



**UNIVERSIDADE FEDERAL DE SANTA MARIA**  
**ORGANIZAÇÃO DE COMPUTADORES**  
**SISTEMAS DE INFORMAÇÃO**

**ISABELLA SAKIS**  
**RHAUANI WEBER AITA FAZUL**

**RELATÓRIO DO JOGO 21 EM LINGUAGEM ASSEMBLY**  
**PARA A CADEIRA ORGANIZAÇÃO DE COMPUTADORES**

Santa Maria, RS, Brasil  
Maio de 2016

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>03</b>
<b>2</b>	<b>OBJETIVOS .....</b>	<b>04</b>
2.1	OBJETIVO GERAL .....	04
2.2	OBJETIVOS ESPECÍFICOS .....	04
<b>3</b>	<b>REVISÃO BIBLIOGRÁFICA .....</b>	<b>05</b>
<b>4</b>	<b>METODOLOGIA .....</b>	<b>06</b>
<b>5</b>	<b>EXPERIMENTO .....</b>	<b>06</b>
<b>6</b>	<b>RESULTADO .....</b>	<b>10</b>
<b>7</b>	<b>DISCUSSÃO .....</b>	<b>10</b>
<b>8</b>	<b>CONCLUSÕES E PERSPECTIVAS .....</b>	<b>11</b>
	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>12</b>
	<b>APÊNDICES .....</b>	<b>13</b>
	<b>APÊNDICE A – CÓDIGOS-FONTE .....</b>	<b>14</b>

## 1. INTRODUÇÃO

O presente relatório foi escrito no intuito de relatar com detalhes o trabalho da cadeira de Organização de Computadores, no qual foi implementada uma simulação do jogo de dados 21, que consiste em uma partida de três rodadas, onde os jogadores possuem a sua disposição dois dados: um branco e um vermelho. Sempre ao final de cada partida, os jogadores podem escolher se querem jogar uma nova partida ou não.

O jogo possui dois módulos: Usuário Vs Usuário ou Usuário Vs Computador. A cada partida, é perguntado ao jogador qual módulo ele deseja jogar. Assim que o jogador escolher o módulo, o jogo inicia. No módulo Usuário Vs Usuário, a cada rodada, os jogadores tem a opção de escolherem se irão jogar os dados ou passar a vez. Já no módulo Usuário Vs Computador, além do jogador poder escolher se vai jogar o dado ou passar a vez, o computador faz a sua escolha sozinho, de uma maneira “inteligente”, onde é feita uma análise não só dos pontos que já possui, mas também dos pontos do seu adversário, para decidir qual escolha o deixará mais próximo de vencer o jogo.

O lançamento dos dados é simulado através de uma geração randômica de valores. Ambos os jogadores começam com 0 pontos. Se o jogador passar a vez, não joga nenhum dos dados. Se o jogador escolher jogar os dados, joga primeiro o dado branco. O valor do dado branco soma-se aos pontos que ele possui (por exemplo, se o jogador tinha 6 pontos e tirou 4 no dado branco, agora ele tem 10 pontos). Logo em seguida, o jogador deve jogar o dado vermelho.

Se o valor do dado vermelho for 6, este valor é duplicado e somado aos pontos que ele já possui (por exemplo, se o jogador tinha 10 pontos, após jogar o dado branco, e tirou 6 no dado vermelho ele agora tem  $10 + (2 \times 6) = 22$  pontos). Outro valor no dado vermelho é simplesmente somado aos pontos do jogador, assim como o dado branco. Uma rodada termina quando todos os jogadores fizerem sua ação (passar a vez ou jogar). Após as três rodadas, o jogador que passar de 21 pontos perde (se os dois passarem, o jogo empata), e caso nenhum dos jogadores passar dos 21 pontos, o que mais se aproximar desse valor ganha.

Pretende-se, com a elaboração deste trabalho, aprimorar os conhecimentos dos alunos envolvidos, auxiliando na absorção da matéria passada em sala de aula, visto que a prática é a melhor maneira para isso. O jogo 21 foi escolhido por ser considerado o mais apropriado para alcançar tal objetivo, pois envolve tanto raciocínio lógico quanto a maioria das instruções vistas em aula.

## 2. OBJETIVOS

### 2.1 OBJETIVO GERAL

A proposta de implementação relatada neste documento é a de realizar a simulação do jogo de dados 21 para dois jogadores (um deles podendo ser o próprio computador), no qual serão gerados aleatoriamente valores de 1 a 6 (faces de um dado), para simular o lançamento de dois dados, o branco e o vermelho, uma vez por rodada, em um total de três rodadas. Vence o jogador que mais se aproximar de 21 pontos, caso ambos passem dos 21, o jogo empata.

### 2.2 OBJETIVOS ESPECÍFICOS

- Implementar a simulação do jogo 21 para dois jogadores;
- Dividir a simulação em dois módulos: Usuário Vs Usuário e Usuário Vs Computador;
- Oferecer ao usuário, ou aos usuários, a opção de jogar novamente;
- Simular uma escolha inteligente para a jogada do computador no módulo Usuário Vs Computador, na qual ele deverá decidir qual opção será melhor para seu desempenho no jogo (jogar ou passar a vez);
- A cada rodada exibir a soma atual de pontos de ambos os jogadores e, ao final do jogo, exibir corretamente o resultado da partida: apontar corretamente o vencedor, caso haja, ou exibir o empate caso a soma dos pontos sejam iguais ou ambos passarem de 21 pontos;
- Separar o código em procedimentos;
- Utilizar *macros* e *eqv's* (equivalências) para melhor leitura e reutilização do código futuramente.

