
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

# Relatório do trabalho de UART

**Arquitetura e Organização de Computadores II**

**Autores:**

David Machado - 475664

Antonio César - 473444

Anderson Moura - 473070

**Professor:** Roberto Cabral

Quixadá - Ceará
01/04/21

# Conteúdo

# 1 Introdução

O universal asynchronous receiver-transmitter (UART), é um dos mais usados protocolos de transmissão entre dispositivos. Varias aplicações de eletrônica e comunicações envolvem o uso desse protocolo. Ele não é como um protocolo comum de redes de computadores, ele é um componente de hardware de um dispositivo ou sozinho, e o seu principal objetivo é a transmissão e recebimento. O nome assíncrono do UART vem do fato que ele não se utilizar de um clock para realizar a transmissão dos dados, em vez disso ele utiliza de bits de controle para simbolizar quando um dado começa ou termina como mostrado a seguir:

| | | |
|---|---|---|
| 0 | BYTE | 1 |

Como pode ser visto, a transmissão começa com um bit zero, ele indica o começo de uma transmissão de um byte, enquanto o bit 1 significa o fim de uma. O trabalho proposto é simular essa comunicação serial implementado em um código em ARM assembly, fazendo uma conversão de um texto em uma comunicação UART no formato de um arquivo binário e o recebendo um arquivo texto no que possui bits que representam uma comunicação UART e converter em um texto.

# 2 Organização e estrutura

Em ARM assembly, se pode definir secções de código e dados, o primeiro é utilizado .text para definir e o segundo .data. O programa possui essas duas secções e a seguir é explicado o que possui em cada umas delas:

## 2.1 Section .data

Na section .data é onde foi definido as variáveis e constantes do programa. Uma explicação de cada variável e constante no programa:

- **user_input**: input do usuário para escolher menu inicial, possuindo um byte de memória pois a chamada de sistema que faz a leitura do teclado retorna um char.
- **buffer**: variável para receber o char dos arquivos, possui um byte de armazenamento.
- **menu_str**: string que representa o menu inicial do programa.
- **size_menu**: constante com o tamanho em bytes da string menu_str.
- **exit_str**: string para indicar que o programa acabou.
- **size_exit**: constante com tamanho em bytes da string exit_str.
- **error_str**: string para indicar que houver error.
- **size_error**: tamanho em bytes da string de error_str.
- **clear_str**: string que contém a palavra clear que vai ser usado como argumento para a função system, para fazer limpar a tela do terminal.

- **bufOut**: Variável para receber os bytes do arquivos, possui um byte de armazenamento.
- **control**: Variável para receber os bits de controle do do protocolo UART.

### 2.2 Section .text

A secção de código é pode ser separado em três partes: o menu inicial que é a interação com o usuário, a opção que é a conversão de um texto que se encontra em um arquivo txt para um arquivo bin que possui a representação em bits de como seria uma transmissão UART daquele arquivo. A opção dois que é receber um arquivo binário que representa uma transmissão de dados usando o protocolo UART é converter em um texto num arquivo de saída do tipo txt. A primeira parte do programa é implementada utilizando chamadas de sistemas SWI e as duas opções de conversão, utilizam de funções implementadas para sua utilização.

## 3 Arquivos

O código trabalha com dois arquivos tipo texto e um arquivo tipo binário, nomeados de: text_input.txt, UART.bin e text_output.txt. Os modo conversão escolhido vai definir qual dos arquivos vai ser usado como entrada ou como saída. Em caso de converter de texto para UART, o arquivo text_input.txt vai ser a entrada e o UART.bin vai ser o retorno. O modo de converter de UART e texto, tem como entrada o arquivo UART.bin e o text_output.txt como saída.

O programa para poder interagir com os arquivos utiliza de chamadas de sistemas, utilizando a instrução swi. Em texto para UART, se tem uma chamada de sistema para leitura do arquivo text_input.txt e uma chamada para escrever em UART.bin, e essa chamada para o UART tem umas condições para caso: UART.bin não existir, criar um e apagar as informações que existiam antes caso o UART.bin já ter sido utilizado anteriormente. Em UART para texto, se tem uma chamada de sistema para leitura do arquivo UART.bin e uma chamada para escrever em text_output.txt, e essa chamada para o UART tem umas condições para caso: text_output.txt não existir, criar um e apagar as informações que existiam antes caso o text_output.txt já ter sido utilizado anteriormente.

