TD : Assembleur x64

1. Registres des processeurs x64

Registres 64 bits		Registres 32 bits		Registres 16 bits		Registres 8 bits	
RAX		EAX		AX		AH	AL
RBX		EBX		BX		BH	BL
RCX		ECX		CX		CH	CL
RDX		EDX		DX		DH	DL
RSI		ESI		SI			SIL
RDI		EDI		DI			DIL
RBP		EBP		BP			BPL
RSP		ESP		SP			SPL
R8		R8D		R8W			R8B
R9		R9D		R9W			R9B
R10		R10D		R10W			R10B
R11		R11D		R11W			R11B
R12		R12D		R12W			R12B
R13		R13D		R13W			R13B
R14		R14D		R14W			R14B
R15		R15D		R15W			R15B
RIP		EIP		IP			
		EFL		Flags			

Le tableau ci-dessus indique les différents registres des processeurs x64. Dans ces processeurs, la lettre finale B (Byte) indique un registre 8 bits, W (Word=mot) indique un registre 16 bits et D (Double word = mot double) un registre 32 bits.

- Quelle est le lien entre le registre RAX et le registre EAX ?
- Quelle est le lien entre le registre EAX et le registre AX ?
- Quelle est le lien entre le registre AX et les registres AH et AL ?
- Si le registre 64 bits RAX contient $(ABCDEF0123456789)_{16}$, quel est le contenu en hexa des registres suivants :
  - AL = ... ?
  - AH = ... ?
  - AX = ... ?
  - EAX = ... ?
- Sachant qu'une case mémoire peut contenir 8 bits, quelle est en octets, la taille de l'espace mémoire qu'on peut adresser avec un registre 16 bits ?
- Même question pour un registre 20 bits ?
- Même question pour un registre 32 bits ?
- Même question pour un registre 64 bits ?
- Le registre RSP (Stack Pointer) sert à pointer vers la prochaine case libre de la pile. Quelle est le rôle de la pile ?
- Que contient le registre des indicateurs EFL ? Quand est-il modifié ?

2. Premier programme

Le programme C++ ci-dessous écrit dans le fichier **main.cpp** fait appel à une fonction en ASM x64 écrite dans le fichier **prog.asm**.

main.cpp

```cpp
#include <iostream>
using namespace std;

extern "C" int somme(int a, int b);

int main()
{
	int a, b;

	cout << "Entrez un premier  entier : "; cin >> a;
	cout << "Entrez un deuxieme entier : "; cin >> b;

	cout << a << " + " << b << " = " << somme(a,b) << endl;

	return 0;
}
```

prog.asm

```
.CODE

    somme   PROC

        MOV EAX,ECX
        ADD EAX,EDX

        RET

    somme   ENDP

END
```

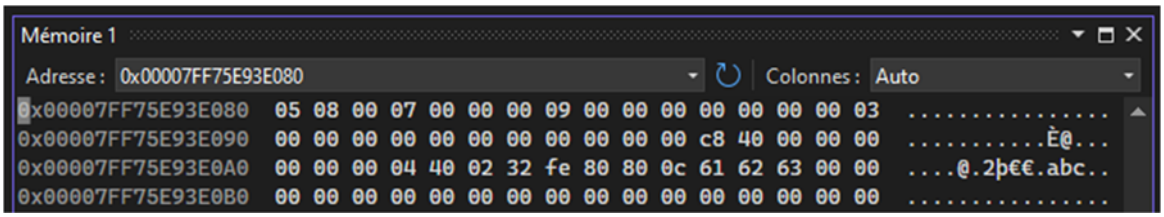
- Expliquez le rôle de la ligne `extern "C" int somme(int a, int b);`
- Que définissent les lignes `.CODE` et `END` ?
- Que définissent les lignes `somme  PROC` et `somme  ENDP` ?
- Quelle est la taille en bits des paramètres a et b et quels registres sont utilisés pour transmettre leurs valeurs à la fonction somme ?
- Quelle est la taille en bits de la valeur retournée par la fonction somme et quel registre est utilisé pour cela ?
- Que fait l'instruction `MOV EAX,ECX` ?
- Que fait l'instruction `ADD EAX,EDX` ?

3. Programme avec section de données

Le programme ci-dessous contient des variables.

```
.DATA
        var1    BYTE    5
        var2    WORD    8
        var3    DWORD   7
        var4    QWORD   9
        var5    TBYTE   3
        var6    REAL4   6.25
        var7    REAL8   2.5
        var8    DB      2
        var9    DB      '2'
        var10   DB      -2, 128, -128, 12
        var11   DB      'abc',0
        var12   DB       ?

.CODE
        fct PROC

            MOV AL,var1
            MOV BL,var8
            ADD AL,BL
            MOV var12,AL

            RET

        fct ENDP

END
```

- Quel est le rôle de la ligne `.DATA` ?
- La section DATA commence à l'adresse mémoire 0x00007FF75E93E080. Le contenu de la mémoire à partir de cette adresse est indiqué ci-dessous.