### 3. REVISÃO BIBLIOGRÁFICA

Um computador é uma máquina capaz de processar dados que são necessários para nós, usuários, pelos mais diversos motivos. Esse processamento de dados é realizado através da coleta e manipulação de dados, para depois fornecer as informações, que são os resultados dessa manipulação, para o usuário.

Os termos dado e informação podem ser tratados como sinônimos, mas também podem ser usados de forma distinta. O termo “dado”, normalmente, é usado para definir a matéria-prima originalmente obtida e, a expressão “informação” é usada, normalmente, para definir o resultado do processamento, ou seja, o dado processado (MONTEIRO, 2007).

Os primeiros programadores se comunicavam com os computadores em números binários, tarefa maçante que logo foi substituída por novas notações mais parecidas com a maneira como os humanos pensam. Essas notações, no início, eram traduzidas para binário manualmente, o que era cansativo demais. A solução então foi usar a própria máquina para ajudar a programá-la, foram inventados então, programas para traduzir da notação simbólica para binário. O primeiro desses programas foi chamado de montador (*assembler*), utilizando a linguagem chamada de *Assembly* para traduzir uma versão simbólica de uma instrução para uma versão binária.

Ou seja, o processamento de dados é realizado pelo computador através da execução de instruções em linguagem de máquina, as quais o processador tem a capacidade de executar. O programador hoje em dia, codifica um algoritmo computacional utilizando linguagens de alto nível, que são mais fáceis de serem escritas, e ao serem ou compiladas, tem suas instruções traduzidas para linguagem de máquina, a qual é “legível” para o computador, graças ao *assembler*.

Resumindo o processo: Primeiramente é elaborado o algoritmo, composto por uma sequência de passos ou ações que determinam a solução de algum problema computacional, em seguida é realizada a codificação do mesmo em uma linguagem de alto nível. Esse código, com o auxílio do interpretador, é traduzido para um código correspondente em linguagem *Assembly*, através da interpretação ou da compilação do programa fonte. Nessa etapa, cada instrução da linguagem de alto nível é interpretada para executar a instrução correspondente, através do *hardware* do computador. Finalmente o programa é executado pelo computador.

## 4. METODOLOGIA

A simulação do jogo 21 foi implementada em linguagem de montagem Assembly. O conjunto de instruções escolhido para a implementação vem da *MIPS Technology*, que é um exemplo elegante dos conjuntos de instruções criados desde a década de 1980, baseado em registradores, ou seja, a CPU usa apenas registradores para realizar operações aritméticas e lógicas. Esse conjunto de instruções é considerado bastante didático, pela sua elegância e simplicidade, e por isso é utilizado com frequência no ensino em faculdades.

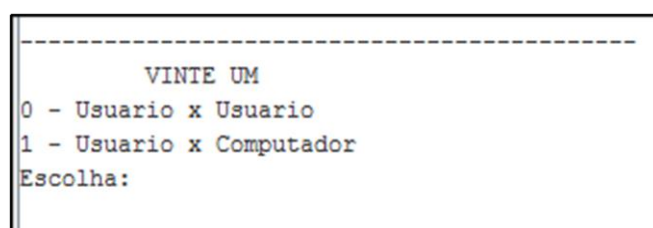
O *software* usado para essa implementação foi o M.A.R.S. (*MIPS Assembler and Runtime Simulator*), um ambiente de desenvolvimento interativo leve (IDE) para a programação em linguagem *Assembly* do MIPS.

## 5. EXPERIMENTO

Nessa seção serão apresentadas as principais telas do jogo exibidas para o usuário em tempo de execução.

Para iniciar o jogo é necessário abrir o arquivo *main.asm* no *software MARS*, marcar a opção '*Assembly all files in directory*' na aba '*Settings*' e em seguida clicar em Assemble (ou clicar na tecla '*F3*').

As telas a seguir apresentadas, serão mostradas na aba '*Run I/O*':



```
-----  
                VINTE UM  
0 - Usuario x Usuario  
1 - Usuario x Computador  
Escolha:
```

Figura 1 - Escolha do modo de jogo.

Fonte: Elaborado pelos autores.

Se o usuário digitar 0 irá selecionar o modo de jogo para dois jogadores, senão irá jogar contra o computador, que escolherá suas jogadas com base nas pontuações atuais, tentando sempre fazer a jogada mais inteligente.

```

Rodada numero 1

Player 1
0 - Passar vez
1 - Jogar
Escolha: 1

Dado branco: 3
Dado vermelho: 1

Player 2
0 - Passar vez
1 - Jogar
Escolha: 0

Placar:
Soma Player 1: 4
Soma Player 2: 0

```

Figura 2 - Exibição ao final da rodada do modo Usuário x Usuário.  
Fonte: Elaborado pelos autores.

Após a escolha do jogador um e exibição correspondente (se decidir jogar exibe o valores dos dados) o jogador dois decide (se decidir jogar exibe os valores dos dados). Em seguida é mostrado o placar (soma dos pontos) de ambos até o momento.

```

Rodada numero 1

Player 1
0 - Passar vez
1 - Jogar
Escolha: 1

Dado branco: 1
Dado vermelho: 1

** Computador decide se vai jogar ou passar a vez:  Irá jogar **

Dado branco: 5
Dado vermelho: 4

Placar:
Soma Player 1: 2
Soma computador: 9

```

Figura 3: Exibição ao final da rodada do modo Usuário x Computador.  
Fonte: Elaborado pelos autores.

Após a escolha do usuário e exibição correspondente (se decidir jogar exibe os valores dos dados) o computador testa qual a melhor jogada (se decidir jogar exibe os valores dos dados). Ao final é mostrado o placar (soma dos pontos) de ambos até o momento.

```
Placar:
Soma Player 1: 12
Soma computador: 13
-----
RESULTADO: Computador ganhou.
-----
```

Figura 4 - Exibição de resultado ao final da partida.

Fonte: Elaborado pelos autores.

Computador possui a pontuação mais alta e não passou de 21 pontos.

```
Placar:
Soma Player 1: 22
Soma Player 2: 20
-----
RESULTADO: Player 2 ganhou.
-----
```

Figura 5 - Exibição de resultado ao final da partida.

Fonte: Elaborado pelos autores.

Player 2 possui pontuação mais alta, porém Player 1 passou do limite (21 pontos).

```
Placar:
Soma Player 1: 23
Soma Player 2: 38
-----
RESULTADO: O jogo empatou.
-----
```

Figura 6 - Exibição de resultado ao final da partida.

Fonte: Elaborado pelos autores.

Ambos os jogadores passaram do limite (21 pontos).



```
Deseja jogar novamente?  
0 - Nao  
1 - Sim  
Escolha: 1  
  
-----  
  
*NOVO JOGO INICIADO*  
  
-----  
  
VINTE UM  
0 - Usuario x Usuario  
1 - Usuario x Computador  
Escolha:
```

Figura 7 – Jogador decidiu jogar novamente.  
Fonte: Elaborado pelos autores.

Após a exibição do resultado final é oferecida a opção de jogar novamente. Caso seja sim, inicia um novo jogo com a opção da escolha do modo escolhido.

```
Deseja jogar novamente?  
0 - Nao  
1 - Sim  
Escolha: 0  
  
-----  
  
*FIM DE JOGO*  
-----
```

Figura 8 – Jogador não escolheu jogar novamente.  
Fonte: Elaborado pelos autores.

Caso usuário escolha não, o jogo é finalizado.

## 6. RESULTADOS

De acordo com os objetivos definidos nesse relatório, o resultado da implementação foi o seguinte:

- A implementação da simulação do jogo para dois jogadores foi realizada com sucesso;
- O jogo foi dividido em dois módulos: Usuário Vs Usuário e Usuário Vs Computador;
- A opção de jogar novamente foi implementada;
- A escolha de jogada do computador no módulo Usuário Vs Computador, foi feita de modo inteligente, na qual ele decide se quer jogar ou passar a vez baseado na pontuação que já possui e na pontuação do seu adversário;
- É exibida, a cada rodada, a soma atual dos pontos de ambos os jogadores e, ao final da partida, é exibido corretamente se há vencedor ou se houve empate;
- O código foi separado em procedimentos e foram utilizadas *macros* e *eqv's* para melhor leitura e reutilização do código futuramente.

## 7. DISCUSSÃO

A meta inicial de implementação de simulação do jogo 21 foi alcançada com sucesso, sendo possível implementar todas funcionalidades do jogo utilizando a linguagem de montagem. As maiores dificuldades encontradas foram em questões mais específicas que a linguagem ou a IDE possuem, como por exemplo, gerar números randômicos, separar o trabalho em arquivos e utilizar *macros* e *.eqvs* globais, porém as dúvidas dessas questões foram sanadas em sala de aula e/ou com auxílio dos livros utilizados de base para estudo nesse trabalho.

A codificação do trabalho contou com o auxílio de uso de procedimentos, ou funções, que foram utilizadas para estruturar o programa, a fim de torná-lo mais claro para que outras pessoas possam entender, de permitir que seja reutilizado com facilidade e de evitar códigos redundantes.

O uso de *macros* e *eqvs* ajudou muito, tanto na correção de erros do código quanto na utilização de serviços, como por exemplo, ler e escrever inteiros, strings, etc. O fato do código em *assembly* ser muito extenso pode apresentar dificuldades para quem for tentar entendê-lo, mesmo com comentários ao lado, logo, o uso de *macros* e equivalências facilita a compressão do código em geral.

## 8. CONCLUSÕES E PERSPECTIVAS

O trabalho aqui relatado foi realizado com êxito, servindo ao propósito inicial do grupo, que era de se beneficiar da realização do trabalho para aprimorar os conhecimentos nos conteúdos da disciplina da cadeira de Organização de Computadores e também desenvolver melhor a lógica computacional.

Através da implementação relatada nesse trabalho, foi possível colocar em prática o que foi aprendido sobre a linguagem *Assembly*, tão importante para programadores que dela o usufruem, direta ou indiretamente. Sendo esta uma linguagem de máquina, com instruções de baixo nível que são importantíssimas na tradução de programas em linguagem de alto nível para instruções em binário, ou seja, que o computador “consegue entender”.

