Universidade de Brasília
Departamento de Cíências da Computação
Professor: Marcus Vínicius Lamar
Disciplina: Organização e Arquitetura de Computadores

Laboratório 1

Adarley Luiz Grando Filho	11/0007344
Guilherme Silva Lemos	13/0142468
Matheus Abrantes Cerqueira	13/0144291
Paulo da Cunha Passos	10/0118577
Rafael Lima	10/0131093

Brasília
2016

Conteúdo

1 Simulador/Montador MARS

1.1 Introdução ao MARS

Pelo Mars obtemos que o programa *sort.c* tem $(126)_{16}$ bytes ou seja 294 bytes. Sendo necessários 305 instruções para mostrar os valores de entrada e saída e 517 para ordenar o vetor, conforme detalhado na tabela 5.

	Instruções	R	I	J
UI	305	51	178	22
Sort	517	212	304	55
Total	822	263	482	77

Tabela 1: Instruções Executadas

1.2 Análise de Complexidade

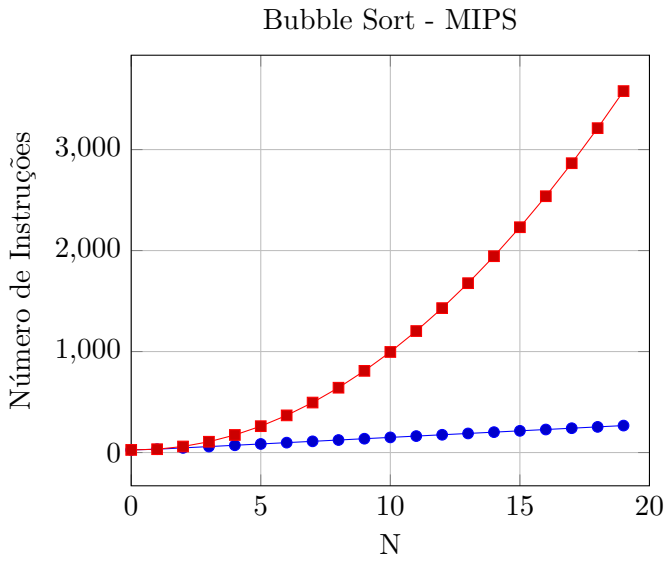

Figura 1: Complexidade Algoritmo

O algoritmo implementado foi o Bubble Sort que têm como complexidade para o pior caso $O(n) = n^2$ e para o melhor caso $O(n) = n$. Nota-se pelo gráfico na figura 1 que o a implementação usada do Bubble Sort teve uma complexidade maior que o esperado.

2 Compilador GCC

2.1 Assembly Mips Cross Compiler

Um cross-compiler é um compilador capaz de criar código executável para uma plataforma diferente daquela em que é executado [3]. O *cross-compiler* MIPS GCC é um compilador capaz gerar código executável para a plataforma MIPS a partir de um código em linguagem C. E pode gerar código *assembly* MIPS, bastando apenas executar o compilador com a diretiva -S. Ex: mips-sde-elf-gcc -S teste.c.
O código fonte em *assembly* MIPS gerado pelo compilador possui uma série de diretivas que são usados pelo montador [2] . Segue abaixo as diretivas do montador presentes no arquivo sort.s gerado a partir do arquivo sort.c:

Diretiva	Descrição
.file	Informa o nome do arquivo que foi compilado, para fins de debug.
.section	Instrui o montador a montar o código em uma seção com o nome informado.
.previous	Troca a seção/subseção atual pela seção/subseção anterior.
.nan	Indica qual codificação de NaN (*not a number*) é usada nos arquivos em assembly, podendo ser a codificação original (*legacy*) ou a do padrão IEEE 754-2008 (2008).
.type	Define o tipo de um símbolo.
.size	Define o tamanho de um símbolo.
.ent	Marca o início de uma função.
.end	Marca o fim de uma função.
.frame	Define o formato da pilha através dos argumentos: registrador do quadro (geralmente $fp), deslocamento, registrador de retorno (geralmente $ra).
.mask	Indica os registradores que devem ser salvos na pilha.
.fmask	Indica os registradores de ponto flutuante que devem ser salvos na pilha.
.ident	Insere *tags* em arquivos objeto.
.rdata	Indica o início do segmento de dados apenas para leitura (*read-only data*).

