

Escola SENAI Fundação Zerrener

Eletrônica

Processador Programável com Circuitos Discretos

**Guilherme Bohnstedt
Lucas Seiji Kido**

**São Paulo
2013**

**Guilherme Bohnstedt
Lucas Seiji Kido**

Processador Programável com Circuitos Discretos

Monografia apresentada como exigência
para obtenção do grau de Ensino Médio
em Eletrônica da Escola SENAI Fundação
Zerrener.

Orientador: Ricardo Gallatti Yasumura

**São Paulo
2013**

AGRADECIMENTOS

Agradecemos ao professor pelo apoio e pelo incentivo.

RESUMO

O objetivo do projeto é ser um kit didático que de ao usuário uma primeira noção de programação e como um processador funciona e o que ele faz com uma variável, manipulando os bits. Ele também pode ser conectado no computador, e possui uma interface onde se pode programar. Nos baseamos em um computador feito em casa, Magic-1, mas utilizando CI's CMOS. Pode se inserir os dados manualmente ou pelo computador, onde se cria um arquivo que mostra a manipulação da variável, e escolher qual operação o processador irá realizar com a sua variável.

Palavras-chave: Kit didático, processador, circuitos discretos

ABSTRACT

The objective of this project is to be a didactic kit that gives to the user his first notion of programming, how a processor works and what it does to the variable manipulating the bits. It also can be connected to computer, and has a interface where you can programming. We have as a base the homebrew computer Magic-1, but we used CMOS circuit integrated. The user can input the data manually or by the computer.

Keywords: Didactic kit, processor, discrete circuit

LISTA DE FIGURAS

Figura 1 - Algoritmo de Divisão por Dois em Binário.....	11
Figura 2 - Arquitetura Interna do Processador.....	17
Figura 3 - Portas Lógicas.....	19
Figura 4 - Multiplexador.....	20
Figura 5 - Demultiplexador.....	21
Figura 6 - Comparadores Binários.....	23
Figura 7 - Layout da placa da entrada.....	39
Figura 8 - Layout da placa da ULA.....	40

LISTA DE TABELAS

Tabela 1 - Número Binários.....	10
Tabela 2 - Complemento de Um.....	12
Tabela 3 - Complemento de Dois.....	13
Tabela 4 - Flip-Flop Saída.....	22
Tabela 5 - Flip-Flop JK.....	22
Tabela 6 - Cloud Code Instruções.....	25
Tabela 7 - Erros e Descrições.....	37
Tabela 8 - Características Técnicas do Processador.....	38
Tabela 9 - Componentes e Preços.....	40

SUMÁRIO

1 INTRODUÇÃO.....	8
2 DESENVOLVIMENTO.....	9
2.1 Proposição.....	9
2.2 Estado da Arte.....	9
2.3 Concepção da Ideia.....	10
2.3.1 Número Binários.....	10
2.3.1.1 Conversão Decimal Binário.....	10
2.3.2 Operações Aritméticas.....	11
2.3.2.1 Soma.....	12
2.3.2.2 Subtração.....	12
2.3.2.3 Multiplicação.....	15
2.3.2.4 Divisão.....	16
2.3.3 Operações Lógicas.....	16
2.3.4 Construção do Hardware.....	17
2.3.5 Componentes Eletrônicos Digitais.....	18
2.3.5.1 Portas Lógicas.....	18
2.3.5.2 Multiplexadores e Demultiplexadores.....	20
2.3.5.3 Flip-Flops.....	21
2.3.5.4 Comparadores Binários.....	22
2.3.6 Unidade Lógica e Aritmética.....	23
2.3.7 Unidade de Controle.....	23
2.3.8 Circuito Clock.....	24
2.3.9 Circuito de Memória de Programa.....	25
2.3.9.1 Cloud Code.....	25
2.3.9.1.1 Assembler.....	26
2.3.9.1.2 Gravação.....	38
2.4 Execução.....	38
2.5 Melhorias Futuras.....	41
3 CONCLUSÃO.....	42
REFERÊNCIAS.....	43

1 INTRODUÇÃO

Os computadores possuem uma longa história. As primeiras máquinas de computar eram todas mecânicas. A primeira máquina de verdade foi construída por Wilhelm Schickard sendo capaz de somar, subtrair, multiplicar e dividir. Anos mais tarde Gottfried Wilhelm Leibniz aprimorou a máquina de Pascal. No entanto, essas máquinas não eram programáveis e por isso estavam longe dos computadores atuais. Além disso, o sistema utilizado para fazer os cálculos não era no sistema binário o que tornava os sistemas confusos. Depois os sistemas mecânicos de cálculos foram substituídos por válvulas, precedentes dos diodos eletrônicos. O primeiro grande computador foi o ENIAC (Electronic Numeric Integrator And Calculator). Somente com John von Neumann o sistema binário passou a ser utilizado em operações. Depois que os transistores foram construídos houve uma revolução na construção de sistemas computacionais. Os circuitos integrados diminuíram seu tamanho e o deixaram mais simples e eficaz. Este trabalho será desenvolvido a base da lógica e dos componentes mais simples, remontando aos circuitos lógicos e utilizando as arquiteturas dos processadores atuais.

2 DESENVOLVIMENTO

2.1 Proposição

Atualmente todos temos contato com computadores. Independente da atividade, os computadores são a grande central de qualquer sistema. Alunos dos cursos técnicos na área de eletrônica aprendem o básico da eletrônica digital e microcontroladores, CLP's, entre outros componentes processadores de dados. Em ensinos universitários ligado a área de computação, diversos temas sobre arquitetura de computadores são abordados, no entanto tudo teoricamente. Um dos componentes mais obscuros e sem material aberto são os processadores. Como sistemas eletrônicos digitais conseguem fazer contas? E como é manipulado os bits dentro de um processador? O objetivo desse trabalho é apresentar uma ideia geral de como um processador pode funcionar.

2.2 Estado da Arte

Magic-1 é um computador feito em casa, mas ele é feito entorno de 200 CI's TTL 74 Series. No verão de 1980, o criador do Magic-1, conhecido como Bill, comemorava seu bacharelado em Jornalismo e então ganhou um computador TRS-80 Model 1, onde ficou indignado que podia interagir com a máquina, isso fez com que ficasse interessado para saber como os computadores funcionavam. Vinte anos depois ele começou a construir seu proprio computador, peça por peça. O computador é um unico endereço com opcodes de um bit, e possui operações de dados 8/16 bits e usa um barramento de dados de 8 bits com 16 bits de endereço. O código e o endereço de dados podem ser compartilhado ou separados. Possui o mod administrador e usuário, memória mapeada de IO e interrupção externa. Foi inteiramente construido por CI's 74LS e 74F da TTL e SRAM e EPROMs. E depois

de alguns anos de redesenhando o mecanismo Magic-1 ou M-1 roda a 4.09 MHz. Agora o computador possui um compilador ANSI C e um sistema operacional Minix 2 e consegue conectar a internet.

2.3 Concepção da Ideia

2.3.1 Número Binários

O sistema de numeração binário é um sistema de sistema posicional, onde as quantidades são representadas por dois dígitos(1 ou 0). Esse sistema desse sistema foi atribuído a Leibniz. Esse sistema é muito prático para ser utilizado em sistemas digitais, onde um sinal ligado ou desligado é convertido em um representante número. Convencionou-se que o 0 representa a falta de sinal, ou tensão igual a 0 e 1 representa a presença de sinal ou 5 Volts. Com algumas convenções os sistemas digitais conseguem representar caracteres, números, imagens, fazer cálculos e etc.

