Estudante: Barbara Letícia da Silva.

Prontuário: CJ3029921.

1° Semestre.

Professor: Josivan Pereira da Silva.

Disciplina: Arquitetura de Computadores - CJOARQC.

Atividade de Revisão – Arquitetura de Computadores

Conversão Binária, Decimal e Hexadecimal.

1. Converta 420_{10} para binário.

A Figura abaixo apresenta a resolução do exercício número 1.

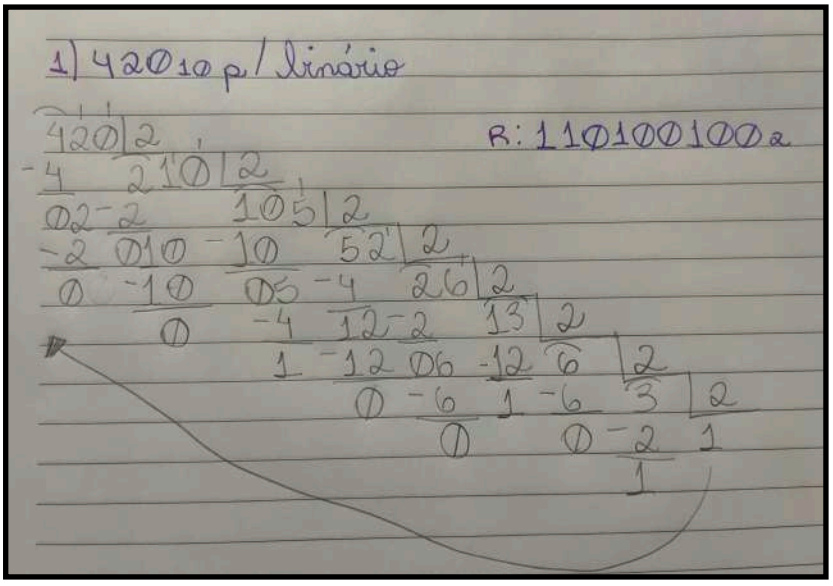

Figura 1

2. Converta 11110101_2 para decimal.

A Figura abaixo apresenta a resolução do exercício número 2.

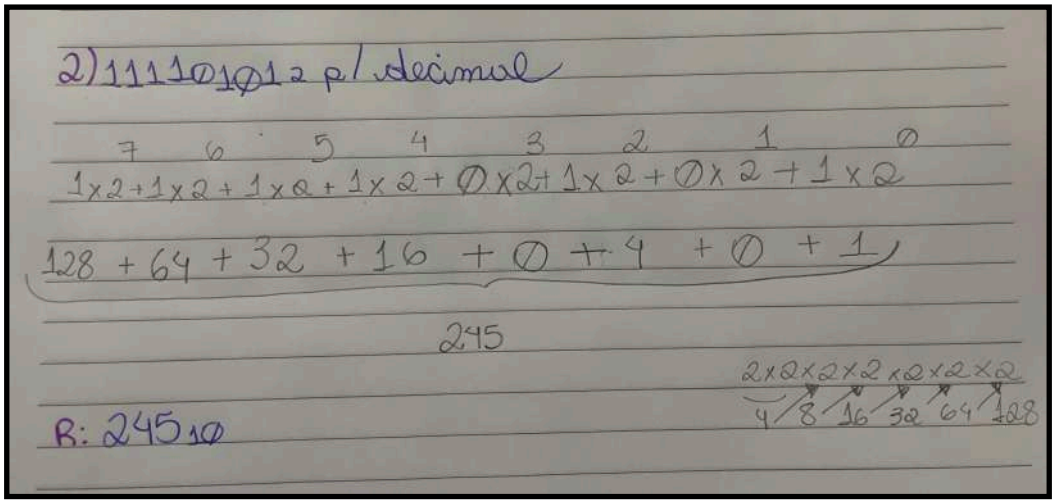

Figura 2

3. Converta $2F_{16}$ para decimal.

A Figura abaixo apresenta a resolução do exercício número 3.