Tabela 2: Diretivas do montador não reconhecidas pelo MARS

Diretiva	Descrição
.globl	Torna um símbolo visível globalmente.
.align	Alinha o endereço a um múltiplo do valor especificado.
.word	Declara um inteiro.
.ascii	Declara uma string.
.data	Indica o inicio do segmento de dados.
.text	Instrui o montador a montar o código no segmento de texto.
.set	Ativa/desativa opções do montador, como a expansão de macros (macro e nomacro) e a reordenação de instruções (*reorder e noreorder*).

Tabela 3: Diretivas do montador reconhecidas pelo MARS

2.2 Assembly MIPS no Mars

O código em *assembly* gerado pelo *cross-compiler* MIPS GCC não é totalmente compatível com o MARS, sendo assim necessita-se realizar alguns ajustes para que o código compile corretamente no MARS.Então com essa finalidade, realizou-se as seguintes modificações no código:

(*1*) Comentou-se as linhas que possuíam as diretivas *.module*, *.type* e *.align*(as demais diretivas não reconhecidas pelo MARS não foram comentas, uma vez que ele foi capaz de ignora-las no momento da compilação);

(*2*) Como as instruções *lui* e *addiu* não podem ter como parâmetro um LABEL do campo .data, substituiu-se cada ocorrência da instrução *lui $rt,%hi(label)* seguida de *addiu $rt,$rs,%lo(label)* pela pseudo-instrução *la $rt, label*;

(*3*) Como a instrução *j* não pode receber um registrador como parâmentro, substitui-se cada ocorrência desta instrução (*j $31*) por *jr $31*;

(*4*) Substituiu-se as ocorrências da instrução *jal printf* pelas instruções *li $rt, Imm* e *syscall*;

(*5*) Substituiu-se as ocorrências da instrução *jal putchar* pelas instruções *la $rt, Addr* e *li $rt, 4*;

(*6*) Colocou-se a função *main* no início do segmento de texto e substituiu-se a instrução *j $31* por *li $v0, 10* mais *syscall* para finalizar o programa.

2.3 Otimizações de Código

Normalmente o compilador busca reduzir os custos de compilação [1], realizando esse processo no menor tempo possível, por isso, por padrão a opção de otimização do compilador fica desligada. Porém, a compilação sem otimização normalmente gera programas maiores e mais lentos do que aqueles gerados com a otimização ligada.

Parâmetro	Nome	Efeito
-O0	Compilação Rápida	Reduz o tempo de compilação e applica o debug do código (Opção Padrão), mas não há otimização.
-O1	Otimização Moderada	Tenta reduzir o tamanho do código e o tempo de execução sem utilizar operações de compilação que demorem muito.
-O2	Otimização Completa	Utiliza todas as otimizações que não envolvem balanço entre tamanho de código e tempo de execução.
-O3	Otimização Completa de alto nível	Acrescenta outras otimizações em relação a *-O2*.
-Os	Otimização de Tamanho	Otimiza o tamanho do código.
-Og		Aborda a necessidade de compilação rápida e uma experiencia de depuração superior, proporciona um nível razoável de desempenho de tempo de execução.

Tabela 4: Descrição Otimizações

Pelo Mars obtemos que o programa *sort.c* tem seu tamanho e número de instruções variado dependendo do parâmetro de otimização usado, variando de 348 a 608 bytes. A tabela abaixo detalha o número de instruções necessárias para cada parâmetro de otimização, além do tamanho do código executável de cada um.

	Instruções	R	I	J	Tamanho
-O0	1845	607	1214	24	608
-O1	750	264	462	24	444
-O2	599	135	461	3	404
-O3	584	133	448	3	404
-Os	872	300	548	24	348

Tabela 5: Instruções Executadas e Tamanho em bytes

2.3.1 O0

Como esta é a compilação sem otimizações, podemos perceber que este é um dos maiores códigos gerados, com 608 bytes. Com este parâmetro de compilação também temos o maior número de instruções executadas, conforme o esperado.

2.3.2 O1

O tamanho do código fica bem menor com a otimização O1 em relação a O0. Para conseguir rodar este código no Mars precisamos ativar a opção *delayed branching*, pois no lugar de instruções *nop* após cada *jump* o compilador alternou entre instruções antes e depois do *jump* para otimizar o tempo de execução.