2.3.1.1 Conversão Decimal Binário

Cada posição do binário corresponde a dois elevado a posição.

Tabela 1 - Número Binários

Base	n (posição)	2^n
2	0	1
2	1	2
2	2	4
2	3	8
2	4	16

Fonte: Autoria própria.

Para converter um decimal em binário existem vários algoritmos. O algoritmo da divisão por dois é o mais utilizado.

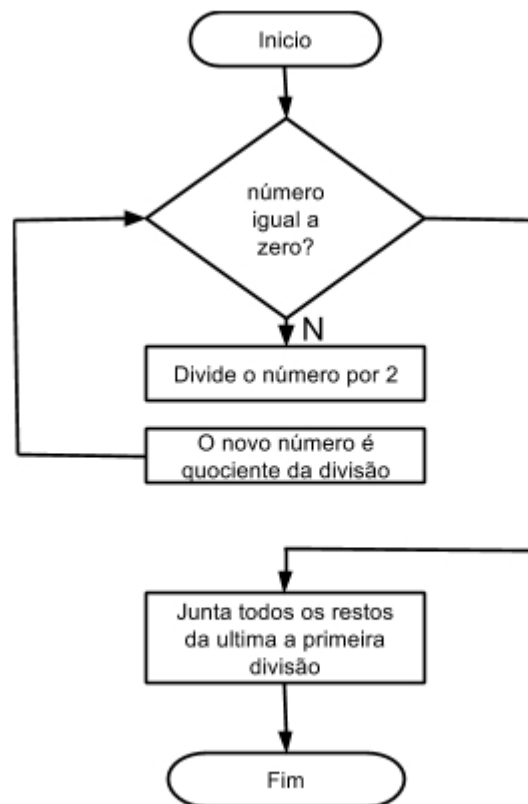


Figura 1 - Algoritmo de Divisão por Dois em Binário
Fonte: Autoria própria.

2.3.2 Operações Aritméticas

Por se tratar de um circuito puramente digital, a base binária foi utilizada no desenvolvimento do trabalho. A soma é a base primária para qualquer operação matemática, pois a subtração é a soma de um número com um outro negativo, a multiplicação é a soma sucessiva, divisão é a subtração sucessiva(que deriva da soma). Portanto, um único circuito somador foi utilizado e uma lógica de controle para a obtenção dos resultados das quatro operações básicas da matemáticas, a saber: soma, subtração, multiplicação e divisão.

Regras básicas para adição de binários:

1. $0 + 0 = 0$
2. $0 + 1 = 1$
3. $1 + 0 = 1$
4. $1 + 1 = 10$ (zero "sobe" um)

2.3.2.1 Soma

```

  11010
+ 10011
-----
101101

```

No exemplo acima, estamos somando o número 26 com o número 19. Pela regra geral da matemática somasse da direita para a esquerda cada unidade. Seguindo a regra básica de adição e binário chegamos ao número 101101(base 2), que é 45 em decimal($26+19=45$). Assim como cotidianamente somamos números da base decimal, os binários seguem a mesma lógica.

2.3.2.2 Subtração

Para fazermos uma subtração, somamos um número positivo com um negativo. São dois os modos de se obter um binário negativo: complemento de dois e complemento de um.

O complemento de um consiste em inverter todos os bits do número.

Tabela 2 - Complemento de Um

Binário	Interpretação de Complemento de Um	Interpretação sem sinal
00000000	+0	0

00000001	+1	1
...
01111111	127	127
10000000	-127	128
10000001	-126	129
...
11111110	-1	254

Fonte: Autoria própria.

Como podemos observar, no complemento de um existe dois zeros(negativo e positivo).

O complemento de dois consiste em inverter o número(complemento de um) e depois somar 1.

Tabela 3 - Complemento de Dois

Binário	Interpretação de Complemento de Dois	Interpretação sem sinal
00000000	0	0
00000001	1	1
...
01111111	127	127
10000000	-127	128
...
11111110	-2	254
11111111	-1	255

Fonte: Autoria própria.

No entanto neste trabalho este padrão foi ligeiramente modificado, onde o bit mais significativo não é sinal. Logo o processador aceita números de 0 a 255, somente positivos. Entretanto, como será abordado, é possível obter números negativos.

Na arquitetura deste processador a operação de subtração possui dois algoritmos. O algoritmo que o circuito usará dependerá dos operandos. O primeiro algoritmo é usado quando o operando positivo for maior que o negativo. O segundo algoritmo é utilizado quando o operando negativo for maior que o positivo.

O primeiro é constituído pelo algoritmo do complemento de dois.

1º passo: pegasse o segundo operando(ele deve ser menor que o primeiro) e é aplicado a inversão binária.

2º passo: somasse o primeiro operando com o inverso do segundo.

3º passo: somasse um ao resultado anterior.

Exemplificando com a seguinte operação (8-4).

8=1000.

4=0100.

1º passo:

4=0100

-4=1011

2º passo

11000

+1011

10011

3º passo:

10011

+ 1

10100

Aplicando o metodo obtivemos 0100 que é 4 em decimal com o 5º bit com valor 1(foram utilizados 4 bits para o exemplo, no entanto para 8 bits o algoritmo funcionará da mesma forma, com o diferencial que o 9º bit terá o valor 1.)

Este ultimo bit indica o valor positivo da operação.

O segundo é constituída em aplicar o algoritmo do complemento de um duas vezes.

1º passo: pegasse segundo operando(ele deve ser maior que o primeiro) e é aplicado a inversão binária.

2º passo: somasse o primeiro operando com o inverso do segundo.

3º passo: complemento de um no resultado anterior.

Exemplificando com a seguinte operação (4-8).

8=1000.

4=0100.

1º passo:

8=1000

$$\begin{array}{r}
 -8=0111 \\
 2^{\circ} \text{ passo} \\
 \begin{array}{r}
 10100 \\
 + \quad 0111 \\
 \hline
 1011
 \end{array}
 \end{array}$$

3º passo:

$$\text{NOT}(1011)=0100=-4$$

Aplicando esse metodo obtemos 0100 que representa 4 em decimal. Notasse que o 5º não está com valor um, configurando assim o número como negativo(Observação importante:A logica para números positivos e negativos é inversa a utilizada pelos processadores usuais, a arquitetura eletronica, assim como a logica utilizada não se assemelha a nenhum outro).

2.3.2.3 Multiplicação

Assim como na base decimal, a multiplicação em binário segue o mesmo algoritmo. Por exemplo: $8 \times 2 = 16$

$$\begin{array}{r}
 8=1000 \\
 2=0010 \\
 16=10000 \\
 \begin{array}{r}
 1000 \\
 \times 0010 \\
 \hline
 0000 \\
 1000+ \\
 0000++ \\
 0000+++ \\
 \hline
 0010000
 \end{array}
 \end{array}$$

No entanto, para um processador ou computador numérico esse algoritmo é longo e complicado de ser implementado. Utilizamos, portanto, o algoritmo das sucessivas adições. O primeiro operando é o número de vezes que o segundo operando será somado.

2.3.2.4 Divisão

Na divisão em binário, utilizamos o mesmo algoritmo da divisão em decimal.

```

10100 | 101
      ----
- 101    100
      ----
        000
  
```

Implementar esse algoritmo em sistemas lógicos é massacrante e completamente impensável para tratar com grandes quantidades de bits. Por isso utilizasse nos processadores o algoritmo das sucessivas subtrações. No entanto, neste trabalho faremos o inverso: trabalharemos com sucessivas adições! O segundo operador será somado a ele mesmo até que ele chegue ou ultrapasse o primeiro operador. Utilizando contadores, saberemos quantas operações foram feitas e teremos um resultado.

