1)

No codigo assembly, .data indica a inicialização de dados globais e logo em seguida nums: está definindo o vetor int nums[] que foi definido em C como uma variável global.

.text inicia a parte das instruções; .globl main indica que a main é uma instrução global. Isso é equivalente a definição da função int main() no código C.

main: inicia as operações que vão ser feitas na main.

movl $0, %ebx

copia o valor 0 para o registrador ebx. Isso é equivalente à declaração da variável int i e atribuição de 0 a i no código C.

movq $nums, %r12

copia o endereço de memória do vetor nums para o registrador %r12. Isso é equivalente à declaração do ponteiro int \*p e atribuição de nums a p no código C.

L1: inicia as operações da instrução L1

cmpl $4, %ebx

atualiza flags com 4 – (valor no registrador %ebx)

je L2

se o flag ZF estiver setado (ou seja, se 4 – (valor no registrador %ebx) == 0), pula para a instrução L2

essas duas últimas linhas são equivalentes à condição i != 4 no for do código C.

movl (%r12), %eax

copia o conteúdo da memória apontada pelo registrador %r12 (nesse caso é um inteiro de 32 bits) para o registrador %eax

addl $1, %ebx

guarda em %ebx o valor atual de %ebx + 1 (equivalente ao i++ do for no código C)

addq $4, %r12

anda 4 bytes na memória que %r12 aponta (ou seja, faz %r12 apontar para o próximo inteiro do array)

jmp L1

volta para o início da instrução L1

assim, o código fica voltando em loop para o início de L1 até que o valor no registrador %ebx seja 4

quando o valor no registrador %ebx é 4, pula para a instrução L2.

2)

Copiei o valor 0 para o registrador %ecx (movl $0, %ecx), depois dentro do loop somei o conteúdo da memória apontada pelo registrador %r12 no registrador %ecx (addl (%r12), %ecx). Movi a parte que imprimia os elementos para dentro da instrução L2 (que é quando termina o loop) e ao invés de copiar o conteúdo apontado por %r12 para o registrador %eax para imprimi-lo, estou copiando o valor de %ecx (que, ao final do loop, contém a soma de todos os elementos) para %eax (movl %ecx, %eax).

o código com essas modificações para imprimir a soma está no arquivo prog1Soma.s

3)

Copiei o conteúdo apontado pelo registrador %r12 para o registrador %ecx (movl (%r12), %ecx) e verifiquei se o o bit menos significativo do valor em %ecx era 1 – ou seja, se esse valor é ímpar (andl $1, %ecx) e pula para a instrução L3 se for 1 (jnz L3).

Coloquei o label L3: depois da parte de impressão do código para poder pular para a parte que incrementa o contador e o ponteiro – ainda dentro do for – caso o elemento do array seja ímpar.

O código com essas modificações para imprimir apenas os elementos pares está no arquivo prog1Par.s

4)

Troquei nums: .int por nums: .byte

Troquei a operação movl que copiava o conteúdo apontado por %r12 para o registrador %eax por movsbl, porque agora o valor apontado por %r12 tem um byte então é preciso estendê-lo para 32 bits para imprimir.

Troquei a parte que incrementa o registrador %r12, porque agora ele aponta para um array de bytes, então para acessar cada elemento é preciso andar apenas 1 byte e não 4. (addq $1, %r12)

O código com essas alterações para um array de bytes está no arquivo prog1Byte.s