INTRODUCTION AU LANGAGE ASSEMBLEUR 1

M. CHANIAT

INTRODUCTION

Avant même d'étudier, ou plutôt de présenter très succinctement quelques remarques d'ordre général à propos du langage assembleur, il convient de s'interroger sur les raisons de son éventuelle utilisation. En effet, la mise en œuvre d'une application, écrite totalement ou partiellement en langage assembleur requiert entre autre une relative maîtrise de ce langage informatique. Il est donc tout à fait indispensable de développer des argumentations susceptibles de justifier l'usage de ce langage, d'autant que sa mise en œuvre ne peut être envisagée qu'à l'issue d'une longue et difficile période d'apprentissage. De plus, il ne saurait être question d'abreuver le lecteur de la connaissance stérile de nombreux langages informatiques, sauf dans le cas où la connaissance de certains de ces langages se révèle nécessaire, en particulier par le caractère très spécifique de l'application ou par les limites imposées par le ou les langages déjà étudiés.

POURQUOI UTILISER LE LANGAGE ASSEMBLEUR ?

L'utilisation du langage assembleur se doit d'être justifiée par des arguments de poids, tant l'investissement intellectuel requis pour sa parfaite connaissance est important. La conception de logiciels, quel que soit le domaine de l'application envisagée, se traduit en particulier, lors de la réalisation informatique du produit, par l'écriture d'algorithme. Or, dans de très nombreux cas, certains algorithmes, tout à fait performants pour le traitement d'une quantité réduite d'informations, deviennent d'une utilisation difficile voire impossible à envisager lorsqu'ils concernent de plus grandes masses d'informations. C'est le cas en particulier d'algorithmes de tri, de recherche, etc.

Dans un premier temps, il peut être envisagé de pallier cette lenteur par la recherche d'algorithmes intrinsèquement plus performants. Les nombreux exemples d'algorithmes de tri témoignent de cette recherche. Cependant, cette voie présente ses limites, propres, et peut se révéler insuffisante, même si les résu1tats apportés contribuent déjà a améliorer notablement les performances.

Une seconde voie consiste alors à écrire ces algorithmes dans un autre langage informatique. En particu1ier,le passage d'un langage interprété à un langage compilé peut, dans certains cas, s'avérer tout à fait justifié, et améliorer encore les performances déjà obtenues. Toutefois, cette seconde voie peut aussi se révéler encore insuffisante, bien que susceptible de répondre à la plupart des applications courantes.

La technique, qui incontestablement permet de faire un bon prodigieux au niveau du temps d'exécution des algorithmes est celle mettant en œuvre le langage assembleur. Selon les cas, le gain de temps peut s'étendre d'un facteur 10 dans les situations, les plus défavorables, à plusieurs centaines. Le gain de temps peut être tel que, dans certains cas, le seul fait d'écrire en langage assembleur un algorithme intrinsèquement très peu performant, puisse suffire à porter ses performances à un niveau tout à fait acceptable, comparable voire même supérieur à celui des algorithmes les plus performants. Enfin, pour certaines applications, en particulier celles qui concernent le génie nucléaire, le calcul vectoriel, etc. les solutions envisagées précédemment se montrent encore insuffisantes. Ainsi, depuis plus de 10 ans, la recherche tente de définir de nouvelles structures pour l'unité arithmétique. C'est ainsi que des équipes de chercheurs travaillent sur des calculateurs à multiprocesseurs, ou avec des unités de calcul mettant en œuvre le principe du parallélisme. Pour passionnants que soient ces travaux, leur présentation sort du cadre de cet article. Le recours au langage assembleur peut aussi, dans certains cas, être justifié par les limites mêmes du langage évolue dont l'emploi a été envisagé. En effet, un langage informatique dit évolué se situe entre le langage naturel d'une part, trop complexe pour être analysé tel quel par un système informatique (du moins dans l'état actuel des recherches), et le langage binaire d'autre part, seul exécutable par le microprocesseur, mais assez hermétique et éloigné des systèmes traditionnels de référence. Un tel compromis engendre quasi nécessairement des contraintes , parmi lesquelles l'impossibilité d'exploiter à l'aide d'un langage évolué toutes les ressources offertes par le système informatique. En outre, un langage informatique est plus ou moins lié à une gamme d'applications. Le Fortran est plus spécifiquement destiné