2.3.3 Operações Lógicas

Como em o todo processador, as operações lógicas são fundamentais. Na lógica, existem quatro operações básicas, a saber: SIM, NÃO, E e OU. A partir delas podemos derivar qualquer outro comparador de sinal digital.

2.3.4 Construção do Hardware

Como notado, todas as operações aritméticas tem como a base a adição. Portanto foi construído um único circuito somador. Neste mesmo circuito, as operações lógicas foram implementadas, portanto criando uma Unidade Lógica e Aritmética(ULA). A Unidade de Controle é responsável por controlar o “caminho” que cada bit fará para que a operação escolhida seja realizada. A Unidade de Clock é a responsável por controlar o número de clocks necessários para que uma operação. A Unidade de Memória de Programa, possui uma EEPROM que executa operações matemáticas automaticamente, de acordo com o que foi codificado pelo programador. Ainda existem a Unidade de Entrada e Unidade de Saída.

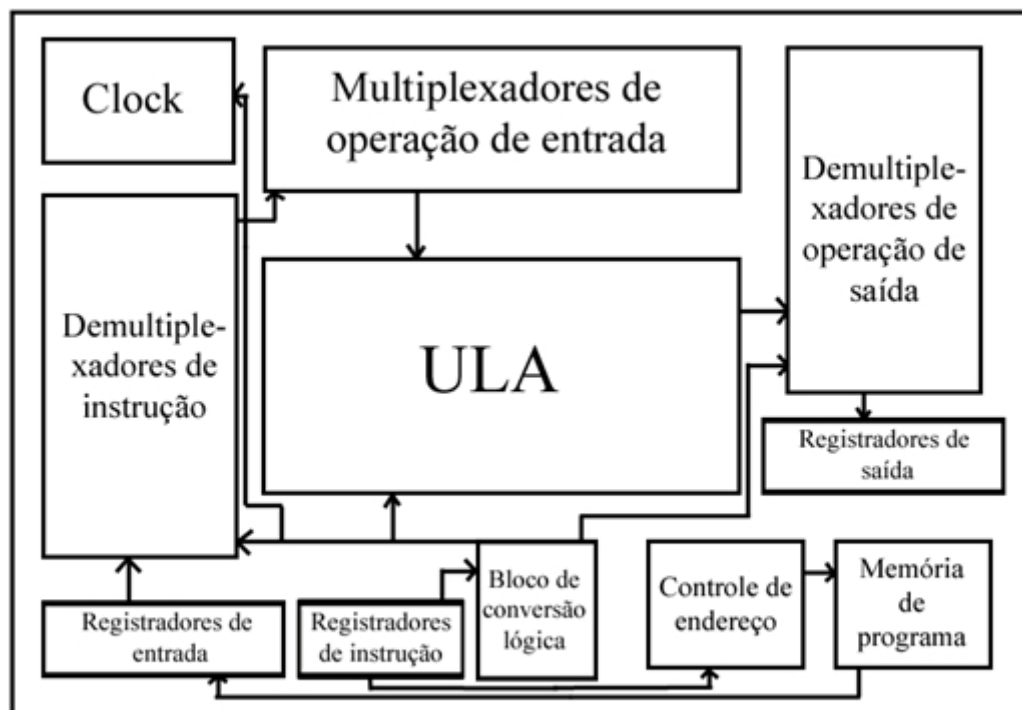


Figura 2 - Arquitetura Interna do Processador
 Fonte: Autoria própria.

A partir dessa arquitetura e dos algoritmos das operações matemáticas citadas, construímos os circuitos lógicos.

2.3.5 Componentes Eletrônicos Digitais

2.3.5.1 Portas Lógicas

Em eletrônica digital, trabalha-se apenas com dois estados: “ligado” ou “desligado”, 1 ou 0. Estes dois tipos de sinais podem ser comparados a partir de portas lógicas. As portas lógicas processam as entradas e produzem uma saída. As portas logicas conhecidas são apresentadas na tabela abaixo.


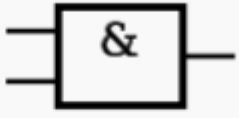

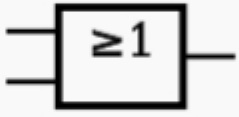



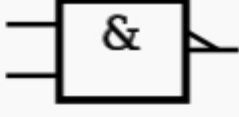

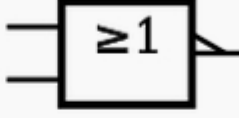

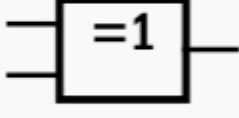

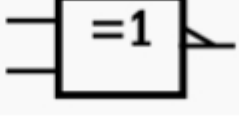
Tipo	Símbolo (Norma ANSI)	Símbolo (Norma IEC)	Função booleana
AND			$A \cdot B$
OR			$A + B$
NOT			\overline{A}
NAND			$\overline{A \cdot B}$
NOR			$\overline{A + B}$
XOR			$A \oplus B$
XNOR			$\overline{A \oplus B}$

Figura 3 - Portas Lógicas
Fonte: Autoria própria.

A partir das portas lógicas são construídos outros componentes como multiplexadores, demultiplexadores, comparadores binários, flip-flops, entre outros. Todos os citados foram utilizados no trabalho.

2.3.5.2 Multiplexadores e Demultiplexadores

MUX e DEMUX são dispositivos digitais que processam informações de diversas formas, servindo como conversores serial/paralelo e vice-versa.

Os Multiplexadores possuem varias entradas e uma única saída. Pelos bits de controle podemos selecionar uma entrada e conectar as informações dessa entrada a saída.

A equação que define um MUX primário é

$$S = (A \text{ and not } C) \text{ or } (B \text{ and } C)$$

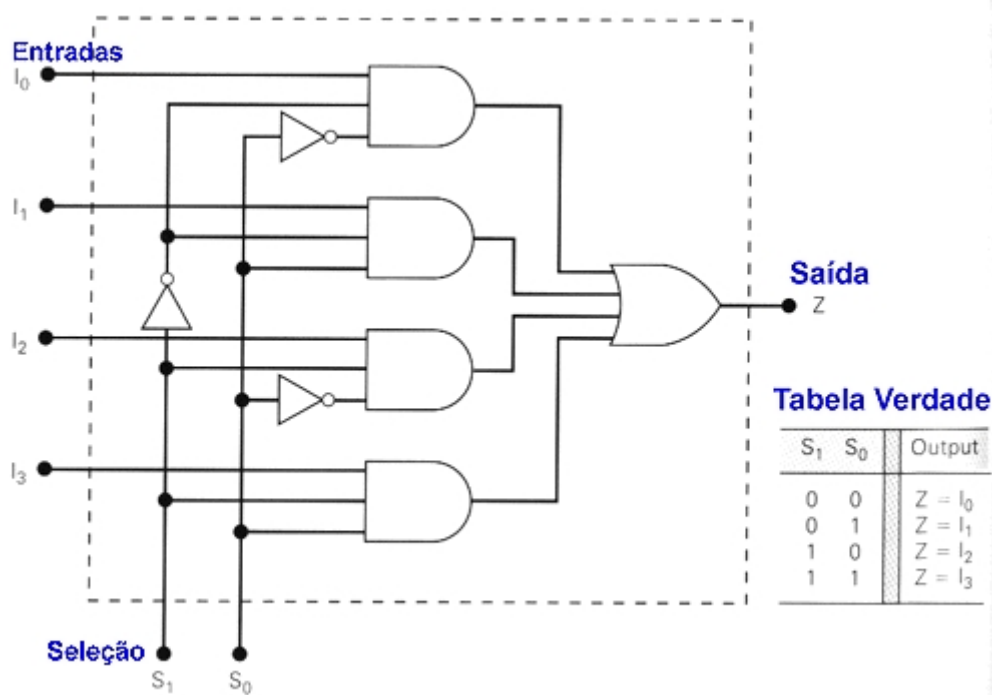


Figura 4 - Multiplexador
 Fonte: Autoria própria.

