PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS - PUC CAMPINAS
CEATEC - CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIA

Projeto MIPS

Sistema de abastecimento

Bruno Guilherme Spirlandeli Marini - RA 17037607
Caio Lima e Sousa Della Torre Sanches - RA 17225285
Gabriela Nelsina Vicente - RA 17757089
Jefferson Meneses da Silva - RA 17230400
Marcos Aurélio Tavares de Sousa Filho - RA 17042284

Campinas-SP
2018

Índice

Introdução

Foi proposto como primeiro projeto do laboratório de Arquitetura de Computadores, a elaboração de um sistema para administração de abastecimentos, feito em linguagem assembly do MIPS (x32 bits). Para isso, utilizamos as funções, cadastrar abastecimento (insere novo abastecimento), exclui abastecimento (exclui abastecimento), exibe abastecimentos, cálculo do consumo médio e cálculo do preço médio.

Detalhes de projeto

Cadastro

Analisando os requerimentos do projeto, chegamos à conclusão de que seria útil um armazenamento já ordenado dos dados. Mas ao mesmo tempo, se para cada inserção/remoção tivéssemos que deslocar o resto dos dados uma posição para trás ou para frente, o tempo de execução seria longo e confuso demais.

Decidimos então por armazenar nossos dados em forma de lista ligada, adicionando um campo na estrutura para armazenar o endereço do próximo elemento, ou 0 no caso do último (ponteiro NULL).

Malloc (`nalloc`)

Analisando a implementação de listas ligadas em Estrutura e Recuperação de Dados A, desenvolvemos um malloc primitivo, capaz de reaproveitar os espaços livres após a exclusão de abastecimentos. Após encontrar um espaço livre para armazenamento da nova estrutura, essa função retornaria a primeira posição válida para o abastecimento.

EPOCH

Para simplificar o processo de comparação de datas (para inserção ordenada e busca de elementos, por exemplo), implementamos métodos para codificação e decodificação de datas (3 valores) em um valor só.

Optamos por utilizar uma adaptação do sistema EPOCH (com precisão de 1 dia). Para isso, também limitamos que todos os anos tenham 12 meses de 30 dias cada (essa regularidade simplifica os métodos).

Exibição

Como nossa lista ligada que armazena as informações cadastradas já é salva de forma ordenada pela data, na hora da exibição de todos os cadastros o trabalho ficou muito mais simples.

Tendo o ponteiro que indica o início da lista, usamos ele para percorrer os componentes da nossa “Struct”, assim exibindo membro a membro. Chegando ao fim da struct temos o campo que armazena a posição do próximo cadastro. Para uma tela mais amigável e uma disposição mais limpa, criamos procedimentos que alinhavam os números na exibição.

Exclusão

Para a parte de exclusão dos registros, é feito uma busca simples pelos elementos da lista ligada, no qual é verificado a data de cada um deles a fim de encontrar um registro compatível com a data digitada pelo usuário.

Caso não haja algum registro armazenado ou a data não tenha sido encontrada é exibida uma mensagem ao usuário. Caso contrário o registro deixa de fazer parte da lista ligada e os ponteiros são reorganizados.

Consumo médio

Para o consumo médio guardamos os valores de toda a distância percorrida e de todo o combustível utilizado, descartando a primeira quilometragem (organizada por data) e a última quantidade de combustível.

Caso o sistema não tenha registrado no mínimo dois abastecimentos uma mensagem aparece ao usuário, informando que as informações guardadas não são suficientes para calcular o consumo médio.

Preço médio

No caso do preço médio, elaboramos a função para procurar todos os postos de abastecimentos que foram cadastrados e assim calcular em tempo real, os valores de preço pela frequência obtida até o momento. Para isso, percorremos a lista ligada (utilizada para armazenar todas as informações), inserindo na pilha a cada nome diferente encontrado e incrementamos a posição de frequência (do nome do posto) quando encontramos um posto igual.

Após todas as informações serem computadas e inseridas na pilha, utilizamos o algoritmo de ordenação por bolha para ordenar todos esses dados em ordem crescente do início da pilha, para então imprimi-los do fim da pilha até o início.

Para todas as informações anteriores, em pilha, utilizamos a seguinte estrutura que chamaremos de *PMCad* (Preço médio do cadastro):

Informação:	$\overline{Valor}$	Frequência	Nome Posto	Σ Preços
Tam. bytes:	4	4	16	4

Detalhes de implementação

Foi utilizado a seguinte estrutura para organizar os dados dentro da memória:

Info.:	Data	Qtq. de Combustível	Preço	Distância	Nome Posto	Ponteiro Prox
Bytes:	2	2	4*	4	16	4

* 4 bytes que correspondem a um float, neste caso.

Cadastro

Primeiro utilizamos uma função (`jal RData`) para automaticamente ler e converter a data do novo cadastro. Lemos então o resto das informações, armazenando-as em registradores temporários.

Então, utilizamos a função de alocação dinâmica (`jal nalloc`), para receber o endereço da nova struct, e utilizamos as funções de armazenamento (`sw`, e `s.s`) para efetivamente guardar os valores na memória.

Após a inserção dos dados, iniciamos o procedimento de armazenamento ordenado em lista ligada:

1. Analisamos o caso mais específico (que necessitaria mudar o ponteiro inicial da lista ligada):
    a. Se a lista estiver vazia (`$s7` for igual a 0), pular para a label `emptyList`;
    b. Se o valor do EPOCH novo for menor que o EPOCH do primeiro item, pular para a label `emptyList`;
    c. Em emptyList, escreve que o ponteiro para a próxima struct do novo (deslocamento de 28 bytes) recebe o antigo `$s6`, e em `$s6` guarda o novo ponteiro.
2. Analisamos então o caso genérico:
    a. Guarda em `$t2` o endereço da próxima struct (deslocamento de 28 bytes da posição atual, em `$t0`);
    b. Se recebeu 0, significa que chegou ao fim da lista ligada (inserção no final, e portanto já encontrou a posição adequada);
    c. Caso contrário, compara o EPOCH novo com o atual. Se for menor, armazena `$t2` em `$t0` e volta para (2.a.);
    d. Caso contrário, encontrou a posição de inserção adequada:
        i. Armazena no novo->próx, o valor do atual->próx;
        ii. Armazena no atual->próx, o endereço atual (em `$t0`).

