Esse é o relatório da questão primeira da segunda avaliação da cadeira de Arquitetura e Organização de Computadores I - 01A - 2020.1, sob o professor Wagner. Sou Pedro Henrique Magalhães Botelho, de matrícula 471047, e venho por meio desse texto informar ao excelentíssimo meu progresso e pensamentos conforme o passar do tempo a respeito da questão primeira.

O enunciado da questão segue abaixo:

Questão 01. Faça um programa para calcular o número de lâmpadas 60 watts necessárias para um determinado cômodo. O programa deverá ler um conjunto de informações, tais como: tipo, largura e comprimento do cômodo.

A tabela abaixo mostra, para cada tipo de cômodo, a quantidade de watts por metro quadrado.

De acordo com a potência exigida para cada tipo de cômodo, a área calculada deverá ser multiplicada pela potência exigida e dividida pela potência das lâmpadas.

O programa termina quando o tipo de cômodo for igual -1.

Quando uma medida negativa for informada, o programa deve imprimir a mensagem "Valor Invalido" e encerrar.

Tipo Cômodo	Potência (watt/m^2)
0	12
1	15
2	18
3	20
4	22

Exemplo de Entrada

0
3.0
10.0
-1

Exemplo de Saída

6.000000

A questão tem um objetivo central: **calcular a quantidade de lâmpadas que devem ser instaladas em um cômodo,** dado seu tipo (e logo a potência por m² exigida), seu comprimento e sua largura.

Considere os seguintes dados:

- **A** a área
- **$C_{comprimento}$** o comprimento
- **L_{argura}** a largura
- **$V_{exigida}$** a potência por m² exigida
- **V** a potência de cada lâmpada

Podemos construir o seguinte cálculo:

$Q_{lampadas} = (A \times V_{exigida})/(V)$, sendo $A = C_{comprimento} \times L_{argura}$.

Veja que nem sempre a área vai ser exata, logo isso irá impactar no cálculo como um todo. Iremos usar a Floating-point Unit (FPU) para realizar os cálculos, já que as medidas da área [e a própria] podem não ser números inteiros.

Houveram algumas preferencias de implementação, como:

- não usar as instruções **ENTER n, m** e **LEAVE**
- usar **macros** predefinidos para facilitar na organização
- comentários em inglês
- para uma organização melhor, o **procedimento principal main** está mais limpao, apenas chamando os procedimentos, que estão devidamente documentados.
- macros e procedimentos em **CAIXA_ALTA**, espaços em memória em **camelCase**, e labels afins **podendo_SER_MISTAS**.
- **indentação** padronizada
- uso do **carriage return** e **line feed** ao final de uma string de output.
- verificação se o valor inserido é maior que 4, no procedimento **ROOM_VERIFY**
- um "miniprocedimento" **EXIT_PROGRAMM_COMMAND** para desfazer a **stack frame** e retornar ao final do procedimento principal
- procedimentos de scan e print com bit de controle [em EBX], para controlar o tipo de dado

Abaixo estão os macros usados:

1. `%DEFINE LINE_FEED 0x0A`
2. `%DEFINE END_OF_STRING 0x0`
3. `%DEFINE CARRIAGE_RETURN 0x0D`
4. `%DEFINE EXIT_PROGRAMM -1`
5. `%DEFINE REMOVE_FOUR_BYTES 4`

```
%DEFINE REMOVE_EIGHT_BYTES 8
%DEFINE REMOVE_TWELVE_BYTES 12
%DEFINE PRINT_STRING 0
%DEFINE PRINT_DOUBLE 1
%DEFINE SCAN_INT 0
%DEFINE SCAN_DOUBLE 1
%DEFINE ERROR_CHECK 0
%DEFINE ROOM_TYPE_0 0
%DEFINE ROOM_TYPE_1 1
%DEFINE ROOM_TYPE_2 2
%DEFINE ROOM_TYPE_3 3
%DEFINE ROOM_TYPE_4 4
%DEFINE POTENCY_TYPE_0 12
%DEFINE POTENCY_TYPE_1 15
%DEFINE POTENCY_TYPE_2 18
%DEFINE POTENCY_TYPE_3 20
%DEFINE POTENCY_TYPE_4 22
```

Foram usados, logicamente, os procedimentos de **printf** e **scanf**, então eles são carregados no "preambulo":

```
EXTERN printf
EXTERN scanf
```

Na memória foram carregados 4 strings (três usadas como "modelo" para scanf/printf com números, e uma mensagem) e apenas um valor numérico de 4 bytes para ser usado nos cálculos mais a frente.

```
SECTION .data
defaultPotency DD 60
intInput DB "%d", END_OF_STRING
doubleInput DB "%f", END_OF_STRING
doubleOutput DB "%lf", CARRIAGE_RETURN, LINE_FEED, END_OF_STRING
exceededLimitErrorMessage DB "Valor Invalido", CARRIAGE_RETURN, LINE_FEED, END_OF_STRING
```

Nos não inicializados estão os espaços reservados para o usuário preencher (3 espaços) e para serem preenchidos pelo programa (2 espaços).

1. SECTION .bss
2. roomType RESD 1
3. roomWidth RESD 1
4. roomLength RESD 1
5. requiredPotency RESD 1
6. lampQuant RESQ 1

O procedimento inicial começa criando uma **stack frame**(moldura de pilha) apenas para o programa, e ao final a destruindo, sem impactar nada exterior a ele. Além do fato dele se repetir, o que ele faz é chamar outros procedimentos para fazerem o serviço.

Vamos dar uma olhada rápida nos procedimentos:

- **main:** como disse antes, apenas chama outros procedimentos e se repete ao final, ainda assim é a espinha dorsal da aplicação.
- **LAMP_CALC:** a estrela da peça. É aqui onde o cálculo da quantidade lâmpadas é feito.
- **ROOM_VERIFY:** apenas verifica se o tipo de cômodo digitado é maior que 0 ou menor que 4, e atribui valores de potência a memória. Caso seja -1 o programa é finalizado, e se for maior que 4, o programa é finalizado e ainda sinaliza um erro. Não fiz o mesmo pra valores menores que -1 para não desviar muito da linha da questão.
- **INPUT_INTERFACE:** Recebe um dado digitado pelo usuário e o salva na memória, podendo ser int ou double (em EBX deverá ser informado isso, por 0 ou 1, respectivamente).
- **OUTPUT_INTERFACE:** Imprime um dado da memória na tela, podendo ser string ou double (em EBX deverá ser informado isso, por 0 ou 1, respectivamente).
- **SIZE_ERROR_CHECK:** Verifica se o valor requisitado é maior que 0. Se não é impresso uma mensagem de erro e o programa é encerrado.
- **EXIT_PROGRAMM_COMMAND:** não é um procedimento, mas sim um incluso dentro de **ROOM_VERIFY**, que vai desfazer alguma stack frame e voltar ao final do procedimento principal, um "mini-procedimento", mas que foi super útil.

Indo direto ao ponto, vamos verificar o funcionamento do procedimento **LAMP_CALC**:

```
PUSH  EBP
```

```
MOV   EBP, ESP
FINIT
FLD    DWORD[roomLength]
FLD    DWORD[roomWidth]
FMUL ST0, ST1
FIMUL DWORD[requiredPotency]
FIDIV  DWORD[defaultPotency]
FSTP  QWORD[lampQuant]
  MOV   ESP, EBP
  POP   EBP
  RET
```

Esse procedimento realiza o cálculo principal da questão. Ele primeiramente cria sua própria **stack frame** e a destrói no final, retornando para quem a chamou.

- **FINIT** irá iniciar a FPU (desnecessário em processadores atuais)
- **FLD DWORD[roomLength]** e **FLD DWORD[roomWidth]** carregam seus respectivos operandos na pilha da FPU, sendo atualmente
  - **ST0 =** roomWidth
  - **ST1 =** roomLength
- **FMUL ST0, ST1** e **FIMUL DWORD[requiredPotency]** iram realizar multiplicações, respectivamente entre seus operandos **ST0** e **ST1**, e de **ST0** com o operando informado.
- **FIDIV DWORD[defaultPotency]** o resultado da ultima multiplicação, em **ST0**, é dividido pelo operando informado (sendo ele um inteiro).
- **FSTP QWORD[lampQuant]** salva o resultado final na memória e o retira do topo da pilha.

Note que o valor final é salvo em 64-bits, ou 8 bytes. Sendo assim, na hora de imprimi-lo devemos passar 32-bits de cada vez:

1. `PUSH DWORD[EAX+4]`
2. `PUSH DWORD[EAX]`
3. `PUSH doubleOutput`
4. `CALL printf`
5. `ADD ESP, REMOVE_TWELVE_BYTES`

Ao final, ao invés de remover 8 bytes (dois endereços de 32 bits) da pilha, nós removemos 12 (três endereços de 32 bits).

A questão foi bem interessante de implementar. Sem muitas dificuldades além das normais, como errar nomes de variáveis, ou organização mesmo. Tive uma grande ajuda. Não tanto na hora da questão, fiz no máximo duas buscas, mas enquanto eu estudo é uma ajuda e tanto. Se trata de uma manual de instruções (literalmente) do x86. Pesquiso sempre lá quando tenho dúvidas, ou nos slides do próprio professor mesmo.