**Laboratório I – Iniciando com o MARS e a Linguagem Assembly MIPS**

Nome: Gabriel Gatti da Silva Matrícula: 2021100336

Nome: Thiago F. N. Lahass Matrícula: 2021100178

Data: 27/09/22

**1. OBJETIVOS**

Carregar e executar programas em assembly MIPS;

Examinar posições de memória;

Examinar registradores;

Executar programas passo-a-passo.

**2. ATIVIDADES**

**2.1. Primeiros passos com o MARS**

Observar o resultado para diferentes números digitados como entrada e preencher a tabela a seguir com 5 diferentes valores de entrada e de saída (valores inteiros, positivos e negativos) para a execução do programa.

| Entrada | Saída |
| --- | --- |
| 1 | 256 |
| 2 | 512 |
| 0 | 0 |
| -3 | -768 |
| -10 | -2560 |

Tabela 1 - Entradas e Saídas do programa executado para dígito igual a 8

-- Q1: Qual é a equação que define a transformação dos valores de entrada para saída?

É possível observar que a função que age sobre os valores é , onde n é o número alterável no código dado (último número da matrícula).

Na tabela acima foi usado n = 8.

-- Q2: Qual o valor armazenado em $s0 e por que ele foi passado a $ra?

O valor salvo em $s0 é 0x00400000, que representa o endereço da primeira instrução do programa. Ele foi passado para $ra para que fosse possível saltar (jump) para a primeira instrução do programa assim que o mesmo chegasse ao fim, permitindo executar as mesmas instruções novamente.

**2.2. Usando o MARS para entender a arquitetura**

Observar a tabela de símbolos do programa. Dois labels devem aparecer (L1 e msg1). Anotar os valores desses labels (eles estão em hexadecimal, prefixados sempre com 0x).

| Label | Valor | Justificativa |
| --- | --- | --- |
| L1 | 0x00400008 | L1 tem o endereço de uma instrução (li $v0, 4), logo o começo fica 0x0…, pois é uma Label de instrução |
| msg1 | 0x10010000 | msg1 salva o endereço do começo da string na memória, e por isso começa com 0x1…, pois é uma Label de dados |

Tabela 2 - Labels e seus valores

-- Q3: O que esses valores significam?

São os endereços para as quais as Labels apontam, ou seja, se referem àquela posição da memória, que no caso de L1, aponta para uma instrução, e msg1 aponta para o início de uma sequência de caracteres (String).

-- Q4: Quantas instruções tem o programa? Quantos Bytes no total o programa ocupa na memória?

15 instruções. Cada instrução ocupa 4 Bytes. A string apontada por msg1 possui 25 caracteres e cada caractere ocupa um Byte, porém sempre é ocupado um número múltiplo de 4 Bytes (palavra/word) na memória, ou seja, são ocupados 28 Bytes pela string.

Portanto o programa ocupa um total de 15 \* 4 + 28 = 88 Bytes na memória.

-- Q5: Para quê serve a instrução li $s0, 0x00400000?

0x00400000 é o endereço da primeira instrução do programa, então o comando “li” salva esse valor no registrador $s0 para que futuramente seja feito o jump para o início do programa novamente.

-- Q6: Observe que durante a montagem essa instrução foi substituída por duas instruções. Quais e por quê?

Instruções:

lui $1, 0x00000040

ori $16, $1, 0x00000000

Motivo:

Como cada instrução possui 32 bits, e um registrador também possui 32 bits, então é impossível que uma instrução seja de tal forma que tenha o opcode que indique a função a ser realizada e ainda tenha como parâmetro os 32 bits do valor a ser colocado no registrador de destino, então são necessárias duas instruções para realizar essa tarefa. Primeiro é carregado a halfword menos significativa do segundo argumento (0x00000040) na halfword mais significativa do registrador $1. Os bits menos significativos do registrador são colocados em 0. Na segunda instrução é feito o OR lógico do registrador $1 e do hexa (0x000000000) e o resultado é colocado no registrador $16.

-- Q7: Observe que durante a montagem a instrução la $a0, msg1 também foi substituída por 2 outras instruções. Quais e por quê?

Instruções:

lui $1, 0x00001001

ori $4, $1, 0x00000000

Motivo:

Mesmo motivo descrito em Q6, ou seja:

Como cada instrução possui 32 bits, e um registrador também possui 32 bits, então é impossível que uma instrução seja de tal forma que tenha o opcode que indique a função a ser realizada e ainda tenha como parâmetro os 32 bits do valor a ser colocado no registrador de destino, então são necessárias duas instruções para realizar essa tarefa. Primeiro é carregado a halfword menos significativa do segundo argumento (0x00001001) na halfword mais significativa do registrador $1. Os bits menos significativos do registrador são colocados em 0. Na segunda instrução é feito o OR lógico do registrador $1 e do hexa (0x000000000) e o resultado é colocado no registrador $4.

| Endereço da instrução syscall | Tipo de serviço | Entrada (registrador) | Saída (registrador) |
| --- | --- | --- | --- |
| 0x00400014 | print\_string | - | msg1($a0) |
| 0x0040001C | read\_int | int($v0) | - |
| 0x00400030 | print\_int | - | int($v0) |

Tabela 3 - Dados das chamadas de syscall

**2.3. Coprocessadores**

-- Q8: Qual registrador indica o erro?

$12 (status).

-- Q9: Qual o erro foi indicado?

O registrador $13 (cause) indica o valor 0x00000020, o que provavelmente deve ser a flag de indicação de que aconteceu um overflow, como o ocorrido.

-- Q10: Como essa indicação poderia ser usada pelo SO?

Poderia ser usada como uma flag que indicaria que ocorreu overflow durante a execução do programa, sendo assim o SO poderia informar o ocorrido ao cliente ou utilizar essa informação para tomar as devidas providências.

-- Q11: Que representação é essa para a variável “x” (não precisa explicar)?

A variável x representa um número em ponto flutuante.

-- Q12: O que representam os valores que foram gerados nos registradores do Coproc1?

O valor gerado em $f0 é o valor que foi carregado da variável x para esse registrador, que tem valor 1.5674, e foi carregado através do comando *l.s $f0, x.* Já o valor que aparece em $f2 é duas vezes o valor que está em $f0, pois foi feita a soma do valor que está no registrador $f0 com o valor que está no registrador $f0, e o resultado foi colocado no registrador $f2, isso foi feito através do comando *add.s $f2, $f0, $f0*, e o número que este registrador representa é 1.5674 + 1.5674 = 3.1348.