Após a inserção adequada, incrementamos `$s7` em 1 (uma vez que foi adicionado um valor à lista).

Malloc (`nalloc`)

Inicializamos um registrador temporário com `lui $t0, 0x1004`, que é a primeira posição do HEAP. Então, verificamos se esta posição da memória está livre (se contém 0). Caso esteja, guarda esse valor em `$v0` e retorna ao ponto anterior. Caso contrário, incrementa o temporário em 32 (equivalente ao tamanho da estrutura), e repete o processo.

```
nalloc: lui $t0, 0x1004          Inicia a busca no início do HEAP
_next:  lw  $t2, 0($t0)          Recebe valor em [$t0]
        beq $t2, $zero, _found   Se for 0, é um espaço livre
        addi $t0, $t0, 32        Adiciona 32 (tamanho da struct)
        j _next                  Repete o processo
_found: add $v0, $t0, $zero      Se achou, guarda em $v0
        jr $ra                   Retorna ao ponto de chamada
```

EPOCH

Codificação

A função de codificação da data recebe três parâmetros (dia, mês e ano em `$a0`, `$a1` e `$a2` respectivamente).

Zeramos o registrador de retorno (EPOCH em `$v0`), subtraímos 2000 do ano (valor escolhido arbitrariamente, o que significa que datas anteriores a 1/1/2000 retornam valores inesperados) e multiplicamos o resultado por 360 (dias no ano), adicionando o resultado em `$v0`.

Subtraímos 1 do mês (por conta de janeiro ser o mês 1), multiplicamos por 30 (dias no mês) e somamos o resultado em `$v0`.

Por fim, adicionamos o valor do dia em `$v0` diretamente, e usamos o `jr $ra` para retornar ao ponto de chamada da função.

```
addi $t1, $a1, -1                Janeiro é mês 1
mul  $t1, $t1, 30                30 dias por mês
addi $t2, $a2, -2000             EPOCH em 2000
```

```
mul  $t2, $t2, 360          360 dias por ano
add  $v0, $t2, $t1          Junta os valores...
add  $v0, $v0, $a0          ...em $v0
jr $ra                      Retorna ao ponto de chamada
```

Decodificação

A função de decodificar o EPOCH funciona ao contrário, recebendo o valor codificado em `$v0`, e devolvendo dia, mês e ano em `$a0`, `$a1` e `$a2`, respectivamente.

Dividimos o valor recebido por 360. Ao valor da divisão inteira, somamos 2000, recuperando então o valor do ano.

Então multiplicamos novamente por 360 e subtraímos do valor recebido originalmente (o resto da divisão). Dividimos por 30, e somamos 1, recebendo então o valor do mês.

Novamente multiplicamos por 30 e subtraímos do resultado anterior, recuperando imediatamente o valor do dia. Utilizamos o `jr $ra` para retornar ao ponto de chamada.

```
div  $a2, $v0, 360          Parte inteira em $a2 (ano)
mul  $t0, $a2, 360
sub  $t0, $v0, $t0          Resto em $t0 (mês e dia)
div  $a1, $t0, 30           Parte inteira em $a1 (mês)
mul  $t1, $a1, 30
sub  $a0, $t0, $t1          Resto em $a0 (dia)
addi $a2, $a2, 2000         EPOCH em 2000
addi $a1, $a1, 1            Janeiro é mês 1
jr $ra                      Retorna ao ponto de chamada
```

Exclusão

Primeiramente é solicitada uma data, e essa é recebida e convertida em seguida pelo algoritmo EPOCH. Com a finalidade de não perder os dados presentes nos registradores `$s6` (início da lista) e `$s7` (contador de registros), estes foram copiados para os temporários `$t5` e `$t6`, respectivamente.

E então é verificado se há algum registro, se sim, verifica se ele é o primeiro, se sim novamente, o registro é excluído e o `$s6` passa a ter o endereço do próximo elemento (posteriormente guardado a partir do byte 28 da “Struct” do registro). Se o elemento não for o primeiro, o endereço presente no ponteiro do registro anterior passa a ser o endereço do ponteiro do elemento excluído.

Caso a data informada não seja a mesma do primeiro registro, ele entra em um loop que passa de registro por registro (adicionando 28 ao registrador temporário e dando load para ir para o próximo), sempre pegando a data (presente no primeiro campo da “Struct”), comparando e decrementando o contador. A condição de parada desse loop é que o contador do registrador `$t6` chegue a *ZERO* ou que o elemento seja encontrado.

A exclusão é simbolizada para o programa colocando um *ZERO* na primeira palavra do registro, e o contador de registros é atualizado adicionando -1 ao seu valor.

Consumo médio

Na função do consumo médio utilizamos registradores temporários para guardar a distância total percorrida e a quantidade total consumida de combustível, além de um outro temporário que inicialmente guarda o ponteiro inicial dos abastecimentos cadastrados.

Dessa forma, cada vez que o usuário chama a função que retorna o consumo médio é feita a soma de todas as quilometragens e todo o combustível registrado, esse loop é feito com a utilização de um jump que verifica se o ponteiro para o próximo registro é nulo, enquanto isso o registrador `$t0` guarda o endereço de memória do próximo abastecimento salvo.

Quando o loop é encerrado nós subtraímos registros que não devem entrar na conta, como a primeira quilometragem e o último abastecimento.

No final da função verificamos se o consumo total é nulo, se for verdade é retornado ao usuário uma mensagem de que não há informações suficientes para o cálculo de consumo médio, caso contrário o consumo médio é apresentado ao usuário, retornando ao menu no final.

Preço médio

Para o preço médio foi utilizado registradores temporários para armazenar cada informação que seria destinada a esta função:

- `$t0` - registrador que percorrerá a lista ligada;
- `$t1` - registrador que aponta para a atual string lida;

- `$t3` - início da pilha;
- `$t4` - quantidade de *PMCad*;
- `$t5` - registrador que aponta para a string guardada, ou seja, que está sendo comparada com a atual (`$t1`);
- `$f31` - registrador utilizado como temporário para armazenar o valor float do preço;

Assim, iniciamos a função lendo o primeiro cadastro para receber em `$t1` o endereço base do nome do posto. Após isso comparamos com os *PMCads* presentes em pilha e tomamos duas rotas. Caso haja um nome igual ao lido no cadastro, atualizamos este *PMCad* incrementando em 1 a sua frequência, somando o novo preço do posto (guardado em `$f31`) e atualizando a média do valor do posto ($\frac{\Sigma\, preços}{frequências}$). Porém se não existir tal nome, inserimos um novo *PMCad* com valores padrões de frequência (1), de preço atual e valor médio (ambos atuais) e o nome do novo posto.

Após todas as leituras de cadastros e inserções de *PMCads* em pilha, realizamos o método de ordenação por bolha para ordenar cada estrutura *PMCad* e finalmente obter os valores de preços médios ordenados, prontos para serem imprimidos na tela.

Exibição

Como nosso código foi estruturado para o registrador `$s6` sempre possuir o endereço do primeiro elemento armazenado e o registador `$s7` sempre possuir a quantidade de cadastros, passamos seus respectivos valores aos registradores temporários `$t5` e `$t6` para poder usá los livremente.

A função funciona de maneira bem simples, caminhamos pelas structs com o registrador temporário $t5 exibindo região por região até alcançar o último elemento que contém o endereço da próxima struct. O registrador $t6 foi usado em um primeiro momento para realizar um índice entre as informações exibidas na tela com o intuito de ficar uma disposição mais agradável ao usuário e em um segundo momento como condição de parada para nosso loop, ou seja, quando nosso registrador chega ao zero sabemos que todos os cadastros já foram exibidos

Resultados

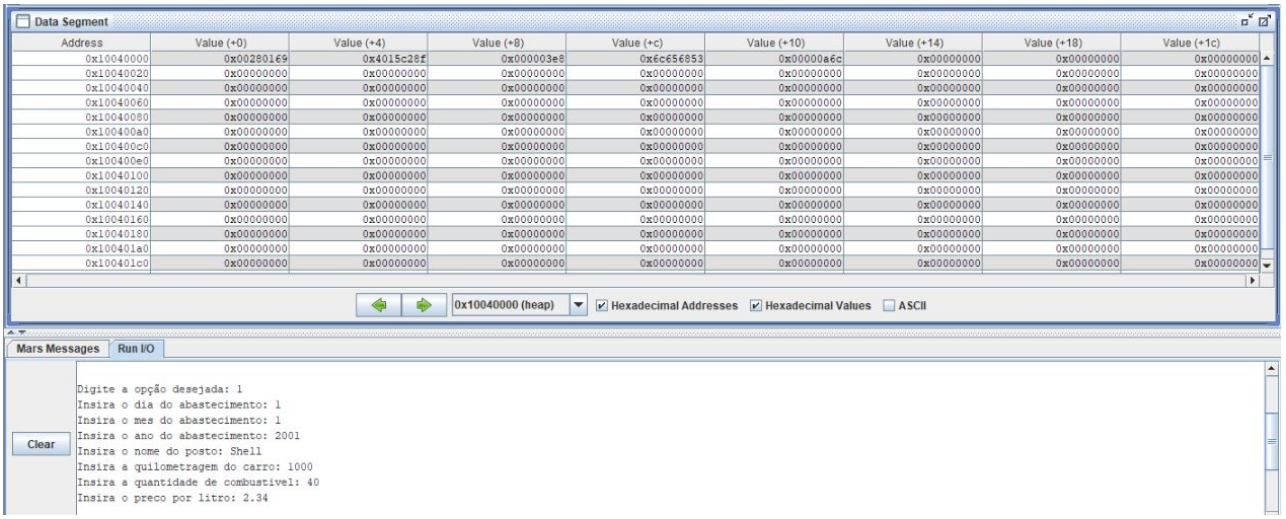
Testes Realizados

Para fase de testes, foi criado uma planilha para organização dos dados a serem cadastrados e então testados em cada função do programa (Cadastro, Exibir cadastro, Exclusão de cadastro, Consumo médio e Preço médio), totalizando 7 cadastros.

1	1	1	2001	Shell	1000	40	2,34
2	2	2	2002	Shell	2000	50	3,12
3	3	3	2003	Ipiranga	3000	50	2,26
4	4	4	2004	Petrobras	4000	50	2,5
5	6	6	2006	Ipiranga	5000	50	3
6	7	7	2007	Petrobras	6000	50	3,67
7	8	8	2008	Petrobras	8000	50	4,1

Resultados e discussão

Para o primeiro teste foi realizado o para verificar a primeira função (1.Cadastro), obtendo êxito ao mostrar que os dados foram guardados na primeira região do HEAP contendo um cadastro.



