Chapitre 1
Introduction

Prolégomènes[1]

Le cours *Organisation des ordinateurs et assembleur*, pour lequel ces notes ont été écrites, a pour objectif de vous présenter l'organisation de la machine qu'est l'ordinateur. Ce n'est pourtant pas un cours de circuits logiques, encore qu'un des chapitres de ces notes leur soit consacré. En effet, le cours vous présente la machine du point de vue du programmeur ; par conséquent, on ne descend pas jusqu'au niveau des circuits électroniques, bien que l'on établisse les bases qui permettent d'y arriver. Pour les étudiants du programme de baccalauréat, un aspect plus profond de l'*Architecture des ordinateurs* sera couvert dans un cours plus avancé, pour lequel vous avez besoin de la préparation que le cours *Organisation des ordinateurs et assembleur* vous donne.

Malgré tout, le cours que vous commencez est essentiellement un **cours de programmation**, qui va vous permettre de comprendre le fonctionnement interne de la machine qu'est l'ordinateur. En effet, les travaux pratiques que vous aurez à faire sont essentiellement centrés sur la production de petits programmes écrits en langage d'assemblage. Et, par le biais de la programmation en assembleur, vous allez être très proche de la machine et, par nécessité, en apprendre le fonctionnement intime. Notez bien, cependant, que l'objectif du cours n'est **pas** de faire de vous un **programmeur en assembleur**. En fait la proportion d'étudiants d'un baccalauréat en informatique qui programmeront en assembleur est probablement inférieure à 1‰, et cela, en étant très optimiste. On vous a peut-être déjà parlé de performance au sujet des programmes que vous écrivez, ne serait-ce que pour vous sensibiliser au fait que le choix d'un algorithme peut avoir un effet considérable sur la vitesse d'exécution du programme correspondant. Lorsque les langages de programmation évolués ont fait leur apparition, et pendant de nombreuses années, on savait que les programmes écrits en assembleur étaient généralement bien plus rapides que ceux écrits en langages évolués. Cependant, notez que ceci n'est plus du tout vrai, car les compilateurs actuels engendrent un code machine d'excellente qualité.

Mais, si par hasard, après avoir établi le *profil d'exécution* d'un de vos gros programmes, vous aviez détecté l'existence de goulots d'étranglement (endroits où le programme passe la majorité de son temps), vous pourriez décider d'améliorer la performance de votre programme. Pour ce faire, vous seriez sans doute amenés à devoir reprogrammer en assembleur les parties de votre système correspondant à ces goulots, afin d'en accélérer l'exécution. Vous pourriez alors faire comme Steve McConnel, l'auteur du livre *Code Complete* et une sommité de la construction de logiciel, l'a fait, lui qui n'avait jamais programmé professionnellement en assembleur, bien qu'il ait eu des notions comparables à celles de ce cours. Tel qu'il le décrit dans la seconde édition de son livre, il a commencé par faire produire par son compilateur la liste de son programme compilé en langage d'assemblage. Il a ensuite étudié cette liste et déterminé quelles étaient les parties visées par la réécriture. Il a alors modifié le code assembleur produit par le compilateur. Cette façon de procéder lui a permis d'arriver à son objectif, sans toutefois être ce qu'on pourrait appeler un expert en langage d'assemblage.

[1] Si le premier mot de ces notes vous embarrasse, ce serait plutôt mauvais signe si cela vous rebutait; mais profitez plutôt de l'occasion pour enrichir votre vocabulaire : les prolégomènes sont une ample préface nécessaire à la compréhension d'un ouvrage, ce qui n'est peut-être pas exagéré ici !

Bien que vous ne puissiez penser être jamais un programmeur professionnel en assembleur, si vous aimez programmer, ce cours vous plaira ; par contre, si vous n'aimez pas programmer, ce cours vous déplaira, car il vous demandera encore plus d'effort pour produire un programme, que le cours de Java ou de C++ que vous avez suivi au préalable ! Les principes de génie logiciel auxquels vous avez été exposés, et que, l'on espère, vous avez assimilés, demeurent valables, quel que soit le langage de programmation que vous utilisiez. En langage d'assemblage, une séquence d'instruction demeure une séquence, une boucle demeure une boucle, une sélection demeure une sélection, et un appel de sous-programmes demeure un appel. Il n'y aura rien de nouveau sur le plan des concepts de la programmation. Cependant, ces concepts vont devoir être appliqués dans un contexte tout différent, où tout est permis, où il n'existe pas de type de données, où les instructions sont mélangées aux données, où l'accès aux divers éléments n'est plus contrôlé strictement.

Dans ces conditions, l'application des principes de génie logiciel est plus vitale que jamais, car la liberté de tout faire sans contrôle, conduit rapidement à la possibilité de faire n'importe quoi, moyen le plus rapide d'arriver à l'échec du programme. La discipline et la méthode imposées par les principes du génie logiciel sont des outils précieux, même s'ils restreignent la liberté totale dont on dispose en langage d'assemblage. Sans elles, vos programmes ressembleront à ce qu'on a appelé, il y a déjà bien longtemps, du *spaghetti logique*. La mise au point de tels programmes est un cauchemar ! Pour vous en sortir, rappelez vous les principes que l'on vous a énoncés dans vos cours de programmation précédents et appliquez-les ! Il sera peut-être étonnant pour vous de constater que ce que vous avez appris dans un cours précédent est utile dans un cours subséquent, mais c'est comme cela qu'on progresse en informatique.

Pour réussir ce cours, il faut vous assurer d'un bon niveau de compréhension de la matière. La meilleure méthode pour bien comprendre, qui est aussi la meilleure préparation aux questions d'examen, est le travail de programmation que vous devrez faire au laboratoire et qui est aussi nécessaire à la production de vos travaux pratiques.

En fait ce cours, comme tout cours, va avoir pour effet de modifier vos **réseaux neuronaux** ; la partie mémoire est présente, mais ce n'est pas elle qui est la plus sollicitée. En effet, pour comprendre, il vous faut emmagasiner les idées et les étudier, ce qui implique un va-et-vient du cortex arrière (mémoire) vers le cortex avant (structuration et décision). C'est uniquement ce va-et-vient qui construit de nouveaux réseaux, modifie des synapses, etc. L'évaluation parfaite des étudiants d'un cours devrait être basée sur une image de vos réseaux neuronaux au début du cours, laquelle serait comparée à une autre image de vos réseaux neuronaux à la fin du cours. Malheureusement chacun d'entre-nous a des réseaux neuronaux très particuliers, dans lesquels on a beaucoup de mal à s'y retrouver, et il est impossible de distinguer les modifications dues uniquement au suivi d'un cours donné… Bien sûr, savoir tout cela ne vous aidera pas vraiment, mais au moins vous serez conscients que vous avez besoin de travail cérébral. Une bonne nouvelle cependant, on peut apprendre durant toute notre vie et notre mémoire n'est jamais pleine, contrairement à celle des ordinateurs ! Une autre bonne nouvelle : il est plus facile d'apprendre si l'on est de bonne humeur ! Comme la vie d'étudiant est parfois ennuyeuse, je vous rappelle que l'être humain possède le rire, comme l'a dit Rabelais dans son « Avis aux lecteurs » de *Gargantua* (1534):

> « Amis lecteurs, qui ce livre lisez,
> Despouillez vous de toute affection;

Et, le lisant, ne vous scandalisez:
Il ne contient mal ne infection.
Vray est qu'icy peu de perfection
Vous apprendrez, si non en cas de rire;
Aultre argument ne peut mon cueur elire,
Voyant le dueil qui vous mine et consomme :
Mieulx est de ris que de larmes escripre,
Pour ce que rire est le propre de l'homme. »

Permettez-vous donc de sourire ou même de rire pendant votre travail ! Votre apprentissage n'en ira que mieux !

