Lista 3 – Felipe Melo – Thalles Nonato

DRE Felipe: 119093752

DRE Thalles: 119058809

**Questão 1) a) O que é armazenado inicialmente em %ecx e em %eax?**

Em %ecx é armazenado o parâmetro n e em %eax é armazenado o parâmetro x.

**Questão 1) b) Qual valor é carregado em %edx na linha 4? Explique a razão deste valor:**

É carregado o valor em (%ecx + 15), ou seja, (n + 15). A razão para isso é para entrar adequadamente em cada case do switch. Para isso, nosso jump computa (4 \* |n + 15|), mas devemos dividir esse resultado por sizeof(int)= 4, resultando em |n + 15|, que nos levará para o respectivo statement do switch.

**Questão 1) c) O que faz a linha 5? Explique a razão desta operação:**

Compara subtraindo 30 do valor de (%edx = n + 15). Isso ocorre pois 30 é o intervalo de instruções .long do código Assembly. No código C, como o valor do nosso menor case é -15 e o maior é 15, o tamanho do intervalo [-15, 15] é 30. No entanto, nem toda instrução nesse intervalo é usada e, por isso, o código Assembly desvia esses casos para o default (.L8).

**Questão 1) d) Quais condições provocam o desvio para .L8? Explique a instrução ja:**

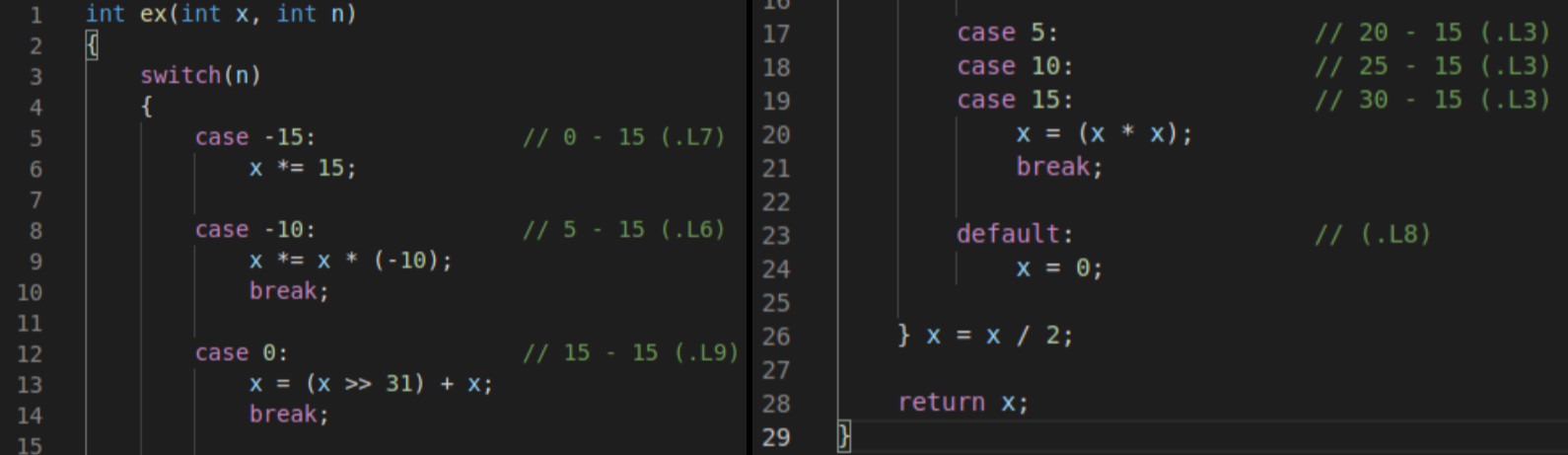
Se (|n + 15| > 30), ocorre o desvio para .L8. A instrução ja realiza um desvio caso a magnitude do termo (n + 15) for maior que 30.

**Questão 1) e) Explique a linha 7 e a necessidade do notrack:**

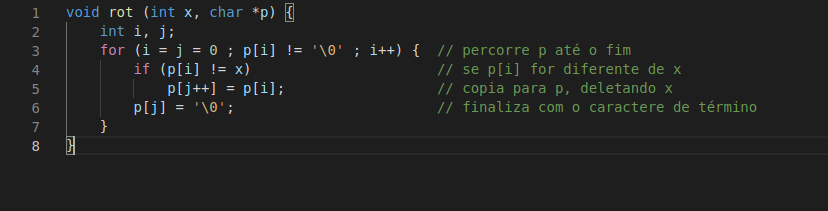
notrack é um prefixo usado para diminuir o número de instruções endbr para realizar saltos indiretos que derivam o endereço de destino de ramificação de informações que não podem ser modificadas.

