CURSO:	ENGENHARIA DE SOFTWARE	
DISCIPLINA:	Fundamentos de Arquitetura de Computadores	TURMA: A
SEMESTRE:	1°/2019	
PROFESSOR:	Tiago Alves da Fonseca	
ALUNOS:	Sara Conceição de Sousa Araújo Silva 16/0144752 Shayane Marques Alcântara 16/0144949	

Relatório Projeto
Manipulação Binária

Introdução

Assembly é a linguagem de montagem que faz uma abstração da linguagem de máquina usada em dispositivos computacionais. Cada declaração em Assembly produz uma instrução de máquina, fazendo uma correspondência um-para-um. A programação em linguagem de montagem é preferível em relação a linguagem de máquina porque a utilização de nomes simbólicos e endereços simbólicos em vez de binários ou hexadecimais facilita a memorização e o aprendizado das instruções. Então, o programador de Assembly só precisa se lembrar dos nomes simbólicos porque o compilador assembler os traduz para instruções de máquina.

Neste trabalho, duas estudantes de Fundamentos de Arquitetura de Software praticaram instruções em Assembly para analisar a paridade de um número inteiro por meio de manipulação binária.

Objetivos

1) Exercitar conceitos da linguagem de montagem (assembly) MIPS.
2) Interagir com ferramentas de desenvolvimento para criação, gerenciamento, depuração e testes de projeto de aplicações.

Contextualização

Um bit de paridade, ou um bit de conferência, é um bit adicionado a uma string de um código binário de forma que a string composta possua um número total de bits 1 par (paridade par) ou ímpar (paridade ímpar). Bits de paridade são uma das formas mais simples de códigos de detecção de erro e foram utilizados nos primórdios da Internet na proteção dos dados transmitidos pelos modems, dados que trafegavam em links de comunicação seriais, geralmente providos pelo par trançado das linhas telefônicas.

Há duas variantes para o cálculo de bits de paridade: a paridade par e a paridade ímpar. No caso da paridade par, para um conjunto de bits, as ocorrências dos bits cujo valor é 1 são contadas. Se a contagem total resultar em um número ímpar, o bit de paridade é ajustado para 1, fazendo com que a quantidade de ocorrências de 1 no conjunto (incluindo o bit de paridade) seja um número par. Se a contagem de 1s em determinado conjunto de

bits já for par, o bit de paridade é ajustado para 0. O bit de paridade pode ser colocado na posição mais significativa da palavra resultante, conforme exemplificado na Tabela 1.

Tabela 1: Cálculo de Bit de Paridade para palavras de 7 bits.

7 bits of data	(count of 1-bits)	8 bits including parity	
		even	odd
0000000	0	**0**0000000 = 0	**1**0000000
1010001	3	**1**1010001 = 209	**0**1010001
1101001	4	**0**1101001 = 105	**1**1101001
1111111	7	**1**1111111 = 255	**0**1111111

Referências Teóricas

David A. Patterson; John Hennessy, Organização e Projeto de Computadores, Campus, 3a Edição, 2005.

Material Necessário

- Computador com sistema operacional programável
  - Windows 10;
  - Manjaro Linux.
- Ambiente de simulação para arquitetura MIPS: MARS.

Roteiro

1) Revisão de conceitos básicos da arquitetura MIPS. Colete o material acompanhante do roteiro do trabalho a partir do Moodle da disciplina e estude os conceitos básicos da arquitetura MIPS.
2) Realizar as implementações solicitadas no questionário do trabalho.

Procedimentos

1) Ler um número inteiro menor que 128. Por exemplo: 127
2) Processar o número lido calculando o valor de bit necessário à ser adicionado para que a quantidade de bits 1 seja um número par.

