

---

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

---

## **Relatório Projeto**

### **Manipulação de Strings**

#### **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 implementar o CRC-16, código corretor de erros de 16 bits, com polinômio gerador 0x8005.

#### **Objetivos**

- 1) Exercitar conceitos da linguagem de montagem para arquitetura MIPS, especialmente aqueles referentes à implementação de solução de problemas em aritmética inteira.
- 2) Interagir com ferramentas de desenvolvimento para criação, gerenciamento, depuração e testes de projeto de aplicações.

#### **Referências Teóricas**

David A. Patterson, John L. Hennessy , Computer Organization and Design: the Hardware/Software Interface, The Morgan Kaufmann series in computer architecture and design ,5th ed

#### **Material Utilizado**

- Computador com sistema operacional programável
  - Manjaro Linux;
  - Ubuntu 18.10.
- Ambiente de simulação para arquitetura MIPS: MARS.

## Roteiro

1. Revisão de técnicas e ferramentas de desenvolvimento usando linguagem de montagem MIPS. Colete o material acompanhante do roteiro do trabalho a partir do Moodle da disciplina e estude os princípios e técnicas de desenvolvimento de aplicações usando linguagem de montagem MIPS
2. Realizar as implementações solicitadas no questionário do trabalho.

## Procedimentos

1. Ler, a partir do terminal, uma string de até 16 caracteres;
2. Calcular o CRC16 da string de entrada usando o polinômio gerador 0x8005;
3. Imprimir o CRC16/BUY-PASS em hexadecimal com 4 dígitos.

## Solução

Primeiro, o programa escrito no simulador MARS cria a mensagem que acompanhará o resultado, espaço para o buffer que pegará a string de entrada, um array que contém os valores da tabela ASCII que correspondem aos dígitos em hexadecimal e uma quebra de linha.

```
.data
crc: .asciiz "CRC16-BUYPASS: 0x"      # String que acompanhará a saída
quebra_linha: .asciiz "\n"           # Quebra de linha
buffer: .space 17                     # Para pegar 16 bytes do buffer
tabela_ascii: .word 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 65, 66, 67, 68, 69, 70 # Valores em ascii corresponde aos digitos da base 16
```

Em seguida, começa a rotina principal que pegará a string de entrada pelo terminal de no máximo 16 bytes e o *loop* que percorrerá a string. Quando a string chegar ao fim, o programa é desviado para a rotina responsável pela impressão do valor calculado.

```
.text
main:
    li $v0, 8                        # Para ler uma string
    la $a0, buffer                   # Para o buffer
    li $a1, 17                       # Tamanho máximo da string de entrada
    move $t0, $a0                    # Move o conteúdo de $a0 para $t0
    syscall                           # Faz a chamada do sistema

loop:
    lb $a0, ($t0)                    # loop para percorrer a string
    beq $a0, 0, imprime_crc          # $t0 vai percorre a string
    # Se $a0 for igual a 0, é porque a string terminou e sai do loop

    j crc16                           # Desvia para rotina que calcula o crc
```