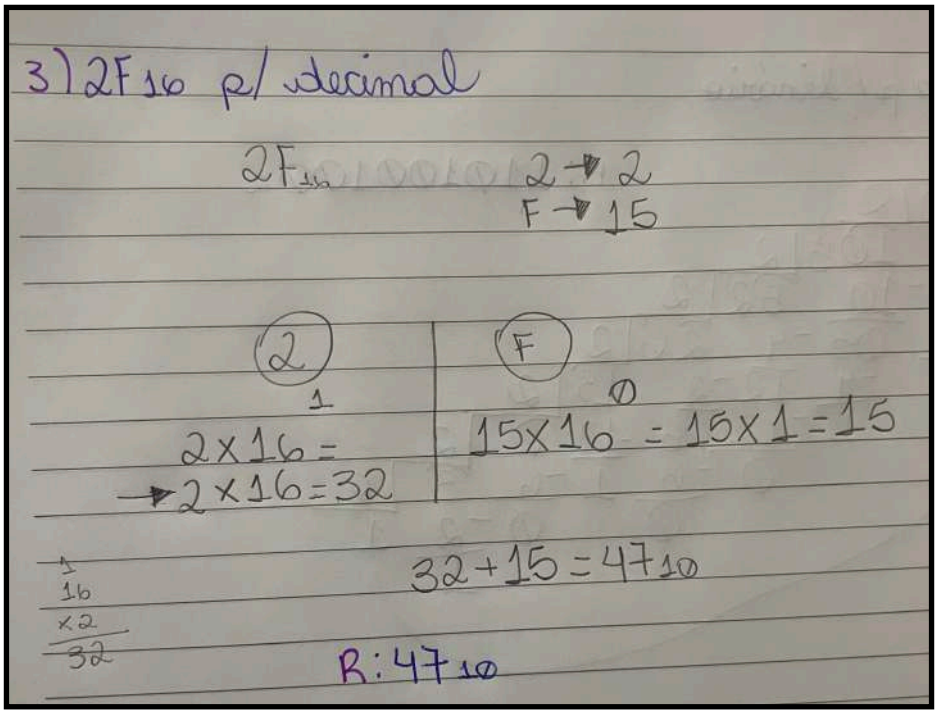

Figura 3

4. Converta 275_{10} para hexadecimal.

A Figura abaixo apresenta a resolução do exercício número 4.

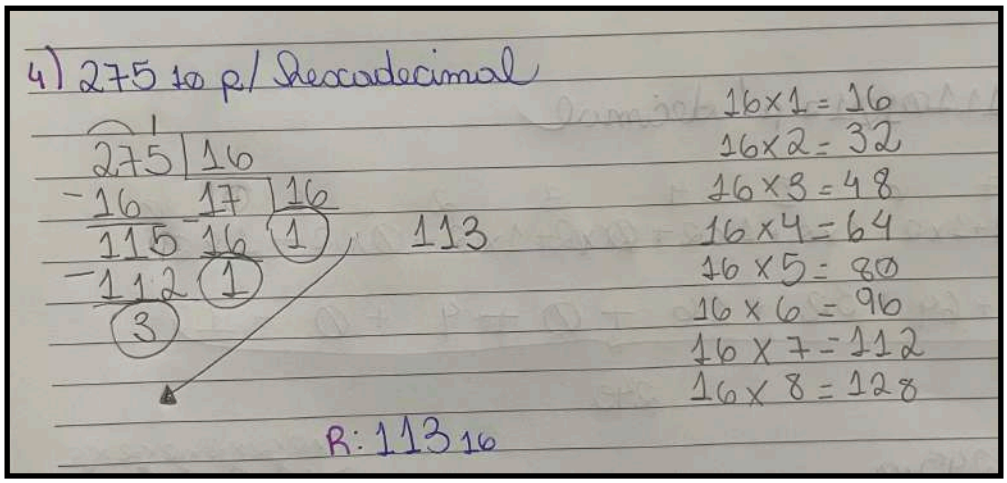

Figura 4

Representação Binária da Informação.

1. Quantos valores diferentes cabem em 12 *bits*?

R: Um *bit* pode assumir dois valores sendo eles, **0** ou **1**.

Dessa forma teremos **2 elevado a n**, na qual **dois** representa os **dois valores** que **um *bit*** pode assumir(**0** e **1**). E **n** será os **12 *bits*** estabelecidos no exercício.

Vamos resolver, 2ˆ12=2x2x2x2x2x2x2x2x2x2x2x2= 4096.

Em 12 *bits* cabem **4096** valores.