O Demultiplexador é o inverso do MUX. Nele há uma única entrada para várias saídas. Novamente com os bits de controle, é selecionada uma saída onde será reproduzida a entrada.

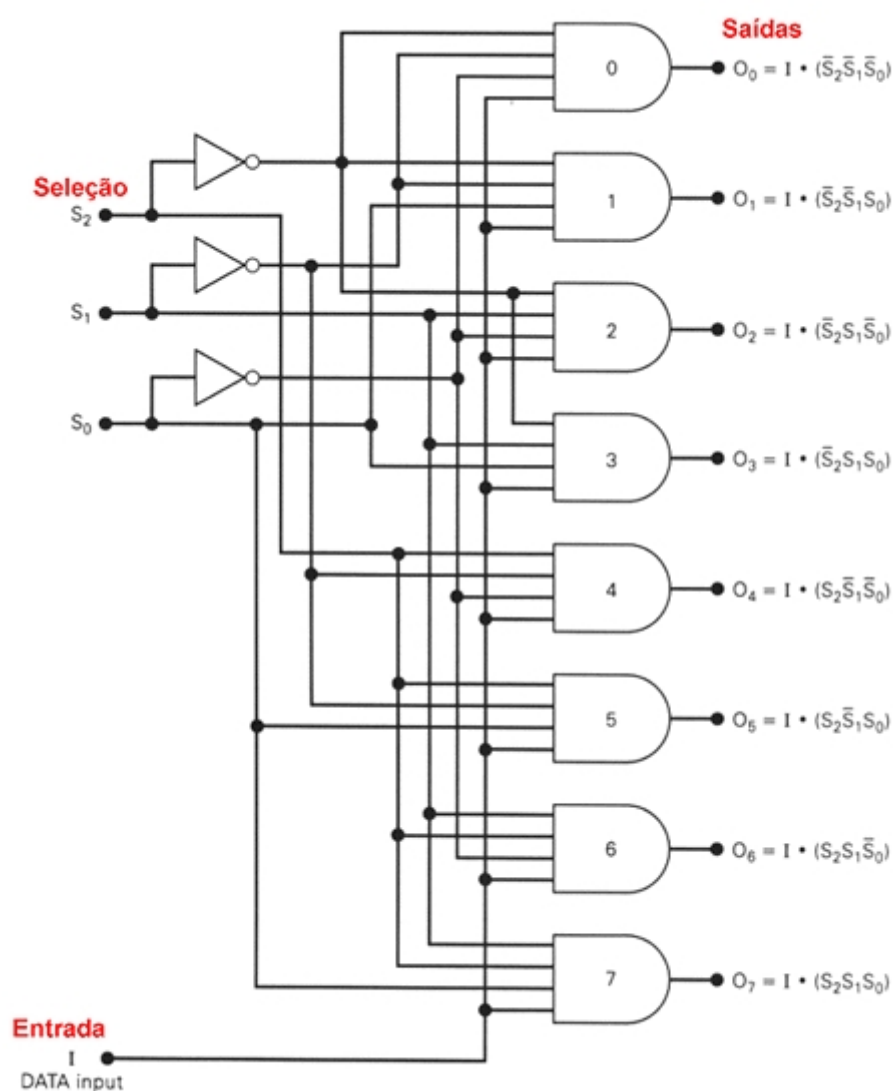


Figura 5 - Demultiplexador
Fonte: Autoria própria.

2.3.5.3 Flip-Flops

Um Flip-Flop é um multivibrador biestável que tem a capacidade de armazenar um bit. Tipicamente, os flips-flops possuem uma entrada, um sinal clock e uma saída. Existem alguns tipos de flip-flops, a saber: Tipo D, Tipo RS, Tipo JK e Tipo T. Cada Flip-Flop possui uma tabela verdade diferente e por esse motivo seu uso é exclusivo. Neste trabalho, foi utilizado apenas os tipos D e JK. Por esse motivo só será abordados sobre eles.

O flip-flop D é utilizado tipicamente como um registrador binário (por isso o D de DATA). Sua entrada reflete na sua saída.

Tabela 4 - Flip-Flop Saída

D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1

Fonte: Autoria própria.

Onde Q* significa o estado posterior de Q depois do clock.

Os flip-flops JK são utilizados neste trabalho para a criação de um contador de oito bits. Ele aprimora o uso do flip-flop RS.

Tabela 5 - Flip-Flop JK

J	K	Qprox	Comentário
0	0	Qanterior	mantém (hold)
0	1	0	reseta
1	0	1	seta
1	1	Qanterior	alterna (Toggle)

Fonte: Autoria própria.

2.3.5.4 Comparadores Binários

Os comparadores binários são componentes que possuem um determinado bit de entrada e retorna a relação entre eles. Os comparadores utilizados nesse projeto são de quatro bits e retorna em suas saídas se os números da entrada são iguais e qual é o maior.

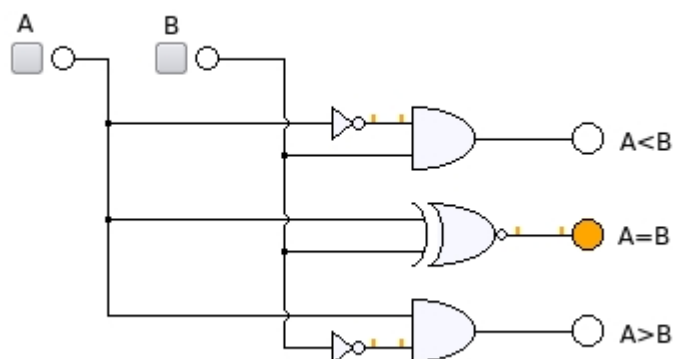


Figura 6 - Comparadores Binários
Fonte: Autoria própria.

2.3.6 Unidade Lógica e Aritmética

O cérebro de todo o projeto, a ULA, é responsável pelas mais básicas operações do processador. Para criar um circuito somador, poderíamos utilizar a álgebra de Boole. No entanto, utilizamos somente lógica.

A porta lógica XOR possui exatamente essa tabela verdade. Então para somarmos dois bits utilizamos essa porta lógica. Quando somamos da segunda posição do binário em diante, precisamos verificar se ocorreu o “vai um”. Para que isso ocorra as duas entradas da XOR precisam ser um. Então com uma porta lógica AND podemos fazer essa verificação. Seguimos essa mesma lógica para saber se o resultado da soma dos dois bits é um e se existe o “vai um” da operação anterior.

O circuito lógico precisa apenas das portas lógicas de cada operação. No entanto, a saída do somador é a mesma do circuito lógico e para isso multiplexadores mudam o “caminho” de cada bit. Demultiplexadores reorganizam os bits para uma única saída.