Para o segundo teste, verificamos a integração da função de exibição após cadastrar todos os valores anteriores citados na tabela.

```
1.Cadastrar abastecimento;
2.Excluir abastecimento;
3.Exibir abastecimento;
4.Consumo médio;
5.Preço médio;
6.Sair

Digite a opção desejada: 3
Lista de abastecimentos:
1. 01/01/2001 | 040 Litros | 2.34 R$/L | 01000 Km | Shell
2. 02/02/2002 | 050 Litros | 3.12 R$/L | 02000 Km | Shell
3. 03/03/2003 | 050 Litros | 2.26 R$/L | 03000 Km | Ipiranga
4. 04/04/2004 | 050 Litros | 2.5 R$/L | 04000 Km | Petrobras
5. 06/06/2006 | 050 Litros | 3.0 R$/L | 05000 Km | Ipiranga
6. 07/07/2007 | 050 Litros | 3.67 R$/L | 06000 Km | Petrobras
7. 08/08/2008 | 050 Litros | 4.1 R$/L | 08000 Km | Petrobras
```

Após obtermos todos os cadastros, realizamos as funções de consumo médio e preço médio com 7 cadastrados.

```
1.Cadastrar abastecimento;
2.Excluir abastecimento;
3.Exibir abastecimento;
4.Consumo médio;
5.Preço médio;
6.Sair

Digite a opção desejada: 4
Consumo médio:    24.137932 Km/L
```

Opção 4 - Consumo médio

```
1.Cadastrar abastecimento;
2.Excluir abastecimento;
3.Exibir abastecimento;
4.Consumo médio;
5.Preço médio;
6.Sair

Digite a opção desejada: 5
  | Nome do Posto     | Preço Médio
1 | Petrobras         | R$ 3.4233334
2 | Shell             | R$ 2.73
3 | Ipiranga          | R$ 2.63
```

Opção 5 - Preço médio

Para finalizar os testes, realizamos a última função restante (Exclusão), mostrando que foi bem sucedida na sua implementação ao executar, em seguida, a função de exibir.

```
1.Cadastrar abastecimento;
2.Excluir abastecimento;
3.Exibir abastecimento;
4.Consumo médio;
5.Preço médio;
6.Sair

Digite a opção desejada: 2
Insira o dia do abastecimento: 2
Insira o mes do abastecimento: 2
Insira o ano do abastecimento: 2002
O registro foi excluido com sucesso!

1.Cadastrar abastecimento;
2.Excluir abastecimento;
3.Exibir abastecimento;
4.Consumo médio;
5.Preço médio;
6.Sair

Digite a opção desejada: 3
Lista de abastecimentos:
1. 01/01/2001 | 040 Litros | 2.34 R$/L | 01000 Km | Shell
2. 03/03/2003 | 050 Litros | 2.26 R$/L | 03000 Km | Ipiranga
3. 04/04/2004 | 050 Litros | 2.5 R$/L | 04000 Km | Petrobras
4. 06/06/2006 | 050 Litros | 3.0 R$/L | 05000 Km | Ipiranga
5. 07/07/2007 | 050 Litros | 3.67 R$/L | 06000 Km | Petrobras
6. 08/08/2008 | 050 Litros | 4.1 R$/L | 08000 Km | Petrobras
```

Após todos os testes é importante ressaltar que mesmo após as explicações da função EPOCH, as datas inseridas para cadastro deveriam ser a partir do ano de 2000.

Bibliografia

- http://www.tfinley.net/notes/cps104/mips.html;
- https://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html;
- Slides disponibilizados no Ambiente Virtual de Aprendizagem (AVA) da disciplina de Arquitetura de Computadores;

