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

Relatório Projeto 1
Operações Aritméticas e Lógicas Básicas

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 básicas em Assembly para realizar operações aritméticas e lógicas entre dois números e então introduzir o aprendizado e memorização do uso das instruções de montagem.

Objetivos

Este trabalho tem como objetivo a familiarização dos estudantes com as operações aritméticas e lógicas básicas em Assembly e assim serem capazes de resolver problemas diversos colocando em prática os conceitos da linguagem de montagem, além de promover a interação com ferramentas de simulação para criação e testes de códigos nesta linguagem.

Recursos utilizados

- Software Mars para simulação da arquitetura MIPS e edição de código Assembly.
- Computadores com sistemas operacionais Linux

Procedimentos

1. Ler dois números menores que 255 a partir do terminal;

2. realizar as operações aritméticas básicas de *soma* e *subtração* entre os dois números;
3. realizar as operações lógicas básicas *and*, *or* e *xor* entre os dois números;
4. realizar o *deslocamento* de 3 bits à esquerda do primeiro número e o *deslocamento* de 1 bit à direita do segundo número.

Entrada

A entrada é composta por dois números inteiros N_1, N_2 ($0 \leq N_1, N_2 \leq 255$).

Saída

A saída é composta dos resultados dos procedimentos de 1 a 4, conforme exemplos abaixo.

Exemplo de Entrada:

- 9
- 2

Exemplo de Saída:

- ADD: 11
- SUB: 7
- AND: 0
- OR: 11
- XOR: 11
- SLL(3): 72
- SRL(1): 1

Solução

Primeiro, o programa escrito no simulador MARS cria mensagens que serão mostradas no terminal para mostrar a qual operação pertence o resultado exibido.

```
5         .data
6
7   $soma: .asciiz  "ADD: " #variável para indicar a operação de soma entre os dois números
8   $subtracao: .asciiz "\nSUB: " #variável para indicar a operação de subtração entre os dois números
9   $and: .asciiz "\nAND: " #variável para indicar a operação AND entre os dois números
10  $or: .asciiz "\nOR: " #variável para indicar a operação OR entre os dois números
11  $xor: .asciiz "\nXOR: " #variável para indicar a operação XOR entre os dois números
12  $sll: .asciiz "\nSLL(3): " #variável para indicar deslocamento de 3 bits à esquerda do primeiro número (shift left logical)
13  $srl: .asciiz "\nSRL(1): " #variável para indicar deslocamento de 1 bit à direita do segundo número (shift right logical)
14  $quebra_linha: .asciiz "\n"  #variável para quebrar a linha após a última saída
```

Em seguida, começa a rotina principal *main*, nela é feita a requisição dos dois números de entrada a partir do terminal, realizando o procedimento 1.

```
18  main:
19          li $v0, 5 #lê o primeiro número a partir do terminal
20          syscall #faz a syscall
21
22          move $s0, $v0 #move conteúdo de $v0 para $s0
23
24          li $v0, 5 #lê o primeiro número a partir do terminaluser
25          syscall #faz a syscall
26
27          move $s1, $v0 #move conteúdo de $v0 para $s1
```

Então as funções referentes às operações propostas foram chamadas. Depois foi realizada a instrução de “sair” para mostrar ao simulador quando parar com a execução do programa.

```
29          #chamada das rotinas
30          jal _soma
31          jal _subtracao
32          jal _and
33          jal _or
34          jal _xor
35          jal _sll
36          jal _srl
37
38          li $v0, 10 #exit
39          syscall #faz a syscall
```

Após a *main*, são implementadas as rotinas necessárias para o procedimento 2. Para a operação de soma, foi implementada a função ***_soma*** que utiliza a instrução *add* para realizar a operação entre os dois números de entrada e depois o resultado é imprimido.

```
42  _soma:  #realiza a soma entre os dois números
43          li $v0,4 #print_string syscall
44          la $a0, $soma #imprime "ADD: "
45          syscall #faz a syscall
46
47          add $s2, $s0, $s1 #soma os valores dos registradores $s0 e $s1 e registra no $s2
48
49          li $v0,1 #carrega syscall print_int em $v0
50          la $a0, ($s2) #imprime o registrador $s2
51          syscall #faz a syscall
```

Para a operação de subtração, foi implementada a função ***_subtracao*** que utiliza a instrução *sub* para realizar a operação entre os dois números de entrada e depois imprime o resultado. Com isso, o procedimento 2 é concluído.

```
54  _subtracao:  #realiza a subtração entre os dois números
55          li $v0,4 #print_string syscall
56          la $a0, $subtracao #imprime "\nSUB: "
57          syscall #faz a syscall
58
59          sub $s2, $s0, $s1 #subtrai os valores dos registradores $s0 e $s1 e registra no $s2
60
61          li $v0,1 #carrega syscall print_int em $v0
62          la $a0, ($s2) #imprime o registrador $s2
63          syscall #faz a syscall
```

