

Universidade de São Paulo

Escola de Artes, Ciências e Humanidades

Amauri de Melo Junior - 8516650

Brenno Cremoini - 9778860

**Relatório EP**

**Organização de Computadores Digitais**

Profa. Dra. Gisele da Silva Craveiro

SÃO PAULO

2017

Sumário

[1. Introdução 2](#_Toc486292160)

[2. O Conjunto de Instruções MIPS 4](#_Toc486292161)

[3. Implementação 6](#_Toc486292162)

[4. Bibliografia 8](#_Toc486292163)

1. **Introdução**

O problema sorteado para a realização deste exercício-programa foi a obtenção de máximos divisores comuns (MDC) seguindo o algoritmo de Euclides. O problema divide-se em duas partes:

* Escrever uma função que receba dois números a e b como parâmetros e retorne o MDC deles;
* Escrever um programa que leia da entrada padrão um inteiro positivo n e uma sequência de n inteiros não-negativos e imprime o MDC de todos os números dados.

O algoritmo de Euclides para determinação de MDC é um dos algoritmos mais antigos que se tem conhecimento (data de cerca de 300 anos A.C) e pode ser encontrado no Livro VII da obra Os Elementos, de Euclides. Ainda nos dias de hoje, o Algoritmo de Euclides é uma das maneiras mais simples e eficientes de se calcular MDC. O processo é também conhecido como processo das divisões sucessivas, pois é a partir de sucessivas divisões que ele é executado, e baseia-se no seguinte resultado:

*Sejam a e b números naturais que, para evitar aborrecimentos desnecessários, suporemos a > b > 0. Se q e r são, respectivamente, o quociente e resto da divisão de a por b, então mdc(a, b) = mdc(b, r).*

Baseado na descrição acima, foi criado o programa em C que atenda as condições exigidas para a solução do problema. Em seguida, baseando-se no código em C, foi criado o programa em assembly correspondente seguindo o Conjunto de Instruções MIPS. O Conjunto de Instruções MIPS será descrito a seguir.

Para desenvolvimento, testes e execução, usamos o simulador MARS, um versátil e leve ambiente de desenvolvimento interativo (IDE, no original) para a elaboração de programas seguindo a arquitetura MIPS. O software foi desenvolvido pela Universidade Missouri State e disponibilizado gratuitamente para uso estudantil.

1. **O Conjunto de Instruções MIPS**

O Conjunto de Instruções MIPS prevê três formatos de instruções, e as instruções podem receber até três operandos. Na arquitetura MIPS, os operandos das instruções são registradores de 32 bits de memória ou palavras endereçadas na memória principal ou pilha de dados. No total, 32 registradores compõem a arquitetura, porém há usos convencionados para cada conjunto que veremos adiante.

Os três formatos de instruções previstos são estes que se seguem:

* Tipo R:

São instruções para as quais todos os operandos se tratam de registradores. Todas as instruções do tipo R possuem o seguinte formato:

|  |  |  |  |
| --- | --- | --- | --- |
| OP | rd | rs | rt |

Onde *OP* corresponde ao código mnemônico da instrução. *rs* e *rt* são registradores de origem, e *rd* é o registrador de destino. Podemos tomar como exemplo a operação add:

add $s1, $s2, $s3

* Tipo I:

São instruções que operam sobre um valor “imediato” e um operando. “Valores imediatos” podem ter no máximo 16 bits de comprimento. Números grandes não devem ser manipulados por instruções imediatas. Todas as instruções do tipo I possuem o seguinte formato:

|  |  |  |  |
| --- | --- | --- | --- |
| OP | rt | rs | IMM |

Onde *OP* corresponde ao código mnemônico da instrução. *rs* é o registrador de origem e *rt* o de destino. *IMM* corresponde ao valor imediato. Podemos tomar como exemplo a operação addi:

addi $s1, $s2, 100

* Tipo J:

São instruções usadas para realizar saltos. Instruções deste tipo possuem o maior espaço para um valor imediato já que endereços são sempre números grandes. Todas as instruções do tipo J possuem o seguinte formato:

|  |  |
| --- | --- |
| OP | LABEL |

Onde OP corresponde ao código mnemônico da instrução e LABEL é o endereço alvo para onde deve ocorrer o salto. Podemos tomar como exemplo a operação j:

j FIM

Os 32 registradores são implementados como de uso geral, ou seja, podem ser usados à vontade pelo programador. Os registradores são precedidos pelo caractere $ nas instruções em assembly. Podem ser endereçados pelo seu endereço numérico (de $0 a $31) ou pelos seus nomes ($t1, $sp, $ra). Dois registradores especiais são reservados e não podem ser diretamente acessados: Lo e Hi, usados para operações com pontos flutuantes e de multiplicação e divisão.