Anexos

Segue abaixo todo código escrito em linguagem de montagem MIPS x32:

```
.data
ReadString:   .space       16     #Nome do Posto
StructInfo:   .space 32     #Número dado em bytes: | Data-2 | QtdCombustível-2 |
Preço-4 | Distancia-4 | NomePosto-16 | PonteiroProx-4

#Declaração de strings
Cadastrar:      .asciiz "\n1.Cadastrar abastecimento;\n"
Excluir:        .asciiz "2.Excluir abastecimento;\n"
EAbastecimento: .asciiz "3.Exibir abastecimento;\n"
EConsumoMedio:  .asciiz "4.Consumo médio;\n"
EPrecoMedio:    .asciiz "5.Preço médio;\n"
SairProg:     .asciiz "6.Sair\n\n"

DigiteOpcao:    .asciiz "Digite a opção desejada: "

Ins_Dia:      .asciiz "Insira o dia do abastecimento: "
Ins_Mes:      .asciiz "Insira o mes do abastecimento: "
Ins_Ano:      .asciiz "Insira o ano do abastecimento: "
Ins_Nome:     .asciiz "Insira o nome do posto: "
Ins_Qlmt:     .asciiz "Insira a quilometragem do carro: "
Ins_Qntd:     .asciiz "Insira a quantidade de combustivel: "
Ins_Prec:     .asciiz "Insira o preco por litro: "

EncontradoReg:  .asciiz "O registro foi excluido com sucesso! \n"
SemReg:             .asciiz "Nenhum registro foi encontrado! \n"

ExibePorData: .asciiz "Lista de abastecimentos:\n"

Kms:          .asciiz " Km"
Litros:             .asciiz " Litros"


Consumo:      .asciiz "Consumo médio:    "
SemRegConsumo:      .asciiz "Não há registros que indiquem algum consumo,
retornando ao menu...\n"
KmL:          .asciiz " Km/L"

ReaisPorLitro:  .asciiz " R$/L"

Reais:        .asciiz "R$ "

Separacao:      .asciiz " | "
Barra:        .asciiz "/"
Espaco:             .asciiz " "
Ponto:        .asciiz "."
FimDeLinha:   .asciiz "\n"
Zero:         .asciiz "0"

Ex_Qlmt:      .asciiz "Quilometragem: "
MenuNomePreco:      .asciiz "  | Nome do Posto    | Preço Médio\n"
.text
main:
```

```
#----------- Inicializando -----------#
       #subi $sp, $sp, -28
       divu $sp, $sp, 32
       mulu $sp, $sp, 32

       and $s7,$s7,$zero   #"Seta" $s7 para 0 pois este contará quantos cadastros
foram feitos
       add $fp,$sp,$zero   #Escreve o valor maximo da pilha em FP
       #lui $s6,0x1004

       add $s6,$zero,$zero #O ponteiro inicial será guardado em $s6

#------------ Exibir Menu ------------#
Menu:
       li  $v0, 4
       la $a0, Cadastrar
       syscall

       la $a0, Excluir
       syscall

       la $a0, EAbastecimento
       syscall

       la $a0, EConsumoMedio
       syscall

       la $a0, EPrecoMedio
       syscall

       la $a0, SairProg
       syscall

       la $a0, DigiteOpcao
       syscall

#------------ Exibir Menu ------------#

#--------- Opção Selecionada ---------#
       li $v0, 5
       syscall

       beq $v0,1,Cadastro
       beq $v0,6,Exit
       bne $s6,$zero,StartOpcao
       li $v0,4
       la $a0,SemReg
       syscall
       j  Menu
StartOpcao:

       beq $v0,2,Exclui
       beq $v0,3,EAbastece
       beq $v0,4,EConsumo
       beq $v0,5,EMedio

       j  Menu

#--------- Opção Selecionada ---------#

#------ Cadastro Abastecimento -------#
Cadastro:
       jal RData
       add $t1,$zero,$v0 # $t1 detém do valor da EPOCH
```

```
	li $v0,4	#Recebe Nome do Posto no Addr. ReadString
	la $a0,Ins_Nome
	syscall
	li $v0,8
	la $a0,ReadString
	li $a1,16
	syscall

	li $v0,4	#Recebe Quilometragem em $s1
	la $a0,Ins_Qlmt
	syscall
	li $v0,5
	syscall
	add $s1,$zero,$v0

	li $v0,4	#Recebe Quantidade de combustível em upper($s0)

	la $a0,Ins_Qntd
	syscall
	li $v0,5
	syscall
	sll $v0,$v0,16
	or $t1,$t1,$v0

	li $v0,4	#Recebe Preço do litro em $f0

	la $a0,Ins_Prec
	syscall
	li $v0,6
	syscall

	la $s4,ReadString

	#addi $sp,$sp,-28

	#jal malloc
	addi $a0, $zero, 8
	jal nalloc

	add $t7,$v0,$zero #current pointer in t7

	sw $t1,0($v0)#Data e Qtd Comb. OK
	s.s $f0,4($v0)	#Preço do litro	OK
	sw $s1,8($v0)#Km Atual	O
	addi $v0,$v0,0xc
	addi $t0,$zero,4
StoreWord:		#Nome do Posto
	lw $t1,0($s4)
	sw $t1,0($v0)
	addi $s4,$s4,4
	addi $v0,$v0,4
	addi $t0,$t0,-1
	bnez $t0,StoreWord

	#Start of linked list insertion
	add  $v0, $t7, $zero
	beq  $s7, $zero, emptyList #if list is empty

	lw   $t4, 0($v0) #store current item epoch
	andi $t4, $t4, 65535 #crop epoch data ???
	lw   $t1, 0($s6)
	andi $t1, $t1, 65535 #crop epoch data ???

	slt  $t3, $t4, $t1 #t3 = 1 if first data < new data
	bne  $t3, $zero, emptyList
```

```
	add  $t0, $s6, $zero	#Pega inicio da lista ligada
findNext:
	lw   $t2, 28($t0) #t2 is current->next
	beq  $t2, $zero, exitFindNext

	lw   $t1, 0($t2) #t1 is current->next->data
	andi $t1, $t1, 65535 #crop epoch data ???