2. Um arquivo de 5.000 *caracteres ASCII* ocupa quantos *KB*?

R: 5000 *caracteres* × 1 *byte*/*caractere* = 5000 *bytes*

5000 *bytes* / 1024 *bytes KB* = 4,88 *KB*

3. Qual a principal vantagem do *Unicode* em relação ao *ASCII*?

R: A principal vantagem do *Unicode* em relação ao *ASCII* é a capacidade de representar praticamente todos os sistemas de escrita do mundo, enquanto o *ASCII* é limitado basicamente a *caracteres* do inglês.

Portas Lógicas Básicas.

• Resolva as expressões lógicas:

• Monte a tabela verdade da expressão

$$(A + B) \cdot \neg C.$$

(A *OR* B) *AND NOT* C

Tabela verdade A, B, C.

Como temos três variáveis então será 2ˆ3= 2x2x2=8 linhas com todas às combinações possíveis de 0(Falso) e 1(Verdadeiro).

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Tabela Verdade: A *OR* B (A+B)

A	B	A + B
0	0	0
0	0	0
0	1	1
0	1	1
1	0	1
1	0	1
1	1	1
1	1	1

Tabela Verdade: *NOT* C (¬C)

C	¬C
0	1
1	0
0	1
1	0
0	1
1	0
0	1
1	0

Resultado Final: (A + B) · ¬C

A	B	C	(A + B) · ¬C
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

A expressão final é **1 (Verdadeira)** quando:

(¬C) o mesmo que: C = 0

Pelo o menos uma entre A e B é 1 por conta do A+B

Sendo assim, é verdade quando **C** é **Falso** e **A** ou **B** é **Verdadeiro**.

Portas *NAND*, *NOR* e *XOR*

1. (A *NAND* B) · (A *XOR* B), A=1, B=0.

OPERAÇÕES	A=1 B=0
A *AND* B	**0**
***NAND*(A ^ B)**	**1**
A *XOR* B	**1**
(A *NAND* B) X (A *XOR* B)	**1**

2. (A *NOR* B) + (A *XOR* B), A=0, B=1.

OPERAÇÕES	A=0 B=1
A *OR* B	**1**
***NOT* (A *OR* B)**	**0**
A *XOR* B	**1**
(A *NOR* B) + (A *XOR* B)	**1**

3. (A *NAND* B) *XOR* C, A=1, B=1, C=0.

OPERAÇÕES	A=1 B=1 C=0
A *AND* B	**1**
***NOT* (A *AND* B)**	**0**
(A *NAND* B) *XOR* C	**0**

4. Explique por que a porta *NAND* é universal.

R: Podemos construir qualquer circuito lógico usando apenas a porta *NAND*. Isso é muito útil na fabricação de circuitos integrados: basta ter um tipo de porta (*NAND*) para construir microprocessadores e memórias, simplificando o design e a produção.

Álgebra Booleana e Circuitos Lógicos

1. Simplifique: **(A · B) + (A · ¬B).**

$$A \cdot (B + \neg B)$$

$$B + \neg B = 1 \text{ (complemento)}$$

$$A \cdot 1 = A$$

R: A

2. Simplifique: **(A + B) · (A + ¬B).**

$$A + (B \cdot \neg B)$$

$$B \cdot \neg B = 0 \text{ (complemento)}$$

$$A + 0 = A$$

R: A

3. Monte o circuito lógico: **S = (A · B) · (B + C).**

A Figura abaixo apresenta a resolução do exercício número 3.