Enfin, une dernière remarque pratique, avant de considérer des choses plus techniques, un cours est comme une tartine, autrement dit, une tranche de pain sec à laquelle il vous faut ajouter la confiture de votre travail. Pour un cours universitaire, vous devez investir 6 heures de travail personnel par semaine. Allez-vous le faire dès la première semaine ? Allez-vous lire le texte des travaux à l'avance pour vous faire une idée ? Selon le modèle habituel, vous n'étudiez pas 6 heures par semaines, mais vous attendez la dernière minute pour faire votre TP, ce qui ne vous laisse pas assez de temps, car nous sommes en programmation et les choses vont souvent de travers, et vous vous plaignez alors d'une surcharge de travail qui vous a fait passer deux nuits blanches de suite. N'oubliez pas l'utilité de la répétition, comme le dit Boileau dans *L'art poétique*:

«Hâtez-vous lentement; et, sans perdre courage,
Vingt fois sur le métier remettez votre ouvrage:
Polissez-le sans cesse et le repolissez;
Ajoutez quelquefois, et souvent effacez.»

En revenant à la tartine, si on peut voir que vous n'étalez pas la confiture et qu'il y a des pics de confiture, cela explique les nuits blanches. Assurez-vous bien que vos méthodes de travail sont bonnes, avant de récriminer contre ce cours, qui ne vous a rien fait !

Programmation en langage d'assemblage

La programmation est un phénomène unique qui transforme bien des gens en maniaques, mais en fait décrocher un bien plus grand nombre. Cependant tous les programmeurs n'écrivent pas nécessairement de bons programmes; les raisons en sont diverses et peuvent venir du manque de méthode du programmeur aussi bien que du fait d'avoir à écrire un programme en vitesse, ou du fait que le programmeur déteste … programmer! Une programmation de qualité demande de la méthode et une certaine discipline; l'effort principal doit être consacré à la conception des algorithmes; il doit être suivi d'une programmation soignée qui rendra la vérification et la mise au point plus faciles. C'est malheureusement loin d'être toujours le cas.

La conception d'un programme doit débuter par la spécification claire et sans ambiguïté de ce que le programme doit faire. Soixante ans après le début de l'ère informatique, ceci n'est encore malheureusement pas toujours le cas. La conception d'un programme se fait généralement par étapes, les algorithmes et les structures des données étant raffinés à chaque étape. Dans ce processus de conception arrive un moment où l'on doit faire le choix du langage de programmation à utiliser, ce choix influençant alors les étapes ultérieures de la conception.

Il faut choisir le langage de programmation le plus approprié au problème posé, celui qui permettra la programmation la plus simple. Ainsi FORTRAN 90 est conçu d'abord pour des applications scientifiques, COBOL pour des applications de gestion, APT pour des applications de commande numérique industrielle, mais les langages de programmation évolués comme C++ ou Ada 95 couvrent un plus grand nombre de champs d'application. Il existe à l'heure actuelle plusieurs centaines de langages de programmation évolués, c'est une véritable tour de Babel. Cependant tous les algorithmes ne sont pas nécessairement programmés de la meilleure façon à l'aide de langages évolués. En effet certains langages spécialisés ne possèdent pas les outils nécessaires à des traitements non directement reliés à leur spécialité et il faut souvent coder un algorithme de façon à tirer le meilleur parti possible des ressources de l'ordinateur. L'accès à certaines ressources peut être impossible ou inefficace à partir d'un langage évolué; le contrôle de la façon dont les calculs sont faits à l'intérieur d'un langage évolué échappe ainsi souvent au programmeur.

On peut toujours utiliser le *langage d'assemblage* à la place des langages évolués. Un langage d'assemblage est une forme symbolique des instructions du *langage machine*. Les premiers langages de programmation étaient en fait des langages d'assemblage! A l'opposé des langages évolués, les langages d'assemblage n'ont pas été conçus pour des catégories d'applications mais reflètent l'architecture des ordinateurs sur lesquels ils existent. La spécification des instructions en langage d'assemblage devra être plus précise qu'en langage évolué, ainsi l'instruction d'affectation ci-dessous:

X = A + B – C;

sera codée de la façon suivante:

```
OBT   A        ;obtenir valeur de A
AJT   B        ;ajouter valeur de B
SOU   C        ;soustraire valeur de C
RAN   X        ;ranger valeur dans X
```