Em seguida são implementadas as rotinas necessárias para o procedimento 3. Na função ***_and***, a instrução *and* é utilizada para fazer a operação lógica entre os dois números de entrada e depois o resultado é imprimido.

```
66  _and:   #realiza a operação and entre os dois números
67          li $v0,4 #print_string syscall
68          la $a0, $and #imprime "\nAND: "
69          syscall #faz a syscall
70
71          and $s2, $s0, $s1 #faz o and entre os valores dos registradores $s0 e $s1 e registra no $s2
72
73          li $v0,1 #carrega syscall print_int em $v0
74          la $a0, ($s2) #imprime o registrador $s2
75          syscall #faz a syscall
```

Na função ***_or***, a instrução *or* é utilizada para fazer a operação lógica entre os dois números de entrada, seguida da impressão do resultado.

```
78  _or:    #realiza a operação or entre os dois números
79          li $v0,4 #print_string syscall
80          la $a0, $or  #imprime "\nOR: "
81          syscall #faz a syscall
82
83          or $s2, $s0, $s1 #faz o or entre os valores dos registradores $s0 e $s1 e registra no $s2
84
85          li $v0,1 #carrega syscall print_int em $v0
86          la $a0, ($s2) #imprime o registrador $s2
87          syscall #faz a syscall
```

Na função ***_xor***, a instrução x*or* é utilizada para fazer a operação lógica entre os dois números de entrada e em seguida o resultado é exibido. Com as últimas 3 funções foi concluído o procedimento 3.

```
90  _xor:   #realiza a operação xor entre os dois números
91          li $v0,4 #print_string syscall
92          la $a0, $xor #imprime "\nXOR: "
93          syscall #faz a syscall
94
95          xor $s2, $s0, $s1 #faz o xor entre os valores dos registradores $s0 e $s1 e registra no $s2
96
97          li $v0,1 #carrega syscall print_int em $v0
98          la $a0, ($s2) #imprime o registrador $s2
99          syscall #faz a syscall
```

Em seguida são implementadas as rotinas necessárias para o procedimento 4. Na função ***_sll*** é realizado o deslocamento de 3 bits à esquerda do 1º número de entrada com o uso da instrução *sll* que se refere a *shift left logical*, em seguida o resultado é exibido.

```
102 _sll:   #realiza deslocamento de 3 bits à esquerda do primeiro número(shift left logical)
103         li $v0, 4 #print_string syscall
104         la $a0, $sll #imprime "\nSLL: "
105         syscall #faz a syscall
106
107         sll $s2, $s0, 3 #faz o sll de 3 bits no registrador $s0 e registra no $s2
108
109         li $v0,1 #carrega syscall print_int em $v0
110         la $a0, ($s2)  #imprime o registrador $s2
111         syscall #faz a syscall
```

Na função ***_srl*** é realizado o deslocamento de 1 bits à direita do 2º número de entrada com o uso da instrução *srl* que se refere a *shift right logical*, em seguida o resultado é exibido e então é concluído o procedimento 4.

```
114 _srl:   #realiza deslocamento de 1 bit à esquerda do segundo número(shift right logical)
115         li $v0, 4 #print_string syscall
116         la $a0, $srl  #imprime "\nSRL: "
117         syscall #faz a syscall
118
119         srl $s2, $s1, 1  #faz o srl de 3 bits no registrador $s1 e registra no $s2
120
121         li $v0,1 #carrega syscall print_int em $v0
122         la $a0, ($s2)  #imprime o registrador $s2
123         syscall #faz a syscall
124
125         li $v0, 4 #print_string syscall
126         la $a0, $quebra_linha #imprime um '\n' no final
127         syscall #faz a syscall
```

A dupla inicialmente teve dificuldades com a sintaxe e com o entendimento da linguagem utilizada neste trabalho, por nunca ter sido utilizada antes pelas estudantes e ter um nível de abstração menor que linguagens usadas habitualmente. Esse problema foi contornado através de estudos e de testes de depuração no simulador MARS.

Conclusão

O trabalho foi uma experiência que proporcionou a familiarização das estudantes com a linguagem de montagem Assembly e com o simulador, além disso, os resultados com a solução proposta foram satisfatórios. Apesar de serem simples as operações realizadas, agora as estudantes terão maior facilidade no desenvolvimento de soluções de futuros problemas cuja implementação usará Assembly.

Referências

Tanenbaum, Andrew S. **Organização Estruturada de Computadores.** 6a Edição. Pearson, 2006

Ellard, Daniel J. **MIPS Assembly Language Programming CS50 Discussion and Project Book.** 1994.