UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM **ENGENHARIA DE COMPUTAÇÃO**
QUARTO SEMESTRE

UARTsim - UM SIMULADOR DE DUPLA DECODIFICAÇÃO DO PROTOCOLO UART

PEDRO HENRIQUE **MAGALHÃES BOTELHO**

FRANCISCO ÍTALO **DE ANDRADE MORAES**

QUIXADÁ - CE
2021

471047 — PEDRO HENRIQUE **MAGALHÃES BOTELHO**

472012 — FRANCISCO ÍTALO **DE ANDRADE MORAES**

UARTsim - UM SIMULADOR DE DUPLA DECODIFICAÇÃO DO PROTOCOLO UART

Orientador: Prof. Msc. Roberto Cabral Rabêlo Filho

Trabalho escrito apresentado no curso de graduação em Engenharia de Computação, pela Universidade Federal do Ceará, campus em Quixadá, para a disciplina de Arquitetura e Organização de Computadores II.

QUIXADÁ - CE
2021

Conteúdo

1 Introdução

Este é o relatório do segundo trabalho da disciplina de **Arquitetura e Organização de Computadores II**, do quarto semestre do curso de graduação em Engenharia de Computação, na Universidade Federal do Ceará, campus de Quixadá.

Nas seções seguintes serão tratados tópicos essenciais na concepção e implementação do tema proposto pelo orientador, a fim de relatar os acontecimentos no decorrer do trabalho, assim como os resultados finais obtidos.

1.1 Proposta

O trabalho proposto consiste na implementação de um programa em ARM-Assembly que irá "simular"o protocolo **UART(Universal Asynchronous Receiver/Transmitter)**, simulando a comunicação entre um pino TX e um pino RX. O trabalho visa a codificação de um caractere no padrão **ASCII** para o padrão UART, para que o caractere possa ser transmitido sob o referido protocolo, bem como a codificação de um dado do padrão **UART** para o padrão ASCII, para que o caractere possa ser imprimido na tela de forma correta.

1.2 Projeto

O projeto foi devidamente testado e implementado para atender aos requisitos e a proposta do orientador. O título foi escolhido baseado no fato de o programa simular a execução do protocolo UART, e *dupla decodificação* pois o programa faz tanto a tradução ASCII-UART quanto UART-ASCII.

O programa conta com um menu inicial que oferece duas opções: a função de codificação dos dados de entrada do padrão **ASCII** para o padrão **UART** e uma função oposta, que é decodificar os dados de entrada do padrão **UART** para o padrão **ASCII**.

São utilizadas manobras, nos procedimentos, para evitar erros, arquivos inexistentes, nomes de arquivos inválidos, opção selecionada inválida, onde estes são tratados no programa, apenas ao pedir ao usuário que repita o procedimento. Um erro em específico não pode ser tratado no programa: pulsos UART inválidos, sendo eles pequenos demais, por exemplo. Na interface, erros são evidenciados por textos na cor vermelha, enquanto mensagens normais estão na cor verde. Erros não tratáveis são evidenciados também por mostrarem a frase **Fatal Error**.

É possível acessar o repositório do projeto[1] e ter acesso a seus arquivos: tanto o executável quanto o código fonte.

2 Desenvolvimento

Nessa seção serão apresentados assuntos relativos à concepção e desenvolvimento do projeto. Cada subseção, e seus tópicos, abordarão um tema diferente, levando sempre em conta todos os aspectos possíveis do desenrolar da construção do projeto, podendo assim esclarecer o funcionamento do programa, bem como a implementação e o planejamento empregado, que foi seguido estritamente.

2.1 Planejamento

O planejamento é parte essencial de um projeto, mesmo que seja individual. A presença ativa de companheiros requer um planejamento mais rígido, que funcione não somente para um membro, mas para todos. Reuniões, padronização, ferramentas, tudo irá interferir em um planejamento, mesmo que de forma positiva.

Os tópicos a seguir irão dissertar em torno da questão:

"Como se deu o planejamento do projeto, tendo em vista as dificuldades, sejam elas de comunicação ou técnicas?"