La programmation en langage d'assemblage demande plus de *soin* et de *précision* que la programmation en langage évolué. Le langage d'assemblage n'est pas réservé à certaines applications, on peut l'utiliser pour tout faire: à partir du langage d'assemblage on a en effet accès a toutes les ressources de l'ordinateur et on a un contrôle complet sur tout le programme. Il ne faut alors pas perdre de vue que la tâche du programmeur est en conséquence **plus lourde**. Un programmeur en langage évolué n'a besoin que d'une connaissance superficielle du fonctionnement de l'ordinateur; ses programmes sont traduits en langage machine, mais il n'a pas à s'en préoccuper. Par contre, le programmeur en langage d'assemblage doit connaître le fonctionnement de l'ordinateur et sa structure et c'est cette connaissance qui fait du langage d'assemblage un outil puissant.

Les langages de haut niveau sont indépendants des ordinateurs ce qui permet idéalement à un même programme de fonctionner sur plusieurs types d'ordinateurs différents (la réalité peut parfois exiger des retouches). Un programme en langage d'assemblage dépend d'un type d'ordinateur donné; d'un type d'ordinateur à un autre il n'y a pas deux langages d'assemblage identiques. Cependant tous les langages d'assemblage ont des caractéristiques communes, de sorte qu'un programmeur connaissant bien un langage d'assemblage peut s'adapter rapidement à un autre.

A part quelques exceptions importantes comme MULTICS (écrit en PL/1) et UNIX (écrit en C), les anciens systèmes d'exploitation des ordinateurs étaient habituellement écrits en langage d'assemblage. On devait donc connaître le langage d'assemblage pour devenir programmeur de systèmes; ceci est

toujours vrai car certaines parties critiques du système (par exemple le noyau ou « kernel ») doivent être écrites en langage d'assemblage pour des raisons d'efficacité. Le langage d'assemblage est aussi un outil valable pour le programmeur en langage évolué qui désire améliorer l'efficacité de ses programmes, encor que de nos jours les compilateurs optimisent très bien leur code. Pour la programmation d'applications il est peu souhaitable de programmer tout un algorithme en langage d'assemblage; cependant en codant certaines parties de l'algorithme en langage d'assemblage on peut parfois en augmenter grandement l'efficacité. Un programmeur connaissant un langage d'assemblage a une meilleure connaissance du fonctionnement interne de l'ordinateur ce qui peut lui permettre d'en tirer le meilleur parti.

```
//  Ce programme calcule et affiche le premier nombre de Fibonacci
//  supérieur à 500.
//    Philippe Gabrini     mai 2005
//
#include <iostream>
using namespace std;

int main()
{
  const int LIMITE = 500;
  int avantDernier = 0, dernier = 1, somme = 1;

  while(somme < LIMITE) {
    avantDernier = dernier;
    dernier = somme;
    somme = avantDernier + dernier;
  }
  cout << "Le premier nombre de la suite de Fibonacci supérieur à "
       << LIMITE << " est " << somme << endl;
  return 0;
}
```

Figure 1.1 Programme de la suite de Fibonacci en C++

Pour illustrer la différence entre langage évolué et langage d'assemblage nous donnons dans la figure 1.1 l'exemple d'un programme C++, suivi de la figure 1.2 qui est sa traduction par le compilateur Metrowerks CodeWarrior en langage d'assemblage pour processeur Intel.