O0	O1

`sw $4,32($fp)`
`sw $5,36($fp)`
`sw $0,16($fp)`
`b .L2`
`nop`
 	
`.L3:`
`lw $2,16($fp)`
`sll $2,$2,2`
`lw $3,32($fp)`
`addu $2,$3,$2`
`lw $2,0($2)`
`move $5,$2`
`lui $2,%hi(.LC0)`
`addiu $4,$2,%lo(.LC0)`
`jal printf`
`nop`
	`sw $17,24($sp)`
	`blez $5,.L2`
	`sw $16,20($sp)`
	
	`move $19,$5`
	`move $16,$4`
	`move $17,$0`
	`lui $18,%hi(.LC0)`
	`addiu $18,$18,%lo(.LC0)`
	`.L3:`
	`lw $5,0($16)`
	`jal printf`
	`move $4,$18`
	
	`addiu $17,$17,1`
	`bne $19,$17,.L3`
	`addiu $16,$16,4`

2.3.3 O2

Com este parâmetro de compilação o código ficou ainda menor se comparado com a otimização O1. Para conseguir rodar este código no Mars precisamos ativar a opção *delayed branching*, pois no lugar de instruções *nop* após cada *jump* o compilador alternou entre instruções antes e depois do *jump* para otimizar o tempo de execução.

O1	O2

show:
blez $5,.L11
nop
addiu $sp,$sp,-40
sw $19,32($sp)
lui $19,%hi(.LC0)
sw $17,24($sp)
move $17,$0
addiu $19,$19,%lo(.LC0)
sw $18,28($sp)
sw $16,20($sp)
move $18,$5
sw $31,36($sp)
move $16,$4
	sort:
	addiu $sp,$sp,-40
	sw $31,36($sp)
	sw $19,32($sp)
	sw $18,28($sp)
	sw $17,24($sp)
	blez $5,.L2
	sw $16,20($sp)
	move $19,$5
	move $16,$4
	move $17,$0
	lui $18,%hi(.LC0)
	addiu $18,$18,%lo(.LC0)

2.3.4 O3

Percebemos uma pequena melhora de otimização com este parâmetro de compilação se comparado com o parâmetro O2. O número de instruções diminuiu um pouco e houve um reposicionamento de registradores atribuídos a cada variável.

O2	O3

.L19:
bltz $10,.L27
addiu $11,$4,4
 	
lw $5,0($4)
lw $6,4($4)
slt $2,$6,$5
beq $2,$0,.L24
nop
 	
move $2,$4
b .L18
move $7,$11
 	
.L25:
lw $5,-4($2)
addiu $7,$7,-4
lw $6,0($2)
slt $8,$6,$5
beq $8,$0,.L17
addiu $2,$2,-4
	.L19:
	bltz $10,.L27
	addiu $11,$4,4
	
	lw $3,0($4)
	lw $7,4($4)
	slt $5,$7,$3
	beq $5,$0,.L24
	nop
	
	move $6,$4
	b .L18
	move $5,$11
	
	.L25:
	lw $3,-4($6)
	addiu $5,$5,-4
	slt $8,$7,$3
	beq $8,$0,.L17
	addiu $6,$6,-4

2.3.5 OS

Com este parâmetro de otimização obtivemos o menor código gerado, com 348 bytes. O número de instruções executadas também é o menor se comparado com todos os outros parâmetros de otimização, pois os *loops* foram otimizados se compados com o código gerado para o parâmetro de otimização O3.

O3	OS
`addiu $18,$18,%lo(.LC0)`	`addiu $19,$19,%lo(.LC0)`
`sw $19,32($sp)`	`sw $18,28($sp)`
`sw $17,24($sp)`	`sw $16,20($sp)`
`move $19,$5`	`move $18,$5`
`sw $31,36($sp)`	`sw $31,36($sp)`
`move $17,$4`	`move $16,$4`
`slt $2,$16,$19`	`.L3:`
`.L7:`	`lw $5,0($16)`
`beq $2,$0,.L6`	`addiu $17,$17,1`
`move $4,$18`	`move $4,$19`
	`jal printf`
`lw $5,0($17)`	`addiu $16,$16,4`
`addiu $16,$16,1`	
`jal printf`	`bne $18,$17,.L3`
`addiu $17,$17,4`	`lw $31,36($sp)`
`b .L7`	
`slt $2,$16,$19`	