2.1.1 Ferramentas

Tendo em vista que os integrantes da equipe moram em cidades diferentes, foram utilizadas diversas ferramentas para trabalho compartilhado de forma online, no intuito de auxiliar no desenvolvimento do projeto. A padronização de ferramentas teve papel fundamental na concepção do projeto.

- Editor de Código: **Visual Studio Code**
- Compilador: **GCC**
- Linguagem de Programação: **ARM Assembly**
- Depurador: **GDB Debugger**
- Automação de Compilação: **Makefile**
- Editor de Documentos: **LaTeX**
- Controle de Versões e Repositório: **GitHub**
- Comunicação Síncrona: **WhatsApp**
- Videoconferência: **Meet**

2.1.2 Equipe

A fim de realizar com êxito o trabalho proposto pelo professor, foi decidido, a princípio, que seria realizada uma primeira reunião para o entendimento da lógica do trabalho e o que seria feito, além de dividir as tarefas entre os membros da equipe. Foi criado um grupo no *WhatsApp* para facilitar a comunicação entre os membros, trocar ideias e sanar dúvidas, além de marcar semanalmente as reuniões para acompanhar o andamento do projeto. Ficou acertado que, primeiramente, seria feita a implementação do programa para depois fazer o relatório.

2.1.3 Dificuldades

No decorrer do trabalho, algumas dificuldades foram travadas pela equipe. No que diz respeito a fatores externos, provavelmente uma dificuldade a ser relatada foi na organização de horários específicos para a implementação do código em equipe, já que com o fim do semestre, os horários estavam muito apertados, então muito do trabalho foi feito individualmente.

A linguagem usada foi a ARM-Assembly, como foi proposta pelo professor, para entendermos melhor como o protocolo funciona.

Com relação ao trabalho em si, uma das dificuldade encontradas foi arranjar horários para os encontros, visto que como foi antes mencionado, os fins de semestre costumam ser apertados. Porém isso não impediu a realização do trabalho por parte da equipe.

A modalidade de ensino remoto também foi um desafio para a realização deste trabalho pela equipe, visto que a interação presencial tem muita importância no aprendizado e desempenho em equipe. Impossibilitados desse recurso, tanto por causa do contexto de pandemia como pelo fato dos dois membros da equipe morarem em cidades diferentes, foi preciso optar por encontros síncronos via *Google Meet* para compensar tal necessidade.

2.2 Funcionamento

O programa foi projetado em ARM-Assembly, logo pode ser compilado e executado em qualquer sistema operacional com suporte ao compilador GCC, visto que há suporte para bibliotecas em C nesta linguagem. Nessa subseção serão tratados alguns tópicos referentes ao funcionamento do programa após sua compilação ser efetuada, ou seja, o programa executável em si.

Para que o programa possa ser executado ele deve ser devidamente compilado para o sistema operacional desejado. Um arquivo **Makefile** facilita esse processo, automatizando a compilação. Tudo a se fazer é ir na pasta raiz do projeto, e pelo terminal escrever **make it**, comando esse que ordenará a compilação do programa, **make run** para executar o programa, e caso seja necessário *depurar* o programa, usa-se o comando **make debug**. O Makefile auxilia principalmente na organização de arquivos. O repositório do projeto tem seis pastas, onde cada uma guarda arquivos importantes:

- Arquivos binários, os executáveis, ficam em bin/
- Arquivos objetos ficam em build/
- Documentações, como este relatório e slides, ficam em doc/
- Arquivos de cabeçalho(*.lib) ficam em inc/
- Exemplos ficam em examples/
- Código-fonte a ser compilado fica em src/

O Makefile, então, irá compilar o código, e separar os componentes em pastas. Irá pegar os arquivos de código-fonte em src/ e o arquivo de cabeçalho em inc/, e ao compila-los irá salvar os arquivos objeto em build/ e o binário final em bin/.

Como exposto, a estrutura do makefile é simples.