Portanto, considera-se que trabalhos desse escopo (implementação + relatório) são fundamentais para que nós, alunos, consigamos compreender melhor, não só as instruções da linguagem, mas também a importância da mesma para a nossa vida como programadores e para a vida de todos que utilizam um computador.

## REFERÊNCIAS BIBLIOGRÁFICAS

CRISTO, Fernando de. **Arquitetura de computadores** / Fernando de Cristo, Evandro Preuss, Roberto Franciscatto. – Frederico Westphalen: Universidade Federal de Santa Maria, Colégio Agrícola de Frederico Westphalen, 2013.

MONTEIRO, Mário A. **Introdução à organização de computadores**. 5. ed. Rio de Janeiro: LTC, 2007.

PATTERSON, David A, HENNESSY, John L . **Organização e projeto de computadores: a interface hardware/software**. 3. ed. Rio de Janeiro: Elsevier, 2005.

PATTERSON, David A, HENNESSY, John L . **Organização e projeto de computadores: a interface hardware/software**. 4. ed. Rio de Janeiro: Elsevier, 2014.

VOLLMAR, Ken. MARS – MIPS Assembly and Runtime Simulator. Disponível em: <<http://courses.missouristate.edu/kenvollmar/mars/>>. Acesso em: maio de 2016.

## APÊNDICES

### APÊNDICE A – CÓDIGOS-FONTE

O jogo contém quatro arquivos-fonte de extensão *.asm*. Todos os códigos-fonte seguem abaixo.

- Arquivo main.asm:

```
#####
.include "macros.asm"          # Inclui arquivo com macros
#####

.text

.globl main

main:
#
#####
#####
# prologo
# Nao precisamos armazenar endereco de retorno ou argumentos

# corpo do procedimento
jogar_novamente: # Reiniciar o jogo caso necessario
    imprime_string(divisor)      # " ----- "

    imprime_string(titulo)       # " VINTE UM "

    imprime_string(pxp)          # " 0 - Usuario x Usuario "

    imprime_string(pxc)          # " 1 - Usuario x Computador "

    imprime_string(escolha)      # " Escolha: "

    le_inteiro()                 # $v0 contem o inteiro lido (escolha do usuario)

    beq $v0, $zero, play1xplay2  # se $v0 = 0 salta para playxplay2 (modo de jogo Jogador Vs Jogador)
    jal userxpc                  # chama procedimento ($v0 = 1)
    j fim

play1xplay2:
    jal userxuser                # chama procedimento

fim:
```

```

imprime_string(divisor)

# Jogar novamente?
imprime_string(novamente)      # " Deseja jogar novamente? 0 - Nao 1 - Sim  Escolha: "

le_inteiro()                    # $v0 contem o inteiro lido (escolha do usuario)

beq $v0, $zero, sair            # se $v0 = 0 (nao deseja jogar novamente) salta para sair
imprime_string(divisor)

imprime_string(again)           # " NOVO JOGO INICIADO "

j jogar_novamente

sair:
    imprime_string(divisor)      # " ----- "
    imprime_string(fim_de_jogo)  # " FIM DE JOGO "
    imprime_string(divisor)      # " ----- "

# epilogo
    # Nao existe quadro para ser destruido

    encerra_programa(programa_OK)

#
#####
#####

.data
titulo:      .ascii "\n\t VINTE UM\n"
pxp:         .ascii "0 - Usuario x Usuario\n"
pxc:         .ascii "1 - Usuario x Computador\n"
escolha:     .ascii "Escolha: "
novamente:   .ascii "\n\nDeseja jogar novamente?\n0 - Nao\n1 - Sim\nEscolha: "
again:       .ascii "\n\n\t*NOVO JOGO INICIADO*\n"
fim_de_jogo: .ascii "\n\n\t*FIM DE JOGO*\n"
#
#####
#####

```

- Arquivo biblioteca.asm:

```

# Funcoes e strings usadas por ambos os procedimentos (userXuse e userXpc)

#####
#include "macros.asm"      # Inclui arquivo com macros
#####

# Procedimentos globais
.globl pergunta
.globl rola_dados
.globl imprime_resultado
#####

