Architecture des ordinateur 2

Cours 4 - Gnu Assembleur pour AVR Atmega 328p

Halim Djerroud <hdd@ai.univ-paris8.fr>

LIASD - Université Paris 8

Janvier 2018

Assembleur

- L'assembleur est un langage dit bas niveau (proche du langage machine)
- Connaître obligatoirement l'architecture (ex. AVR , PIC, Intel, PowerPC . . .).
- Connaître un assembleur, dans notre cas GNU Assembleur

Connaître l'architecture, c'est comprendre le fonctionnement du processeur, des registres, d'adressage et l'organisation de la mémoire, les interruptions . . . et tout ce que vous avez appris dans le cours d'architecture des ordinateurs 2.

Assembleur

Le langage assembleur, ou simplement l'assembleur, est une représentation symbolique du langage machine.
Deux Syntaxes :

- Assembleur Intel : l'assembleur principal utilisant cette syntaxe est NASM ;
- Assembleur AT&T : l'assembleur principal est l'assembleur GNU ou simplement as.

Sections assembleur

- **.text** (read only) : la section du code. Elle contient les instructions et les constantes du programme. Un programme assembleur doit contenir au moins la section .text
- **.data** (read-write) : la section des données (data section). Elle décrit comment allouer l'espace mémoire pour les variables initialisables du programme (variables globales)
- **.bss** (read-write) : contient les variables non initialisées. Dans notre code, cette section est vide. On peut donc l'éliminer.

L'ordre des sections dans le code source n'est pas important.
N'importe quelle section peut être vide !

Exemple code Assembleur

```
.data

.bss

.text
```

Commentaires

eux méthodes pour commenter un code assembleur

- /* commentaire sur plusieurs lignes */
- # commentaire jusqu'à la fin de ligne

```
/*
 commentaires
*/

# commentaire
```

Les déclarations

- Une déclaration se termine par le caractère saut de ligne n ou par le caractère « ; »
- Une étiquette peut être suivie d'un symbole clé qui détermine le type de la déclaration
- Si le symbole clé est préfixé par un point, alors la déclaration est une directive assembleur
- Les attributs d'une directive peuvent être un symbole prédéfini, une constante ou une expression

Exemple déclarations

Les déclarations peuvent prendre quatre formes :

```
                .nom directive
label_1 :       .nom directive attribut
label_2 :
                 expression
                 instruction        op1 , op2 , ...
```

Directives

- Les directives sont des pseudo-opérations
- Elles sont utilisées pour simplifier des opérations complexes du programmeur
- Une directive est un symbole préfixé par un point (.)

```
.text
   .directive
   .directive attribut
```

Symbole

- Les lettres de l'alphabet : a .. z, A .. Z
- Les chiffres décimaux : 0 .. 9
- Les caractères : (point) . $.

Les constantes

- Un nombre binaire est un 0b ou 0B suivi de zéro ou plusieurs chiffres binaires {0, 1}.
  (ex : 0b10110101)
- Un nombre octal est un 0 suivi de zéro ou plusieurs chiffres octaux 0, 1, 2, 3, 4, 5, 6, 7.
  (ex : 04657)
- Un nombre décimal ne doit pas commencer par 0. Il contient zéro ou plusieurs chiffres décimaux 0..9.
  (ex : 19351)
- Un nombre hexadécimal est un 0x ou 0X suivi de zéro ou plusieurs chiffres hexadécimaux 0..9, A, B, C, D, E, F.
  (ex : 0x4F)

Les caractères

Table ASCII (man ascii dans un terminal)

- 'A' est le code ASCII 65 de A ;
- '9' désigne le code ASCII 100 de 9

Les chaînes caractères

Table ASCII (man ascii dans un terminal)

- Une chaîne de caractères (string) est une séquence de caractères écrite entre guillemets
- Elle représente un tableau contigu d'octets en mémoire. Exemple : « Hello, World ! »
- '9' désigne le code ASCII 100 de 9

```
        .data
msg :     .asciz "Hello, World !\n"
```

Autres directives

- **.ascii** (chaîne de caractères sans le Zero de fin)
- **.asciz** (chaîne de caractères avec le Zero de fin)
- **.byte** (Définit et initialise un tableau d'octets)
- etc (consulter la documentation **avr-as**)

Étiquette (Labels)