Le programme équivalent en langage d'assemblage produit par le compilateur vous semble obscur et indûment compliqué? C'est normal, puisque vous n'avez encore jamais vu de programme en langage d'assemblage. Et même si vous vous y connaissiez en langage d'assemblage, les instructions produites par les compilateurs ont une forme spéciale et le système C++ ajoute de son côté quelques complications au code. Ce programme n'est donné que comme exemple, on ne s'attend pas à ce que vous en saisissiez les complexités intimes!

```
                    *** COFF HEADER (Fibonacci.cpp) ***

machine            = 0x014c (i386)
numsections        = 15
timestamp          = 0x428c64fb (Thu May 19 10:05:47 2005)
symboltableoffset  = 0x000013d6
numberofsymbols    = 95
optionalheadersize = 0
characteristics    = 0x00000000 ()


                    *** SECTION HEADER TABLE ***
```

```
no   offset      size        vaddr       vsize       flags       name
     lines       nlines      relocs      nrelocs

1    0x0000026c  0x000000c7  0x00000000  0x00000000  0x00100a00  .drectve
     0x00000333  0           0x00000333  0
2    0x00000333  0x00000078  0x00000000  0x00000000  0x42100048  .debug$S
     0x000003ab  0           0x000003ab  4
3    0x000003d3  0x00000bd0  0x00000000  0x00000000  0x42100048  .debug$T
     0x00000fa3  0           0x00000fa3  0
4    0x00000fa3  0x0000001d  0x00000000  0x00000000  0xc0400080  .bss
     0x00000fa3  0           0x00000fa3  0
5    0x00000fa3  0x00000097  0x00000000  0x00000000  0x60501020  .text
     0x0000103a  12          0x00001082  13
6    0x00001104  0x00000028  0x00000000  0x00000000  0xc0101040  .data
     0x0000112c  0           0x0000112c  0
7    0x0000112c  0x00000010  0x00000000  0x00000000  0xc0101040  .data
     0x0000113c  0           0x0000113c  0
8    0x0000113c  0x00000001  0x00000000  0x00000000  0x40101040  .rdata
     0x0000113d  0           0x0000113d  0
9    0x0000113d  0x00000008  0x00000000  0x00000000  0x40301040  .exc$BBB
     0x00001145  0           0x00001145  2
10   0x00001159  0x00000088  0x00000000  0x00000000  0x42101048  .debug$S
     0x000011e1  0           0x000011e1  6
11   0x0000121d  0x00000047  0x00000000  0x00000000  0x60501020  .text
     0x00001264  2           0x00001270  12
12   0x000012e8  0x00000004  0x00000000  0x00000000  0xc0300040  .CRT$XCU
     0x000012ec  0           0x000012ec  1
13   0x000012f6  0x00000001  0x00000000  0x00000000  0x40101040  .rdata
     0x000012f7  0           0x000012f7  0
14   0x000012f7  0x00000008  0x00000000  0x00000000  0x40301040  .exc$BBB
     0x000012ff  0           0x000012ff  2
15   0x00001313  0x00000087  0x00000000  0x00000000  0x42101048  .debug$S
     0x0000139a  0           0x0000139a  6


                    *** SYMBOL TABLE ***

no     value       type      storage   shndx  aux  name

0      0x00000005  NULL      FILE      65534  2    .file
3      0x00000000  NULL      STATIC    2      1    .debug$S
5      0x0000000f  NULL      FILE      65534  5    .file
11     0x00000000  NULL      STATIC    4      1    .bss
13     0x00000000  NULL      STATIC    4      0    @7071
14     0x0000000c  NULL      STATIC    4      0    ?__msl_ios_base_ninit@std@@3V__nInit@1@A
15     0x00000017  NULL      FILE      65534  5    .file
21     0x00000010  NULL      STATIC    4      0    @7072
22     0x0000001c  NULL      STATIC    4      0    ?__msl_ios_base_winit@std@@3V__wInit@1@A
23     0x00000030  NULL      FILE      65534  2    .file
26     0x00000000  NULL      STATIC    5      1    .text
28     0x00000000  FUNC      EXTERNAL  5      1    _main
30     0x00000000  NULL      STATIC    6      1    .data
32     0x00000000  NULL      STATIC    6      0    @7114
33     0x00000000  NULL      STATIC    7      1    .data
35     0x00000000  NULL      STATIC    7      0    @7115
36     0x00000000  NULL      STATIC    8      1    .exc$T
38     0x00000000  NULL      STATIC    9      1    .exc$BBB
40     0x00000000  NULL      STATIC    9      0    exc$bbb_9
41     0x00000000  NULL      FUNCTION  5      1    .bf
43     0x0000000c  NULL      FUNCTION  5      0    .lf
44     0x00000097  NULL      FUNCTION  5      1    .ef
46     0x00000000  NULL      STATIC    10     1    .debug$S
48     0x0000003f  NULL      FILE      65534  5    .file
54     0x00000000  NULL      STATIC    11     1    .text
