Imperial College London
Department of Computing

Computer Systems (113)
Exercises – *Pentium Programming*

1. Translate the following high-level language program into Pentium assembly language.

   Use register *edi* for p and register *esi* for q. Assume **int**'s are 32-bit.

```
int  p,  q ;
int  vec [10];

for (p = 9;  p >= 1;  p--) {
        for (q = 1;  q <= p;  q++)  {
                if  (odd p) {              // assume odd p returns true if p is an odd number
                        vec [p] = vec [q];
                }
        }
}
```

2. Write a sequence of *commented* Pentium instructions to reverse the bytes in a memory block:

   Memory Block Before Reversal

Byte 1	Byte 2	...	Byte n-1	Byte n

   Memory Block After Reversal

Byte n	Byte n-1	...	Byte 2	Byte 1

   Assume that
   register `esi` holds the address of the first byte in the memory block, and
   register `ecx` holds the number of bytes in the memory block.

Imperial College London
Department of Computing

Computer Systems (113)
Solutions – *Pentium Programming*

1.
```
vec    resd 10                  ; declare int vec [10]

                                ; for (p = 9; p >= 1; p--)
       mov  edi, 9              ; p = 9
nextp: cmp  edi, 1              ; compare p and 1
       jl   endp                ; jump to endp if p < 1

                                ; for  (q = 1; q <= p; q++)
       mov  esi, 1              ; q = 1
nextq: cmp  esi, edi            ; compare q and p
       jg   endq                ; jump to endq if q > p

                                ; if (odd p)
       test edi, 1              ; p and 1;   result will be 1 if odd, 0 if even
       jz   endif               ; jump to endif if result was even

                                ; vec[p] = vec[q]
       mov  eax, [vec+4*esi]    ;      eax = vec[q]
       mov  [vec+4*edi], eax    ;      vec[p] =vec[q]
endif:

       inc  esi                 ; q ++
       jmp  nextq

endq:  dec  edi                 ; p --
       jmp  nextp

endp:
```

2. Here's one solution that maintains a start pointer and an end pointer, advancing the start pointer from the start address and the end pointer backwards from the last byte in the memory block. The bytes at start pointer and end pointer are swapped until the pointers cross. Other solutions are possible too.

```
       ; ecx  = number of bytes to reverse (given)
       ; esi  = start address (given)
       ; edi  = end pointer

       mov  edi, esi       ; end pointer = start pointer + bytes - 1
       add  edi, ecx
       dec  edi

while:
       cmp  esi, edi       ; while  (start pointer < end pointer)
       jge  endwhile

       mov  ah, [esi]            ; swap bytes at start pointer and end pointer
       mov  al, [edi]
       mov  [esi], al
       mov  [edi], ah

       inc  esi                  ; advance start pointer to next byte
       dec  edi                  ; advance end pointer to previous byte
       jmp  while

endwhile:
```