Question 1

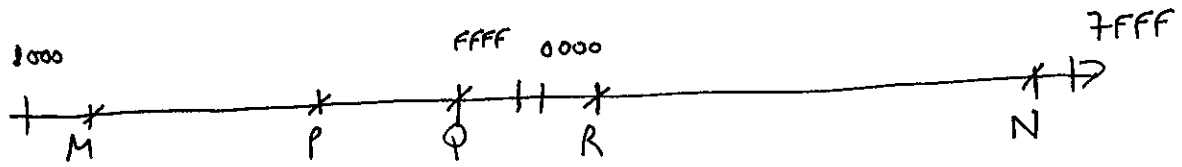
1.1) Le bit de poids le plus fort est le bit de signe.

Les nombres négatifs sont ceux pour lesquels ce bit est à 1.

Ce sont donc :

M, P, Q.

1.2) En notation complément à deux, les nombres sont triés de la sorte :



Donc :

$M < P < Q < R < N$.

1.3) M+P :

```
   8  6  3  2
+  A  B  B  A
```

```
   1000 0110 0011 0010
+  1010 1011 1011 1010
(1) 0011 0001 1110 1100
    3    1    E    C
```

Il y a débordement

1.4) En notation complément à deux, l'opposé s'obtient en complémentant les bits et en ajoutant 1. Donc :

Q : F0D3

```
complément ( 1111 0000 0000 1101
             0000 1111 1111 0010
           + 0000 0000 0000 0001
             0000 1111 1111 0011
             0    F    F    3
```

$-Q = 0FF3$

M-Q : 8632
+ ØFF3 car M-Q = M+(-Q)

```
  1000 0110 0011 0010
+ 0000 1111 1111 0011
-------------------
  1001 0110 0010 0101
```

9 6 2 5

Question 2

2.1) La fonction map prend deux paramètres.
On le voit lors de son appel : on fait deux fois PUSHL, aux adresses ØØE et Ø16. On utilise ces paramètres aux adresses Ø34 et Ø3A.

2.2) Le mrmovl lit la donnée stockée à l'adresse fournie : c'est une lecture de variable.
Le irmovl charge le registre avec la valeur de l'étiquette : on lit l'adresse de la variable placée à l'étiquette en question. Cela servira à faire des accès indirects.
Les paramètres étant empilés dans l'ordre inverse de leur usage, le prototype de la fonction map est :

```
int map (int * t, int n);
```

↑ Car on utilise la valeur de %eax à l'adresse Ø23.

2.3) Le rôle de ces instructions est de mettre en place le contexte de pile de la fonction f. À travers %ebp, on pourra accéder à la fois aux paramètres de la fonction ainsi qu'à ses variables locales.

2.4) La fonction f semble ne prendre qu'un seul paramètre, car on ne fait qu'une seule lecture à partir de %ebp : à l'adresse Ø76.

2.5) L'instruction à l'adresse Ø7C est une lecture par adressage indirect : on place dans %eax la donnée contenue en mémoire à l'adresse contenue dans le registre %edx. Le paramètre de f est donc une adresse

2.6) La fonction f ne prend qu'un seul paramètre (cf Q. 2.4). Ce paramètre est utilisé comme l'adresse pour effectuer la lecture d'une donnée qui est passée deux fois à la fonction mul. Le résultat de l'appel de la fonction mul est renvoyé tel quel à la fonction appelante.

Le prototype de f est donc :

```
int f (int * p);
```

La fonction f calcule (*p)*(*p), c'est-à-dire le carré de la valeur dont l'adresse est passée en paramètre.

2.7) Ces instructions servent à sauvegarder les registres "callee save" que la fonction map va utiliser.

2.8) La fonction map passe à la fonction f les adresses de zones mémoire consécutives. L'adresse contenue dans le registre %esi est incrémentée de 4 octets à chaque tour de boucle.

2.9) On fait autant de tours de boucle que la valeur du deuxième paramètre de la fonction map. Cette valeur est placée dans le registre %edi, qui est décrémenté à chaque tour de boucle (adresse Ø42) et testé dans la foulée (adresse Ø48).

2.10) La fonction f calcule le carré de la valeur dont l'adresse lui est passée en paramètre.

La fonction map calcule la somme des carrés des valeurs d'un tableau dont l'adresse lui est passée en tant que premier paramètre et la taille en tant que second paramètre.

Dans ce cas précis, le tableau commence à l'adresse t et est de taille 4 (car n = 4).

Le résultat de l'appel de map est donc :

$2^2 + 3^2 + 1^2 + 6^2 = 4 + 9 + 1 + 36 = 50 \quad (= 32_{16})$

2.11) On peut écrire une version simple en n'utilisant que des registres "caller save" et sans mettre en œuvre de contexte de pile :

```
mul:    mrmovl  4(%esp), %ecx
        mrmovl  8(%esp), %edx
        xorl    %eax, %eax
mul1:   isubl   1, %ecx
        jl      mul2
        addl    %edx, %eax
        jmp     mul1
mul2:   ret
```