Problem 1: Programming in C [20 Points]

Write a C program that takes as input a positive integer number and converts it to base 4. Some examples of input and output of this program are as follow:

Example 1:
Input: 10
Output: 22
Example 2:
Input: 16
Output: 100
Example 3:
Input: 5485
Output: 1111231
Example 4:
Input: 3
Output: 3

Note 1: a pseudo code is not acceptable. Your code should have a very precise syntax in C.
Note 2: write neatly. If your code is not easily readable, we will not give you the points.

```
#include <stdio.h>
#include <math.h>

int showNum(int digits[], int digitCount);

int main()
{
   int number;
   int digitCount = 0;
   int digits[100];
   printf("Input: ");
   scanf("%d", &number);
   while (number != 0)
   {
      digits[digitCount++] = number % 4;
      number = number / 4;
   }
   printf("Output: %d\n", showNum(digits, digitCount));
   return 0;
}

int showNum(int digits[], int digitCount)
{
   int i, temp = 0;
   for (i = digitCount - 1; i >= 0; i--)
   {
      temp += digits[i] * pow(10, i);
   }
   return temp;
}
```

Problem 2: Assembly Programming in C [20 Points]

The following program, using c and inline assembler, is supposed to:

- define an array, name *table*, of 10 bytes containing digits 0 to 9,
- then, in a loop receives from the console two numbers, name *n* and *m*, between 0 and 10,
- if *n* or *m* is not between 0 and 10, the program ends,
- if *n* or *m* is 10, the program prints the whole *table*,
- if *n* and *m* are between 0 and 9, it calls a function, name *addElements*,
- function *addElements* **adds *table*[m] and *table*[*n*] and stores it in table[n]**.

Complete the body of *addElements.* It must be all in assembly.
Important note: You are not allowed to use register DI.

```
#include "stdafx.h"
#include "stdlib.h";
#include "stdio.h";

void addElements(char *tablePtr, int n, int m);

int _tmain(int argc, _TCHAR* argv[])
{
	char table[10];		// Defines an array of 10 elements.
	int m, n, i, done;

	//Fills the array with 0 to 9.
	for (i = 0; i < 10; i++){
		table[i] = i;
	}

	done = 0;
	while(done == 0){
		printf("Please enter two numbers between 0 and 10: ");
		scanf("%d %d", &n, &m);

		if (n < 0 || n > 10 || m < 0 || m > 10){
			done = 1;
		}
		else if (n == 10 || m == 10){
			for (i = 0; i < 10; i++){
				printf("%d ", table[i]);
			}
			printf("\n");
		}
		else
		{
			addElements(table, n, m);
		}
	}

	system("PAUSE");
	return 0;
}
```

```
void addElements(char *table, int n, int m)
{
  _asm {
    // complete this function in assembly
    // remember you are not allowed to use register DI.

       {
         MOV EBX, table        // copies the start address of table in EBX
         DEC n                 // Decrement n by 1
         DEC m                 // Decrement m by 1
         MOV ECX, n            // copies the value of n, which is n-1 actually, into ECX
         MOV ESI, m            // copies the value of m, which is m-1 actually, into ESI
         MOV AL, [EBX + ECX]   // copies the contents of position table[n-1] into AL
         ADD AL, [EBX + ESI]    // adds table[m-1] into AL
         MOV [EBX+ECX], AL     // copies the contents of AL into the contents addressed
                                // by EBX+ESI (table[n-1])
       }

  }
}
```

Problem 3: Short Assembly Programming [30 points]

a) Write a short sequence of instructions that copy 100 bytes of data from an area of memory addressed by DATA1 into an area of memory addressed by DATA2.

```
CLD
MOV SI, DATA1
MOV DI, DATA2
MOV CX, 100
REP MOVSB
```

b) Complete the following far procedure that copies contents of the double-word-sized memory location CS:DATA3 into EAX, ECX, and ESI.

```
COPY    PROC    FAR
                ;complete this procedure:

        MOV EAX, [CS:DATA3]
        MOV ECX, EAX
        MOV ESI, EAX

        RET
COPY    ENDP
```

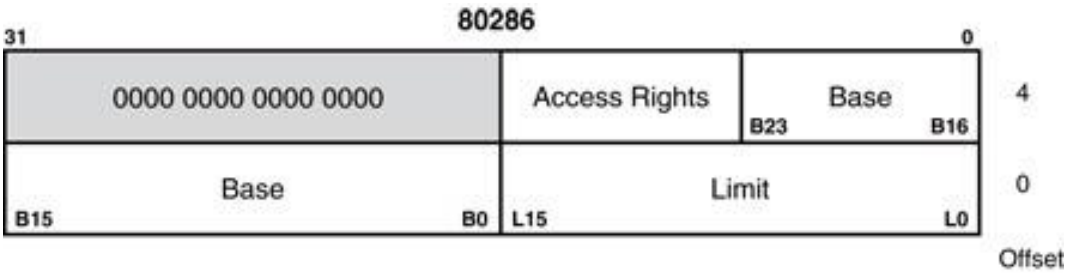
Problem 4: Descriptors: [20 Points]

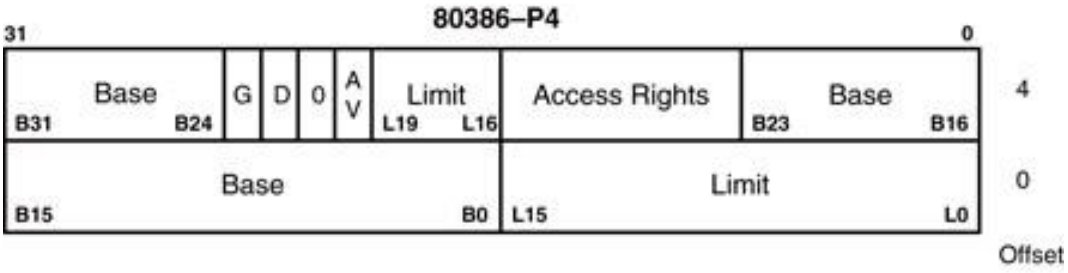
Code a descriptor that describes a memory segment that begins at location 04000000H and ends at location 05FFFFFH. This memory segment is a data segment that grows downward in the memory system and can NOT be written. The descriptor is for a Pentium 4 microprocessor and has the lowest privilege level.

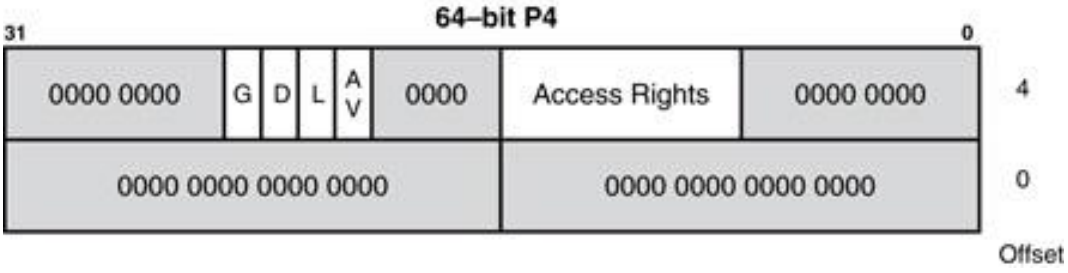
To show your answer, complete the following 8 bytes by 0s and 1s.

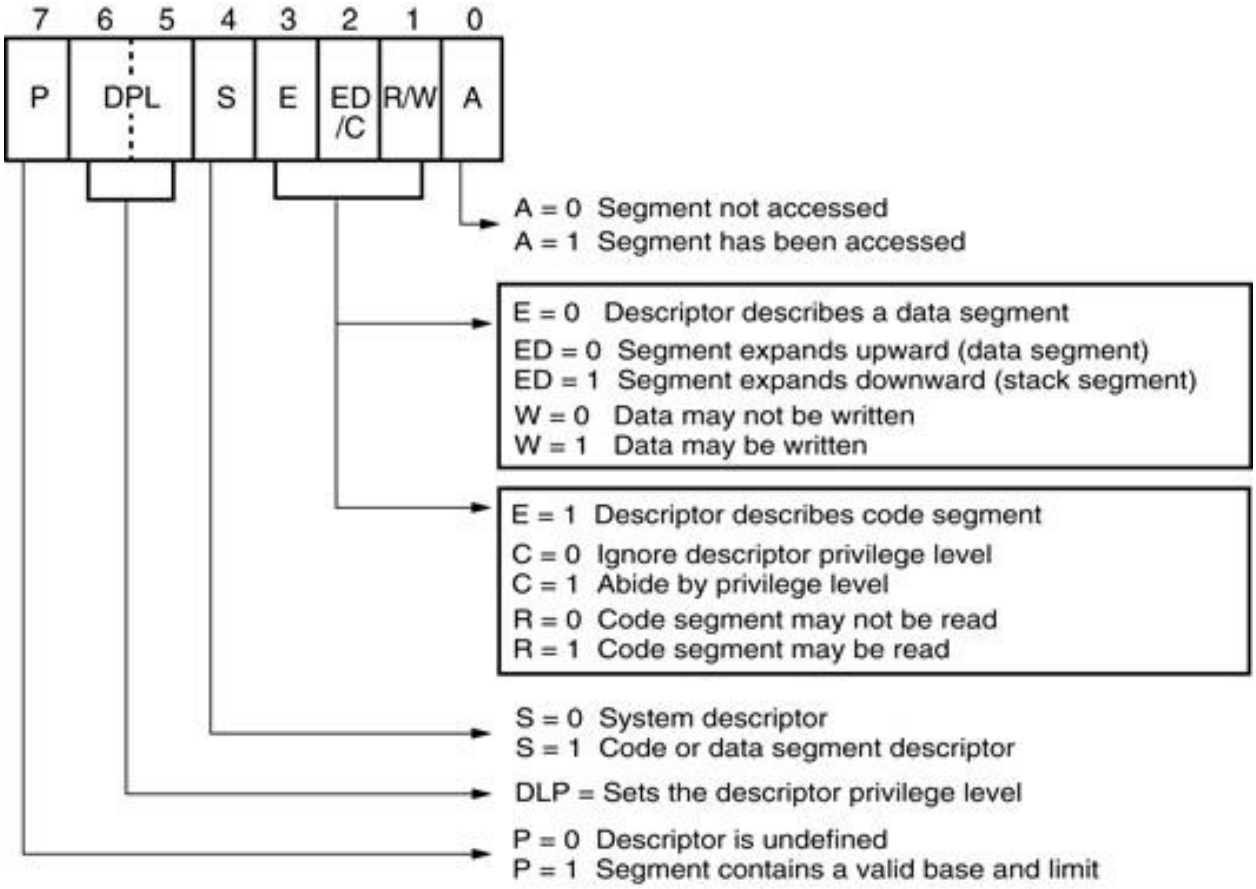
0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Hint: if the size of a memory segment is more than 64K, the granularity bit should be 1; and the size of the limit will be multiplied by 4k bytes (this means the size of the segment will automatically be appended with FFFH). You may want to use the following figures.









Note: Some of the letters used to describe the bits in the access rights bytes vary in Intel documentation.

Problem 5: Short Answer Questions: [10 Points]

Note: some instruction may not be correct and you are required to identify them.

a) Explain the **MOV AX,DL** instruction:

Incorrect: mixed register sizes is not allowed

b) Explain **MOVSB** instruction:

MOV string function with byte data (memory to memory transfer):
ES:[DI] = DS:[SI]; DI = DI +- 1; SI = SI +- 1

c) Explain **MOV CS, AX** instruction:

Copy contents of AX to the code segment.
Not allowed: It may or may not cause the program to crash depending on the contents of AX

d) Explain the **MOV RAX, [RDX]** instruction:

RAX gets content from address contained in RDX and DS

e) How much memory is found in the Windows system area?

2GB