**Questão 1) f) Escreva o código C completo da rotina ex, inclusive preenchendo o comando switch corretamente. O único**

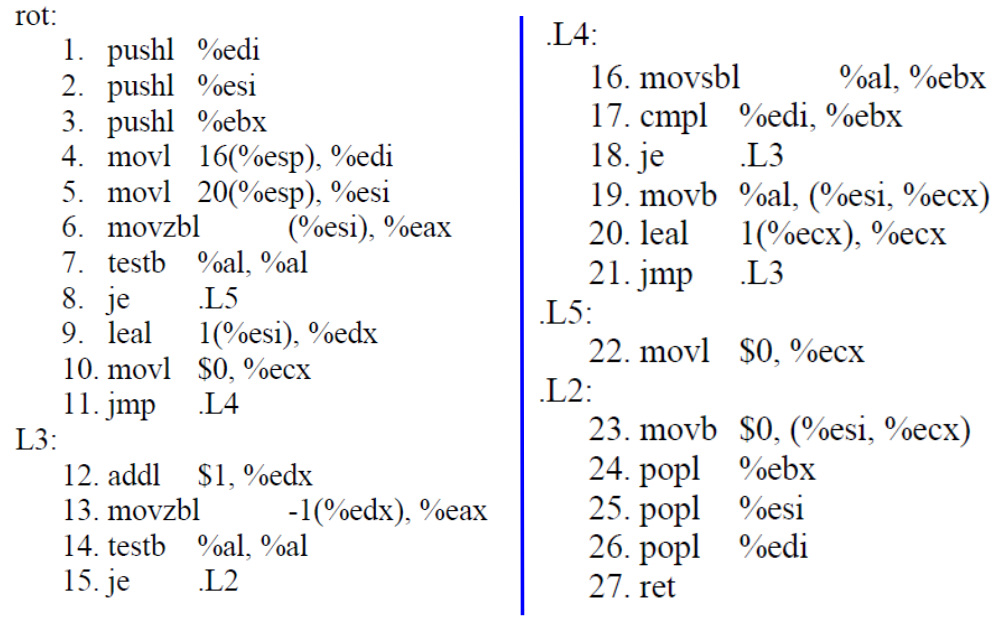
**return na rotina está mostrado no código C. Atenção e não insira return dentro do switch:**

****

**Questão 2) a) Fazendo engenharia reversa no código de montagem, preencha as lacunas do código C. Todas as linhas devem ser preenchidas e nenhuma acrescentada. Reticências podem conter expressões e não apenas um valor, uma variável ou um operador. Atenção aos tipos dos parâmetros de rot:**



**Questão 2) b) Faça a engenharia reversa, comentando todas as linhas do código de montagem. É preciso associar ao C, e não indicar simplesmente o que a instrução faz, que sabemos. Justifique a presença de instrução no código de montagem se não estiver diretamente relacionada com o C:**



L1: salva %edi na pilha

L2: salva %esi na pilha

L3: salva %ebx na pilha

L4: pega o parâmetro x

L5: pega o parâmetro p

L6: copia o parâmetro p para %eax. Com essa instrução, sabemos que p é ponteiro para char, pois

sizeof(char) = 1 e a instrução move 1 byte para um long (movzbl).

L7: realiza um AND bit a bit de %eax com %eax

L8: se o último byte de %eax for zero, ou seja, se chegou ao fim da lista, desvia para .L5

L9: realiza (%esi + 1), ou seja, (p + 1), que é o próximo caractere

L10: realiza (j = 0)

L11: faz desvio incondicional para .L4

L12: incrementa (i = i + 1)

L13: pega p[i] após o incremento

L14: realiza um AND bit a bit de %eax com %eax

L15: se p[i] = ‘\0’, desvia para .L2

L16: copia p[i] para %ebx

L17: compara x com p[i]

L18: se (x = p[i]), desvia para .L3

L19: copia de %al o caractere para p[j]

L20: incrementa (j = j + 1)

L21: faz desvio incondicional para .L3

L22: zera a variável j

L23: realiza p[j] = 0

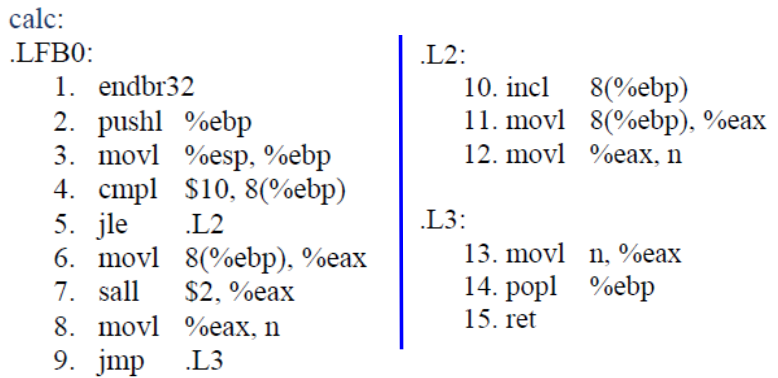
L24: restaura %ebx

L25: restaura %esi

L26: restaura %edi

L27: retorna

**Questão 3) a1) Comente todas as linhas do código, associando ao código C:**



L2 e L3: cria registro de ativação

L4: compara x com 10, computando a expressão (x – 10)