Convencionalmente, os registradores são organizados por uso e por comportamento esperado conforme o quadro abaixo:

|  |  |  |
| --- | --- | --- |
| Número do Registrador | Nome Alternativo | Descrição |
| 0 | zero | O valor 0 |
| 1 | $at | (temporário assembler) reservado pelo assembler |
| 2-3 | $v0-$v1 | (valores) Resultados de expressões e retornos de funções |
| 4-7 | $a0-$a3 | (argumentos) Primeiros quatro parâmetros de uma subrotina. Não são preservados entre chamadas de funções. |
| 8-15 | $t0-$t7 | (temporários) Salvos pelo escopo invocador, se necessário. Subrotinas podem usar sem salvar previamente. Não são preservados entre chamadas de funções. |
| 16-23 | $s0-$s7 | (valores salvos) – Salvos pelo escopo invocado. Uma subrotina que use um destes registradores deve salvar o valor original e restaurá-lo antes de retornar. São preservados entre chamadas de funções. |
| 24-25 | $t8-$t9 | (temporários) Salvos pelo escopo invocador, se necessário. Subrotinas podem usar sem salvar previamente. Não são preservados entre chamadas de funções. |
| 26-27 | $k0-$k1 | Reservados para uso dos tratamentos de interrupções/exceções. |
| 28 | $gp | Ponteiro global. Aponta para o meio do bloco (64kb) de memória no segmento de memória estática. |
| 29 | $sp | Ponteiro de pilha. Aponta para a última posição em uso da pilha. |
| 30 | $s8/$fp | Valor salvo/Ponteiro da moldura. É preservado entre chamadas de funções. |
| 31 | $ra | Endereço de retorno. |

1. **Implementação**

A parte A do problema foi resolvida implementando a função recursiva abaixo em C, que reproduz o algoritmo de Euclides:

int mdc(int x, int y)

{

if (y == 0) {

return x;

}

if (x == y) {

return x;

}

if (x < y) {

return mdc(y,x);

}

int r = x % y;

if (r == 0) {

return y;

}

return mdc(y, r);

}

E seu código correspondente em assembly MIPS:

mdc:

# primeiro de tudo, salvamos o valor de ra na pilha (para saber de onde viemos)

addiu $sp, $sp, -4

sw $ra, 0($sp)

# agora podemos começar

# o segundo parametro é zero? se sim, pula para o trecho que retorna o primeiro parametro

beqz $a1, mdc\_retorna\_a0

# os parametros são iguais? se sim, pula para o trecho que retorna o primeiro parametro

beq $a0, $a1, mdc\_retorna\_a0

# o primeiro parametro é menor do que o segundo? se sim, pula para o trecho que os inverte e chama mdc recursivamente

blt $a0, $a1, mdc\_retorna\_mdc\_invertido

# divide o primeiro parametro pelo segundo

div $a0, $a1

# copia o resto da divisão para a variável temporária t0

mfhi $t0

# o valor de t0 é zero? se sim, significa que são múltiplos entre si, portanto pula para o trecho que retorna o segundo parametro

beqz $t0, mdc\_retorna\_a1

# se não pulamos até aqui, o que resta é chamar mdc recursivamente, para isso, colocamos os novos parametros nas posicoes a0 e a1 e pulamos para o começo de mdc

la $a0, ($a1)

la $a1, ($t0)

jal mdc

# quando voltarmos, precisamos desempilhar o endereço e voltar, por isso, copiamos o que está no topo da pilha nesse instante para ra, movemos o ponteiro da pilha e pulamos para o ponto de ra

lw $ra, 0($sp)

addiu $sp, $sp, 4

jr $ra

mdc\_retorna\_mdc\_invertido:

# copia o valor do primeiro parametro para a variável temporária t0

la $t0, ($a0)

# copia o valor do segundo parametro para a variável de parameto a0

la $a0, ($a1)

# copia o valor da variável temporária t0 para a variável de parametro a1

la $a1, ($t0)

# pulamos para o começo de mdc

jal mdc

# quando voltarmos, precisamos desempilhar o endereço e voltar, por isso, copiamos o que está no topo da pilha nesse instante para ra, movemos o ponteiro da pilha e pulamos para o ponto de ra

lw $ra, 0($sp)

addiu $sp, $sp, 4

jr $ra

mdc\_retorna\_a0:

# armazenamos em v0 o valor do primeiro parametro, armazenado até então na variável de parametro a0

la $v0, ($a0)

# desempilhamos o endereço para voltar e pulamos

lw $ra, 0($sp)

addiu $sp, $sp, 4

jr $ra

mdc\_retorna\_a1:

# armazenamos em v0 o valor do primeiro parametro, armazenado até então na variável de parametro a0

la $v0, ($a1)

# desempilhamos o endereço para voltar e pulamos

lw $ra, 0($sp)

addiu $sp, $sp, 4

jr $ra

Os 32 registradores são implementados como de uso geral, ou seja, podem ser usados à vontade pelo programador. Os registradores são precedidos pelo caractere $ nas instruções em assembly. Podem ser endereçados pelo seu endereço numérico (de $0 a $31) ou pelos seus nomes ($t1, $sp, $ra). Dois registradores especiais são reservados e não podem ser diretamente acessados: Lo e Hi, usados para operações com pontos flutuantes e de multiplicação e divisão.

Convencionalmente, os registradores são organizados por uso e por comportamento esperado conforme o quadro abaixo:

1. **Bibliografia**

[**http://www.ic.unicamp.br/~pannain/mc542/aulas/ch3\_arq.pdf**](http://www.ic.unicamp.br/~pannain/mc542/aulas/ch3_arq.pdf)

[**http://clubes.obmep.org.br/blog/sala-de-estudos-algoritmo-de-euclides-para-determinacao-de-mdc/**](http://clubes.obmep.org.br/blog/sala-de-estudos-algoritmo-de-euclides-para-determinacao-de-mdc/)

[**https://en.wikibooks.org/wiki/MIPS\_Assembly/Instruction\_Formats**](https://en.wikibooks.org/wiki/MIPS_Assembly/Instruction_Formats)

[**http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm**](http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm)