CIÊNCIA DA COMPUTAÇÃO

Programação para Interfaceamento de Hardware e Software (9792)

TRABALHO 2

Professor: Ronaldo Augusto de Lara Gonçalves

Discentes

RA	NOME
130099	GUSTAVO HENRIQUE TRASSI GANAZA
129182	YOSHIYUKI FUGIE

Maringá

2025

Conteúdo

1 Visão Geral do Sistema

Este relatório detalha a implementação do sistema de gerenciamento de produtos em Assembly 32 bits, com foco nas modificações realizadas para suportar ponto flutuante e syscalls. O sistema utiliza uma lista encadeada simplesmente ligada para armazenar os produtos, implementando operações CRUD, consultas financeiras e geração de relatórios. As principais mudanças incluem o uso de valores monetários em float e a substituição de funções de biblioteca por chamadas de sistema para operações de E/S em arquivos.

A entrega inclui um arquivo makefile para compilar o código-fonte. O makefile utiliza *gcc -m32* para gerar código 32 bits e *-lm* para linkar a biblioteca matemática, portanto é necessário ter instalado o pacote *gcc-multilib* no sistema (em distribuições baseadas no Debian/Ubuntu: *sudo apt install gcc-multilib*). Para compilar o código, execute o comando *make* no terminal. Para rodar o programa, execute *./supermercado*.

2 Estrutura de Dados Modificada

A estrutura de dados foi otimizada para suportar valores monetários em ponto flutuante:

- **next**: ponteiro para o próximo nó (4 bytes)
- **tipo**: inteiro (4 bytes)
- **quantidade**: inteiro (4 bytes)
- **valor_compra**: float (4 bytes) - **MODIFICADO**
- **valor_venda**: float (4 bytes) - **MODIFICADO**
- **nome**: string (50 bytes)
- **lote**: string (20 bytes)
- **dia**: inteiro (4 bytes)
- **mês**: inteiro (4 bytes)
- **ano**: inteiro (4 bytes)
- **fornecedor**: string (50 bytes)

Offsets dos campos no nó:

- OFFSET_COMPRA: 12 bytes (campo float)
- OFFSET_VENDA: 16 bytes (campo float)
- Demais campos mantiveram seus offsets originais

O tamanho total permanece 152 bytes, mas agora com precisão decimal para valores monetários.

3 Implementação de Ponto Flutuante

3.1 Leitura de Valores Float

A entrada de valores monetários foi modificada para usar scanf com especificador `%f`:

```
# Leitura do valor de compra (float)
pushl $str_compra_prompt
call  printf
addl  $4, %esp
leal  OFFSET_COMPRA(%ebx), %eax
pushl %eax
pushl $str_float_input      # "%f"
call  scanf
addl  $8, %esp
```

3.2 Impressão de Valores Float

Para impressão, os valores float são convertidos para double na pilha da FPU:

```
# Imprime valor de compra (float)
flds  OFFSET_COMPRA(%ebx)   # Carrega float para FPU
subl  $8, %esp              # Espaço para double
fstpl (%esp)                # Converte para double
pushl $fmt_compra           # "Compra: %.2f\n"
call  printf
addl  $12, %esp
```

3.3 Operações Aritméticas com FPU

As funções financeiras foram completamente reescritas para usar a Floating Point Unit:

Instruções FPU utilizadas:

- `fldz`: carrega 0.0 no topo da pilha FPU
- `flds`: carrega float de 32 bits da memória
- `fildl`: carrega inteiro de 32 bits e converte para float
- `faddp`: soma e remove do topo da pilha
- `fmulp`: multiplica e remove do topo da pilha
- `fsubp`: subtrai e remove do topo da pilha
- `fstpl`: armazena como double e remove da pilha

4 Implementação de Syscalls para E/S de Arquivo

4.1 Definições de Syscalls

O código define constantes para syscalls do Linux 32-bit:

```
.set SYS_OPEN, 5
.set SYS_CLOSE, 6
.set SYS_READ, 3
.set SYS_WRITE, 4

.set O_RDONLY, 0
.set O_WRONLY, 1
.set O_CREAT, 64
.set O_TRUNC, 512
```

4.2 Gravação com Syscalls (save_list)

A função foi reescrita para usar syscalls em vez de `fopen/fwrite/fclose`:

1. **Abertura**: Usa syscall `open` com flags `O_WRONLY | O_CREAT | O_TRUNC`
2. **Escrita**: Para cada nó, copia 148 bytes (dados sem ponteiro) para buffer e usa syscall `write`
3. **Fechamento**: Usa syscall `close`

```
# Abrir arquivo
movl  $SYS_OPEN, %eax
movl  $filename, %ebx
movl  $(O_WRONLY | O_CREAT | O_TRUNC), %ecx
movl  $0644, %edx           # Permissões
int   $0x80

# Escrever dados
movl  $SYS_WRITE, %eax
# %ebx já tem o file descriptor
movl  $buffer, %ecx
movl  $dados_size, %edx
int   $0x80
```

4.3 Leitura com Syscalls (load_list)

Processo similar para leitura:

1. **Abertura**: Syscall `open` com flag `O_RDONLY`

2. **Leitura**: Loop que lê blocos de 148 bytes com syscall `read`
3. **Processamento**: Para cada bloco lido, aloca nó e insere na lista
4. **Término**: Quando `read` retorna menos que 148 bytes (EOF)

5 Funções Financeiras com Ponto Flutuante

5.1 Função total_venda

Implementada de forma similar à `total_compra`, mas utilizando o campo `valor_venda`:

```
total_venda:
    pushl %ebp
    movl  %esp, %ebp
    pushl %ebx

    fldz                          # ST(0) = 0.0
    movl  head, %ebx

venda_loop_float:
    testl %ebx, %ebx
    jz    venda_done_float

    fildl OFFSET_QUANTIDADE(%ebx)  # Quantidade como float
    flds  OFFSET_VENDA(%ebx)       # Valor de venda (float)
    fmulp %st, %st(1)              # quantidade * valor_venda
    faddp %st, %st(1)              # Acumula total

    movl  (%ebx), %ebx             # Próximo nó
    jmp   venda_loop_float

venda_done_float:
    # Resultado em ST(0)
    popl  %ebx
    leave
    ret
```

5.2 Função lucro_total

Calcula o lucro estimado subtraindo o total de compras do total de vendas:

```
lucro_total:
    pushl %ebp
    movl  %esp, %ebp
```

```
    call  total_compra  # ST(0) = total_compra
    call  total_venda   # ST(0) = total_venda, ST(1) = total_compra
    fsubp %st, %st(1)   # ST(0) = total_venda - total_compra

    leave
    ret
```

A ordem das chamadas é crucial - `total_compra` primeiro, depois `total_venda`, para que a subtração `fsubp` resulte em `venda - compra`.

5.3 Função capital_perdido

Esta função agora implementa a lógica de cálculo de capital perdido com ponto flutuante:

```
capital_perdido:
    # ... leitura da data atual ...

    movl  head, %ebx
    fldz                    # ST(0) = 0.0 (acumulador)

capital_loop:
    testl %ebx, %ebx
    jz    capital_done

    # Carregar data do produto e comparar
    # ... código de comparação de datas ...

    cmpl  $-1, %eax         # Se vencido (-1)
    jne   next_capital

    # Calcular perda usando FPU
    fildl OFFSET_QUANTIDADE(%ebx)   # Quantidade como float
    flds  OFFSET_COMPRA(%ebx)       # Valor de compra (float)
    fmulp %st, %st(1)               # quantidade * valor_compra
    faddp %st, %st(1)               # Acumula perda total

next_capital:
    movl  (%ebx), %ebx      # Próximo produto
    jmp   capital_loop
```

5.4 Função print_float_currency

Nova função para formatação de valores monetários:

```
print_float_currency:
    pushl %ebp
    movl  %esp, %ebp

    subl  $8, %esp          # Espaço para double
    fstpl (%esp)            # Converte ST(0) para double
    pushl 8(%ebp)           # Formato ("%.2f")
    call  printf
    addl  $12, %esp

    leave
    ret
```

6 Atualização de Valor de Venda com Float

A função `update_product_interactive` foi modificada para suportar corretamente a atualização de valores monetários em ponto flutuante:

```
update_venda:
    pushl $str_nova_venda           # Prompt para novo valor
    call  printf
    addl  $4, %esp

    leal  OFFSET_VENDA(%ebx), %eax  # Usa constante simbólica
    pushl %eax
    pushl $str_float_input          # "%f" - formato correto
    call  scanf                     # Lê valor como float
    addl  $8, %esp
    call  clear_input_buffer
```

7 Relatórios e Consultas

- **Relatório texto**: Agora usa `fprintf` com `%.2f`
- **Funções financeiras**: Todas migradas para FPU
- **Capital perdido**: Corrigido para usar float em vez de centavos
- **Ordenação**: Sem impacto (ordena por campos não monetários)

8 Arquitetura Híbrida: Syscalls + Biblioteca

O sistema adota uma abordagem híbrida otimizada:

- **Arquivos binários**: Syscalls (maior controle e eficiência).
- **Formatação de texto**: Funções de biblioteca (simplificação).
- **Entrada/saída console**: Funções de biblioteca (compatibilidade).

9 Tratamento de Erros

9.1 Verificações Implementadas

- **Syscalls**: Verificação de retorno negativo indica erro
- **malloc**: Teste de ponteiro nulo antes de uso
- **FPU**: Stack da FPU gerenciada corretamente para evitar overflow

9.2 Limitações Conhecidas

- **Validação de entrada**: Não há verificação de ranges para valores float
- **Overflow FPU**: Operações com valores muito grandes podem causar exceções
- **Precisão**: Float de 32 bits tem limitações para valores monetários muito grandes
- **Portabilidade**: Syscalls são específicos do Linux 32-bit

10 Conclusão

As principais modificações incluíram a conversão dos campos monetários de inteiros (centavos) para float e a substituição de funções de biblioteca por syscalls do Linux para operações de E/S em arquivos binários. Todas as funções financeiras foram reescritas para utilizar a Floating Point Unit (FPU) do processador, incluindo `total_compra`, `total_venda`, `lucro_total` e `capital_perdido`. A função `update_product_interactive` foi corrigida para aceitar valores float na atualização de preços, e a geração de relatórios foi adaptada para formatar corretamente os valores monetários usando `fprintf` com especificador `%.2f`.