L5: se (x ≤ 10), desvia para .L2

L6: copia o parâmetro x para o registrador %eax

L7: realiza x = 4 \* x

L8: faz (n = 4 \* x)

L9: faz desvio incondicional para .L3

L10: realiza ++x

L11: copia ++x para %eax

L12: copia x para n

L13: copia o valor n para %eax, preparando o retorno da função

L14: restaura a base %ebp

L15: retorna n

**Questão 3) a2) Indique o custo (? m) + (? n) desta compilação. Comente sobre as ineficiências que observar:**

7m + 7n. São 14 instruções, nas quais a metade delas acessa a memória, o que diminui a velocidade de execução, tornando ineficiente.

**Questão 3) b1) Comente as linhas do código, associando ao código C:**

calc:

.LFB0:

1. endbr32
2. movl 4(%esp), %edx
3. leal 1(%edx), %eax
4. cmpl $10, %edx
5. jle .L3
6. leal 0(, %edx, 4), %eax

.L3

1. movl %eax, n
2. ret

L2: pega o parâmetro x e salva em %edx

L3: move (x = x + 1) para %eax

L4: compara x com 10, computando a expressão (x - 10)

L5: se (x ≤ 10), desvia para .L3

L6: copia (%edx \* 4), ou seja, (x \* 4), para %eax

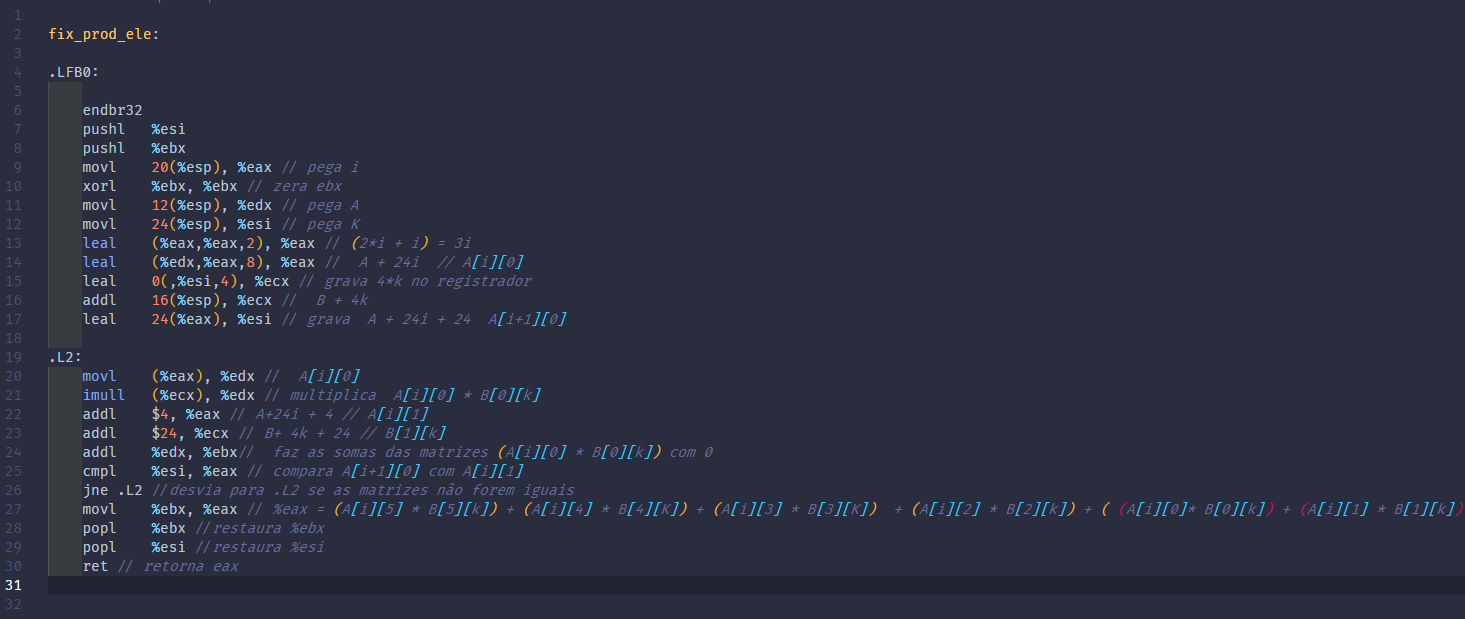
L7: realiza n = x

L8: retorna

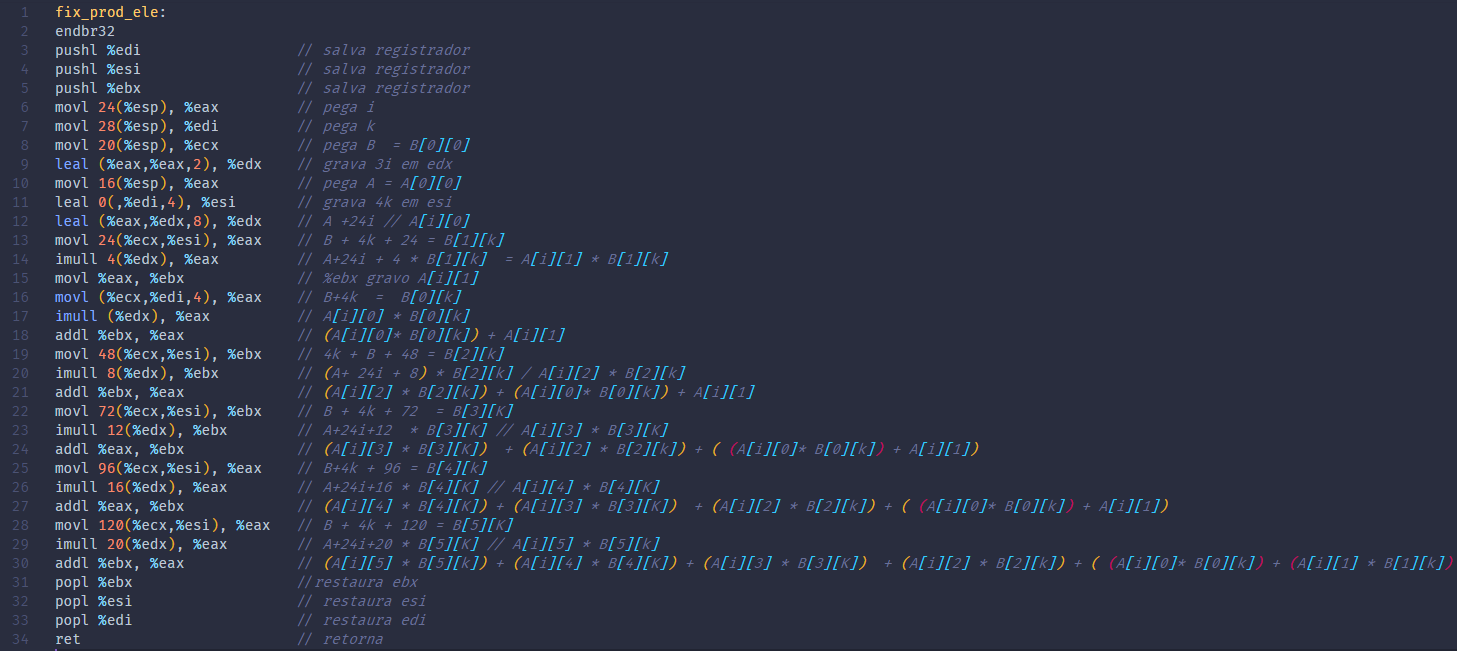
**Questão 3) b2) Indique o custo (? m) + (? n) desta compilação. Compare com o anterior sem otimização. Cite as principais alterações observadas que favorecem o desempenho:**

3m + 4n. Além de acessar menos vezes a memória, possibilitando uma execução mais rápida, possui menos instruções.

**Questão 4) a) Comente cada linha, justifique sua existência e associe ao código C. Identifique o uso de ponteiros para acesso facilitado aos elementos das matrizes:**



**Questão 4) b) Feita a otimização -O3, comente cada linha, justifique sua existência e associe ao código C:**



**Questão 4) c) Tente justificar a otimização feita pelo GCC, pensando na eficiência da execução. Analise e conclua as razões deste novo código ser mais eficiente do que o anterior. Compare os custos dos dois códigos (em função do custo definido na questão 1):**

A otimização -O3 trata o código com mais linearidade, ou seja, ao invés de buscar por .L2 diversas vezes no código, como na otimização -O2, ela é muito mais direta, possibilitando uma execução clara e objetiva. Além disso, o código com maior otimização, as operações fazem mais uso de registradores e, assim, são mais rápidas que acessar a memória diversas vezes como na otimização -O2.