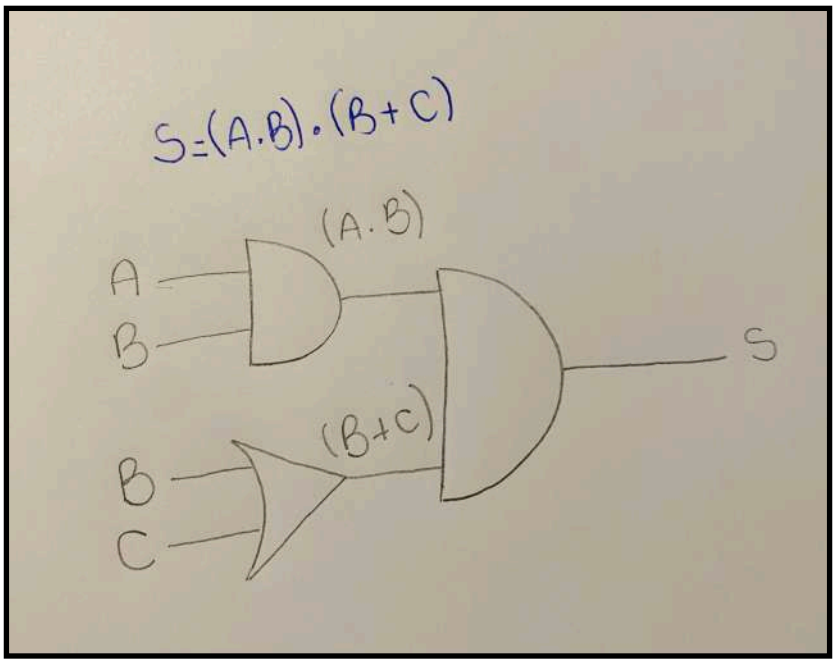

Figura 3 (Álgebra Booleana e Circuitos Lógicos)

Arquiteturas de Computadores

1. Diferença entre *Von Neumann* e *Harvard*.

R: A principal diferença entre as arquiteturas *Harvard* e *Von Neumann* está na organização da memória e dos barramentos.

Von Neumann: Usa uma única memória para instruções e dados, com um único barramento. Isso simplifica o sistema, mas cria um gargalo, impedindo o acesso simultâneo. Já a arquitetura de *Harvard*, possui memórias separadas, cada uma com seu barramento próprio. Isso permite acesso simultâneo, eliminando gargalos e melhorando o desempenho, mas com maior complexidade de implementação.

2. Vantagem e desvantagem de cada arquitetura.

R: Arquitetura *Von Neumann*

- **Vantagens:** Mais barata, flexível (memória unificada para dados e instruções), simples.
- **Desvantagens:** Gargalo no barramento único, menor desempenho, menos segura.

Arquitetura *Harvard*

- **Vantagens:** Maior velocidade (acesso simultâneo a dados e instruções), mais segura, melhor para tempo real.
- **Desvantagens:** Custo mais alto, complexa, uso rígido da memória.

3. O que é o gargalo de *Von Neumann*?

R: Basicamente, a via de transmissão de dados entre a *CPU* e a memória limita de certa forma a velocidade do processamento de um computador.

Os barramentos têm esta função e a troca de dados entre o processador e a memória fica limitada pela taxa de transferência de dados que esses barramentos são capazes de proporcionar, que em geral são bem menores que a capacidade dos processadores, sendo um fator limitador da velocidade atingida no processamento das informações.

Esse problema aumenta a cada nova geração e o desenvolvimento da tecnologia com maior número de barramentos é uma das soluções adotadas pelos fabricantes da tecnologia.

Unidade de Controle

1. Diferencie *Hardwired* vs Microprogramada.

R: A diferença fundamental está na forma como a Unidade de Controle (o "cérebro" do processador) gera os sinais que comandam todas as outras partes do *CPU*.

Unidade de Controle *Hardwired* (Por Fiação):

Ela é implementada como um circuito lógico fixo e permanente, construído com portas lógicas (*AND*, *OR*, *NOT*) e *flip-flops*. Cada instrução do conjunto de instruções (*ISA*) é "queimada" no silício. O caminho dos sinais é físico e definido pelo projeto do *hardware*. É como ter um painel de controle com fiações e interruptores fixos, onde apertar um botão (executar uma instrução) aciona uma sequência predeterminada de eventos de forma direta e imediata.

Unidade de Controle Microprogramada:

Ela é, na essência, um computador dentro de outro computador. No núcleo do *CPU* existe uma memória *ROM* especial chamada **Memória de Controle**, que armazena um programa de baixíssimo nível chamado **microprograma**. Cada instrução do conjunto de instruções que nós programamos (como *ADD* ou *MOV*) é, na verdade, uma "chamada" para uma pequena rotina desse microprograma, composta por várias **microinstruções**. A unidade de controle busca e executa essas microinstruções uma a uma para realizar a tarefa complexa. É como ter um intérprete que, para cada comando que você dá, consulta um manual interno (o microprograma) para saber quais pequenos passos executar.

2. Qual é mais rápida? Qual é mais flexível?

R: A implementação ***Hardwired* é significativamente mais rápida.** Por ser puramente *hardware*, não há o atraso inerente de ter que buscar e decodificar múltiplas microinstruções de uma memória. Os sinais são gerados e propagados quase que instantaneamente através dos circuitos, resultando em uma execução muito mais veloz.