```

.text

```
#####
#####
# Este procedimento le a escolha do usuario (jogar ou passar)
#
# Parametro:
# $a0 <- string a ser impressa (user 1 ou user2)
#
# Valor de retorno:
# $v0 <- retona sim (1) ou nao (0)
#
# historico:
# 08/05/2016    Primeira versao do procedimento
# 10/05/2016    Segunda versao do procedimento
# 14/05/2016    Terceira versao do procedimento
#
pergunta:
#
#####
#####

# prologo
# procedimento folha. Nao precisa armazenar endereco de retorno ou argumento
# nao ha variaveis locais

# corpo do procedimento
# $a0 <- contem string a ser impressa (Player 1 ou Player 2)
li $v0, servico_imprime_string
syscall

le_inteiro()          # v0 contem o inteiro lido
# epilogo
# nao existe quadro para ser destruido
jr $ra                # retorna ao procedimento chamador

#

#####
#####
# Este procedimento rola os dados do usuario randomicamente
#
# Valor de retorno:
# $v0 <- possui a soma do valor do dado branco + dado vermelho

# historico:
# 08/05/2016    Primeira versao do procedimento
# 14/05/2016    Segunda versao do procedimento
#
rola_dados:
```

```

#
#####
#####

# prologo
# procedimento folha. Nao precisa armazenar endereco de retorno ou argumento

# corpo do procedimento

# Dado branco
gera_numero_random(max_random) # $a0 contem o valor randomico

bne $a0, $zero, rand_ok # se rand nao for 0 desvia para rand_ok
li $a0, 1 # se rand for 0 muda-se valor para 1
rand_ok:
move $t0, $a0 # $t0 <- valor dado branco

imprime_string(branco) # " Dado branco: "
imprime_inteiro($t0) # Imprime valor dado branco

# Dado vermelho
gera_numero_random(max_random) # $a0 contem o valor randomico
bne $a0, $zero, rand_ok2 # testa se rand = 0
li $a0, 1
rand_ok2:
move $t1, $a0 # $t1 <- valor dado vermelho

imprime_string(vermelho) # " Dado vermelho: "
imprime_inteiro($t1) # Imprime valor do dado vermelho

li $t3, 6 # $t3 = 6
beq $t1, $t3, dobro # testa se dado vermelho = 6, se for desvia para dobro
j retorna
dobro:
li $t1, 12 # dado vermelho vale o dobro dos pontos, ou seja 12
retorna:
addu $v0, $t0, $t1 # $v0 <- valor dado branco + valor dado vermelho

# epilogo
# nao existe quadro para ser destruido
jr $ra # retorna ao procedimento chamador
#

#####
#####
# Este procedimento imprime o resultado da partida
#
# Argumentos:
# $a0 <- guarda pontos do player1
# $a1 <- guarda pontos do player2/computador
# $a3 <- guarda a string a ser impressa (user 2 ou computador)

```



```

#
# historico:
# 08/05/2016    Primeira versao do procedimento
# 11/05/2016    Segunda versao do procedimento
# 14/05/2016    Terceira versao do procedimento
#
imprime_resultado:
#
#####
#####

# prologo
# procedimento folha. Nao precisa armazenar endereco de retorno ou argumento
# nao ha variaveis locais

# corpo do procedimento
# condicional composta, equivale a if(pontos_user > 21 && pontos_pc > 21) -> houve empate
li $t0, 21
slt $t5, $a0, $t0          # $t5 = 1 se pontos_user < 21, ou seja, $t5 = 0 se pontos_user > 21
bne $t5, $zero, menor21   # se %t5 = 1 desvia para naoempatou

slt $t5, $a1, $t0          # $t5 = 0 se pontos_pc > 21
bne $t5, $zero, menor21   # se $t5 = 0 nao desvia, ou seja, empatou

imprime_string(empate)     # " RESULTADO: O jogo empatou. "

j fim

menor21:
# se pontos_user = pontos_pc -> tambem significa que empatou

bne $a0, $a1, haganhador   # se pontos dos dois forem iguais nao desvia
imprime_string(empate)

j fim

haganhador:
# como nao houve empate, testaremos quem venceu. se pontos_user1 > 21 -> user_2/computador venceu
li $t0, 21
slt $t5, $t0, $a0          # $t5 = 1 se 21 < pontos_user, ou seja, pontos_user > 21
beq $t5, $zero, prox       # se $t5 = 0 desvia para prox, senao significa que user perdeu

li $v0, servico_imprime_string
move $a0, $a3              # $a0 recebe string a ser impressa (Player 2 ou PC) contida em $a3
syscall

j fim

prox:
# se pontos_user2/computador > 21 -> user1 venceu
slt $t5, $t0, $a1          # $t5 = 1 se pontos_pc > 21
beq $t5, $zero, maisalto   # se $t5 = 0 desvia para maisalto

```

```

    imprime_string(user1_win)

j fim

maisalto:
    # ultimo caso: testa pontuacao mais alta
    slt $t5, $a0, $a1          # $t5 = 1 se pontos_user < pontos_Pc, ou seja, pontos_pc > pontos_user
    beq $t5, $zero, uservenceu # se $t5 = 1 -> pc venceu. se $t5 = 0 -> desvia para user_venceu

    li $v0, servico_imprime_string
    move $a0, $a3              # $a0 recebe string a ser impressa (Player 2 ou PC) contida em $a3
    syscall

j fim

uservenceu:
    imprime_string(user1_win)  # " RESULTADO: Player 1 ganhou."

fim:
# epilogo
    # nao existe quadro para ser destruido
    jr $ra                    # retorna ao procedimento chamador
#
#####
#####

# Stings disponiveis para ambos os procedimentos

.globl rodada
.globl divisor
.globl decidindo
.globl pc_joga
.globl pc_passou
.globl joga_ou_passa
.globl joga_ou_passa2
.globl branco
.globl vermelho
.globl msg_placar
.globl pontos_user1
.globl pontos_user2
.globl pontos_pc
.globl empate
.globl pc_win
.globl user1_win
.globl user2_win

.data
rodada: .asciiz "\n\n\tRodada numero  "
divisor: .asciiz "\n-----"
decidindo: .asciiz "\n\n** Computador decide se vai jogar ou passar a vez:  "
pc_joga: .asciiz " Irá jogar **\n"
pc_passou: .asciiz " Passou a vez **\n"