O programa irá realizar traduções em duas vias: tradução do padrão ASCII para o padrão UART e vice-versa.
O menu de opções dá ao usuário a opção de traduzir ou de um modo, ou de outro. O menu é composto de duas opções, sendo a opção número 2 a de traduzir os caracteres ASCII para o padrão UART, e a opção número 1 traduz a sequência de bits do padrão UART para o padrão ASCII, assim sendo possível que a mensagem seja impressa na tela. Se o usuário não apertar os números correspondentes às duas opções, a interface entrará em *loop* até que uma das opções válidas sejam pressionadas.
O programa então pede ao usuário que insira o nome dos arquivos de entrada e saída para que a operação possa se concretizar. E de novo, se a resposta dada pelo usuário não for a esperada pelo sistema, sua interface entrará em *loop* até que uma saída válida seja dada.
Após isso, o programa executará sua função, usando uma sub-rotina diferente em cada situação. Se o usuário optar por passar uma sequência de caracteres ASCII para o padrão UART,

então a função usada será a de tradução dos pulsos UART para caracteres ASCII. Caso a opção 2 seja a escolhida, então será ativada uma rotina que irá transformar cada caractere ASCII numa sequência de bits ordenados segundo o protocolo ASCII. O resultado então é colocado em uma posição de memória, e passado a um arquivo ao final.

É interessante que os arquivos de entrada não tenham quebras de linhas, pois elas, de toda forma, serão ignoradas com o decorrer do programa e não serão processadas.

2.3 Implementação

Como já dito antes, o código foi totalmente escrito na Assembly para ARM. Usando do compilador GCC para compilar o código, o programa tem 3 arquivos de código, sendo uma principal e duas de sub-rotinas, e dois *headers*, usados pelos arquivos de sub-rotinas.

Nessa subseção será tratado a implementação do algoritmo de tradução, bem como a utilização do Assembly para alcançar os propósitos desse trabalho.

2.3.1 Procedimento _decodeUART

Iniciado caso a opção selecionada pelo usuário seja (1), decodifica pulsos UART descritos em uma string(previamente retirada de um arquivo de entrada por meio do procedimento _fileToString) em caracteres ASCII. Basicamente é verificado se há quebra de linhas(se sim, são ignoradas), se os bits de controle estão posicionados corretamente, se os caracteres no arquivo estão corretos(caracteres 0 ou 1), e, por meio de um contador, é possível converter os 10 caracteres de um código UART em apenas um caractere ASCII. Há uma string s de onde os caracteres do código UART são retirados, um contador *count* que inicia com 10 e a cada interação decresce, um valor *uart* que guarda cada código retirado de s, e um valor final *ascii*, que guarda o valor final em ASCII. Logo, após todas as verificações:

Seja um fator de deslocamento *shift_factor*:

$shift\_factor = 8 - count$

$uart = uart \ \& \ 1$

$uart = uart \ll shift\_factor$

$ascii = ascii \mid uart$

2.3.2 Procedimento _decodeCHAR

Iniciado caso a opção selecionada pelo usuário seja (2), decodifica caracteres ASCII descritos em uma string em pulsos UART. Realiza o processo inverso do procedimento tratado anteriormente: percorre uma string, transformando caracteres em códigos UART.

2.3.3 Padronização de Código

O código foi implementado no formato de cinco arquivos: dois arquivos de cabeçalho, dois arquivos de funções e uma função principal. A função principal é responsável pela interação com o usuário, enquanto os arquivos de funções que contém as sub-rotinas se encarrregam da operação de tradução, verificação de erros, etc... O código apresenta a padronização de cores ANSI, com a cor verde para textos normais e a cor vermelha para textos de erro, bem como utilização de diretivas de pré-processamento.

3 Conclusão

Este trabalho foi bastante útil no que diz respeito a prática no manuseio do código em Assembly, bem como o entendimento do modo de operação do padrão UART, o que para nós do curso de Engenharia de Computação é muito importante, pois é usado por algumas placas de programação.

A experiência no presente trabalho foi desafiadora e ao mesmo tempo divertida, já que foi preciso estudar algumas instruções no ARM, mas que serviu para aprimorarmos mais a nossa capacidade de programação nessa linguagem. Sendo assim, a realização deste trabalho foi bastante proveitosa.

Referências

[1] P. Botelho, I. Andrade. *Repositório UARTsim.*