3 Plotador de Gráficos no Mars

3.1 Procedimento

Os gráficos foram plotados em uma imagem com resolução de 320x240 e 8bits/pixels para codificação de cores. Os parâmetros de entradas são o intervalo real de x [Linf ,Lsup] e a equação desejada em registradores, assim o procedimento pode ser divido em: Cálculo de resolução cartesiana e equacionamento, determinação de limites superiores e inferiores de y e plotagem de pontos. Uma vez que a imagem possui 320 pixels em x, adota-se que o pixel 0 corresponde ao valor Linf e em 320 o valor Lsup e uma vez que a variação é constante pode-se calcular a resolução de um pixel, ou seja, quando que o acréscimo de um pixel contribui na escala cartesiana :

$$resol_x = \frac{L_{sup} - L_{inf}}{320}$$

Assim pode-se obter os valores da função para coordenadas cartesianas yc = f(xc) e utilizando um laço que varia de Linf a Linf com incrementos de resolx determina-se os limites que yc atinge (Ymin, Ymax) que são atualizados a cada interação do laço de repetição.

Com os valores limites de yc é possível realizar a conversão de coordenadas cartesianas para pixels de forma análoga a realizada em x, onde:

$$y_e = \frac{y_c}{resol_y}$$

em que a resolução de y é dada por:

$$resol_y = \frac{y_{max} - y_{min}}{240}$$

Contudo, a fórmula acima deve ser atualizada, uma vez que diferentemente de X os valores de pixels em Y variam de 0 a 240 de cima para baixo, ou seja, contrário ao eixo cartesiano. Para ajustar o valor, deve-se levar em consideração a imagem da função no domínio desejado. Caso a função tenha uma imagem estritamente positiva, o valor é ajustado de acordo com a função abaixo:

$$y_p = 240 - \frac{y_c}{resol_y}$$

Caso a função tenha uma imagem com valores negativos e positivos, a posição do eixo X é definida no meio da tela, ou seja, em (120, 0). Dessa forma, o ajuste é feito da seguinte maneira:

$$y_p = 120 - \frac{y_c - y_{min}}{resol_y}$$

Por fim, caso a função tenha uma imagem puramente negativa, o eixo Y é definido no topo da tela, ou seja, sem nenhum ajuste extra:

$$y_p = \frac{y_c - y_{min}}{resol_y}$$

3.2 Plotando Funções

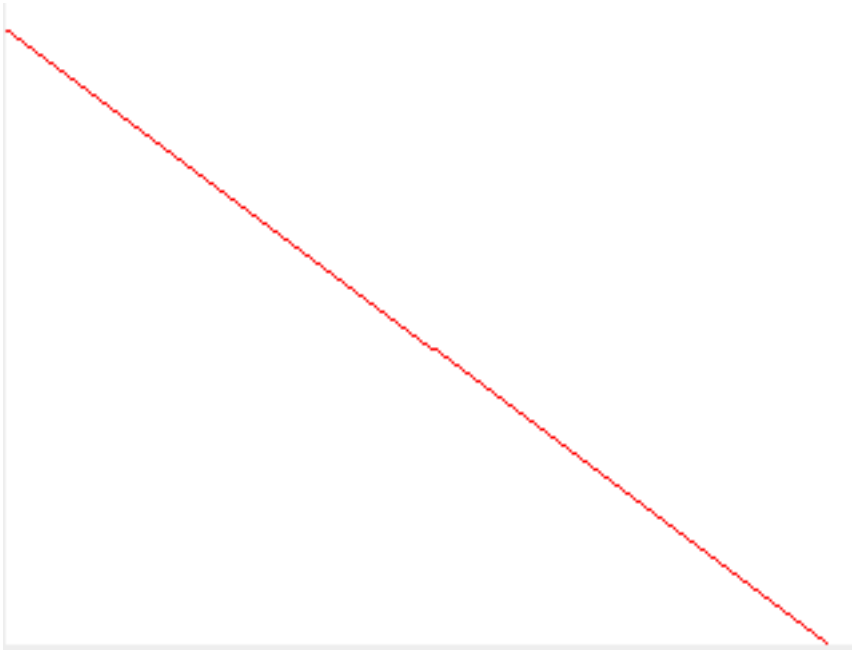

Figura 2: $F(x) = -x$, $x \in [-1, 1]$

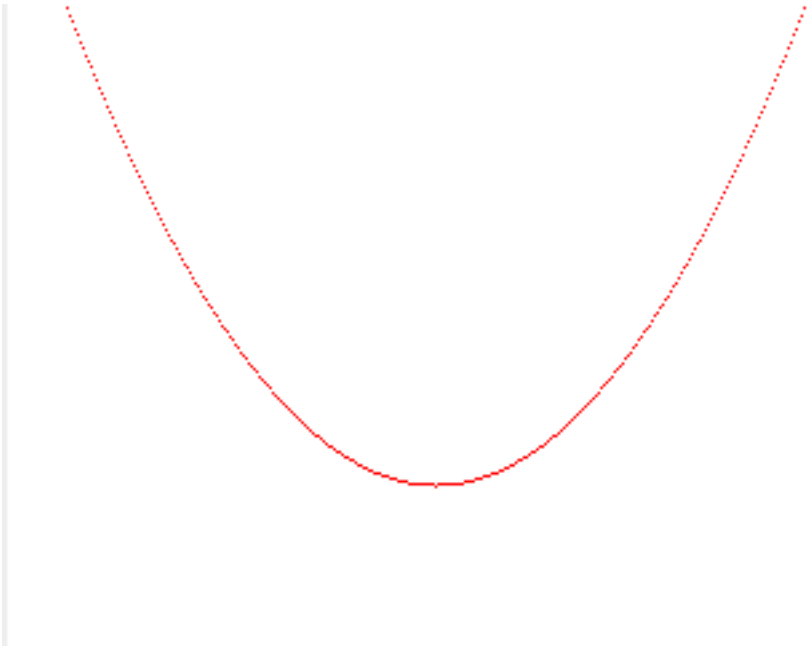

Figura 3: $F(x) = x^2 + 1,\ x \in [-2, 2]$

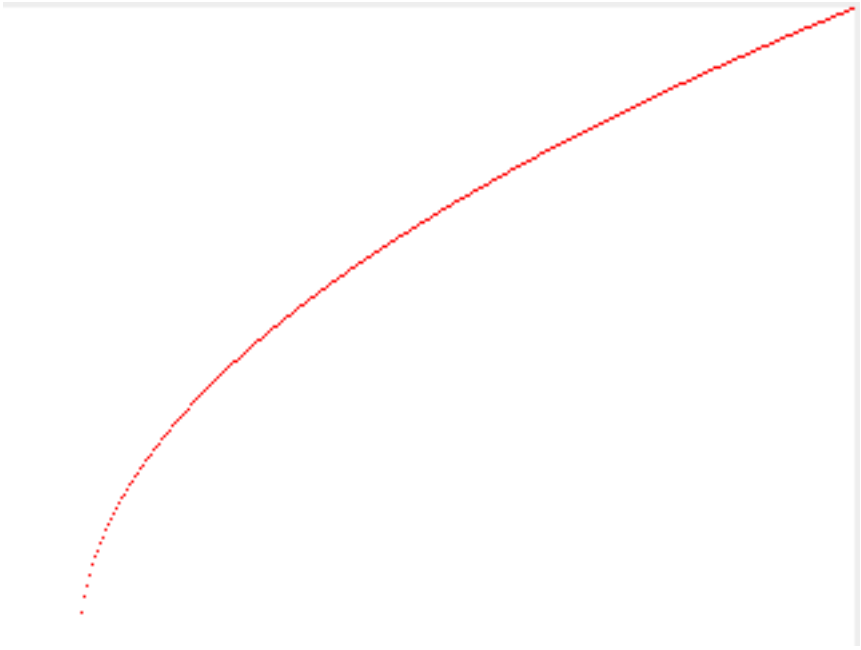

Figura 4: $F(x) = \sqrt{x},\ x \in [-1, 10]$

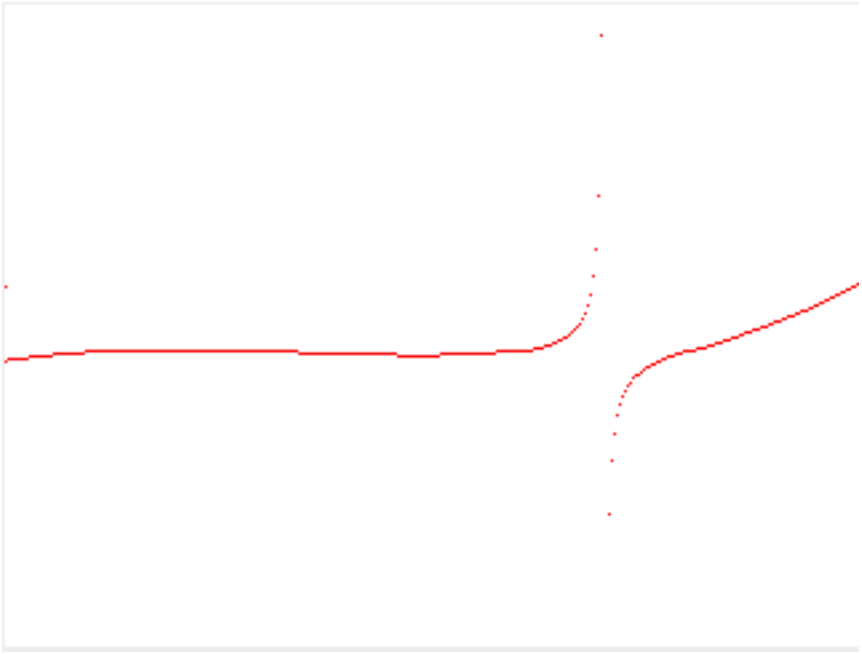

Figura 5: $F(x) = (x+1)^2(x-1)(x-2)/(x-1.5)$, $x \in [-2, 3]$

3.3 Símbolo Batman

Para desenhar o símbolo do Batman, foi escolhido o símbolo de 1998. O plot final é uma combinação de quatro funções distintas. O resultado foi demonstrado na figura 6. Diferentemente dos gráficos anteriores, o símbolo do Batman teve resolução fixa, para não afetar a apresentação visual e as funções utilizadas são exemplificadas abaixo:

$$f(x) = |x/2| - 0.09x^2 - 3 + (1 - (||x| - 2| - 1)^2)^{\frac{1}{2}}$$

$$g(x) = (4.71 - 0.5|x| - 1.36(4 - (|x| - 1)^2)^{\frac{1}{2}} \left(\frac{||x| - 1|}{|x| - 1}\right)^{\frac{1}{2}} + 0.9$$