```

```
joga_ou_passa: .asciiiz "\n\n Player 1\n0 - Passar vez\n1 - Jogar\nEscolha: "
joga_ou_passa2: .asciiiz "\n\n Player 2\n0 - Passar vez\n1 - Jogar\nEscolha: "
branco: .asciiiz "\nDado branco: "
vermelho: .asciiiz "\nDado vermelho: "
msg_placar: .asciiiz "\n\n          Placar:"
pontos_user1: .asciiiz "\n          Soma Player 1: "
pontos_user2: .asciiiz "\n Soma Player 2: "
pontos_pc: .asciiiz "\n          Soma computador: "
pc_win: .asciiiz "\n\n\tRESULTADO: Computador ganhou.\n"
user1_win: .asciiiz "\n\n\tRESULTADO: Player 1 ganhou.\n"
user2_win: .asciiiz "\n\n\tRESULTADO: Player 2 ganhou.\n"
empate: .asciiiz "\n\n\tRESULTADO: O jogo empatou.\n"
#
```

```
#####
#####
```

- Arquivo macros.asm:

# Macros e equivalências usados por todos arquivos

```
#####
```

# Diretivas .eqv

```
.eqv programa_OK                0
.eqv servico_imprime_string     4
.eqv servico_imprime_inteiro    1
.eqv servico_le_inteiro         5
.eqv servico_encerra_programa   17
.eqv servico_random_int         42
.eqv max_random                 7      # valor topo do random (ou seja, apenas numeros < 7)
```

#

```
#####
```

# Diretivas .macro

```
.macro encerra_programa(%valor_fim_programa)
```

```
    li $a0, %valor_fim_programa
```

```
    li $v0, servico_encerra_programa
```

```
    syscall
```

```
.end_macro
```

#

```
.macro imprime_string(%string_para_impressao)
```

```
    la $a0, %string_para_impressao
```

```
    li $v0, servico_imprime_string
```

```
    syscall
```

```
.end_macro
```

#

```
.macro imprime_inteiro(%inteiro_para_impressao)
```

```
    move $a0, %inteiro_para_impressao
```

```
    li $v0, servico_imprime_inteiro
```

```
    syscall
```

```
.end_macro
```

#

```
.macro le_inteiro()
```

```

        li $v0, servico_le_inteiro
        syscall
        # $v0 ira conter inteiro lido
.end_macro
#
.macro gera_numero_random(%valor_max_random)
    li $v0, servico_random_int
    li $a1, %valor_max_random      # $a1 <- valor topo do random (ou seja, apenas numeros < 7)
    syscall
    # $a0 ira conter numero randomico
.end_macro
#
#####

```

- Arquivo userXuser.asm:

```

# Modulo de jogo: Player vs Player
#     Ambos os jogadores podem decidir jogar ou passar a vez a cada rodada
#

#####

.include "macros.asm"      # Inclui arquivo com macros
#####

.text
#####
# Este procedimento chamado pelo main é a base do jogo versao Usuario Vs Usuario
#
# Valor de retorno:
#
# historico:
# 08/05/2016    Primeira versao do procedimento
# 09/05/2016    Segunda versao do procedimento
# 14/05/2016    Terceira versao do procedimento
#

#####
# QUADRO USERXUSER:      #
#   | $a0 |   $sp + 60   #
#   | $a1 |   $sp + 56   #
#   | $a2 |   $sp + 52   #
#   | $a3 |   $sp + 58   #
#   | $ra |   $sp + 44   #
#   | $s0 |   $sp + 40   #
#   | $s1 |   $sp + 35   #
#   | $s2 |   $sp + 32   #
#   | $s3 |   $sp + 38   #
#   | $s4 |   $sp + 24   #
#   | $s5 |   $sp + 20   #
#   | $s6 |   $sp + 16   #

```

```

#      | $s7  |      $sp + 12  #
#      | rodada |      $sp + 8    #
#      | pontos_p1 |      $sp + 4    #
#      | pontos_p2 |      $sp + 0    #
#####

.globl userxuser # chamada pelo main

userxuser:
# prologo
    # Salvando elementos na pilha para restauramos seus valores antes de voltar ao procedimento chamador
    sw $a0, 60($sp)          # guarda os argumentos $a0, $a1, $a2, $a3
    sw $a1, 56($sp)          #
    sw $a2, 52($sp)          #
    sw $a3, 48($sp)          #
    sw $ra, 44($sp)          # guarda endereco de retorno
    sw $s0, 40($sp)          # guarda registradores $s0 - $s7
    sw $s1, 36($sp)          #
    sw $s2, 32($sp)          #
    sw $s3, 28($sp)          #
    sw $s4, 24($sp)          #
    sw $s5, 20($sp)          #
    sw $s6, 16($sp)          #
    sw $s7, 12($sp)          #
    li $t0, 1                # $t0 <- 1
    sw $t0, 8($sp)           # rodada = 1
    li $t1, 0                # $t1 <- 0
    sw $t1, 4($sp)           # pontos_p1 = 0
    li $t2, 0                # $t2 <- 0
    sw $t2, 0($sp)           # pontos_p2 = 0

# corpo do procedimento
LOOP:
    imprime_string(divisor)    # " ----- "

    li $a2, 4                 # rodada deve ser menor que 4
    lw $s0, 8($sp)            # $s0 <- rodada
    slt $t5, $s0, $a2         # $t5 = 1 se rodada < 4
    beq $t5, $zero, EXIT      # desvia para EXIT se rodada > 3

    imprime_string(rodada)     # " Rodada numero "
    imprime_inteiro($s0)       # Imprime numero da rodada

    # Player 1 deseja jogar ou passar a vez?
    la $a0, joga_ou_passa      # argumento para o procedimento com a string correta (player1)
    jal pergunta               # chama procedimento
    move $t1, $v0               # $t1 <- valor de retorno do procedimento (1 = jogar, 0 = passar a vez)
    beq $t1, $zero, passa_vez   # se $t1 = 0 desvia para passa_vez

    # Jogar os dados
    jal rola_dados              # chama procedimento
    lw $t1, 4($sp)             # $t1 <- pontos_user

