Relatório 1 – Organização de Computadores I – INE5411
Alunas: Fernanda Bertotti Guedes (24100601) e Laísa Ágathe Bridi Dacroce (24100598)
Data: 16/04/25

1 INTRODUÇÃO

O presente relatório se destina a descrever a resolução dos exercícios propostos no Laboratório 1. Em primeiro momento, será descrita a resolução das atividades 1 e 2 via console, com um panorama geral sobre o código e seu funcionamento, uma análise sobre a quantidade de linhas de instrução e limitações da nossa solução. Posteriormente, será descrita a resolução das atividades 1 e 2 via Digital Lab Sim, assim como as principais dificuldades encontradas.

2 ATIVIDADES VIA CONSOLE

Nessas atividades, foram implementadas em Assembly as operações $a = b + 35$ e $c = d^3 - (a + e)$ em dois exercícios diferentes. Para armazenar as variáveis b, d e e na memória de dados, conforme solicitado, foi necessário declará-las como palavras (.word) na seção de dados do código.

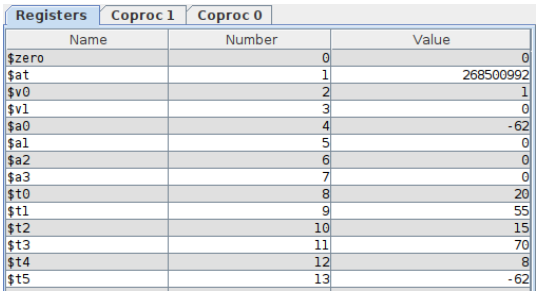
2.1 ATIVIDADE 1

Na primeira atividade, tomamos os valores de b, d e e como sendo 20, 2 e 15, respectivamente. Em especial, a escolha de d = 2 se mostrou conveniente para a implementação da potenciação ao cubo, tendo em vista que essa operação foi possível por meio de uma instrução shift left logical, com o deslocamento de 2 bits à esquerda para obter o valor 8.

Para esse conjunto de valores, o código retorna o resultado de -62 no console, o que atesta seu correto funcionamento, por se tratar da seguinte operação:

$a = 20 + 35 = 55$; $d^3 = 8$;
$c = 8 - (55 + 15) = 8 - 70 \rightarrow c = -62$;

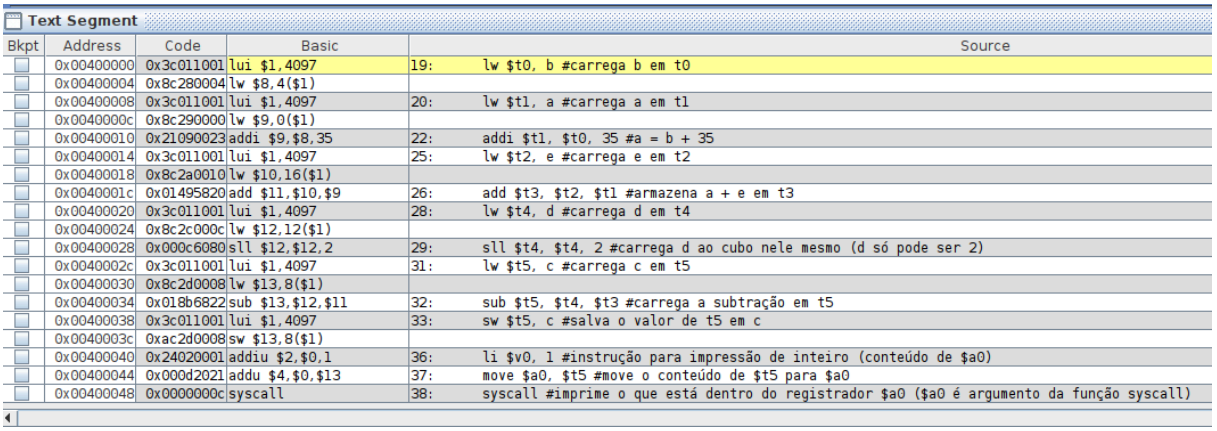
Dessa forma, na coluna de Registradores do Mars, é possível notar o valor -62 em $t5, utilizado para calcular o valor de c, e em $a0, passado como argumento para escrita no console após a chamada de sistema:



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	1
$v1	3	0
$a0	4	-62
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	20
$t1	9	55
$t2	10	15
$t3	11	70
$t4	12	8
$t5	13	-62

O código da resolução da atividade 1 possui 38 linhas, contando linhas em branco e comentários. Descontando estes, somam-se 21 linhas. Ao comparar as colunas Source e Basic no Mars, foi possível notar que Basic apresenta mais linhas de código, conforme a imagem abaixo.