Para a entrada igual a 127, a saída é:

```
bit paridade: 1
saída: 255
```

Outros exemplos:

```
128
========
entrada incorreta
0
```

```
========
bit-paridade: 0
saida: 0
1
========
bit-paridade: 1
saida: 129
```

Solução

Primeiro, o programa escrito no simulador MARS cria mensagens que serão mostradas no terminal para acompanhar as saídas dos programa

```
        .data

bit_paridade: .asciiz "bit-paridade: " # Mensangem que indicará o valor de bit necessário para paridade par
saida: .asciiz "\nsaida: " # Mensagem que acompanhará o valor final do número
mensagem_erro: .asciiz "entrada incorreta\n" # Mensagem de erro caso a entrada seja maior ou igual a 128
quebra_linha: .asciiz "\n" # Quebra de linha após última linha
```

Em seguida, começa a rotina principal *main*. Nela é feita a requisição do número de entrada a partir do terminal. Ocorre a validação do número de entrada, em que este deve estar no intervalo de maior ou igual a 0 e menor ou igual a 127. Caso a entrada esteja fora desse intervalo, o programa desvia para a *se_invalida,* que faz o devido tratamento, e se a entrada for 0, desvia para *imprime_resultado.*

```
main:
        li $v0, 5                           # Para ler o inteiro
        syscall                             # Faz a chamada de sistema

        move $t2, $v0                       # Move o valor de entrada no registrador $t2

        beq $v0, 0, imprime_resultado       # Se a entrada for 0, desvia para a rotina de impressão do resultado
        bge $v0, 128, se_invalida           # Se a entrada for igual ou maior que 128, desvia para a rotina de tratamento de entr
        ble $v0, -1, se_invalida            # Se a entrada for negativa, desvia para a rotina de impressão do resultado

        move $t4, $t2                       # Move o valor de entrada no registrador $t4
```

A *faz_divisao* realiza o procedimento de pegar os bits correspondentes ao número binário do número de entrada por meio de divisões sucessivas do valor de entrada por 2, para ver quais bits estão ligados, ou seja, quais são 1, o resto da divisão é comparado. Se o resto da divisão for igual a 1, desvia para a função *faz_icremento,* para icrementar na quantidade de bits ligados..

```
faz_divisao:    # Rotina que faz a divisão sucessiva do número de entrada em
        div $t2, $t2, 2                     # faz a divisão do inteiro pror 2
        mfhi $t3                            # Salva o resto da divisão no registrador $t3
        beq $t3, 1, faz_icremento           # Se o resto for 1, desvia para rotina que faz o icremento
        j faz_divisao                       # Retorna ao inicio da rotina para fazer o loop
```

Função que realiza o incremento do resultado do resto da divisão quando ele for 1, ou seja, calcula a quantidade de bits ligados.

```
faz_icremento:  # Rotina que soma a quantidade de restos 1 da divisão
        add $t1, $t1, 1                         # Soma 1 no registrador $t1
        beq $t2, 0, calcula_paridade            # Se o $t2 chegou em zero, desvia para rotina que calcula a paridade
        j faz_divisao                           # Vai para a rotina que faz a divisão do número
```

A *calcula_paridade*, verifica a paridade do valor de entrada. Se a quantidade de bits ligados for ímpar, então desvia para função que fará a paridade ser par, se não, já imprime o resultado

```
calcula_paridade: # Rotina para verificar a paridade é par ou ímpar
        beq $t1, 1, paridade_par                # Se a quantidade for 1, desvia para rotina que transfomar em par
        div $t1, $t1, 2                         # Faz a divisão da quantidade de bits ligados por 2, para verificar se é par ou impar
        mfhi $t3                                # Salva o resto da divisão no registrador $t3

        beq $t3, 0, imprime_resultado           # Se for 0, desvia para rotina que imprime o resultado
        beq $t3, 1, paridade_par                # Se for 1, desvia para rotina que transfomar faz a paridade par
```

A *paridade_par* liga mais um bit no número para que a quantidade fique par, isso feito com a adição de 128(2^7) e então desvia para *imprime_resultado.*

```
paridade_par:   # Rotina para ligar o bit mais significativo do número por meio da adiçã de 128
        add $t4, $t4, 128                       # Adiciona o 128 para que o número de bits ligados seja ímpar
        j imprime_resultado
```

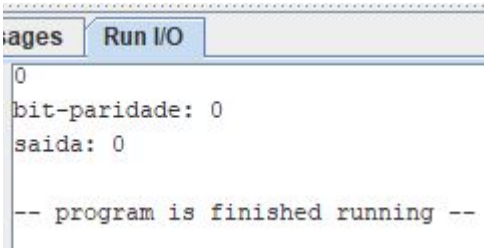
A *imprime_resultado* imprime a o valor de bit necessário para que o número fique com paridade par e o valor final do número de entrada.