$$p(x) = -3\left(1 - \left(\frac{x}{7}\right)^2\right)^{0.5} \left(\frac{||x| - 4|}{|x| - 4}\right)^{0.5}$$

$$h(x) = 2\left(\frac{-||x| - 1| * |3 - |x||}{(|x| - 1)(3 - |x|)}\right)^{\frac{1}{2}} \left(1 + \frac{||x| - 3|}{|x| - 3}\right)\left(1 - \left(\frac{x}{7}\right)^2\right)^{0.5} +$$

$$+(5 + 0.97(|x - 0.5| + |x + 0.5|) - 3(|x - 0.75| + |x + 0.75|))\left(1 + \frac{|1 - |x||}{1 - |x|}\right)$$

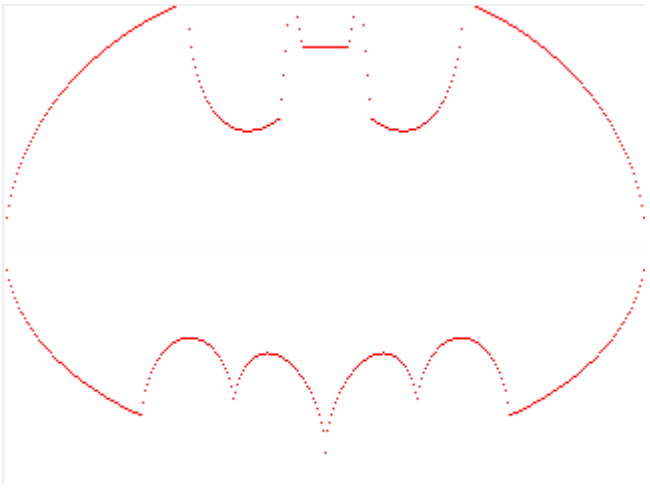

Figura 6: Símbolo Batman no MARS

Referências

[1] Options that control optimization. `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`, 2016. [Online; acessado em 02-10-2016].

[2] A. M. Neto. Aceleração da implementação do algoritmo de criptografia aes-128 em um processador mips. *CEP*, 70910:900, 2016.

[3] Wikipedia. Cross compiler — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Cross_compiler&oldid=731465390`, 2016. [Online; acessado 01-10-2016].