2.3.7 Unidade de Controle

Ela é a responsável por organizar as informações que estão sendo processadas. Nela multiplexadores e demultiplexadores organizam cada bit para que a operação desejada seja feita com sucesso.

Os primeiros demultiplexadores são da entrada do circuito. É neles que cada bit da entrada mudará o percurso dependendo da operação. Na soma os bits das entradas iram direto para a ULA, sem passar por nenhum processamento. Na subtração, os bits do segundo operando sofreram complemento de um (inversão dos bits). Caso o primeiro operando seja maior que o segundo, ao resultado da adição será somado um. Configurando um complemento de dois. Caso o segundo operando seja o maior, ao resultado da soma será aplicado um outro complemento de um. Na multiplicação os bits do primeiro operando são enviados para os comparadores no circuito de clock. Os bits do segundo operando fazem o mesmo caminho da soma (não passam por nenhum processamento). Na ULA é somado o conteúdo dos registradores de saída com o segundo operando da entrada. A divisão segue a mesma lógica que a multiplicação, no entanto a diferença está na comparação feita no circuito clock.

2.3.8 Circuito Clock

O circuito clock é responsável por gerenciar a quantidade de clock's necessários para que cada operação seja efetuada. Na soma uma porta lógica AND verifica se a condição do botão apertado e se o contador já contou um clock. Assim que a condição é verdadeira o registrador de igual é resetado e o clock para de ser enviado para o circuito. No caso da subtração há casos. No caso do primeiro operando ser maior que o segundo (configurando uma operação com resultado positivo) são dados dois clock's. É feita uma verificação no nono bit do somador, e no bit de saída dos registradores. Caso o segundo operando seja maior que o primeiro, essa verificação retorna valor um e assim o circuito gera dois clock's. Na multiplicação e na divisão os comparadores são habilitados. No caso da multiplicação o primeiro operando dos registradores de entrada é comparado com o conteúdo do contador. Enquanto eles forem diferentes o segundo operando da entrada é somado a si mesmo. Quando o circuito lógico dos comparadores retornar

um, o registrador de igual é resetado e clock para. Na divisão ocorre a mesma comparação, no entanto a comparação é entre os registradores de entrada do primeiro operando e o resultado do somador. Assim que o resultado for maior que o primeiro operando, o circuito lógico dos comparadores retorna um e cessa o pulso de clock.

No caso das operações lógicas somente é necessário um pulso para que a operação ocorra, então somente o bloco lógico de pulso único é habilitado.

2.3.9 Circuito de Memória de Programa

O circuito de memória de programa possui uma eeprom onde é gravado o código de programa. Um contador é responsável pela mudança de endereços da eeprom. Dependendo da instrução, o clock na memória ficará inativo. Por exemplo: Caso foi programado fazer uma multiplicação, os dados que estão em determinado byte da eeprom será gravado nos registradores de entrada e o clock ficará inativo até que a operação seja concluída. E para isso um circuito lógico habilita ou desabilita o clock.

A gravação é feita através da porta paralela ou serial. Caso seja pela porta paralela, os dados serão gravados diretamente. Caso seja pela serial, um microcontrolador fará a interface de conversão. O modo de gravação é escolhido no programa assembler que será considerado posteriormente.

2.3.9.1 Cloud Code

Para que o trabalho seja um processador e não apenas uma calculadora, ele deve ser programável. Para isso uma linguagem assembly foi desenvolvida que por meio do montador Assembler Cloud Code constrói a linguagem de máquina.

O Cloud Code possui 8 instruções:

Tabela 6 - Cloud Code Instruções

Instrução	Representação em binário
sim	00010001
nao	00010010
and	00010100
ou.	00011000
adc	00000001
sub	00000010
mul	00000100
div	00001000

Fonte: Autoria própria.

Pra escrever um código em cloud code, algumas regras tem de ser seguidas:

1-Antes de qualquer instrução, pode utilizar infinitos espaços

2-As instruções são somente em letra minúscula.

3-Uma instrução de operação tem a seguinte formatação:

instrução [operando de uma casa decimal]

Exemplos:

adc 1

mul 2

div 9

Sempre há espaço entre a instrução e o operando, caso contrário haverá erro na montagem.

4-A variável var deve ser declarada para uso de operações aritméticas. A instrução var é um diretiva do assembler, onde um valor é guardado nela e todas as operações aritméticas são executadas no valor atual da variável.

2.3.9.1.1 Assembler

O programa montador segue uma lógica simples para traduzir o assembly para código de máquina. Cada linha é lida separadamente e processada. Se alguma das regras citadas acima não for cumprida, o programa retornará erro. A aplicação foi escrito em C e C++. Sua interface foi criada no C++ Builder, uma IDE de desenvolvimento para desenvolvimento de software em C/C++.

```

void __fastcall TForm1::Montar1Click(TObject *Sender)
{
    //Gera o Arquivo com o programa
    ofstream txtFile;
    txtFile.open("txtFile.txt");
    txtFile << txtCodigo->Lines->GetText();
    txtFile.close();
    // Le o arquivo gerado para processar as informações
    txtFile2.open("txtFile.txt");
    char i=0;
    int continst=0;
    Error=false;
    while(txtFile2.get(c)){
        while(i!=3){//Guarda a instrução
            inst[i]=c;
            if(c!=' '){
                i++;
            }
            txtFile2.get(c);
        }
        continst++;
        i=0;
        if((inst[0]=='v')&&(inst[1]=='a')&&(inst[2]=='r')){
            variavel=true;
            txtFile2.get(c);
            goto Fim;
        }
        if(!VerificaInst()){
            Application->MessageBox("Instrução não reconhecida!", "Erro 001",
MB_OK);
            Error=true;
            goto Fim;
        }
    }
}

```

```

        if( c != ' '){
            Application->MessageBox("Espaço não encontrado depois da
instrução!", "Erro 002", MB_OK);
            Error=true;
            goto Fim;
        }
        instruction();
        Fim:
        if(Error){
            break;
        }
    }
    txtFile2.close();
    if(c=='\0'){
        Application->MessageBox("Arquivo em branco!", "Erro 000", MB_OK);
    }else{
        if(Error){
            char *mensagem; //Variavel que guardará a mensagem de erro
            char buff[3];
            itoa(continst,buff,10);
            mensagem=strcat("Montagem concluída com erro na instrução ",buff);
            Application->MessageBox(mensagem, "Fim", MB_OK);
        }else{
            Application->MessageBox("Montagem concluída!", "Fim", MB_OK);
        }
    }
}
}

```

O bloco principal do programa abre um arquivo texto e escreve todo o conteúdo do Memo. Memo são caixas de textos expandidas, que contém lines. Essas lines são extraídas e gravadas em um arquivo texto que será usado para o processamento. O arquivo é reaberto. Primeiramente a instrução é lida e armazenada no vetor inst. Depois é verificado se a instrução existe. Se ela existir é

executado a função `instruction()` que grava no arquivo `CodBin` o código de máquina correspondente a instrução. Se ela não existir, uma mensagem de erro aparece para o usuário.

```
bool VerificaInst(void){
    bool verifica=false;
    if((inst[0]=='a')&&(inst[1]=='d')&&(inst[2]=='c')){
        verifica=true;
        return true;
    }
    if((inst[0]=='s')&&(inst[1]=='u')&&(inst[2]=='b')){
        verifica=true;
        return true;
    }
    if((inst[0]=='m')&&(inst[1]=='u')&&(inst[2]=='l')){
        verifica=true;
        return true;
    }
    if((inst[0]=='d')&&(inst[1]=='i')&&(inst[2]=='v')){
        verifica=true;
        return true;
    }
    if((inst[0]=='a')&&(inst[1]=='n')&&(inst[2]=='d')){
        verifica=true;
        return true;
    }
    if((inst[0]=='o')&&(inst[1]=='u')&&(inst[2]=='.')){
        verifica=true;
        return true;
    }
    if((inst[0]=='n')&&(inst[1]=='a')&&(inst[2]=='o')){
        verifica=true;
        return true;
    }
}
```

```

    }
    if((inst[0]=='s')&&(inst[1]=='i')&&(inst[2]=='m')){
        verifica=true;
        return true;
    }
    if(verifica==false){//Verifica se a instrução existe ou nao
        return false;
    }
}