	slt  $t3, $t1, $t4 #t3 is 1 if current->next->data < new data
	beq  $t3, $zero, exitFindNext

	add  $t0, $t2, $zero #keep walking along list
	j findNext
exitFindNext:
	sw   $t2, 28($v0) #new node -> next = current -> next
	sw   $v0, 28($t0) #current -> next = current addr
	j doneAdding

emptyList:
	sw  $s6,28($v0) #new node -> next = old pointer
	add $s6, $v0, $zero #old pointer = new node addr
	#j doneAdding

	#sw $zero,0($v0) #Ponteiro "seta prox"
	#addi $sp,$sp,-32
doneAdding:

	addi $s7,$s7,1

	j Menu
#------ Cadastro Abastecimento -------#

#-------- Excluir Abstecimento -------#

	#ponteiro inicial $s6
	#$s7 qtd registros

	#$t1 valor da data digitado em EPOCH
	#$t2 valor da data na lista ligada
	#$t3 Ponteiro anterior
	#$t4 ->prox do item excluido
	#t5 ponteiro da lista
	#$t6 decrementador

Exclui:
	jal RData
	add $t1,$zero,$v0 # valor da EPOCH em $t1

	add $t5, $s6, $zero # Ponteiro da lista em $t5
	add $t6, $s7, $zero # Quantidade de registros em $t6
	beq $t6, $zero, MsgSemReg


	lh $t2, 0($t5)	# Carrega a data da lista
	la $t3, ($t5)
	beq $t1, $t2, ExcluiPrimeiroElemento
	addi $t6, $t6, -1

LoopProcuraData:
	beq $t6, $zero, MsgSemReg

	addi $t5, $t5, 28 # Avança pra prosição do ponteiro
	addi $t6, $t6, -1
	lw $t5, 0($t5) # Proximo elento da lista
```

```
	lh $t2, 0($t5)	# Carrega a data da lista
	beq $t1, $t2, ExcluiRealmente
	la $t3, ($t5)

	bne $t1, $t2, LoopProcuraData

MsgSemReg:
	li $v0, 4
	la $a0, SemReg
	syscall
	j FimExclui

MsgEncontradoReg:
	li $v0, 4
	la $a0, EncontradoReg
	syscall
	j FimExclui

ExcluiPrimeiroElemento:
	sw $zero, 0($t5)
	lw $t3, 28($t5)
	add $s6, $t3, $zero
	addi $s7, $s7, -1
	j MsgEncontradoReg

ExcluiRealmente:

	sw $zero, 0($t5)
	lw $t4, 28($t5)
	sw $t4, 28($t3)

	addi $s7, $s7, -1
	j MsgEncontradoReg

FimExclui:
	j Menu
#-------- Excluir Abstecimento -------#

#--------- Exibe Abastecimento -------#
EAbastece: # FORMAT <DD>/<MM>/<AAAA> | <INT>Km | <INT> litros (<FLOAT> R$/l) |
Posto <posto>
	#Número dado em bytes: | Data-2 | QtdCombustível-2 | Preço-4 | Distancia-4
| NomePosto-16 | PonteiroProx-4
	#ponteiro inicial $s6
	#$s7 qtd registros
	#t5 ponteiro da lista
	#$t6 decrementador
	#$t7 contador de registro

	li $v0,4	#Recebe Nome do Posto no Addr. ReadString
	la $a0, ExibePorData
	syscall

	add $t5, $s6, $zero


	addi $t7, $zero, 1
	add $t6, $s7, $zero

LoopExibe:
	beq $t6, $zero, FimExibe


	li $v0, 1
	add $a0, $t7, $zero
```

```
syscall                 #exibe o indice
jal PrintaPonto
jal PrintaEspaco

lh $v0, 0($t5)
#and $v0, $v0, 65535
jal EpochToDate     #pega a data e desconverte do epoc


add $t0, $a0, $zero
jal IdentaData
add $a0, $t0, $zero

li $v0, 1
syscall             #printa dia
jal PrintaBarra

add $a0, $a1, $zero
add $t0, $a1, $zero
jal IdentaData
add $a0, $t0, $zero
li $v0, 1

syscall             #printa mes
jal PrintaBarra

li $v0, 1
add $a0, $a2, $zero
syscall             #printa ano

jal PrintaSeparacao

addi $t5, $t5, 2

lh $a0, 0($t5)

add $t3, $a0, $zero
add $t4, $a0, $zero
jal identacaoCombustivel
add $a0, $t4, $zero

li $v0, 1
syscall                 #printa qtd combustivel

jal PrintaLitros
jal PrintaSeparacao

addi $t5, $t5, 2

lwc1 $f12, 0($t5)
li $v0, 2
syscall                 #printa preco

jal PrintaReaisPorLitro
jal PrintaSeparacao

addi $t5, $t5, 4

lw $a0, 0($t5)

add $t0, $a0, $zero
add $t1, $a0, $zero
jal IdentaDistancia
add $a0, $t1, $zero
```

```
	li $v0, 1
	syscall                 #printa distancia

	jal PrintaKm
	jal PrintaSeparacao

	addi $t5, $t5, 4

	la $a0, ($t5)
	li $v0, 4
	syscall

	#jal PrintaFimDeLinha

	addi $t5, $t5, 16
	lw $t5, 0($t5)
	addi $t6, $t6, -1
	addi $t7, $t7,  1

	j LoopExibe

FimExibe:
	j Menu

PrintaEspaco:
	li $v0, 4
	la $a0, Espaco
	syscall
	jr $ra

PrintaBarra:
	li $v0, 4
	la $a0, Barra
	syscall
	jr $ra

PrintaPonto:
	li $v0, 4
	la $a0, Ponto
	syscall
	jr $ra

PrintaSeparacao:
	li $v0, 4
	la $a0, Separacao
	syscall
	jr $ra

PrintaLitros:
	li $v0, 4
	la $a0, Litros
	syscall
	jr $ra

PrintaReaisPorLitro:
	li $v0, 4
	la $a0, ReaisPorLitro
	syscall
	jr $ra

PrintaKm:
	li $v0, 4
	la $a0, Kms
	syscall
```

```
	jr $ra

PrintaFimDeLinha:
	li $v0, 4
	la $a0, FimDeLinha
	syscall
	jr $ra

identacaoCombustivel:
	div $t0, $t3, 100
	bne $t0, $zero, Identado
	li $v0, 4
	la $a0, Zero
	syscall
	mul $t3, $t3, 10

	j identacaoCombustivel

Identado:
	jr $ra

IdentaData:
	div $t1, $t0, 10
	bne $t1, $zero, Identadoo
	li $v0, 4
	la $a0, Zero
	syscall
Identadoo:
	jr $ra

IdentaDistancia:
	div $t2, $t0, 10000
	bne $t2, $zero, Identadooo
