Le C et l'ASM, quelle diffÃ©rence ?

CatÃ©gorie : Le C et l'ASM
PubliÃ© par Charly le 02/04/2013

Vous verrez souvent des rÃ©fÃ©rences Ã du code en "C" ou du code en "ASM" (ou assembleur) pour donner vie Ã nos micro-contrÃ´leurs prÃ©fÃ©rÃ©s.

Quelle est la diffÃ©rence ? Pourquoi programmer en C plutÃ´t quâ□□en assembleur ?

Cet article a pour objectif de vous aider Ã comprendre simplement de quoi on parle afin que vous ne puissiez pas dire quâ□□il y en a un meilleur que lâ□□autre â□¦

Vous trouverez quelques liens vers des cours magistraux qui vous aideront Ã approfondir les deux sujets.

Un ÂµC Ã§a fonctionne comment ?

Pour comprendre il faut revenir aux bases : un micro-contrÃ´leur exÃ©cute un programme qui est chargÃ© dans sa mÃ©moire interne sous la forme dâ□□une suite dâ□□instructions. Ces instructions correspondent Ã des actions que le micro-contrÃ´leur sait rÃ©aliser. Elles sont reprÃ©sentÃ©es par une succession de valeurs hexadÃ©cimales.

Dans nos projets Freescale CodeWarrior, ce fichier sera un .s19 pour les MC9S08 de Freescale. Il contient ces instructions en hexadÃ©cimal et est illisible (Ã moins dâ□□Ãªtre trÃ¨s bon et dâ□□y passer un temps infini et/ou dâ□□Ãªtre un pirate).

Ainsi, vous lâ□□avez compris, quand nous programmons nous cherchons Ã dire "en franÃ§ais" ce qui deviendra de lâ□□hexadÃ©cimal, une succession dâ□□instructions dans le micro-contrÃ´leur.

Lâ□□assembleur, une TRADUCTION

L'assembleur nâ□□est rien dâ□□autre quâ□□ une traduction des instructions hexadÃ©cimales dans un langage comprÃ©hensible pour nous les Hommes.

Ainsi pour mettre tous les bits du port A Ã 0 sauf le premier on Ã©crira :

```
LDA #$01      ; On charge lâ□□accumulateur A avec la valeur hexadÃ©cimale 0x01
STA $1000     ; On donne au registre placÃ© Ã  lâ□□adresse 0x1000 (
le registre du portA) la valeur de lâ□□accumulateur A.
```

Cela se traduira en langage machine par :
0xA601 et 0xC71000 (respectivement LDA(A6) #$01 et STA(C7) $1000)
(note : "0x" signifie que l'on dÃ©signe une valeur hexadÃ©cimale)

LDA #$01 (assembleur) ou 0xA601 (opÃ©ration code), câ□□est la mÃªme chose mais "LDA et STA" se lisent mieux.

Mais ne rentrons pas dans les dÃ©tails ce ces deux instructions, ce n'est pas l'objet de cet article !

"On charge lâ□□accumulateur A avec la valeur hexadÃ©cimale 0x01" Argh, quel rapport avec mon application ? !
RÃ©ponse : Le matÃ©riel, puisque ces instructions correspondent Ã ce que la structure du micro-contrÃ´leur sait faire, Ã ce que les transistors qui le composent sont capables de faire.

La programmation en assembleur implique donc dâ□□une part une parfaite maitrise du matÃ©riel, de ses possibilitÃ©s et de sa structure. Elle implique dâ□□autre part une Ã©criture assez lourde qui imposera au programmeur de rÃ©flÃ©chir murement Ã lâ□□impact de ses instructions sur tout le reste de son programme.
⚠
Il existe un assembleur diffÃ©rent pour tous les matÃ©riels. Un micro-contrÃ´leur Freescale S08 aura son assembleur avec un package dâ□□instruction quâ□□il saura exÃ©cuter. Un micro-contrÃ´leur Freescale Kinetis Mx aura un autre assembleur, votre PC aura encore un autre assembleurâ□¦

Le langage C, COMPILÃ□

Le langage C est un langage dit de "haut niveau", qui nâ□□a rien Ã voir avec un quelconque matÃ©riel (Freescale ou autre). On peut avoir le mÃªme programme (ou en tous cas une partie) en C pour un pc, un micro-contrÃ´leur Freescale ou un Âµc de toute autre marque concurrente.