56     0x00000000  FUNC      STATIC    11     1    _$static_initializer$1
58     0x00000000  NULL      STATIC    13     1    .exc$T
60     0x00000000  NULL      STATIC    14     1    .exc$BBB
62     0x00000000  NULL      STATIC    14     0    exc$bbb_14
63     0x00000048  NULL      FILE      65534  5    .file
69     0x00000000  NULL      FUNCTION  11     1    .bf
71     0x00000002  NULL      FUNCTION  11     0    .lf
```



```
72      0x00000000  NULL      FILE      65534  5    .file
78      0x00000000  NULL      FUNCTION  11     0    .lf
79      0x00000047  NULL      FUNCTION  11     1    .ef
81      0x00000000  NULL      STATIC    15     1    .debug$S
83      0x00000000  NULL      EXTERNAL  0      0
?endl@?$@DU?$char_traits@D@std@@@2@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@2@AAV32@@Z
84      0x00000000  NULL      EXTERNAL  0      0
?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
85      0x00000000  NULL      EXTERNAL  0      0
??6?$@U?$char_traits@D@std@@@1@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@PBD@Z
86      0x00000000  NULL      EXTERNAL  0      0
??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z
87      0x00000000  NULL      EXTERNAL  0      0
?cin@std@@3V?$basic_istream@DU?$char_traits@D@std@@@1@A
88      0x00000000  NULL      EXTERNAL  0      0
??5?$basic_istream@DU?$char_traits@D@std@@@std@@QAEAAV01@AAH@Z
89      0x00000000  NULL      EXTERNAL  0      0
??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@H@Z
90      0x00000000  NULL      EXTERNAL  0      0    ??0__nInit@std@@QAE@XZ
91      0x00000000  NULL      EXTERNAL  0      0    ??1__nInit@std@@QAE@XZ
92      0x00000000  NULL      EXTERNAL  0      0    ___register_global_object
93      0x00000000  NULL      EXTERNAL  0      0    ??0__wInit@std@@QAE@XZ
94      0x00000000  NULL      EXTERNAL  0      0    ??1__wInit@std@@QAE@XZ


                    *** STRING TABLE ***

offset      string

0x00000004  ?__msl_ios_base_ninit@std@@3V__nInit@1@A
0x0000002d  ?__msl_ios_base_winit@std@@3V__wInit@1@A
0x00000056  exc$bbb_9
0x00000060  _$static_initializer$1
0x00000077  exc$bbb_14
0x00000082
?endl@?$@DU?$char_traits@D@std@@@2@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@2@AAV32@@Z
0x000000dd  ?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
0x00000116
??6?$@U?$char_traits@D@std@@@1@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@PBD@Z
0x00000170  ??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z
0x000001bd  ?cin@std@@3V?$basic_istream@DU?$char_traits@D@std@@@1@A
0x000001f5  ??5?$basic_istream@DU?$char_traits@D@std@@@std@@QAEAAV01@AAH@Z
0x00000234  ??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@H@Z
0x00000271  ??0__nInit@std@@QAE@XZ
0x00000288  ??1__nInit@std@@QAE@XZ
0x0000029f  ___register_global_object
0x000002b9  ??0__wInit@std@@QAE@XZ
0x000002d0  ??1__wInit@std@@QAE@XZ


                    *** LINKER DIRECTIVES (.drectve #1) ***


-defaultlib:MSL_Runtime_x86 -defaultlib:MSL_C_WINSIOUX -defaultlib:MSL_C++_x86
-defaultlib:MSL_Extras_WINSIOUX -defaultlib:comdlg32,winspool -defaultlib:gdi32
-defaultlib:kernel32 -defaultlib:user32


                    *** EXECUTABLE CODE (.text #5) ***


Symbols associated with section:

   28: GLOBAL_CODE "_main"
   41: .bf
   26: .text
   43: .lf
   44: .ef


Source file:          C:\INF3105\Progs\Fibonacci.cpp
```

```
;
;    8: int main()
;    9: {
;   10:   const int LIMITE = 500;
;

       _main:

0x00000000: 55              push      ebp
0x00000001: 89E5            mov       ebp,esp
0x00000003: 83EC0C          sub       esp,0xc
0x00000006: B8CCCCCCCC      mov       eax,0xcccccccc
0x0000000b: 890424          mov       dword ptr [esp],eax
0x0000000e: 89442404        mov       dword ptr [esp+0x4],eax
0x00000012: 89442408        mov       dword ptr [esp+0x8],eax
;
;   11:   int avantDernier = 0, dernier = 1, somme = 1;
;   12:
;
0x00000016: C745FC00000000  mov       dword ptr [ebp-0x4],0x0
0x0000001d: C745F801000000  mov       dword ptr [ebp-0x8],0x1
0x00000024: C745F401000000  mov       dword ptr [ebp-0xc],0x1
;
;   13:   while(somme < LIMITE) {
;
0x0000002b: EB15            jmp       $+0x17 (0x42)
0x0000002d: 8B45F8          mov       eax,dword ptr [ebp-0x8]
;
;   14:     avantDernier = dernier;
;
0x00000030: 8945FC          mov       dword ptr [ebp-0x4],eax
0x00000033: 8B45F4          mov       eax,dword ptr [ebp-0xc]
;
;   15:     dernier = somme;
;
