1)

.data -> seção de dados

S2: .byte -> cria um array de bytes. Corresponde à inicialização do vetor char S2[] no código C

.text -> seção de instruções

.globl main -> indica que a main é global

main : -> inicio das operações dentro da main. Corresponde à declaração da função int main() no código C.

movq $S2, %r12 -> copia o endereço de memória do array S2 para o registrador %r12. Corresponde à inicialização do ponteiro char \*pc no código C. (char \*pc = S2).

L1: -> inicia as operações da instrução L1

cmpb $0, (%r12) -> atualiza os flags com 0 – (conteúdo da memória de %r12). Nesse foi preciso acrescentar o sufixo ‘b’ na instrução cmp porque estamos comparando um inteiro de 32 bits com um byte.

je L2 -> pula para a instrução L2 se o flag ZF estiver setado (ou seja, se 0 – (conteúdo da memoria de %r12 == 0).

Essas duas últimas operações correspondem ao while (\*pc) do código C.

movsbl (%r12), %eax -> copia o conteúdo da memoria apontada por %r12 para o registrador %eax. É preciso acrescentar o sufixo ‘sbl’ à operação mov porque estamos movendo um byte para um registrador de inteiros de 32 bits.

addq $1, %r12 -> incrementa o endereço de memoria apontado por %r12, ou seja, faz o registrador apontar para o próximo byte do array.

jmp L1 -> volta para o início da instrução L1. Essa parte também corresponde ao while(\*pc) do código C: enquanto o flag ZF não estiver setado, refaz toda a instrução L1 em loop.

L2: -> inicio da instrução L2. O código só chega aqui quando o registrador %r12 apontar para o último elemento do array (que é 0, então 0-0 seta o flag ZF e a operação je L2 acontece).

2)

Modificação no código C: como o array não tem nenhum valor maior que 122 sem ser os que correspondem às chaves, acrescentei logo abaixo do while um if (\*pc > 122) pc++; para passar direto desse elemento. Código C com a modificação está no arquivo questao2.c.

Modificação no código assembly: dentro da instrução L1, acrescentei as linhas

Cmpb $122, (%r12)

jg L3

compara o valor 122 com o conteúdo da memoria apontada pelo registrador %r12 e pula para a instrução L3 se o conteúdo de %r12 for maior que 122. (implementei a mesma ideia do código C). A instrução L3 é um label que eu acrescentei logo após a parte que imprime, para pular esta. O código está no arquivo questao2.s.

3)

Programa em C no arquivo questao3.c e tradução para assembly no arquivo questao3.s (também tem o código C comentado neste último).

Na main coloquei o valor 1 no registrador %ebx. Depois, na instrução L1, compara o valor 11 e o valor de %ebx: se forem iguais, pula para a instrução L2; se não, copia o valor de %ebx para o registrador %eax. Depois usei a operação imull para colocar em %eax o valor de %eax \* %eax (que é a mesma coisa que %ebx ao quadrado) e imprime esse valor. Em seguida, incrementa 1 em %ebx e volta para o inicio da instrução L1. Assim, o programa fica em loop enquanto o valor de %ebx for menor do que 11.

4)

Inicializei o vetor nums como um array de inteiros de 32 bits na parte de dados globais.

Na main, coloquei o valor 0 nos registradores %ebx e %ecx (pois usarei %ebx como contador do loop e %ecx para guardar a soma dos elementos do array). Coloquei o registrador %r12 apontando para o endereço de nums.

Depois entra na instrução L1, que começa comparando %ebx com 4: se forem iguais, pula para a instrução L2; se não, coloca no registrador %ecx o valor de %ecx + o conteúdo da memoria apontada por %r12; em seguida, incrementa %ebx, soma 4 no valor de %r12 (ou seja, anda 4 bytes no endereço da memória apontada por %r12, assim apontando para o próximo elemento do array já que é um array de inteiros de 32 bits) e volta para o inicio da instrução L1. Assim, o programa fica em loop até %ebx ser igual a 4.

Quando sair do loop, vai para a instrução L2, onde copia o valor do registrador %ecx (nesse caso a soma dos elementos do array nums) para o registrador %eax e imprime esse valor.