, XOR, e adição (clique na imagem para explicação)

https://pt.wikipedia.org/wiki/Unidade_l%C3%B3gica_e_aritm%C3%A9tica

Instruções

- Instruções definem a unidade de trabalho
- Cada instrução especifica
  - Uma operação ou opcode a ser efetuado
  - Operandos de origem e destino
- Como assim?

Instruções

- Escrevam um programa em C para calcular o fatorial de N

fatorial(N) = N! = N*(N-1)*…*1

- Podemos executar este código em hardware?

```
C
int a = 1
int b = N
while (b != 0){
        a = a * b
        b = b – 1
}
```

Instruções

- Escrevam um programa em C para calcular o fatorial de N

fatorial(N) = N! = N*(N-1)*…*1

```
C
int a = 1
int b = N
while (b != 0){
        a = a * b
        b = b – 1
}
```

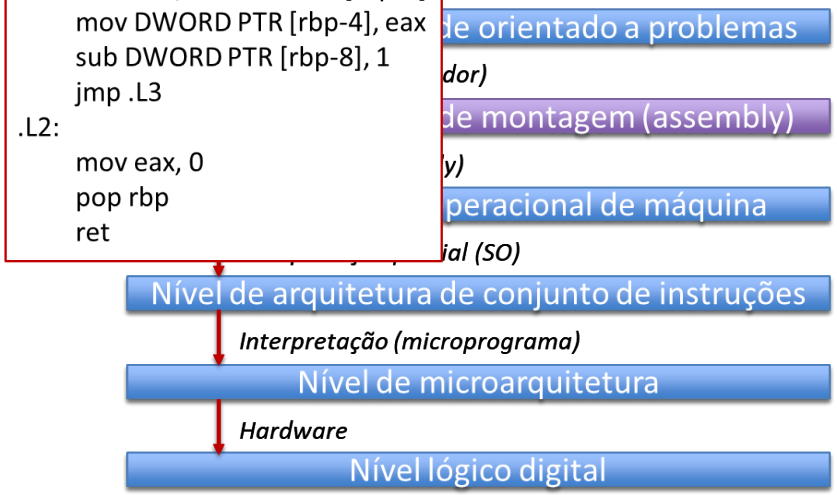
- Podemos executar este código em hardware?
- Compilador traduz o Código para nível *Assembly*

Instruções

- Compilador traduz o Código para nível *Assembly*
  - *Traduzam o Código para Assembly em* https://godbolt.org/

```
C
int main(){
    int a = 1;
    int b = 10;
    while (b != 0){
        a = a * b;
        b = b - 1;
    }
    return 0;
}
```

```
main:
        push rbp
        mov rbp, rsp
        mov DWORD PTR [rbp-4], 1
        mov DWORD PTR [rbp-8], 10
.L3:
        cmp DWORD PTR [rbp-8], 0
        je .L2
        mov eax, DWORD PTR [rbp-4]
        imul eax, DWORD PTR [rbp-8]
        mov DWORD PTR [rbp-4], eax
        sub DWORD PTR [rbp-8], 1
        jmp .L3
.L2:
        mov eax, 0
        pop rbp
        ret
```



Instruções

- Assembly é uma linguagem de baixo nível mais próxima ao hardware
  - *Hardware não executa o assembly!*
  - *Hardware interpreta apenas 0s e 1s!*
- *SO trata especificidades de hardware*
- *Assembly é traduzido para opcode*
- *Hardware executa opcode*
  - http://shell-storm.org/online/Online-Assembler-and-Disassembler/
  - *Assemble (x86)*
  - *Disassemble (x86)*

```
main:
        push rbp
        mov rbp, rsp
        mov DWORD PTR [rbp-4], 1
        mov DWORD PTR [rbp-8], 10
.L3:
        cmp DWORD PTR [rbp-8], 0
        je .L2
        mov eax, DWORD PTR [rbp-4]
        imul eax, DWORD PTR [rbp-8]
        mov DWORD PTR [rbp-4], eax
        sub DWORD PTR [rbp-8], 1
        jmp .L3
.L2:
        mov eax, 0
        pop rbp
        ret
```