```

A função VerificaInst() verifica se a instrução lida do arquivo existe. Ela é do tipo bool, ou seja tem como valor de retorno um dado lógico, verdadeiro ou falso. Caso a string armazenada no vetor inst[] não estiver dentro dessa função o programa retorna erro.

```

//Verifica a instrução e executa os parametros
void instruction(void){

//Abre o arquivo onde será gravado o conteudo em binario
ofstream txtBinario;
txtBinario.open("CodBin.txt", ios::app);

char bin[8]; // guardara o valor em binario
int numero;

if((inst[0]=='a')&&(inst[1]=='d')&&(inst[2]=='c')){
    if(variavel==true){
        txtFile2.get(c);

        numero=v;//Valor do acumulador
        itoa(numero,bin,2);//Converte para base dois o numero da operação
        txtBinario << bin;
        txtBinario << "\n";//Número em binario tradicional
    }
}

```

```

txtBinario << "00000001\n"; //Código da instrução

numero=convert(c); //Valor a ser operacionado.
itoa(numero,bin,2); //Converte para base dois o numero da operação
txtBinario << bin;
txtBinario << "\n";

v=v+convert(c);
txtBinario << "00100000\n"; //Botão de igual
txtBinario << "01000000\n"; //Botão de resete
txtFile2.get(c);
txtFile2.get(c);
}else{
    Application->MessageBox("Você tentou utilizar uma instrução operacional
sem declarar variável!", "Erro 003", MB_OK);
    Error=true;
}
}
if((inst[0]=='s') && (inst[1]=='u') && (inst[2]=='b')){
    if(variavel==true){
        txtFile2.get(c);

        numero=v; //Valor do acumulador
        itoa(numero,bin,2); //Converte para base dois o numero da operação
        txtBinario << bin;
        txtBinario << "\n"; //Número em binario tradicional

        txtBinario << "00000010\n"; //Código da instrução

        numero=convert(c); //Valor a ser operacionado.
        itoa(numero,bin,2); //Converte para base dois o numero da operação

```



```

txtBinario << bin;
txtBinario << "\n";

v=v-convert(c);
txtBinario << "00100000\n";//Botão de igual
txtBinario << "01000000\n";//Botão de reset
txtFile2.get(c);
txtFile2.get(c);
}else{
    Application->MessageBox("Você tentou utilizar uma instrução operacional
sem declarar variável!", "Erro 003", MB_OK);
    Error=true;
}
}
if((inst[0]=='m')&&(inst[1]=='u')&&(inst[2]=='l')){
    if(variavel==true){
        txtFile2.get(c);

        numero=v;//Valor do acumulador
        itoa(numero,bin,2);//Converte para base dois o numero da operação
        txtBinario << bin;
        txtBinario << "\n";//Número em binario tradicional

        txtBinario << "00000100\n";//Código da instrução

        numero=convert(c);//Valor a ser operacionado.
        itoa(numero,bin,2);//Converte para base dois o numero da operação
        txtBinario << bin;
        txtBinario << "\n";

        v=v*convert(c);
        txtBinario << "00100000\n";//Botão de igual
        txtBinario << "01000000\n";//Botão de reset

```

```

    txtFile2.get(c);
    txtFile2.get(c);
}else{
    Application->MessageBox("Você tentou utilizar uma instrução operacional
sem declarar variável!", "Erro 003", MB_OK);
    Error=true;
}
}
if((inst[0]=='d')&&(inst[1]=='i')&&(inst[2]=='v')){
    if(variavel==true){
        txtFile2.get(c);
        if(c=='0'){
            Application->MessageBox("Divisão por 0!", "Erro 004", MB_OK);
        }else{

            numero=v;//Valor do acumulador
            itoa(numero,bin,2);//Converte para base dois o numero da operação
            txtBinario << bin;
            txtBinario << "\n";//Número em binario tradicional

            txtBinario << "00001000\n";//Código da instrução

            numero=convert(c);//Valor a ser operacionado.
            itoa(numero,bin,2);//Converte para base dois o numero da operação
            txtBinario << bin;
            txtBinario << "\n";

            v=v/convert(c);
            txtBinario << "00100000\n";//Botão de igual
            txtBinario << "01000000\n";//Botão de reset
            txtFile2.get(c);
            txtFile2.get(c);
        }
    }
}

```

```

    }else{
        Application->MessageBox("Você tentou utilizar uma instrução operacional
sem declarar variável!", "Erro 003", MB_OK);
        Error=true;
    }
}

if((inst[0]=='a')&&(inst[1]=='n')&&(inst[2]=='d')){
    txtFile2.get(c);
    if((c<48) || (c>=52)){
        Application->MessageBox("Operação lógica inválida", "Erro 005",
MB_OK);
        Error=true;
    }else{

        numero=c;//Valor a ser operacionado.
        if(c=='0'){
            txtBinario << "00000000\n";
            txtBinario << "00010100\n";//Código da instrução
            txtBinario << "00000000\n";
        }else if(c=='1'){
            txtBinario << "00000001\n";
            txtBinario << "00010100\n";//Código da instrução
            txtBinario << "00000000\n";
        }else if(c=='2'){
            txtBinario << "00000000\n";
            txtBinario << "00010100\n";//Código da instrução
            txtBinario << "00000001\n";
        }else if(c=='3'){
            txtBinario << "00000001\n";
            txtBinario << "00010100\n";//Código da instrução
            txtBinario << "00000001\n";
        }
        txtBinario << "00100000\n";//Botão de igual
    }
}

```

```

        txtBinario << "01000000\n";//Botão de reset
    }
    txtFile2.get(c);
    txtFile2.get(c);
}
if((inst[0]=='o')&&(inst[1]=='u')&&(inst[2]=='.')){
    txtFile2.get(c);
    if((c<48) || (c>=52)){
        Application->MessageBox("Operação lógica inválida","Erro 005",
MB_OK);
        Error=true;
    }else{

        numero=c;//Valor a ser operacionado.
        if(c=='0'){
            txtBinario << "00000000\n";
            txtBinario << "00011000\n";//Código da instrução
            txtBinario << "00000000\n";
        }else if(c=='1'){
            txtBinario << "00000001\n";
            txtBinario << "00011000\n";//Código da instrução
            txtBinario << "00000000\n";
        }else if(c=='2'){
            txtBinario << "00000000\n";
            txtBinario << "00011000\n";//Código da instrução
            txtBinario << "00000001\n";
        }else if(c=='3'){
            txtBinario << "00000001\n";
            txtBinario << "00011000\n";//Código da instrução
            txtBinario << "00000001\n";
        }
        txtBinario << "00100000\n";//Botão de igual
        txtBinario << "01000000\n";//Botão de reset
    }
}