Text Segment

Bkpt	Address	Code	Basic		Source
	0x00400000	0x3c011001	lui $1,4097	19:	lw $t0, b #carrega b em t0
	0x00400004	0x8c280004	lw $8,4($1)		
	0x00400008	0x3c011001	lui $1,4097	20:	lw $t1, a #carrega a em t1
	0x0040000c	0x8c290000	lw $9,0($1)		
	0x00400010	0x21090023	addi $9,$8,35	22:	addi $t1, $t0, 35 #a = b + 35
	0x00400014	0x3c011001	lui $1,4097	25:	lw $t2, e #carrega e em t2
	0x00400018	0x8c2a0010	lw $10,16($1)		
	0x0040001c	0x01495820	add $11,$10,$9	26:	add $t3, $t2, $t1 #armazena a + e em t3
	0x00400020	0x3c011001	lui $1,4097	28:	lw $t4, d #carrega d em t4
	0x00400024	0x8c2c000c	lw $12,12($1)		
	0x00400028	0x000c6080	sll $12,$12,2	29:	sll $t4, $t4, 2 #carrega d ao cubo nele mesmo (d só pode ser 2)
	0x0040002c	0x3c011001	lui $1,4097	31:	lw $t5, c #carrega c em t5
	0x00400030	0x8c2d0008	lw $13,8($1)		
	0x00400034	0x018b6822	sub $13,$12,$11	32:	sub $t5, $t4, $t3 #carrega a subtração em t5
	0x00400038	0x3c011001	lui $1,4097	33:	sw $t5, c #salva o valor de t5 em c
	0x0040003c	0xac2d0008	sw $13,8($1)		
	0x00400040	0x24020001	addiu $2,$0,1	36:	li $v0, 1 #instrução para impressão de inteiro (conteúdo de $a0)
	0x00400044	0x000d2021	addu $4,$0,$13	37:	move $a0, $t5 #move o conteúdo de $t5 para $a0
	0x00400048	0x0000000c	syscall	38:	syscall #imprime o que está dentro do registrador $a0 ($a0 é argumento da função syscall)

Ao compreender que a coluna Basic designa o que é de fato operado pela máquina, enquanto a coluna Source apresenta nosso código em Assembly descontando diretivas, nossa suposição inicial foi que essa diferença seria resultado, majoritariamente, do uso de pseudo-código, como a instrução addi. Contudo, foi possível notar que mesmo instruções reais da arquitetura, como lw, podem ser desmembradas em mais instruções na seção Basic, se envolverem o uso de rótulos. Por esse motivo, no nosso exercício, as instruções lw e sw envolvendo as variáveis declaradas na área de dados, foram as que representaram a maior diferença entre a quantidade de linhas de Source e Basic.

2.2 ATIVIDADE 2

Na segunda atividade, os valores de b, d e e devem ser fornecidos pelo usuário via teclado, e o resultado, armazenado em c, deve ser mostrado no terminal. Para essa atividade, salvamos todos os valores na memória, utilizando instruções de store word.

Na área de dados (.data), declaramos individualmente b, c, d e e como .word de valor inicial 0. Para que os valores pudessem ser lidos do teclado, utilizamos a chamada de sistema 5 (para ler inteiro), seguida da instrução syscall.

Como os valores lidos do teclado são armazenados em registradores do tipo $v, após cada leitura de um valor, utilizamos a instrução move para mover o dado armazenado em $v0 para um registrador temporário, para então, realizar uma operação de store word, como mostra o exemplo abaixo:

```
li $v0, 5 #comando (5) para ler inteiro (b)
syscall
move $t0, $v0 #resultado salvo em $t0
sw $t0, b #armazena o dado lido em b
```

A implementação das operações iniciais segue a mesma lógica do exercício 1. Ainda há a limitação de escolha do valor de d, que, nesse caso, pode ser apenas 2 ou 0. Por isso, para valores diferentes dos citados, não há a execução correta do código.