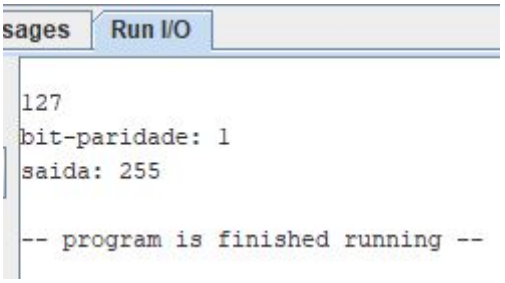
```
imprime_resultado: # Rotina que imprime os resultados
        li $v0, 4                               # Para imprimir mensagem
        la $a0, bit_paridade                    # Imprime "bit-paridade: "
        syscall                                 # Faz a chamada do sistema

        li $v0, 1                               # Para imprimir inteiro
        move $a0, $t3                           # Imprime o valor do bit de paridade
        syscall                                 # Faz a chamada do sistema

        li $v0, 4                               # Para imprimir mensagem
        la $a0, saida                           # Imprime "saida: "
        syscall                                 # Faz a chamada do sistema

        li $v0, 1                               # Para imprimir inteiro
        move $a0, $t4                           # Imprime valor final do número
        syscall                                 # Faz a chamada do sistema

        li $v0, 4                               # Para imprimir mensagem
        la $a0, quebra_linha                    # Imprimre "\n"
        syscall                                 # Faz a chamada do sistema

        li $v0, 10                              # termina o programa
        syscall                                 # Faz a chamada do sistema
```

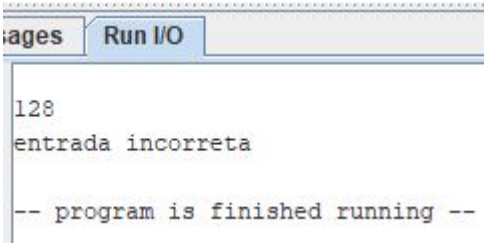
Função que exibe a mensagem de erro de “Entrada inválida”, caso o número esteja fora do intervalo pré-estabelecido (0 <= N <= 127) e finaliza o programa.

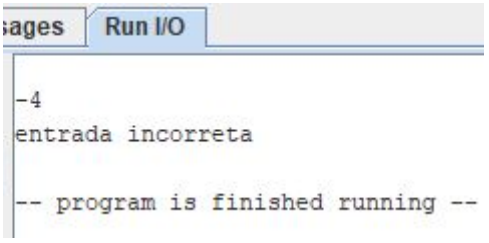
```
se_invalida:    # Rotina que trata a entrada incorreta
        li $v0, 4                       # Para imprimir mensagem
        la $a0, mensagem_erro           # imprime a mensagem de erro
        syscall                         # Faz a chamada do sistema

        li $v0, 10                      # Termina o programa
        syscall                         # Faz a chamada do sistema
```

Para executar as instruções programadas, foi necessário montar o arquivo e então começar a execução. Foi esperada a entrada de um número inteiro no intervalo de 0 a 127 para ser usado nas operações.









Limitações

A dupla encontrou dificuldades para tratar os casos em que a quantidade de bits ligados era par. Inicialmente, foi feito um *if* para tratar mas por algum motivo que se foi conseguido identificar, o programa estava dando adicionado 128 mesmo a quantidade de 1s sendo par. A estratégia foi mudada e o quando havia esse caso, o desvio já era feito para função de imprimir o resultado, sem fazer nenhuma alteração no número de entrada.