```

```

    }
    txtFile2.get(c);
    txtFile2.get(c);
}
if((inst[0]=='n')&&(inst[1]=='a')&&(inst[2]=='o')){
    txtFile2.get(c);
    if((c<48) || (c>=50)){
        Application->MessageBox("Operação lógica inválida","Erro 005",
MB_OK);
        Error=true;
    }else{
        numero=c;//Valor a ser operacionado.
        if(c=='0'){
            txtBinario << "00000000\n";
            txtBinario << "00010010\n";//Código da instrução
            txtBinario << "00000000\n";
        }else if(c=='1'){
            txtBinario << "00000001\n";
            txtBinario << "00010010\n";//Código da instrução
            txtBinario << "00000000\n";
        }
        txtBinario << "00100000\n";//Botão de igual
        txtBinario << "01000000\n";//Botão de reset
    }
    txtFile2.get(c);
    txtFile2.get(c);
}
if((inst[0]=='s')&&(inst[1]=='i')&&(inst[2]=='m')){
    txtFile2.get(c);
    if((c<48) || (c>=50)){
        Application->MessageBox("Operação lógica inválida","Erro 005",
MB_OK);
        Error=true;
    }

```

```

}else{

    numero=c;//Valor a ser operacionado.
    if(c=='0'){
        txtBinario << "00000000\n";
        txtBinario << "00010001\n";//Binário da instrução
        txtBinario << "00000000\n";
    }else if(c=='1'){
        txtBinario << "00000001\n";
        txtBinario << "00010001\n";//Binário da instrução
        txtBinario << "00000000\n";
    }
    txtBinario << "00100000\n";//Botão de igual
    txtBinario << "01000000\n";//Botão de reset
}
txtFile2.get(c);
txtFile2.get(c);
}

txtBinario.close();

}

```

A função acima escreve no arquivo texto o código de máquina correspondente a cada instrução. Ao lado de cada linha de escrita, há um comentário do que cada binário significa.

Durante o processo de programação podem ocorrer alguns erros. A tabela abaixo lista esses erros e suas descrições.

Tabela 7 - Erros e Descrições

Número do erro	Mensagem
000	Arquivo em branco
001	Instrução não reconhecida
002	Espaço não encontrado

	<i>depois da instrução</i>
003	<i>Você tentou utilizar uma instrução operacional sem declarar variável</i>
004	<i>Divisão por zero</i>
005	<i>Operação Lógica inválida</i>
---	<i>Montagem concluída com erro na instrução (999)</i>
---	<i>Montagem concluída</i>

Fonte: Autoria própria.

2.3.9.1.2 Gravação

Gravar a memória EEPROM é feita com gravadores já existentes no mercado. Um arquivo binário gerado pelo Cloud Code Assembler é usado para gerar o arquivo de gravação. O programa Hex Editor Neo é um gerador de tipos de arquivos com diversas extensões .bin, .hex, .dll. Ele cria automaticamente um arquivo que pode ser gravado na EEPROM.

2.4 Execução

Um teclado com 15 teclas são conectados nas respectivas entradas: Entradas de dados, entradas de operação, entrada do botão igual e entrada de resete. Ao ligar o processador ele esperará até que uma instrução seja dada a ele. A primeira informação lida é o primeiro operando. Depois inserimos a instrução que queremos(soma, subtração, multiplicação, divisão, sim, não, e ou ou). Depois colocamos o segundo operando. Ele esperará para processar até que o registrador do sinal de igual. A operação será processada e as saídas serão ligadas de acordo com o resultado calculado.

Tabela 8 - Características Técnicas do Processador

Características	Valor
-----------------	-------

Tensão de alimentação	5V
Corrente elétrica	500mA
Número de conexões	28
Clock	1 Hz
Via de Dados	8 bits
Memória de Dados	8 bits
Memória de Programa	2K
Número de Registradores	4
Interrupção Externa	Parada de clock

Fonte: Autoria própria.

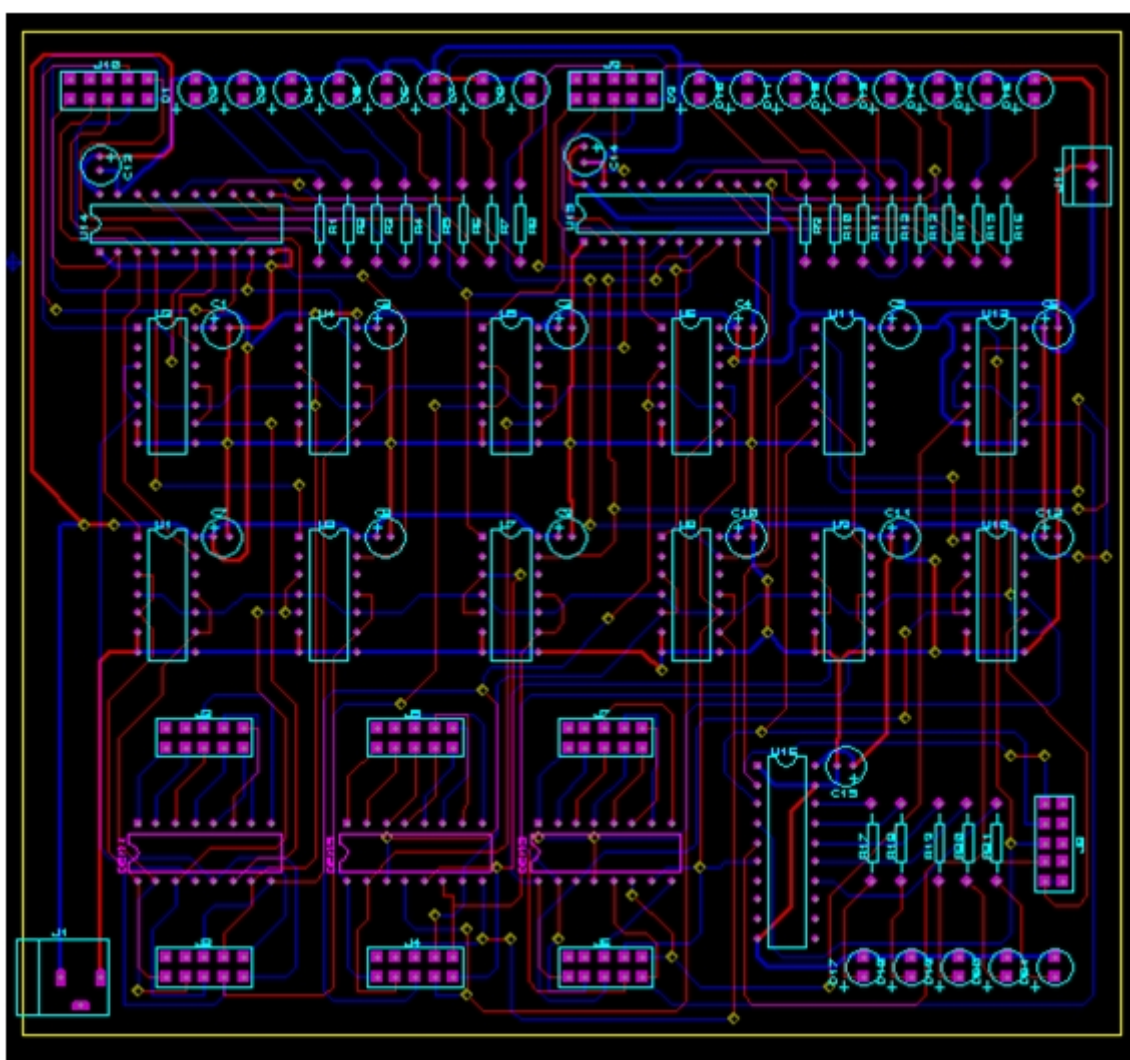


Figura 7 - Layout da placa da entrada
Fonte: Autoria própria.