En vous aidant de cette copie d'écran, remplissez le tableau ci-dessous.

Variable	Adresse de début	Nombre d'octets	Contenu des octets en hexa	Explication
var1				
var2				
var3				
var4				
var5				
var6				
var7				
var8				
var9				
var10				
var11				
var12				

- Que remarquez-vous en général pour toutes les variables ?
- Que remarquez-vous en particulier pour la variable var10 ?
- Quel autre type peut-on utiliser à la place respectivement de BYTE, WORD, DWORD, QWORD, TWORD ?
- Le tableau ci-dessous indique le code machine généré pour les instructions du programme.

Adresse mémoire	Code Machine	Instruction
0x00007FF75E931CD0	8A 05 AA C3 00 00	MOV AL , var1
0x00007FF75E931CD6	8A 1D C9 C3 00 00	MOV BL , var8
0x00007FF75E931CDC	02 C3	ADD AL , BL
0x00007FF75E931CDE	88 05 CB C3 00 00	MOV var12 , AL

Sachant que :

a) L'adresse de l'instruction MOV AL , var1 est `0x00007FF75E931CD0`

b) L'adresse de la variable var1 est `0x00007FF75E93E080`

c) Le code opératoire de l'instruction MOV AL , <variable> est `8A 05`

comment pouvez-vous expliquer que le champ DATA ou Adresse de cette instruction est égal à `AA C3 00 00` ?

- Que fait ce programme ?

4. Si … alors… sinon

Considérons le programme ci-dessous.

main.cpp

```cpp
#include <iostream>
using namespace std;
extern "C" unsigned char majorite(unsigned int age);

int main()
{
	unsigned int age;
	cout << "Quel est votre age ? : "; cin >> age;
	if (majorite(age)) cout << "Vous etes Majeur." << endl;
	else               cout << "Vous etes Mineur." << endl;

	return 0;
}
```

prog.asm

```
1   .CODE
2
3           majorite    PROC
4
5               si_supegal18 :  CMP ECX,18
6                               JAE  alors_majeur
7
8               sinon_mineur :  MOV AL,0
9                               JMP fin_si
10
11              alors_majeur :  MOV AL,1
12
13              fin_si       :  RET
14
15          majorite    ENDP
16
17  END
```

a) Comparer les instructions en lignes 6 et 9.

b) Que fait le programme ?

5. Boucles et tableaux

```
1   .DATA
2       tab_src     DWORD   15 , 80 , 99 , 45 , 8 , 51 , 3 , 19 , 75 , 10
3       tab_dest    DWORD   10 DUP (?)
4
5   .CODE
6       multiple3   PROC
7                           MOV RSI,0
8                           MOV RDI,0
9                           MOV EBX,3
10          boucle :
11                          MOV EDX,0
12                          MOV EAX,tab_src[RSI*4]
13
14                          DIV EBX
15
16                          CMP EDX,0
17                          JNE suivant
18          multiple :
19                          MOV ECX,tab_src[RSI*4]
20                          MOV tab_dest[RDI*4],ECX
21                          INC RDI
22          suivant :
23                          INC RSI
24                          CMP RSI,9
25                          JBE boucle
26
27                          RET
28      multiple3   ENDP
29  END
```

Le programme ci-dessus utilise un tableau **tab_src** de dix nombres, recherche les multiples de 3 et les copie dans un autre tableau **tab_dest**.

- Comment est déclaré le deuxième tableau **tab_dest** ?
- Quels sont les registres qui sont utilisés comme indice pour ces tableaux ?
- Quel est le rôle de la ligne 9 ?
- Quel est le rôle des lignes 11 et 12 ?
- Pourquoi les indices RSI et RDI sont-ils multipliés par 4 dans les lignes 12, 19 et 20 ?
- Expliquez comment fonctionne la division de la ligne 14.
- Quel est le rôle de la ligne 17 ?
- Est-il possible de remplacer les 2 instructions des lignes 19 et 20 par une seule instruction `MOV tab_dest[RDI*4] , tab_src[RSI*4]` ?
- Quel est le rôle de la ligne 25 ?
- Quel est le contenu du tableau **tab_dest** à la fin du programme ?
- Expliquez ce que fait le programme.
- Que faudrait-il changer dans le programme si on veut qu'il fonctionne aussi avec des valeurs négatives ?

6. A vous de jouer

Soit un nombre entier naturel n sur 64 bits. Ecrire un programme en ASM x64 qui, **en utilisant une seule boucle**, calcule la somme S = 1 + 2 +…+ n et le produit P = 1 x 2 x…x n.