0x00000036: 8945F8          mov       dword ptr [ebp-0x8],eax
;
;   16:     somme = avantDernier + dernier;
;
0x00000039: 8B55FC          mov       edx,dword ptr [ebp-0x4]
0x0000003c: 0355F8          add       edx,dword ptr [ebp-0x8]
0x0000003f: 8955F4          mov       dword ptr [ebp-0xc],edx
;
;   17:   }
;
0x00000042: 817DF4F4010000  cmp       dword ptr [ebp-0xc],0x1f4
0x00000049: 7CE2            jl        $-0x1c (0x2d)
;
;   18:   cout << "Le premier nombre de la suite de Fibonacci sup\u99FBieur \u30FB"
;   19:        << LIMITE << " est " << somme << endl;
;
0x0000004b: 6800000000      push      offset
?endl@?$@DU?$char_traits@D@std@@@2@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@2@AAV32@@Z
0x00000050: FF75F4          push      dword ptr [ebp-0xc]
0x00000053: 6800000000      push      offset @7116
0x00000058: 68F4010000      push      0x1f4
0x0000005d: 6800000000      push      offset @7117
0x00000062: 6800000000      push      offset
?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
0x00000067: E800000000      call
??6?$@U?$char_traits@D@std@@@1@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@PBD@Z (0x6c)
0x0000006c: 59              pop       ecx
0x0000006d: 59              pop       ecx
0x0000006e: 89C1            mov       ecx,eax
0x00000070: E800000000      call      ??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@H@Z
(0x75)
0x00000075: 50              push      eax
0x00000076: E800000000      call
??6?$@U?$char_traits@D@std@@@1@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@PBD@Z (0x7b)
0x0000007b: 59              pop       ecx
```



```
0x0000007c: 59              pop       ecx
0x0000007d: 89C1            mov       ecx,eax
0x0000007f: E800000000      call      ??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@H@Z
(0x84)
0x00000084: 89C1            mov       ecx,eax
0x00000086: E800000000      call
??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z (0x8b)
;
;   20:   return 0;
;
0x0000008b: B800000000      mov       eax,0x0
;
;   21: }
;
0x00000090: C9              leave
0x00000091: C3              ret       near


                    *** INITIALIZED DATA (.data #6) ***


Symbols associated with section:

   32: @7116
   30: .data


       @7116:

0x00000000: 20 65 73 74 20 00 00 00.                        ' est ...'


                    *** INITIALIZED DATA (.data #7) ***


Symbols associated with section:

   35: @7117
   33: .data


       @7117:

0x00000000: 4C 65 20 70 72 65 6D 69.65 72 20 6E 6F 6D 62 72. 'Le premier nombr'
0x00000010: 65 20 64 65 20 6C 61 20.73 75 69 74 65 20 64 65. 'e de la suite de'
0x00000020: 20 46 69 62 6F 6E 61 63.63 69 20 73 75 70 E9 72. ' Fibonacci sup.r'
0x00000030: 69 65 75 72 20 E0 20 00.                        'ieur . .'

                    *** EXECUTABLE CODE (.text #11) ***


Symbols associated with section:

   69: .bf
   54: .text
   56: GLOBAL_CODE "_$static_initializer$1"
   78: .lf
   71: .lf
   79: .ef


Source file:          C:\Program
Files\Metrowerks\CodeWarrior\MSL\MSL_C++\MSL_Common\Include\wiostream


       _$static_initializer$1:

0x00000000: 55              push      ebp
0x00000001: 89E5            mov       ebp,esp
0x00000003: B900000000      mov       ecx,offset ?__msl_ios_base_ninit@std@@3V__nInit@1@A
0x00000008: E800000000      call      ??0__nInit@std@@QAE@XZ (0xd)
```

```
0x0000000d: 6800000000       push      offset @7071
0x00000012: 6800000000       push      offset ??1__nInit@std@@QAE@XZ
0x00000017: 6800000000       push      offset ?__msl_ios_base_ninit@std@@3V__nInit@1@A
0x0000001c: E800000000       call      ___register_global_object (0x21)
0x00000021: 83C40C           add       esp,0xc
0x00000024: B900000000       mov       ecx,offset ?__msl_ios_base_winit@std@@3V__wInit@1@A
0x00000029: E800000000       call      ??0__wInit@std@@QAE@XZ (0x2e)
0x0000002e: 6800000000       push      offset @7072
0x00000033: 6800000000       push      offset ??1__wInit@std@@QAE@XZ
0x00000038: 6800000000       push      offset ?__msl_ios_base_winit@std@@3V__wInit@1@A
0x0000003d: E800000000       call      ___register_global_object (0x42)
0x00000042: 83C40C           add       esp,0xc
0x00000045: C9               leave
0x00000046: C3               ret       near


                    *** INITIALIZED DATA (.CRT$XCU #12) ***


0x00000000: 00 00 00 00                                  '....'
```

