UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Relatório projeto 1: Jogo em Assembly

João Vitor Belmonte Rates

Disciplina: Organização de Computadores [ELC1011]
Professor: Giovani Baratto

2022/2

1. Introdução

Este projeto propôs o desenvolvimento de um jogo da velha escrito em assembly MIPS em que o jogador tem como adversário a máquina. O objetivo deste projeto é de avaliar o conhecimento adquirido na disciplina de Organização de Computadores até o momento.

2. Desenvolvimento

2.1 Referencial Teórico

Para realização deste projeto, foi usado como, referencial teórico, os slides da disciplina, disponibilizados no Moodle, o livro Computer Organization and Desings, e a lista de *syscalls* do software MARS acessível no site oficial do mesmo.

2.2 Especificações do Programa

Ao iniciar o programa já será selecionado de forma aleatório que fará o primeiro movimento(jogador ou máquina), então o jogo seguirá até que a partida acabe, quando terminar será mostrado na tela o placar e o menu, que esperará que o usuário tecle "*e"*(*exit*) para finalizar o programa, ou tecle "n" (*new*) para jogar uma nova partida.

As células do tabuleiro são indexadas da esquerda para direita, de cima para baixo, iniciando de 0.

O Jogador sempre será o "X" e a maquina será "O".

2.3 Estrutura do Código

O código está dividido em diversos arquivos, um arquivo para cada função (exceto funções auxiliares), todas as funções seguem o esperado, receber os parâmetros nos registradores $a*, colocar a saída em $v* e manter os valores de $a* e $s* antes de fazer o retorno (*jump register*).

2.4 Padrão de Cabeçalho

Todos os arquivos possuem em seu cabeçalho a mesma estrutura de comentários.
Ex.:

main.asm

```
1  # This code is a constituent part of work 1 of the Computer Organization Discipline [ELC1011]
2  # https://github.com/Jvbrates/TIC_TAC_TOE-MarsMips/
3  # This program is free software under GNU GPL V3 or later version
4  # see http://www.gnu.org/licences
5  #
6  # Autor: João Vitor Belmonte Rates(Jvbrates) - UFSM - CT
7  # e-mail: jvrates%inf.ufsm.br
8
9  # 0/14
10 # Prólogo:
11 # Este arquivo contèm a main(função raiz) do projeto
12 # Prologue:
13 # This code is the main of project
14
15 #IsCaller? Yes
16 #IsCallee? No
17 #ChangeRegisters? Yes
18 #ManipulateStack? No
19 #ManipulateHeap? No
20 #ManipulateDataSegment? Yes | Allocate 3 bytes for SCORE
21
22 ################################################################################
23 #        1         2         3         4         5         6         7         8
24 #2345678901234567890123456789012345678901234567890123456789012345678901234567890
25
```

Linha 1: Informação do projeto;
Linha 2: Endereço web para repositório do projeto no *github;*
Linha 3-4: Informações de licença;
Linha 6-7: Informações de Contato;
Linha 9: Numeração do arquivo;
Linha 10-13: Prólogo, descrição do trecho do código;
Linha 15-20: Informações úteis para facilitar o *debugging*;
Linhas 22-24: Numeração de Colunas.

2.5 Funções auxiliares

Devido à necessidade em várias funções armazenar e restaurar os registradores $a*, $s* e $ra da *stack* foram criadas duas funções auxiliares, *stack_push* e *stack_pop*, facilitando a escrita do código e diminui a ocorrência de erros.

```
.text
.globl stack_push
stack_push:
addi $sp, $sp, -48
sw $s7, 44($sp)
sw $s6, 40($sp)
sw $s5, 36($sp)
sw $s4, 32($sp)
sw $s3, 28($sp)
sw $s2, 24($sp)
sw $s1, 20($sp)
sw $s0, 16($sp)

sw $a3, 12($sp)
sw $a2, 8($sp)
sw $a1, 4($sp)
sw $a0, ($sp)

jr $ra
```

```
#restaura os registradores d
.globl stack_pop
stack_pop:
lw $s7, 44($sp)
lw $s6, 40($sp)
lw $s5, 36($sp)
lw $s4, 32($sp)
lw $s3, 28($sp)
lw $s2, 24($sp)
lw $s1, 20($sp)
lw $s0, 16($sp)

lw $a3, 12($sp)
lw $a2, 8($sp)
lw $a1, 4($sp)
lw $a0, ($sp)

addi $sp, $sp, 48
jr $ra
```

Como ambas funções alteram o registrador *$ra* o seu uso deve ser após o mesmo ser armazenado e antes de ser restaurado, ficando assim o *modelo*:

```
funcao_generica:
#ARMAZENAMENTO DOS REGISTRADORES
#----------------------
addi $sp, $sp, -4
sw $ra, ($sp)
jal stack_push
#----------------------

AQUI É IMPLEMENTADA A FUNÇÃO,
SEM PREOCUPAR-SE EM MANTER OS VALORES DOS REGISTRADORES



#RESTAURAÇÃO DOS REGISTRADORES
#----------------------
jal stack_pop
lw $ra, ($sp)
addi $sp, $sp, 4
#----------------------

jr $ra
```

3. Conclusão

Com o desenvolvimento deste projeto foi possível assegurar uma base sólida de conhecimento sobre a organização de computadores, compreendendo o fluxo de dados e a organização das informações na memória. Assim como, durante o desenvolvimento associou-se o que se estava escrevendo, com implementações semelhantes em linguagens de

alto nível (ex. funcionamento de escopo em C). Também é importante evidenciar que este projeto me instigou a pesquisar mais sobre esta área da computação.

4. Referências

MIPS syscall functions available in MARS. Disponível em: <https://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html>. Acessado em: 23 nov. 2022.

Moodle Presencial - UFSM: Material do Curso - ELC1011. Disponível em: <https://ead06.proj.ufsm.br/mod/folder/view.php?id=1800901>. Acessado em: 23 nov. 2022.

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization And Design The Hardware/Software Interface**. [s.l.] Cambridge, Ma Morgan Kaufman Publishers, 2018.