aux applications numériques, alors que le Cobol s'adresse plutôt au domaine de la gestion. Comme dans d'autres domaines d'ailleurs, il n'existe donc pas de langage informatique universel, c'est-à-dire apte à traduire des applications de toutes natures (cependant, certains langages de conception récente tentent de présenter ce caractère universel, comme c'est le cas du langage ADA). Soit le langage est conçu en fonction d'un type d'application donné, comme c'est le cas du Fortran, soit ce langage est trop général, et ne peut plus convenir à certaines applications dès lors qu'elles deviennent trop pointues, c'est le cas du Basic ou du LSE.

DÉPENDANCE DU LANGAGE ASSEMBLEUR

Plus que tout autre, le langage assembleur est très fortement dépendant de plusieurs facteurs.

En effet, par sa nature même, ce langage est directement lié au jeu d'instructions du microprocesseur. Or, le plus souvent, les jeux d'instructions des divers microprocesseurs sont totalement incompatibles, sauf cas très particulier où les concepteurs de ces circuits intégrés ont souhaité une compatibilité totale ou partielle, pour des raisons évidentes de coût de développement, ou pour des raisons historiques.

D'autre part, les applications écrites en langage assembleur sont très fortement dépendantes de l'environnement hardware du microprocesseur. Plus encore que la précédente cette dépendance est susceptible de remettre en cause gravement la portabilité des programmes utilisant totalement ou le plus souvent partiellement le langage assembleur, même entre système informatique organisé autour du même microprocesseur. Cependant, le recours aux ressources offertes par un système d'exploitation peut, dans une certaine mesure, limiter cette dépendance vis à vis de l'environnement hardware. La conception de certains systèmes d'exploitation est telle que l'environnement matériel qui entoure le microprocesseur est représenté par une série de fonctions, ce qui tend à rendre cet environnement transparent pour l'application.

CONDITIONS D'UTILISATION DU LANGAGE ASSEMBLEUR

La réalisation d'application, que ce soit totalement ou le plus souvent partiellement, à l'aide du langage assembleur, requiert des connaissances multiples et variées, ainsi que la mise en œuvre d'un

environnement logiciel, sans lequel tout développement de l'application devient très vite impossible.

Les connaissances requises concernent en particulier les points suivants :

- maîtrise parfaite de la structure interne du microprocesseur
- maîtrise et connaissance complète du fonctionnement du microprocesseur, et en particulier de ses modes d'adressage
- connaissance du jeu complet d'instructions lié au micro processeur,
- connaissance de l'environnement hardware du microprocesseur, ou des fonctionnalités offertes par le système d'exploitation ou les logiciels de développement.

Connaissance de la structure interne du microprocesseur

Chaque titre de microprocesseur possède sa propre structure interne. La connaissance de celle-ci a donc une influence considérable sur la traduction de l'algorithme. Par exemple, certains microprocesseurs, comme le 6502, ne possèdent que très peu de registres (sortes de mémoires très rapides internes au microprocesseur), alors que d'autres, comme le célèbre Z80, en ont un nombre si important que la plupart des applications ne les exploitent pas tous. Dans le premier cas, des échanges fréquents entre les registres du microprocesseur et la mémoire centrale seront donc nécessaires, alors que dans le second cas, ces échanges concerneront davantage les registres, donc pourront éventuellement contribuer à une vitesse d'exécution encore plus grande.

Il est facile de comprendre qu'un microprocesseur disposant de 7 registres internes sur 8 bits pourra contenir simultanément 7 octets, ce qui ne peut pas être le cas d'un microprocesseur qui ne dispose que de 2 registres.