- Si dans la section **.bss** ou **.data** alors ce label fait référence à une adresse mémoire
- Si dans la section **.text** alors il fait référence au compteur de programme (On peut alors utiliser l'étiquette pour se référer dans le programme)

Exemple étiquette (Labels)

```
 .data
msg :    .asciz "Hello, World !\n"
len = . - msg
.text
f1:
  /*
  code f1
  */
main:
  f1
```

Étiquettes locales

- L'assembleur GNU propose dix noms de symboles locaux de 0 à 9
- Forme « N : », avec N = 0, 1 .. 9.
- Pour se référer au dernier symbole N défini, on spécifie « Nb » comme opérande de l'instruction jmp
- Pour se référer au premier symbole N défini, on spécifie « Nf » comme opérande de l'instruction jmp
- La lettre « b » dans « Nb » est l'abréviation du mot anglais backwards. La lettre « f » dans « Nf » est celle du mot forwards.

Exemple étiquettes locales

```
1:
                ##
                jmp   1b
                jmp   1f
1:
                jmp   1b
                jmp   2f
2:
                ##
2:
```

Symbole point (.)

- Le symbole spécial « . » peut être utilisé comme une référence à une adresse au moment de l'assemblage.

```
 .data
msg :    .asciz "Hello, World !\n"
len = . - msg
```

Les expressions

- Une expression spécifie une adresse ou une valeur numérique
- Une expression entière est un ou plusieurs arguments délimités par des opérateurs
- Les arguments sont des symboles, des chiffres ou des sous-expressions
- Une sous-expression est une expression écrite entre deux parenthèses
- Les opérateurs peuvent être des préfixes ou des infixes

Les expressions

Les opérateurs préfixes prennent un argument absolu

- L'opérateur ~(tild) (complément à 1 vu en cours) complément bit-à-bit (bitwise not).
  (ex : $\sim$ 0b10101000 = 0b01010111)
- L'opérateur négation - (complément à deux) : $-N = \sim N + 1$.
  (ex : $-142 = 01110001 + 1 = 0111001$)

Les expressions

- Un opérateur de type infixe prend deux arguments :

`*`, `/`, `%`, `<`, `<<`, `>`, `>>`, `+`, `-`, `==` ...

Les opérandes

- Les registres (R0 .. R30)
- Les opérandes immédiats (Constantes)
- Les opérandes mémoire (Adresse mémoire)

L'ordre des opérandes

```
op  source,destination
```

Instruction main

- La première fonction appelé est la fonction **main** (dans la section **.text** on appel le label **main :**
- Les Labels utilisables à l'extérieur du fichier doivent être signalé par la directive **.global**
- Donc : le label **main :** doit toujours être déclaré dans une directive **.global main**

```
   .text
.global main
main:
  #code main
```

- Si vous souhaiter changer le point d'entré pour pouvez spécifier au compilateur avec l'option **-s** ou l'option longue **–enrey**

Le compilateur

- Le compilateur Assembleur Gnu se nome **avr-as**
- Les fichier assembleur compilé par **avr-as** doivent avoir une extension **.s**

Compilation

```
avr-as -g -mmcu=atmega328p -o hello.o hello.s
```

- L'option -**mmcu** permet de choisir le type du microcontrôleur
- L'option -g permet d'insérer dans le code généré les **symboles de debug**, cette option est seulement nécessaire dans le cas où vous allez utiliser le dans la suite le débogueur GDB (**avr-gdb**)

Éditeur de lien

```
avr-ld -o hello.elf hello.o
```

- Permet d'assembler un ou plusieurs fichier objet dans un seul fichier binaire.

format Intel HEX

```
avr-objcopy -O ihex -R .eeprom hello.elf hello.hex
```

- Converti dans le format Intel HEX
- Format ASCII

Téléversement du programme

```
avrdude -C /etc/avrdude.conf -p atmega328p \
-c arduino -P /dev/ttyACM0 -b 115200  \
-D -U flash:w:hello.hex:i
```

- **-C** permet de spécifier le fichier de configuration
- **-p** permet de sélectionner le microcontrôleur
- **-P** permet de sélectionner le port (dans notre cas USB)
- **-b** permet de sélectionner la vitesse (bauds)0
- **-P** permet de sélectionner le port (dans notre cas USB)
- (voir main avrdude) ...