	li $v0, 4
	la $a0, Zero
	syscall
	mul $t0, $t0, 10
	j IdentaDistancia

Identadooo:
	jr $ra
#--------- Exibe Abastecimento -------#

#---------- Exibe Consumo ------------#
EConsumo:
	add $t0, $s6, $zero
	add $t4, $zero, $zero
	add $t2, $zero, $zero

	bne $t0, $zero, LConsumo

	mtc1 $t0, $f0
	cvt.s.w $f0, $f0
	add.s $f12, $f0, $f0
	li $v0, 2
	syscall

	j    Menu

	LConsumo:

	lh  $t1, 2($t0)
	add $t2, $t2, $t1          #salva a quantidade de combustivel

	lw  $t3, 8($t0)
```

```
	add $t4, $zero, $t3		#salva Km

	add $t1, $zero, $t0
	lw  $t0, 28($t0)
	bne $t0, $zero, LConsumo

	lh  $t0, 2($t1)
	sub $t2, $t2, $t0
	add $t0, $zero, $s6
	lw  $t3, 8($t0)
	sub $t4, $t4, $t3

	beq $t4, $zero, SemConsumo

	mtc1 $t4, $f0
	cvt.s.w $f0, $f0
	mtc1 $t2, $f1
	cvt.s.w $f1, $f1
	div.s $f12, $f0, $f1

	li $v0, 4
	la $a0, Consumo
	syscall

	li $v0, 2
	syscall

	li $v0, 4
	la $a0, KmL
	syscall

	jal PrintaFimDeLinha
	j Menu

SemConsumo:
	li $v0, 4
	la $a0, SemRegConsumo
	syscall

	j Menu
#---------- Exibe Consumo ------------#

#-------- Exibe Preco Medio ----------#
EMedio:	#$t0 - reg. temp. para percorrer lista; $t1 - reg. para guardar
string temp.; $t3 - inicio pilha; $t4 - quantidade em pilha; $t5 - temp. para
receber word; $f31 - preço temp.
	#Data Format, bytes - | val medio - 4 | freq - 4 | nome - 16 | sigma
preços - 4|
	add $t0,$s6,$zero
	add $fp,$sp,$zero
	add $t4,$zero,$zero
	add $t3,$sp,$zero
	la $t1,ReadString

ReadNewEntry: #Lê nova entrada
	l.s $f31,4($t0)
	addi $t0,$t0,12
	addi $t2,$zero,4	#$t2 - neste caso, para contar 1 word

CarregaStringEMedio:
	lw $t5,0($t0)
	sw $t5,0($t1)
	addi $t0,$t0,4
	addi $t1,$t1,4
	addi $t2,$t2,-1
```

```
	bne $t2,$zero,CarregaStringEMedio
	addi $t1,$t1,-16

	beq $t3,$sp,AnotherEntry
	add $fp,$zero,$t3   #$fp recebe o endereço do inicio da pilha
	la $a0,($t1)
CheckNameEMedio:    #Procura se há ocorrencia da palavra então carregada
	la $a1,-24($fp)
	jal cmpstr
	addi $fp,$fp,-28
	bne $v0,$zero,IncrementEntry
	beq $fp,$sp,AnotherEntry
	j CheckNameEMedio

IncrementEntry:            #Contabiliza novos resultados daquela palavra
	l.s $f30,0($fp)
	add.s  $f31,$f31,$f30
	s.s $f31,0($fp)
	lw $t2,20($fp)
	addi $t2,$t2,1
	sw $t2,20($fp)
	mtc1 $t2,$f30 #Move $t2 para COPROCESSOR 1 (FLU-floating point unit)
	cvt.s.w $f30,$f30 #Move os 32-bit inteiros da direita para um ponto
flutuante (reg. esquerda)
	div.s $f31,$f31,$f30
	s.s $f31,24($fp)
	j PrepareToReadNewEntry

AnotherEntry:       #Nova palavra, novo espaço na pilha...
	addi $sp,$sp,-28    #libera 28 bytes
	s.s $f31,0($sp)
	addi $fp,$fp,-24    #pos 24
	addi $t2,$zero,4    #$t2 - para contar 4 words
StoreFirstEMedio:
	lw $t5,0($t1)
	sw $t5,0($fp)
	addi $fp,$fp,4
	addi $t1,$t1,4
	addi $t2,$t2,-1
	bne $t2,$zero,StoreFirstEMedio
	addi $t1,$t1,-16

	addi $t2,$zero,1    #pos 8 e $t2 - "seta" frequencia
	sw $t2,0($fp)
	addi $fp,$fp,4             #pos 4 - guarda valor medio atual
	s.s $f31,0($fp)
	add $t4,$t4,1
	#$lw $t2,-4($fp)
	#l.s $f31,-24($fp)
	#div.s
	#sw $f31,0($fp)

PrepareToReadNewEntry:            #Vamos para o próximo cadastro...
	lw $t2,0($t0)       #$t2 = end. do prox
	beq $t2,$zero,ExitEMedio
	add $t0,$t2,$zero
	j ReadNewEntry

ExitEMedio:
	#Todo processo de imprimir em decrescente e entao voltar para o menu
	add $a0,$zero,$t3   #inicio da pilha em $a0
	add $a1,$zero,$t4   #quantidade de infos da pilha em $a1

	jal BSort
	#$v0 - inicio da pilha; $v1 - quantidade de infos
```

```
	#$t0 - comparador de \n; $t1 - reg. para guardar string temp.; $s5 -
inicio da pilha; $t5 - temp. contador de null; $t6 - recebe bytes; $t7 - contador
de índice.
	#Data Format, bytes - | val medio - 4 | freq - 4 | nome - 16 | sigma
preços - 4|
	add $s5,$zero,$v0

	li  $v0, 4
	la $a0, MenuNomePreco
	syscall

	addi $t7,$zero,1
	la $t1,ReadString
LoopPrintEMEdio:
	li $v0, 1
	add $a0, $t7, $zero
	syscall                 #exibe o indice
	jal PrintaSeparacao

	addi $t2,$zero,16   #numero de repetiçoes do loop abaixo
	add $t5,$zero,$zero #contador de null
	addi $t0,$zero,10   #comparador de \n
	addi $sp,$sp,4
LoopPrintNomePostoEMEdio:
	lb $t6,0($sp)
	bne $t6,$t0,ContinueStoringEMedio
	add $t6,$zero,$zero
ContinueStoringEMedio:
	sb $t6,0($t1)
	addi $t1,$t1,1
	addi $sp,$sp,1
	addi $t2,$t2,-1
	bne $t6,$zero,DontCountNullEMedio
	addi $t5,$t5,1
DontCountNullEMedio:
	bne $t2,$zero,LoopPrintNomePostoEMEdio

	addi $t1,$t1,-16
	addi $sp,$sp,4

	li $v0,4
	add $a0,$t1,$zero
	syscall

AligningTextEMedio:
	jal PrintaEspaco
	addi $t5,$t5,-1
	bne $t5,$zero,AligningTextEMedio

	jal PrintaSeparacao