O cálculo do CRC de 16 bits foi feito baseado no código em C disponibilizado pelo professor. Foram feitos deslocamentos à direita e à esquerda para fazer a divisão pelo polinômio gerador 0x8005. E para simular os tipos *uint\_8* e *uint\_16* da linguagem C foram feitos *ands* lógicos do número com o tamanho limite desses tipos, para reiniciar o número quando o ele ultrapassar o limite (overflow).

```

crc16:                                     # Rotina que calcula o crc16
move $s0, $a0                             # $s0 recebe o caracter que está em $a0
beq $s0, 10, imprime_crc                  # Desvia para rotina de impressão caso o caracter seja "\n", 10 na tabela ascii

# $s0 = d, caracter da string
# $t1 = c, o estado
# $t2 = r, o valor calculado do crc
# $t3 = p, $t4 = s, $t5 = t, valores intermediarios

srl $t6, $t1, 8                           # $t6 = c >> 8
xor $t4, $s0, $t6                         # s = d ^ (c >> 8)
and $t4, $t4, 255                         # s = s and 255, máscara para tamanho máximo que s pode ter

srl $t6, $t4, 4                           # $t6 = s >> 4
xor $t3, $t4, $t6                         # p = s ^ (s >> 4)
and $t3, $t3, 255                         # p = p and 255, máscara para tamanho máximo que p pode ter

srl $t6, $t3, 2                           # $t6 = p >> 2
xor $t3, $t3, $t6                         # p = p ^ (p >> 2)
and $t3, $t3, 255                         # p = p and 255, máscara para tamanho máximo que p pode ter

srl $t6, $t3, 1                           # $t6 = p >> 1
xor $t3, $t3, $t6                         # p = p ^ (p >> 1)
and $t3, $t3, 255                         # p = p and 255, máscara para tamanho máximo que p pode ter

and $t3, $t3, 1                           # p = p & 1
and $t3, $t3, 255                         # p = p and 255, máscara para tamanho máximo que p pode ter

sll $t6, $t4, 1                           # $t6 = s << 1
or $t5, $t3, $t6                          # t = p | (s << 1)
and $t5, $t5, 65535                       # t = t and 65535, máscara para tamanho máximo que t pode ter

sll $t6, $t1, 8                           # $t6 = c << 8
sll $t7, $t5, 15                          # $t7 = t << 15
sll $t8, $t5, 1                          # $t8 = t << 1

xor $t9, $t6, $t7                         # $t9 = (c << 8) ^ (t << 15)
xor $t2, $t9, $t5                         # r = (c << 8) ^ (t << 15) ^ t
xor $t2, $t2, $t8                         # r = r ^ (t << 1)
and $t2, $t2, 65535                       # r = r and 65535, máscara para tamanho máximo que r pode ter

move $t1, $t2                             # c = r

add $t0, $t0, 1                           # Incrementando $t0 que percorre a string
j loop                                    # Volta para o loop

```

Para a impressão de apenas 4 dígitos do valor hexadecimal do CRC, usou-se a estratégia de fazer um *and* lógico com as máscaras correspondente a cada dígito e deslocamentos à direita para pegar cada valor. O resultado dessas operações será a posição do array *tabela\_ascii* onde estará o valor da tabela ASCII correspondente ao dígito hexadecimal, então é feita a *syscall* 11, que fará a impressão do caracter respondente ao valor. Em seguida, o programa é finalizado.

```

imprime_crc:
    li $v0, 4
    la $a0, crc
    syscall

    la $t8, tabela_ascii
    and $t6, $t2, 0xf000
    srl $t6, $t6, 12
    mul $t6, $t6, 4
    add $t8, $t8, $t6
    lw $t7, 0($t8)

    li $v0, 11
    la $a0, ($t7)
    syscall

    la $t8, tabela_ascii
    and $t6, $t2, 0xf000
    srl $t6, $t6, 8
    mul $t6, $t6, 4
    add $t8, $t8, $t6
    lw $t7, 0($t8)

    li $v0, 11
    la $a0, ($t7)
    syscall

    la $t8, tabela_ascii
    and $t6, $t2, 0xf000
    srl $t6, $t6, 4
    mul $t6, $t6, 4
    add $t8, $t8, $t6
    lw $t7, 0($t8)

    li $v0, 11
    la $a0, ($t7)
    syscall

    la $t8, tabela_ascii
    and $t6, $t2, 0xf000
    mul $t6, $t6, 4
    add $t8, $t8, $t6
    lw $t7, 0($t8)

    li $v0, 11
    la $a0, ($t7)
    syscall

# Rotina para imprimir o valor do crc convertido em string
# Para imprimir string
# Imprime "CRC16-BUYPASS: 0x"
# Faz a chamada do sistema

# $t8 vai receber o primeiro endereço do tabela_ascii
# $t6 = r & 0xf000, aplica máscara para separar o valor do primeiro dígito do hexa
# $t6 = $t6 >> 12, obter o valor do primeiro dígito do hexa
# $t6 = $t6 * 4, para $t6 corresponder ao valor correto do endereço do array
# $t8 = [(0xf000 & r) >> 12] * 4
# $t7 vai receber o valor de tabela_ascii[$t8]

# Para imprimir um caracter
# Imprime o caracter correspondente ao valor de $t7 na tabela ascii
# Faz a chamada de sistema

# $t8 vai receber o primeiro endereço do tabela_ascii
# $t6 = r & 0xf000, aplica máscara para separar o valor do segundo dígito do hexa
# $t6 = $t6 >> 8, obter o valor do segundo dígito do hexa
# $t6 = $t6 * 4, para $t6 corresponder ao valor correto do endereço do array
# $t8 = [(0xf000 & r) >> 8] * 4
# $t7 vai receber o valor de tabela_ascii[$t8]

# Para imprimir um caracter
# Imprime o caracter correspondente ao valor de $t7 na tabela ascii
# Faz a chamada de sistema

# $t8 vai receber o primeiro endereço do tabela_ascii
# $t6 = r & 0xf000, aplica máscara para separar o valor do terceiro dígito do hexa
# $t6 = $t6 >> 4, obter o valor do terceiro dígito do hexa
# $t6 = $t6 * 4, para $t6 corresponder ao valor correto do endereço do array
# $t8 = [(0xf000 & r) >> 4] * 4
# $t7 vai receber o valor de tabela_ascii[$t8]

# Para imprimir um caracter
# Imprime o caracter correspondente ao valor de $t7 na tabela ascii
# Faz a chamada de sistema

# $t8 vai receber o primeiro endereço do tabela_ascii
# $t6 = r & 0xf000, aplica máscara para separar o valor do quarto dígito do hexa
# $t6 = $t6 * 4, para $t6 corresponder ao valor correto do endereço do array
# $t8 = [(0xf000 & r)] * 4
# $t7 vai receber o valor de tabela_ascii[$t8]

# Para imprimir um caracter
# Imprime o caracter correspondente ao valor de $t7 na tabela ascii
# Faz a chamada de sistema

