

Programmation Baremetal sur Micro:bit v2

■ ■ ■ Découverte du Micro:bit v2 et de la programmation assembleur ARM

Vous vérifierez que vous disposez du compilateur `arm-none-eabi-gcc` :

```
xterm
$ arm-none-eabi-gcc
arm-none-eabi-gcc: fatal error: no input files
compilation terminated
```

Vous vérifierez qu'en connectant votre Micro:bit sur votre ordinateur en USB, un nouveau volume est monté et peut être ouvert dans votre interface graphique :

- ▷ pour flasher le firmware, c-à-d programmer le micro:bit, il suffit de glisser-déposer le fichier sur ce volume ;
- ▷ automatiquement, le volume disparaît et le micro:bit redémarre et exécute le firmware.

Vous récupérez les exemples écrits par Mike Spivey : https://en.wikipedia.org/wiki/Michael_Spivey

```
xterm
$ git clone https://github.com/Spivoxity/baremetal-v2.git
cd baremetal-v2
~/baremetal-v2 master $ ls
microbian  x04-numbers  x09-pureasm  x14-processes  x19-servos  x33-clock
setup      x05-subrs     x10-serial   x15-messages  x20-radio
x01-echo   x06-arrays   x11-interrupt x16-sync      x21-car
x02-instrs x07-hack     x12-intrmech x17-driver    x31-adc
x03-loops  x08-heart    x13-neopixels x18-level     x32-infrared
```

1 – Vous allez tester le premier programme du répertoire `x01-echo` :

```
xterm
$ cd x01-echo
~/baremetal-v2/x01-echo master $ ls
Makefile  echo.c  hardware.h  nRF52833.ld  startup.c
```

Vous compilerez l'exemple avec `make` et vous obtiendrez le firmware `echo.hex` que vous pourrez copier sur le micro:bit pour le flasher.

L'objet du firmware est de faire un **écho**, c-à-d de répéter tout texte transmis sur le port série.

Pour utiliser le port série vous utiliserez le programme `tio` :

```
xterm
$ /home/bonnep02/Public/tio /dev/ttyACM0 -b 9600
tio /dev/ttyACM0 -b 9600
[22:52:59.716] tio v2.6
[22:52:59.716] Press ctrl-t q to quit
[22:52:59.716] Connected

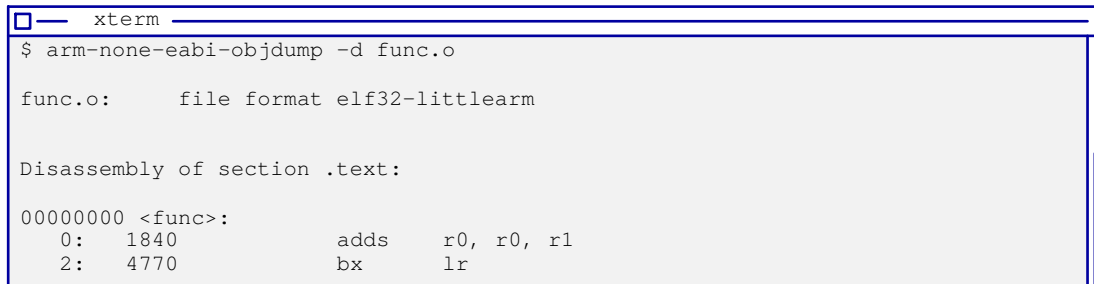
Hello micro:world!
> Cryptis
--> Cryptis
```

Étudiez le code du programme `echo.c` :

- a. À quelle adresse se trouve le périphérique UART ?
- b. Pouvez-vous passer la communication à 115200 bauds ?
- c. Comment se fait la réception d'un caractère ? Est-ce par interruption ?
- d. Modifiez le programme pour afficher le texte reçu à l'envers.

2 – On va passer au programme `x02-instrs` dont le but est de permettre l'accès à une fonction écrite en assembleur ARM depuis un programme écrit en C.

- Étudiez le contenu du fichier `fmain.c` :
 - où est définie la fonction `func` ?
 - comment sont passés les arguments entre le C et l'assembleur ? Est-ce conforme à l'ABI ?
 - comment la valeur de retour est passée à la fonction `init()` ?
- Comment est mesuré le temps d'exécution ?
- Pouvez vous modifier le programme assembleur pour augmenter la durée en réalisant plusieurs fois la somme ?
- Est-ce que le nombre de cycle augmente linéairement ?
- Utilisez l'outil de désassemblage `arm-none-eabi-objdump` :



```
xterm
$ arm-none-eabi-objdump -d func.o

func.o:      file format elf32-littlearm

Disassembly of section .text:

00000000 <func>:
 0:  1840          adds    r0, r0, r1
 2:  4770          bx      lr
```

Le résultat est-il comparable ?

3 – Notre nouveau programme à étudier est le `x03-loops`.

- Que fait-il ?
- Vous suivrez l'exécution du code assembleur donné dans le fichier `mult.s`.
- vous comparerez avec le firmware `x04-numbers`, est-ce qu'il y a des similarités ?

4 – Le programme `x05-subrs` vous propose de réaliser une factorielle en assembleur.

- Quels sont les rapports entre la fonction `func` et `mult` ?
- Est-ce que le second argument passé à `func` est nécessaire ?
- Pourquoi dans `func`, on sauvegarde les valeurs des registres :
 - `r4` et `r5` ?
 - `lr` ?

5 – Le programme `x06-arrays` vous propose d'utiliser les accès indexés à la mémoire :

- ▷ chargement d'une adresse dans un registre ;
- ▷ consultation de la valeur stockée à l'adresse indiquée dans ce registre ;
- ▷ utilisation d'un décalage par rapport à cette adresse pour accéder à des données à la manière d'un tableau.

- Ajoutez dans la fonction `init()` et juste avant l'appel à la fonction `func`, une **initialisation** des valeurs du tableau `account` défini dans le code assembleur du fichier `bank.s`.
- En vous servant de ce que vous avez appris sur les différents exercices, modifiez le firmware pour réaliser le passage des caractères d'une chaîne de **minuscule** en **majuscule**.

Vous tiendrez compte du fait que :

- une chaîne termine par un octet à zéro ;
- seul un caractère alphabétique minuscule peut être décalé en caractère alphabétique majuscule.