	li $v0,4
	la $a0,Reais
	syscall

	l.s $f12,0($sp)
	li $v0,2
	syscall
	addi $sp,$sp,4

	jal PrintaFimDeLinha

	addi $t7,$t7,1
	bne $sp,$s5,LoopPrintEMEdio
	j Menu
```

```
#-------- Exibe Preco Medio ----------#

#------ Converte Data para EPOCH -----#
DateToEpoch: #DD em $a0 - MM em $a1 - AAAA em $a2
       addi $t1, $a1, -1 # Janeiro é mes 1
       mul  $t1, $t1, 30

       addi $t2, $a2, -2000 # EPOCH em 2000
       mul  $t2, $t2, 360

       add  $v0, $t2, $t1
       add  $v0, $v0, $a0 # Result em $v0

       jr $ra
#------ Converte Data para EPOCH -----#

#------ Converte EPOCH para Data -----#
EpochToDate: #EPOCH em $v0
       div  $a2, $v0, 360 # Parte inteira em $a2 (ano)
       mul  $t0, $a2, 360
       sub  $t0, $v0, $t0 # Resto em $t0 (mes e dia)

       div  $a1, $t0, 30 # parte inteira em $a1 (mes)
       mul  $t1, $a1, 30
       sub  $a0, $t0, $t1 # Resto em $a0 (dia)

       addi $a2, $a2, 2000 # EPOCH em 2000
       addi $a1, $a1, 1 # Janeiro é mes 1

       jr $ra
#------ Converte EPOCH para Data -----#

#------------ Recebe data ------------#
RData:
       li $v0,4      #Recebe Dia em $a0
       la $a0,Ins_Dia
       syscall
       li $v0,5
       syscall
       add $t7,$zero,$v0

       li $v0,4      #Recebe Mês em $a1

       la $a0,Ins_Mes
       syscall
       li $v0,5
       syscall
       add $t6,$zero,$v0

       li $v0,4      #Recebe Ano em $a2
       la $a0,Ins_Ano
       syscall
       li $v0,5
       syscall
       add $t5,$zero,$v0

       add $a0,$zero,$t7
       add $a1,$zero,$t6
       add $a2,$zero,$t5

       add $t4,$zero,$ra
       jal DateToEpoch

       jr $t4
#------------ Recebe data ------------#
```

```
#------------- new malloc ------------#
nalloc:
       lui  $t0, 0x1004    #Start search at the beginning of heap
_next: lw   $t2, 0($t0)
       beq  $t2, $zero, _found    #If found zeroed-out position, check for
contiguous
       addi $t0, $t0, 32
       j _next
_found:
       add $v0, $t0, $zero
       #addi $t0, $t0, 4   #Check next for continuity
       jr $ra
#------------- new malloc ------------#

#--------------- cmpstr --------------#
cmpstr:      #$a0 - End. String 1, $a1 - End. String 2
       lb $t6,0($a0)
       lb $t7,0($a1)

       bne $t6,$t7,cmpstrExitFalse
       beq $t6,$zero,cmpstrExitTrue

       addi $a0,$a0,1
       addi $a1,$a1,1

       j cmpstr
cmpstrExitTrue:
       addi $v0,$zero,1    #Retorna 1 caso a string seja igual
       jr $ra
cmpstrExitFalse:
       add $v0,$zero,$zero #Retorna 0 caso a string seja diferente
       jr $ra
#--------------- cmpstr --------------#

#--------------- BBSort --------------#
#$a0 - inicio pilha; $a1 - n infos;
#compare floating point number:
#c.eq.s fs,ft (compare if fs is equal to ft)
#bc1t LABEL (if true branch to the LABEL)
BSort:
       add $t6, $a0, $zero # save $a0 into $t6
       add $t5, $a1, $zero # save $a1 into $t5
       addi $t7, $zero,1 # i = 1
for1tst:
       slt $t0, $t7, $t5 # $t0 = 0 if $t7 ? $t5 (i ? n)
       beq $t0, $zero, exit1 # go to exit1 if $t7 ? $t5 (i ? n)
       addi $s1,$t7,-1  # j = i - 1
for2tst:
       slti $t0, $s1, 0 # $t0 = 1 if $s1 < 0 (j < 0)
       bne $t0, $zero, exit2 # go to exit2 if $s1 < 0 (j < 0)
       mulu $t1, $s1, 28 # $t1 = j * 28
       sub $t2, $t6, $t1 # $t2 = v - (j * 28)
       l.s $f4, -4($t2) # $f4 = v[j]
       l.s $f5, -32($t2) # $f5 = v[j + 1]
       c.lt.s $f5,$f4 # flag 0 = false if $f5 ? $f4
       bc1f exit2 # go to exit2 if $f5 ? $f4

       add $a0,$zero,$t6 # 1st param of swap is v (old $a0)
       add $a1,$zero,$s1 # 2nd param of swap is j
       j BSortSwap # call swap procedure
ReturnBSortSwap:
       addi $s1,$s1,-1 # j -= 1
       j for2tst # jump to test of inner loop
exit2:
```

```
       addi $t7, $t7, 1 # i += 1
       j for1tst # jump to test of outer loop
exit1:
       add $v0,$zero,$t6
       add $v1,$zero,$t5
       jr $ra

BSortSwap:    #trocar posições!
       addi $a1,$a1,1
       mulu $t1, $a1, 28    # $t1 = k * 4
       sub $t1, $a0, $t1    # $t1 = v-(k*4)(address of v[k])
       addi $t4, $zero, 7

BSortLoopSwap:
       lw $t0, 0($t1)              # $t0 (temp) = v[k]
       lw $t2, -28($t1)            # $t2 = v[k+1]
       sw $t2, 0($t1)              # v[k] = $t2 (v[k+1])
       sw $t0, -28($t1)            # v[k+1] = $t0 (temp)
       addi $t1,$t1,4
       addi $t4,$t4,-1
       bne $t4,$zero,BSortLoopSwap
       j ReturnBSortSwap
#--------------- BBsort --------------#

Exit:
       li $v0,17
       li $a0,0
       syscall
```