## 4 Chamadas essenciais

- **_openIn**: Abre arquivo de entrada apenas para leitura (flag 0x0 - O_RDONLY) e retorna o descritor do arquivo de entrada em r0.
- **_openOut**: Abre arquivo de saída apenas para leitura (flag 0x1 - O_WRONLY), truncando o seu tamanho para 0 (flag 0x200 - O_TRUNC). Se o arquivo de saída não existir,então é criado outro (flag 0x40 - O_CREAT) com todas as permissões (0x1FF). Retorna fd do arquivo de saída em r0.
- **_close**: Fecha arquivos abertos.
- **_readByte**: Ler apenas um byte de um arquivo e retorna a quantidade de Bytes lido.

- **_writeByte**: Escreve apenas um byte no arquivo de saída e retorna a quantidade de Bytes escritos.
- **_print**: Imprime algo no terminal. Entretanto, o descrito do arquivo, a mensagem e quantidade de dados são colocados em seus respectivos registradores antes de chegar aqui.
- **errorOpen**: Imprime mensagem de erro ao abrir o arquivo entrada e finaliza a execução.
- **errorUart**: Imprime mensagem de erro por ter menos bits do que 8 para compor o char de Uart para txt.
- **_exit**: Finaliza execução.

# 5 Modos de operação

## 5.1 Texto para UART

Opção 1 começa abrindo o arquivo input_txt, que é o primeiro arquivo de entrada onde está o texto e salvar o descritor desse arquivo em InFileFd. Depois abre o arquivo de saída uart, onde vai ser colocado o texto transformado em binário e seu descritor é salvo em OutFileFd. Por último, é lido um byte do arquivo de entrada e a função de conversão e escrita, no arquivo binário, é chamada.

Em asciiUart, a primeira coisa a ser feita salvar o contexto atual do programa, para que quando o byte lido terminar de ser convertido, o próximo byte possa ser lido. Depois é colocado 0 em r2, esse será o primeiro bit de controle, logo depois é chamada a label bit Control.

Em bitControl, primeiro, é salvo o contexto atual do código, depois o conteúdo de r2 é salvo em control para a função de impressão ser chamada, por fim o contexto anterior é recuperado para depois da chamada de bitControl.

Voltando para asciiUart, o byte lido será salvo em r6 e em r3 será colocado 8. Então a execução entrará em um loop chamado l1_asciiUart. Nesse loop, primeiro, é feito um and lógico entre o valor em r6 e 1, essa operação vai revelar e salvar em r1 o bit menos significativo de r6, depois em bit é impresso no arquivo de saída. Por fim, é feito um deslocamento para direita em r6 para quer o próximo bit fique na posição certa de ser lida, depois r3 é subtraído em 1. Esse loop continuará executando enquanto r3 for maior que 0, pois serão impressos 8 bits.

A última parte de asciiUart, chamada de end_l1_asciiUart, vai imprimir o segundo bit de controle e retornar ao contexto salva no começo de asciiUart, ou seja, voltará para o laço readValue_opt1 para que o próximo byte do arquivo de entrada seja lido.

```
Abre arquivo de entrada "input_txt";
if r0 é menor que 0 then
    imprime mensagem de erro;
else
    salva r0 em InFileFd;
    Abre arquivo de saída "uart";
    Salva r0 em OutFileFd;
    while r0 diferente de 0 do
        Ler 1 byte do arquivo de entrada;
        Pula para "Imprime 0"
    end
    Imprime 0;
    Salva o byte lido em r6;
    r3 ← 8;
    while r3 maior que 0 do
        r6 and 1;
        Imprime o bit encontrado;
        r6 » 1;
        r3 ← r3 − 1;
    end
    Imprime 1;
    Pula para o primeiro while;
end
```

**Algorithm 1:** Lógica para opt1

## 5.2 UART para texto

Opção 2 começa abrindo o arquivo uart, que é o arquivo binário de entrada onde está os bits que serão convertidos para ascii e em seguida salva o descritor desse arquivo em InFileFd. Depois abre o arquivo de saída output_txt, onde vai ser colocado o binário transformado em texto e em seguida seu descritor é salvo em OutFileFd. Por último, é lido um byte do arquivo uart, que corresponde ao bit de controle 0 e a função de conversão (uartAscii) é chamada.