A implementação **Microprogramada é muito mais flexível.** Como a lógica de controle está armazenada em uma memória (a *ROM* de microcódigo), alterar ou corrigir o funcionamento do processador é uma questão de modificar esse microprograma. Isso permite corrigir *bugs* (os famosos "micro*code updates*" que seu sistema operacional aplica), adicionar novas instruções ou até mesmo fazer um processador emular o conjunto de instruções de outro, tudo sem a necessidade de redesenhar e fabricar um novo *chip*, o que é incrivelmente caro e demorado.

3. Associe:

RISC **—> Unidade de Controle *Hardwired***

A arquitetura *RISC* é caracterizada por um conjunto de instruções simples, regulares e de ciclo único. Essa simplicidade permite que a unidade de controle seja implementada diretamente em *hardware* (*hardwired*), resultando em uma lógica de controle simples, rápida e extremamente eficiente. Consequentemente, a velocidade é um dos pilares fundamentais do *design RISC*.

CISC **—> Unidade de Controle Microprogramada**

A arquitetura *CISC* é definida por um conjunto de instruções complexas, poderosas e que exigem múltiplos ciclos para serem executadas. A solução ideal é o uso de uma unidade de controle microprogramada, onde cada instrução complexa é "quebrada" em uma sequência de microinstruções mais simples, armazenadas em uma memória interna. Essa abordagem facilita enormemente o projeto e a manutenção de um conjunto de instruções tão vasto e complexo, oferecendo maior flexibilidade.

Dispositivos de Entrada e Saída

Classifique como Entrada, Saída ou Entrada/Saída:

1. **Teclado -** Entrada.

É um dispositivo de **entrada** porque envia os dados digitados pelo usuário **para** o computador.

2. Impressora Multifuncional - Entrada/Saída.

É um dispositivo **entrada/saída** porque atua como **saída** ao imprimir e como **entrada** ao digitalizar documentos.

3. Projetor Multimídia - Saída.

É um dispositivo de **saída** porque recebe a imagem do computador e a exibe **para** o usuário em uma tela ampliada.

4. *Smartphone* - Entrada/Saída.

É um dispositivo **entrada/saída** porque sua tela e microfone capturam dados (entrada), enquanto sua tela e alto-falante exibem e tocam sons (saída).

5. *Joystick* com *feedback* vibratório - Entrada/Saída.

É um dispositivo **entrada/saída** porque envia os comandos do usuário **para** o computador (entrada) e recebe o sinal de vibração **do** computador (saída).

Ciclo de Execução

Simule o ciclo de execução do programa:

- AX ← 4
- BX ← 7
- *ADD* AX, BX

Explique:

1. O que foi buscado?

R: AX ← 4

- **Busca:** Instrução "*MOV* AX, 4"
- **Decodificação:** Operação *MOV* para carregar valor em AX
- **Execução:** Carregou 4 no registrador
- **Armazenamento:** Valor 4 em AX

2. O que foi decodificado?

R: BX ← 7

- **Busca:** Instrução "*MOV* BX, 7"
- **Decodificação:** Operação *MOV* para carregar valor em BX
- **Execução:** Carregou 7 no registrador
- **Armazenamento:** Valor 7 em BX

3. O que foi executado?

R: *ADD* AX, BX

- **Busca:** Instrução "*ADD* AX, BX"
- **Decodificação:** Operação de soma entre registradores
- **Execução:** Somou AX(4) + BX(7) = 11
- **Armazenamento:** Resultado 11 em AX

4. Onde foi armazenado?

R: AX = 11

BX = 7

Referências Bibliográficas

AtividadeDePesquisa_*Harvard-RISC-CISC-ARM*.pdf

Disponível em: *https://github.com/leticiabarbar/CJOARQ/blob/main/AtividadeDePesquisa_Harvard-RISC-CISC-ARM.pdf* .

Acesso em 30 de Setembro de 2025.

AtividadedePesquisa*VonNewmann*_CicloExecução.pdf

Disponível em: *https://github.com/leticiabarbar/CJOARQ/blob/main/AtividadedePesquisaVonNewmann_CicloExecução.pdf* .

Acesso em 30 de Setembro de 2025.

PortasLógicas.pdf

Disponível em: *https://github.com/leticiabarbar/CJOARQ/blob/main/PortasLógicas.pdf* .

Acesso em 01 de Outubro de 2025.

MateriasdeAula-CJOARQC

Disponível em: *https://suap.ifsp.edu.br/edu/sala_virtual/374710/?tab=materiais* .

Acesso em 01 de Outubro de 2025.