Figure 1.2 Programme de Fibonacci traduit en langage d'assemblage

Cet exemple (figure 1.1) du programme C++ de la suite de Fibonacci et de sa traduction en assembleur par le compilateur (figure 1.2) peut cependant être compris lorsqu'on connaît un peu mieux le système. Le programme traduit comprend plusieurs parties: des tables de symboles, des instructions et des données. Les instructions (section EXECUTABLE CODE) sont elles-mêmes divisées en deux parties: une partie d'initialisation par défaut due au système C++ (adresses 0x00000000, ou 0, à 0x00000015, ou 15 hexadécimal, indiquées dans la colonne de gauche) et une partie traduisant directement les instructions du programme (adresses 16 à 91 – les adresses sont toujours données en hexadécimal). Une dernière partie définit les données; on y retrouve les chaînes de caractères utilisées dans les instructions de sortie. Les variables du programme `avantDernier`, `dernier` et `somme` se trouvent repérées par les décalages hexadécimaux 0x4, 0x8 et 0xC d'une zone de mémoire attribuée au programme.

L'objectif de ce cours n'est pas de faire de vous des programmeurs spécialisés en assembleur, mais plutôt de vous révéler le fonctionnement du processeur en réduisant le niveau de votre programmation à celui du processeur. L'objectif étant essentiellement de vous faire apprendre les concepts qui sous-tendent le fonctionnement des processeurs, le choix du langage d'assemblage n'est pas d'une importance capitale, dans la mesure où l'assembleur choisi illustre bien la plupart des concepts. Les assembleurs sont liés à des processeurs particuliers et chaque processeur possède ses forces et ses faiblesses. Une fois que vous aurez acquis les concepts liés à la programmation du processeur, il vous sera facile de passer à un autre processeur et à son langage d'assemblage particulier.