Connaissance du jeu d'instructions lié au microprocesseur

Chaque type de microprocesseur met à la disposition du programmeur une série plus ou moins importante d'opérations élémentaires dont l'ensemble constitue le jeu d'instructions.

En fonction du jeu d'instructions disponible, certaines tâches seront plus ou moins faciles à traduire. par exemple, le zao dispose d'instructions qui permettent des déplacements de zones complètes de la

mémoire centrale. Une telle tâche sera donc traduite facilement sur ce type de microprocesseur. En revanche, le 6502 ne dispose pas, dans son jeu d'instructions, d'opérations similaires. La traduction de cette tâche sera donc plus difficile et sera réalisée en faisant appel à des instructions élémentaires.

Parmi les instructions disponibles sur un microprocesseur, certaines intéressent en particulier les échanges entre les registres du microprocesseur et l'unité centrale, et réciproquement. Or, ces échanges peuvent s'effectuer selon plusieurs modalités appelées modes d'adressage. Certains microprocesseurs, comme le 6502, sont plus riches en mode d'adressage que d'autres, comme le zao. Là encore, un microprocesseur représente un compromis, dont la connaissance des limites est indispensable.

Connaissance de l'environnement hardware

Dans ce domaine, une connaissance à caractères multiples semble nécessaire.

Au niveau général, il est indispensable de connaître les différentes modalités de dialogue et de gestion des divers périphériques. Par exemple, le contrôle d'un écran peut se faire sous des formes très diverses selon les cas. On peut à ce sujet remarquer le caractère pluridisciplinaire des connaissances requises, ainsi que le niveau de complexité atteint. Le programmeur en langage évolué peut pratiquement tout ignorer de la façon dont l'écran est géré, puisque c'est l'ensemble langage informatique et système d'exploitation qui prend en charge cette gestion.

Au niveau particulier, il est indispensable de connaître l'interface entre les périphériques et le microprocesseur.

Une telle connaissance' n'est souvent disponible que dans les documentations techniques des constructeurs, quand ceux ci acceptent de les diffuser. Si tel n'est pas le cas, un travail ardu et souterrain est nécessaire' pour tenter de découvrir comment se réalise la gestion de tel ou tel périphérique. De plus un tel investissement n'est exploitable que dans le cas très particulier du système testé, ce qui peut en partie expliquer le manque d'intérêt pour le langage assembleur.

NÉCESSITÉ D'UN ENVIRONNEMENT LOGICIEL

Après l'acquisition et la maîtrise des connaissances mentionnées ci-avant, la phase pratique d'une application peut alors être envisagée. Cependant, par sa nature même, la mise en œuvre du langage assembleur, comme d'ailleurs d'autres langages, s'accompagne de la perte du caractère interactif disponible avec un langage informatique évolué interprété, comme c'est le cas pour le Basic ou le LSE. C'est d'ailleurs en partie ce caractère interactif de la mise au point sous ces langages qui permet à certains "programmeurs" d'improviser devant le clavier de la machine telle ou telle solution. L'absence de cette interactivité rendant la mise au point, accompagnée de la difficulté de conception inhérente au langage lui-même, qui par définition n'est pas évolué, implique donc de disposer d'un logiciel dont le rôle est en quelque sorte de recréer artificiellement ce caractère interactif. C'est l'objet même de l'existence de logiciel appelé débuggeur, ou metteur de mise au point. Ce type de logiciel est donc tout à fait indispensable pendant la phase de mise au point de l'application écrite en langage assembleur. L'absence d'un tel logiciel ou ensemble de logiciels entraîne, dans le meilleur des cas, un accroissement considérable de la phase de mise au point, et dans le pire des cas l'abandon pur et simple, contraint et forcé, de l'implémentation de l'application en langage assembleur.

IMPORTANCE DE L'INTERFACAGE AVEC UN LANGAGE ÉVOLUÉ

Le plus souvent, les différentes tâches que doit remplir un logiciel ne nécessitent pas le recours au langage assembleur.