```

```

        addu $t1, $t1, $v0          # $t1 <- $t1 + valor de retorno do procedimento (valores dos dados)
        sw $t1, 4($sp)              # pontos_user = $t1

passa_vez: # Player 1 passou a vez
        # Player 2 deseja jogar ou passar a vez?
        la $a0, joga_ou_passa2      # argumento para o procedimento com a string correta (player2)
        jal pergunta                 # chama procedimento
        move $t1, $v0               # $t1 <- valor de retorno do procedimento (1 = jogar, 0 = passar a vez)
        beq $t1, $zero, passa_vez2   # se $t1 = 0 desvia para passa_vez

        # Jogar os dados
        jal rola_dados               # chama procedimento
        lw $t1, 0($sp)               # $t1 <- pontos_user2
        addu $t1, $t1, $v0           # $t1 <- $t1 + valor de retorno do procedimento (valores dos dados)
        sw $t1, 0($sp)               # pontos_user2 = $t1

passa_vez2: # Player 2 passou a vez

        # Imprime placar da rodada
placar: # " Placar: "
        imprime_string(msg_placar)

        # Usuario 1
        imprime_string(pontos_user1) # " Soma Player 1: "
        lw $t0, 4($sp)               # &t0 <- pontos user1
        imprime_inteiro($t0)         # Imprime pontos usuario1

        # Usuario 2
        imprime_string(pontos_user2) # " Soma Player 2: "
        lw $t0, 0($sp)               # &t0 <- pontos user2
        imprime_inteiro($t0)         # Imprime pontos usuario2

        # Acrescenta rodada em 1
        lw $s0, 8($sp)               # $s0 <- rodada
        addiu $s0, $s0, 1             # $s0 <- $s0 + 1
        sw $s0, 8($sp)               # rodada <- $s0

        j LOOP

# fim das 3 rodadas
EXIT:
        lw $a0, 4($sp)               # a0 <- ponto_user
        lw $a1, 0($sp)               # a1 <- pontos_pc
        la $a3, user2_win            # $a3 <- string para possivel impressao de vencedor
        jal imprime_resultado        # chama procedimento

# epilogo
        # Restauramos valores dos elementos para voltarmos ao procedimento chamador
        lw $a0, 60($sp)               # restaura $a0, $a1, $a2, $a3
        lw $a1, 56($sp)               #
        lw $a2, 52($sp)               #
        lw $a3, 48($sp)               #

```

```

lw $ra, 44($sp)          # restaura o endereco de retorno
lw $s0, 40($sp)          # restaura $s0 - $s7
lw $s1, 36($sp)          #
lw $s2, 32($sp)          #
lw $s3, 28($sp)          #
lw $s4, 24($sp)          #
lw $s5, 20($sp)          #
lw $s6, 16($sp)          #
lw $s7, 12($sp)          #

addiu $sp, $sp, 64       # restauramos a pilha
jr $ra                   # retorna ao procedimento chamador
#
#####
#####

```

- Arquivo userXpc.asm:

```

# Modulo de jogo: Player vs Computador
# Após o Player jogar, o computador ira avaliar qual a melhor jogada (rolar os dados ou passar a vez) e
# realiza-la
#
#####
#include "macros.asm"    # Inclui arquivo com macros
#####

.text
#####
# Este procedimento chamado pelo main é a base do jogo versao Usuario Vs Computador
#
# Valor de retorno:
#
# historico:
# 08/05/2016    Primeira versao do procedimento
# 10/05/2016    Segunda versao do procedimento
# 11/05/2016    Terceira versao do procedimento
# 14/05/2016    Quarta versao do procedimento
#
#####
# QUADRO USERXPC:      #
# | $a0 | $sp + 60 | #
# | $a1 | $sp + 56 | #
# | $a2 | $sp + 52 | #
# | $a3 | $sp + 58 | #
# | $ra | $sp + 44 | #
# | $s0 | $sp + 40 | #
# | $s1 | $sp + 35 | #
# | $s2 | $sp + 32 | #