```

## Execução

Para executar as instruções do programas foi necessário montar o arquivo e então executá-lo no ambiente MARS.

sages	Run I/O	ages	Run I/O
Alo mundo		a123456b	
CRC16-BUYPASS: 0x7FBB		CRC16-BUYPASS: 0xFDBB	
-- program is finished running --		-- program is finished running --	

Entrada vazia:

ages	Run I/O
CRC16-BUYPASS: 0x0000	
-- program is finished running --	

Entrada com 16 bytes:

ages	Run I/O
1234567812345678CRC16-BUYPASS: 0xB466	
-- program is finished running --	

Outros casos de testes:

Pages	Run I/O
0x800b	Trabalho 4- FAC
CRC16-BUYPASS: 0xD7F5	CRC16-BUYPASS: 0x6936
-- program is finished running --	-- program is finished running --

  

Pages	Run I/O
CRC16-BUYPASS	Sara e Shayane
CRC16-BUYPASS: 0xF119	CRC16-BUYPASS: 0x9E8B
-- program is finished running --	-- program is finished running --

## Limitações

A dupla encontrou dificuldades para simular os tipos `uint_8` e `uint_9` da linguagem C, primeiro foi usada a estratégia de desviar para outra rotina quando ocorresse *overflow* e nessa rotina o valor seria reiniciado, mas não foi bem sucedida. Outra mais simples foi utilizada, que é apenas fazer um `and` entre o registrador e o valor máximo permitido. Outro problema foi a impressão do valor hexadecimal, inicialmente foi usada a `syscall 34`, que imprime um inteiro em hexadecimal com 8 dígitos, e não tem o que fazer para ela imprimir apenas 4 dígitos. Então foi feita uma tabela com os valores da tabelas ASCII dos dígitos hexadecimais para ser acessada de acordo com o crc calculado.

Além disso, o programa se limita a calcular CRC de apenas strings de no máximo 16 caracteres e apenas com o polinômio gerador 0x8005.