Dans la plupart des applications, seules quelques tâches sont écrites à l'aide du langage assembleur, le reste, c'est-à-dire la presque totalité du logiciel, est écrit dans le langage évolué choisi pour traduire l'application. Cette cohabitation, entre le langage évolué d'une part, et le langage d'autre part, au sein d'un même logiciel ne semble pas s'accommoder d'un hermétisme total. En effet, il semble indispensable que la (ou les) tâche écrite en langage assembleur ait accès aux variables gérées par le langage évolué, du moins si la fonction remplie par cette tâche le nécessite. C'est le cas par exemple d'un algorithme de tri écrit en assembleur. A quoi pourrait bien servir une telle fonction si elle ne pouvait accéder au vecteur qu'elle doit trier, Mais un tel interfaçage ne doit pas être unilatéral. En effet, une fonction écrite en langage

assembleur peut avoir pour objet d'informer sur l'état d'un périphérique. Il convient donc que cette information puisse être transmise depuis le langage assembleur jusqu'au langage évolué.

Ainsi, une application, écrite en langage assembleur, destinée à être intégrée dans un logiciel écrit en langage évolué, nécessite-t-elle :

- un passage facile des paramètres de toute nature et en quantité quelconque, entre le langage évolué et le langage assembleur. Ces paramètres constituent donc les informations d'entrée vues depuis le langage assembleur.
- un passage tout aussi facile entre le langage assembleur et le langage évolué. Ce passage permet donc à la fonction écrite en langage assembleur de transmettre des informations au reste du logiciel.

A ce sujet, il convient de noter et d'insister sur le fait que l'intégration de procédures écrites en langage assembleur, interfacées avec un logiciel écrit en langage LSE, est grandement facilitée dans la mesure où le système LSE lui-même prend en charge ces passages de paramètres. On ne saurait en dire autant des sous-programmes écrits en langage assembleur interfacés avec la plupart des Basic, puisque le langage Basic lui-même ne contient pas la notion de passation de paramètres entre sous-programmes, sauf pour des versions "pascaliennes" récentes et peu diffusées de ce langage.

CONCLUSION

Ces quelques généralités, relatives au langage assembleur, et à ses conditions de mise en œuvre, ne sauraient en aucun cas faire référence en la matière, et encore moins représenter une liste exhaustive. Avant d'envisager le recours au langage assembleur, tout programmeur, non spécialiste de ce langage, doit ne pas ignorer les quelques remarques suivantes :

- les connaissances préalables à la mise en œuvre d'un langage assembleur sont d'une acquisition et d'une maîtrise longue et difficile. Pas question de développer le moindre embryon d'application après quelques heures ou quelques jours.
- Certaines connaissances acquises présentent un caractère très particulier, et ont donc un champ d'application très réduit. Le jeu d'instructions ne concerne qu'un seul type de microprocesseur.

L'interfaçage microprocesseur-périphérique est spécifique d'un type de machine. Une actualisation des connaissances est donc nécessaire, à chaque passage d'un type de matériel à un autre, ou d'un type de microprocesseur à un autre.

– une méthodologie rigoureuse doit être appliquée pour mener à bien le développement d'une application écrite dans ce langage. Ici, pas de place à l'improvisation ou au "bidouillage". En particulier, une démarche modulaire est absolument indispensable. A ce sujet, l'apprentissage de la programmation dans un langage assembleur peut présenter un intérêt pédagogique certain, dans la mesure où par exemple, certaines "libertés" ou "mauvaises habitudes" permises avec certains langages évolués ne sont plus de mise ici.

Malgré ces contraintes, qui d'ailleurs ne présentent ce caractère pesant que pour les adeptes, s'il y en a, de la programmation "sauvage", la mise en œuvre d'un langage assembleur se révèle d'un intérêt à la hauteur des difficultés qu'il engendre.

M. CHANIAT
Formateur en informatique
C.E.S. Champs Plaisants
SENS