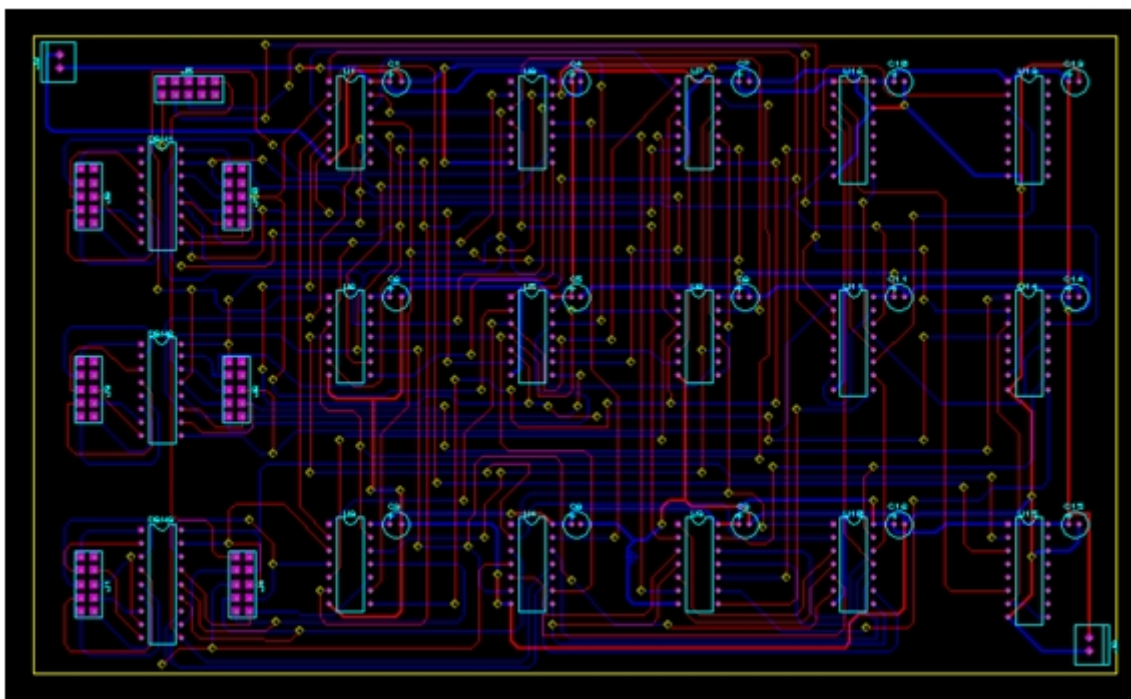


Figura 8 - Layout da placa da ULA
Fonte: Autoria própria.

Tabela 9 - Componentes e Preços

Componente	Quantidade	Preço(R\$)	Preço Total(R\$)
Portas Lógicas	79	1,30	26,00
Portas Lógicas OU	37	1,30	13,00
Portas Lógicas XOR	16	1,30	5,20
Portas Lógicas NOT	39	1,30	13,00
DEMUX 4555	20	1,80	36,00
MUX 4053	12	0,80	9,60
MUX 4052	12	0,80	9,60
Oscilador 555	1	0,70	0,70
Flip Flop JK 4027	23	1,40	16,80
Flip Flop D 4013	25	0,90	11,70
Comparador 4585	2	1,40	2,80
EEPROM 28c16	1	9,40	9,40
LED's	26	0,15	3,90
Resistores 470	46	0,20	9,20
MUX 74238	1	0,80	0,80
Chave de 16 terminais	2	1,50	3,00
Botão Tátil	18	1,00	18,00
Capacitor 40uF	1	0,50	0,50
Capacitor 1uF	1	0,50	0,50
FotoDrill	1	40,00	40,00
Placa Dual Layer	2	9,00	18,00
Preço Total	----	----	247,70

Fonte: Autoria própria.

2.5 Melhorias Futuras

Fazer as placas de todos os circuitos, criar uma linguagem de alto nível, ampliar a via de dados.

3 CONCLUSÃO

Podemos analisar como o processador funciona, assim podemos compreender melhor os circuitos integrados e como manipular os bits, para realizar operações aritméticas. Conseguimos uma noção o que o processador faz com as instruções que lhe é dada. Ter uma experiencia de como construir placas de circuito impresso, e utilizar o software Protheus. Obtivemos um amplo conhecimento de eletrônica digital.

REFERÊNCIAS

Comparador binário de 4 bits. Disponível em:

<http://pdf.datasheetcatalog.net/datasheets2/15/159699_1.pdf> Acesso em: 30 set. 2013

Computador - Wikipedia. Disponível em: <<http://pt.wikipedia.org/wiki/Computador>> Acesso em: 9 set. 2013

Eletrônica Digital - Demultiplexadores. Disponível em:

<http://www.eletronicadigital.xpg.com.br/digi_aula11_11_2.htm#Demultiplexadores> Acesso em: 7 out. 2013

Eletrônica Digital - Multiplexadores. Disponível em:

<http://www.eletronicadigital.xpg.com.br/digi_aula10_11_2.htm#Multiplexadores> Acesso em: 7 out. 2013

Evolução dos Circuitos Integrados. Disponível em:

<<http://www.guanabara.info/wp-content/uploads/2009/07/circuitos-integrados.jpg>> Acesso em: 1 nov. 2013

Foto Optica do Processador 8086 da Intel. Disponível em:

<http://www.electronics-lab.com/blog/wp-content/uploads/2010/09/visual_6502.jpg> Acesso em: 21 nov. 2013

Multiplexadores. Disponível em: <<http://pt.wikipedia.org/wiki/Multiplexador>> Acesso em: 4 out. 2013

Multiplexadores e Demultiplexadores. Disponível em:

<<http://www.newtoncbraga.com.br/index.php/como-funciona/1214-art0159>> Acesso em: 3 out. 2013

Operações aritméticas com números sinalizados. Disponível em:

<http://pt.wikipedia.org/wiki/Opera%C3%A7%C3%B5es_aritm%C3%A9ticas_com_n%C3%BAmeros_bin%C3%A1rios_sinalizados> Acesso em: 18 set. 2013

Porta lógica. Disponível em: <http://pt.wikipedia.org/wiki/Porta_l%C3%B3gica>

Acesso em: 4 out. 2013

Programar em C++. Disponível em:

<http://pt.wikibooks.org/wiki/Programar_em_C%2B%2B/Entrada_e_saida_de_dados_2> Acesso em: 9 set. 2013

Representação de números com sinal. Disponível em:

<http://pt.wikipedia.org/wiki/Representa%C3%A7%C3%A3o_de_n%C3%BAmeros_com_sinal> Acesso em: 18 set. 2013

Sistema de numeração binária. Disponível em:

<<http://www.brasilescola.com/matematica/sistema-numeracao-binaria.htm>> Acesso em: 3 out. 2013

The simple datapath with the control unit. Disponível em:

<http://dc357.4shared.com/doc/K3XaP1lo/preview_html_m36bfe485.png> Acesso em: 21 nov. 2013