Pour obtenir le programme en hexadÃ©cimal qui sera exÃ©cutÃ© par le micro-contrÃ´leur il ne suffit pas de faire une traduction mais le compilateur va compiler le programme. Ce compilateur va sÃ©lectionner et optimiser les instructions du micro-contrÃ´leur choisi afin dâ□□accomplir la tache dÃ©crite par votre programme en C.

Le C offrira dâ□□une part une grande lisibilitÃ© Ã votre programme et dâ□□autre part une portabilitÃ© de votre code vers un autre matÃ©riel, par exemple un micro-contrÃ´leur plus puissant.

⚠
Il existe beaucoup dâ□□autres langages compilÃ©s comme le C : Le C++, le Pascal, ainsi que beaucoup dâ□□autres.
Dâ□□autres langages de haut niveau comme le C ne compilent pas le programme en langage machine mais gÃ©nÃ¨rent un code "interprÃ©tÃ©" qui sera exÃ©cutÃ© par un moteur sur la machine.
On trouve dans cette catÃ©gorie Java, le BASIC (qui peut Ãªtre compilÃ© aussi) etc mais dans tous les cas nous sommes loin du matÃ©riel, comme avec le C, et un outil de compilation va travailler pour donner Ã manger au matÃ©riel.

Nâ□□oublions jamais le matÃ©riel qui est en dessousâ□¦

Lâ□□assembleur, qui exploite directement le matÃ©riel, permet de contrÃ´ler prÃ©cisÃ©ment ce qui va Ãªtre demandÃ© au matÃ©riel. Il est ainsi possible dâ□□Ãªtre plus rapide sur certaines opÃ©rations et de garantir la durÃ©e de certaines tÃ¢ches.

Le C est une couche logicielle au-dessus ; le compilateur aura la charge de dÃ©terminer Ã notre place les instructions qui seront exÃ©cutÃ©es par le micro-contrÃ´leur, mais si on code en C des choses inadaptÃ©es aux capacitÃ©s du matÃ©riel on aura les mÃªmes problÃ¨mes quâ□□en codant directement en assembleur.
Il faut garder Ã©galement en tÃªte que câ□□est le compilateur qui va faire le gros du travail en cherchant une successions dâ□□instructions qui rÃ©pondra au besoin fonctionnel que vous avez dÃ©crit en C ; ce compilateur peut Ãªtre "bon" ou "mauvais", il peut aussi Ãªtre paramÃ©trÃ©, par exemple pour choisir des sÃ©quences dâ□□instructions qui rendront votre programme plutÃ´t plus rapide ou plutÃ´t moins gourmand en espace mÃ©moireâ□¦

Dans tous les cas, il est donc important pour un dÃ©butant de comprendre les bases de lâ□□assembleur afin de cerner ce quâ□□il est possible de demander au micro-contrÃ´leur et les consÃ©quences que peut avoir son code sur le comportement ou la charge du micro-contrÃ´leur.

On rÃ©servera ensuite lâ□□usage de lâ□□assembleur aux quelques situations oÃ¹ on peut profiter de son avantage (prÃ©cis, proche du matÃ©riel). Dans tous les autres cas (la grande majoritÃ©) on choisira le C, plus simple, plus flexible, plus lisible et universel (portable).
⚠
A noter que lâ□□on peut insÃ©rer des fonctions en assembleur au cÅ□ur dâ□□un programme en C, on impose ainsi au compilateur le choix des instructions pour une partie du programme. Le choix n'a rien de dÃ©finitif ! Pour plus de prÃ©cisions

Lien vers le WIKI de l'assembleur.
Lien vers le WIKI du langage C
lien vers le manuel de rÃ©fÃ©rence du CPU08 sur le site de Freescale(on y trouve la description des instructions que le CPU 08 sait exÃ©cuter)

A venir : Lien vers un article exemple d'application en assembleur : Faire clignoter une LED.
A venir : Lien vers un article exemple d'application en C : Faire clignoter une LED.

Si ce type d'articles vous a plu, faites-le nous savoir dans vos commentaires (toute critique est Ã©galement bonne Ã prendre).