Disassembly

Disassembly

```
0x0000000000000000:  55                    push rbp
0x0000000000000001:  48 89 E5              mov  rbp, rsp
0x0000000000000004:  C7 45 FC 01 00 00 00  mov  dword ptr [rbp - 4], 1
0x000000000000000b:  C7 45 F8 0A 00 00 00  mov  dword ptr [rbp - 8], 0xa
0x0000000000000012:  8B 45 FC              mov  eax, dword ptr [rbp - 4]
0x0000000000000015:  0F AF 45 F8           imul eax, dword ptr [rbp - 8]
0x0000000000000019:  89 45 FC              mov  dword ptr [rbp - 4], eax
0x000000000000001c:  83 6D F8 01           sub  dword ptr [rbp - 8], 1
0x0000000000000020:  83 7D F8 00           cmp  dword ptr [rbp - 8], 0
0x0000000000000024:  5D                    pop  rbp
0x0000000000000025:  C3                    ret
```

Instruções

- Instruções definem a unidade de trabalho
- Cada instrução especifica
  - Uma operação ou opcode a ser efetuado
  - Operandos de origem e destino
- Em uma arquitetura von Neumann intruções de máquina são executadas sequencialmente
  - CPU implementa este loop logicamente
  - Por padrão, o próximo CP é dado por

    CP + (tamanho da instrução atual)

Desafio

- Implemente o seguinte algoritmo em assembly

```
N = 200
for i in range(5):
    N = N - i
```

- http://asmdebugger.com/

instrução	Descrição	Exemplo
mov	Atribui segundo parâmetro para o primeiro	mov eax, 10 / mov eax, ebx
sub	Subtrai segundo parâmetro do primeiro	sub ebx, 10 / sub ebx, eax
add	Adiciona segundo parâmetro para o primeiro	add ebx, 10 / add ebx, eax
inc	Incrementa registrador	inc ebx
dec	Decrementa registrador	dec ebx
cmp	Compara valores, ZF = 1 se iguais	cmp ebx, eax
label:	Cria um label para uma região de código	main:
jmp	Muda o fluxo do código para o label	jmp main
jz	Muda o fluxo se ZF = 1	jz main
jnz	Muda o fluxo se ZF = 0	jnz main

Desafio

- Implementem um algoritmo em assembly para calcular o décimo numero na sequencia de Fibonacci e armazena-lo no registrador edx

Sequencia	1	2	3	4	5	6	7	8	9	10	...
Valor	0	1	1	2	3	5	8	13	21	34	

- http://asmdebugger.com/

instrução	Descrição	Exemplo
mov	Atribui segundo parâmetro para o primeiro	mov eax, 10 / mov eax, ebx
sub	Subtrai segundo parâmetro do primeiro	sub ebx, 10 / sub ebx, eax
add	Adiciona segundo parâmetro para o primeiro	add ebx, 10 / add ebx, eax
inc	Incrementa registrador	inc ebx
dec	Decrementa registrador	dec ebx
cmp	Compara valores, ZF = 1 se iguais	cmp ebx, eax
label:	Cria um label para uma região de código	main:
jmp	Muda o fluxo do código para o label	jmp main
jz	Muda o fluxo se ZF = 1	jz main
jnz	Muda o fluxo se ZF = 0	jnz main

Desafio

- Implemente o seguinte algoritmo em assembly (N=5)

$$fatorial(N) = N! = N*(N-1)*\ldots*1$$

- http://asmdebugger.com/

instrução	Descrição	Exemplo
mov	Atribui segundo parâmetro para o primeiro	mov eax, 10 / mov eax, ebx
sub	Subtrai segundo parâmetro do primeiro	sub ebx, 10 / sub ebx, eax
add	Adiciona segundo parâmetro para o primeiro	add ebx, 10 / add ebx, eax
inc	Incrementa registrador	inc ebx
dec	Decrementa registrador	dec ebx
cmp	Compara valores, ZF = 1 se iguais	cmp ebx, eax
label:	Cria um label para uma região de código	main:
jmp	Muda o fluxo do código para o label	jmp main
jz	Muda o fluxo se ZF = 1	jz main
jnz	Muda o fluxo se ZF = 0	jnz main
mul	Multiplica o registrador eax pelo parâmetro	mul 10 <- eax = eax * 10

Continua ...