Essa função basicamente converte 8 bits em um carácter ascii. Ela consiste em ler 0 ou 1 em cada interação de um laço que será controlado por r3, indo de 0 até 7. Logo, r3 se comporta como o contador desse laço, incrementando em 1 enquanto for menor do que 8.

Em cada interação, após a leitura do bit que é colocado em buffer e dentro do laço carregado para r1, é carregado para o registrador r6 o endereço de bufOut, que como o próprio nome já sugere trata-se de um buffer de saída que será utilizado para imprimir o carácter ascii no arquivo de saída. Além disso, em cada interação o valor em r1 será descolado 'r3' vezes para esquerda e adicionado em r6, para que com isso seja implementado cada bit em seu respectivo nível de significância.

Lembrando que os bits presentes no arquivo binário estão em uma ordem inversa, por isso necessitando que o primeiro valor lido seja colocado no bit 0 de bufOut, o segundo valor lido seja colocado no bit 2 de bufOut, o terceiro no bit 3 e assim por diante até que o último valor lido (0 ou 1), quando r3 é igual a 7, esteja na posição do bit 7 de bufOut.

Fora do laço é escrito no arquivo de saída o valor presente em bufOut e feito a leitura de um novo dado presente no arquivo binário para que seja lido o bit de controle 1 e seja possível proceder para uma nova conversão.

```
Abre arquivo binário de entrada "uart";
if r0 é menor que 0 then
    imprime mensagem de erro: Sem arquivo de entrada;
else
    salva r0 em InFileFd;
    Abre arquivo de saída "output_txt";
    Salva r0 em OutFileFd;
    while ler algo do
        @Realizou a leitura do bit de controle de inicio (0);
        Pula para "uartAscii"
    end
end
```

**Algorithm 2:** Lógica para opt2

```
Data: 0 10000110 1
Result: caracter ascii: 01100001
inicialização;
r3 ⟵ 0;
r5 ⟵ bufOut;
limpa bufOut;
while r3 < 8 do
    ler um novo valor e salva em buffer;
    if leu algo then
        r1 ⟵ [buffer];
        r6 ⟵ [r5];
        r1 ⟵ r1 AND 1;
        r6 ⟵ r6 + (r1 << r3);
        salva r6 em bufOut;
    else
        Erro: Falta um ou mais bits dos 8 necessário;
    end
    r3 ⟵ r3 + 1;
end
escreve bufOut no arquivo de saída;
ler o bit de controle 1;
```

**Algorithm 3:** Função convertendo uart para ascii (uartAscii)

# 6 Compilação e execução

O trabalho vem acompanhado de um Makefile que permite a execução dos códigos de maneira simples, basta executar o comando make que vai ser feita a compilação e gerar o executável exe. Antes de executar o comando ./exe, necessário que o nome do arquivo de entrada seja text_input e o seja um .txt, caso for executar o modo de texto para UART. Caso for para fazer uma execução de UART para texto, é necessário que o nome do arquivo com os bits UART seja UART e o tipo dele .bin. Com isso visto, pode executar o programa com ./exe.
Comandos para executar o programa com o Makefile:

```
$ make
$ ./exe
```

Agora para compilar sem o Makefile

```
$ as -o main.o main.s
$ gcc -o main.o exe
$ ./exec$
```

No inicio do programa vai aparecer um menu com três opções: txt para UART, UART para txt e sair. Aperte 1 para converter texto para UART e gerar/alterar o output UART.bin, aperte 2 para converter de UART para texto gerar/alterar o output text_output.txt ou aperte 3 para sair do programa. Quando o programa for executando, vai ser mostrado uma mensagem que o foi terminado o processo com sucesso e que foi finalizado com sucesso.

# 7 Conclusão

Foi feita a conversão completa de um texto em ascii, em um arquivo de entrada, para uart e colocado em um arquivo binário, e de uart para ascii. Adquirimos mais experiência em ARM e entendemos melhor o protocolo UART.