```

```

#      | $s3 |      $sp + 38  #
#      | $s4 |      $sp + 24  #
#      | $s5 |      $sp + 20  #
#      | $s6 |      $sp + 16  #
#      | $s7 |      $sp + 12  #
#      | rodada |      $sp + 8  #
#      |pontos_user|      $sp + 4  #
#      |pontos_pc |      $sp + 0  #
#####

.globl userxpc    # chamada pelo main

userxpc:
# prologo
    addi $sp, $sp, -64      # ajusta a pilha para 16 elementos
    # Salvando elementos na pilha para restauramos seus valores antes de voltar ao procedimento chamador
    sw $a0, 60($sp)        # guarda os argumentos $a0, $a1, $a2, $a3
    sw $a1, 56($sp)        #
    sw $a2, 52($sp)        #
    sw $a3, 48($sp)        #
    sw $ra, 44($sp)        # guarda endereco de retorno
    sw $s0, 40($sp)        # guarda registradores $s0 - $s7
    sw $s1, 36($sp)        #
    sw $s2, 32($sp)        #
    sw $s3, 28($sp)        #
    sw $s4, 24($sp)        #
    sw $s5, 20($sp)        #
    sw $s6, 16($sp)        #
    sw $s7, 12($sp)        #
    li $t0, 1              # $t0 <- 1
    sw $t0, 8($sp)         # rodada = 1
    li $t1, 0              # $t1 <- 0
    sw $t1, 4($sp)         # pontos_user = 0
    li $t2, 0              # $t2 <- 0
    sw $t2, 0($sp)         # pontos_pc = 0

# corpo do procedimento
LOOP:
    imprime_string(divisor)    # " ----- "

    li $a2, 4                # rodada deve ser menor que 4
    lw $s0, 8($sp)           # $s0 <- rodada
    slt $t5, $s0, $a2        # $t5 = 1 se rodada < 4
    beq $t5, $zero, EXIT     # desvia para EXIT se rodada > 3

    imprime_string(rodada)    # " Rodada numero "
    imprime_inteiro($s0)      # Imprime numero da rodada

    # Usuario deseja jogar ou passar a vez?
    la $a0, joga_ou_passa    # argumento para o procedimento com a string correta (player1)
    jal pergunta             # chama procedimento
    move $t1, $v0             # $t1 <- valor de retorno do procedimento (1 = jogar, 0 = passar a vez)

```



```

beq $t1, $zero, passa_vez # se $t1 = 0 desvia para passa_vez

# Jogar os dados
jal rola_dados           # chama procedimento
lw $t1, 4($sp)           # $t1 <- pontos_user
addu $t1, $t1, $v0       # $t1 <- $t1 + valor de retorno do procedimento (valores dos dados)
sw $t1, 4($sp)           # pontos_user = $t1

passa_vez: # Usuario passou a vez
    imprime_string(decidindo)    # " ** Computador decide se vai jogar ou passar a vez: "

# Testes para computador fazer jogadas mais inteligentes

# Se rodada = 1, computador joga
li $t3, 1                # $t3 <- 1
lw $t1, 8($sp)           # $t1 <- rodada
beq $t3, $t1, joga       # Se rodada = 1 salta para joga

# Se pontos do computador > pontos do usuario, computador passa
lw $t1, 4($sp)           # $t1 <- pontos_user
lw $t2, 0($sp)           # $t2 <- pontos_pc
slt $t4, $t2, $t1        # $t4 = 1 se pontos_pc < pontos_user
beq $t4, $zero, passa    # desvia para passa se pontos_pc > pontos_user

# Se usuario passou de 21, computador passa
li $t3, 21
slt $t4, $t3, $t1        # $t4 = 1 se 21 < pontos_user ou seja pontos_user > 21
bne $t4, $zero, passa    # desvia para passa se usuario ja tiver perdido (pontos > 21)

joga:
    # Computador decide jogar
    imprime_string(pc_joga)    # " Irá jogar ** "

    jal rola_dados           # chama procedimento
    lw $t1, 0($sp)           # $t1 <- pontos_pc
    addu $t1, $t1, $v0       # $t1 <- $t1 + valor de retorno do procedimento (valores dos dados)
    sw $t1, 0($sp)           # pontos_pc = $t1

    j placar

passa: # Computador passou a vez
    imprime_string(pc_passou)  # " Passou a vez ** "

# Imprime placar da rodada
placar:
    imprime_string(msg_placar) # " Placar: "

# Usuario
    imprime_string(pontos_user1) # " Soma Player 1: "
    lw $t0, 4($sp)              # &t0 - pontos usuario
    imprime_inteiro($t0)        # Imprime pontos usuario

```

```

# Computador
imprime_string(pontos_pc)      # " Soma computador: "
lw $t0, 0($sp)                 # $t0 <- pontos pc
imprime_inteiro($t0)           # Imprime pontos computador

# Acrescenta rodada em 1
lw $s0, 8($sp)                 # $s0 <- rodada
addiu $s0, $s0, 1              # $s0 <- $s0 + 1
sw $s0, 8($sp)                 # rodada <- $s0
j LOOP

# fim das 3 rodadas
EXIT:
lw $a0, 4($sp)                 # $a0 <- ponto_user
lw $a1, 0($sp)                 # $a1 <- pontos_pc
la $a3, pc_win                 # $a3 <- string para possivel impressao de vencedor
jal imprime_resultado          # chama procedimento

# epilogo
# Restauramos valores dos elementos para voltarmos ao procedimento chamador
lw $a0, 60($sp)                # restaura $a0, $a1, $a2, $a3
lw $a1, 56($sp)                #
lw $a2, 52($sp)                #
lw $a3, 48($sp)                #
lw $ra, 44($sp)                # restaura o endereco de retorno
lw $s0, 40($sp)                # restaura $s0 - $s7
lw $s1, 36($sp)                #
lw $s2, 32($sp)                #
lw $s3, 28($sp)                #
lw $s4, 24($sp)                #
lw $s5, 20($sp)                #
lw $s6, 16($sp)                #
lw $s7, 12($sp)                #

addiu $sp, $sp, 64             # restauramos a pilha
jr $ra                         # retorna ao procedimento chamador

#
#####
#####

```