Houve a tentativa da implementação de um código mais otimizado para a realização do cálculo de d^3, entretanto, devido às limitações de instruções permitidas para essa atividade, não foi possível a conclusão. Uma das nossas tentativas para contornar a falta da instrução de multiplicação (mult), foi implementar essa operação por meio de somas deslocadas usando o operador lógico and e comparação bit a bit, o que, contudo, não surtiu o efeito desejado. Outra ideia que tivemos foi utilizar a instrução sllv, para podermos passar o próprio valor de d (armazenado em um registrador) como o número de bits deslocados à esquerda. Essa instrução, embora não contemplada pelos slides da disciplina, poderia nos permitir incluir um terceiro valor possível para d, pois funcionaria também para o número 4. Entretanto, por conformidade com as orientações propostas, optamos por manter o código com a instrução sll, conforme a implementação inicial.

Para mostrar o resultado salvo em c, foi utilizada a chamada de sistema 1 para a escrita de inteiro no terminal, seguida de uma instrução move para que c fosse armazenado em $a0:

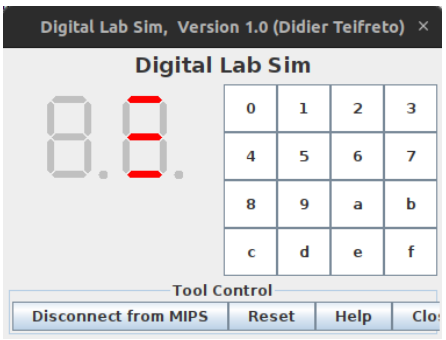
```
li $v0, 1
move $a0, $t4
syscall
```

O código da resolução da atividade 2 possui 39 linhas, com diferenças na quantidade de linhas nas colunas Basic e Source, pelo mesmo motivo da atividade 1.

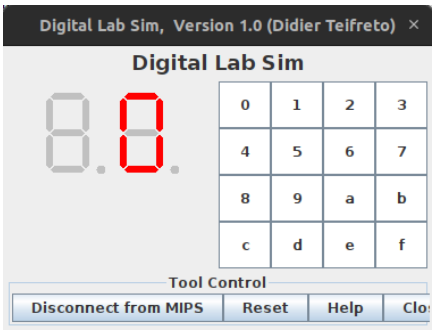
3 ATIVIDADES VIA DIGITAL LAB SIM

3.1 ATIVIDADE 1

Na realização das atividades via Digital Lab Sim, encontramos grande dificuldade em compreender o que está descrito na seção Help da ferramenta. Também foi um desafio a correta descrição dos bytes presentes na seção de dados: inicialmente, pensamos que, por se tratar de .byte, poderíamos atribuir valores em binário às variáveis mapeadas. Porém, esse erro fez com que nosso 0 fosse representado da seguinte forma errônea no display de 7 segmentos:



Então, com o auxílio da monitoria, foi possível consertar nosso código para que as variáveis fossem inicializadas com o valor correspondente ao binário em hexadecimal, o que resultou na correta exibição dos números de 0 a 9, conforme o exemplo abaixo.



Acreditamos que nossa implementação poderia se beneficiar de loops e de tempos de espera da exibição dos números no display, porém, não foi possível implementar tais otimizações em tempo para esta entrega. Por este motivo, temos a limitação de que a totalidade dos números somente pode ser visualizada na execução do código passo a passo, pois a execução completa passa direto para o número 9, por se tratar da manipulação do conteúdo no mesmo endereço de memória.

3.2 ATIVIDADE 2

Não foi possível finalizar a realização da atividade 2, por conta da dificuldade de implementar a lógica que relaciona as linhas e colunas do teclado com os endereços informados pela ferramenta, no caso, 0xFFFF0012 e 0xFFF0014. Também enfrentamos dificuldades com relação à implementação do loop, tendo em vista que não foi possível pensar em uma forma de finalizá-lo.

O esqueleto da nossa ideia consta no arquivo enviado e baseia-se na comparação entre o endereço lido e o código da tecla informado pela própria ferramenta. Assim, há uma série de instruções beq que direcionam o código à exibição do valor associado à tecla no display de sete segmentos, reutilizando os bytes calculados no exercício anterior. Além disso, outra limitação diz respeito a utilizarmos somente o display da direita, de modo que a codificação de A, B, C, D, E e F é feita com o objetivo de mostrar o próprio caractere hexadecimal, em vez do número decimal equivalente com dois dígitos.