Outils de compilation croisés pour le processeur ARM

Philippe Waille

1 Localisation des outils ARM

Sur mandelbrot, les binaires (liés statiquement) des outils pour ARM sont stockés dans l'arborescence de fichiers **/opt/gnu/arm**. Les sources sont dans **/opt/gnu/src/arm**.

Sur votre pc sous linux (avec un noyau pas trop ancien), vous pouvez simplement installer ces répertoires à partir des archives **opt_gnu_arm.tgz** et **opt_gnu_src_arm.tgz** (extraction par **tar xvzf archive.tgz**).

Vous pouvez ensuite utiliser ces commandes en mode manuel ou installer l'environnement contenu dans envarm.tar pour utiliser les commandes simplifiées.

2 Compilation et exécution en mode manuel

Vous préférez certainement installer envarm et utiliser les commandes de compilation simplifiées. Dans le cas contraire, voici les outils et les options spécifiques à utiliser :

- as (assembleur) : **/opt/gnu/arm/bin/arm-elf-as –gdwarf2**
- gcc (compilateur) : **/opt/gnu/arm/bin/arm-elf-gcc -gdwarf-2**
- (Exécuter avec le programme) simulateur de ARM :
  /opt/gnu/arm/bin/arm-elf-run votre_exécutable liste_arguments
- gdb (mode textuel) : **/opt/gnu/arm/bin/arm-elf-gdb**
- ddd (mode graphique) : **ddd –gdb –debugger /opt/gnu/arm/bin/arm-elf-gdb**

3 Installation et configuration de envarm

Il s'agit de créer un répertoire de configuration des outils ARM **.envarm** dans un répertoire *ARM_ENVDIR* de votre choix (par défaut, choisisser votre HOMEDIR : *ARM_ENVDIR* = **~/.envarm**).

3.1 Installation initiale (une seule fois à la première séance)

Récupérez le fichier envarm.tar dans le placard (via le WEB ou en le copiant sur hopper), puis désarchivez dans le répertoire père de *ARM_ENV_DIR*. Dans *ARM_ENV_DIR*, lancer **make**.

```
pcnt35> cd ~; tar xvzf  envarm.tgz
pcnt35> cd ~/.envarm; make
```

3.2 Configuration (à chaque séance)

Utilisez la commande **ps** pour vérifier quel type de shell vous utilisez (sh/bash ou csh/tcsh). Les fichiers **setenvarm.csh** et **setenvarm.sh** contiennent une liste d'instructions à appliquer avec la commande **source**.

La commande source est à lancer à **chaque séance** et dans **chaque fenêtre**[1] utilisée.

```
pcnt35> source ~/.envarm/setenvarm.csh   (ou setenvarm.sh pour bash)
```

La liste des variables d'environnement et des pseudos-commandes nouvellement définies est affichée.

Passez directement à la section décrivant les commandes simplifiées.

3.3 En cas de problème d'accès à votre HOMEDIR

Si le montage des espaces de fichiers des utilisateurs sur les pc locaux pose problème, créez un répertoire à votre nom (exemple : martinp) dans un système de fichier local accessible à tout le monde (/tmp), définissez la variable HOME en conséquence, puis créez dans ce dernier les répertoires *ARM_ENVDIR* et *TP_ARM*.

Suivez ensuite la procédure d'installation initiale, puis recompilez (**make clean all**) tous vos fichiers dans **~/ALM/TP_ARM** . En fin de séance, n'oubliez pas de recopier le contenu de **~/ALM/TP_ARM** sur une clé usb ou sur votre espace de travail sur un serveur. Scp permet de copier dans les deux sens.

```
pcnt35> mkdir /tmp/martinp /tmp/martinp/ALM /tmp/martinp/ALM/TP_ALM
pcnt35> setenv HOME /tmp/martinp          // uniquement pour csh/tcsh
pcnt35> HOME = /tmp/martinp; export HOME  // uniquement pour sh/bash
pcnt35> ssh hopper
...  // suite de la procédure normale d'installation initiale

pcnt35> scp -rv hopper:ALM/TP_ARM ~/ALM   // recopie serveur vers pc
pcnt35> scp -rv ~/ALM/TP_ALM hopper:ALM/TP_ARM   //recopie pc vers serveur)
```

4 Compilation et exécution (commandes simplifiées)

La commande **source setenvarm.csh** définit de nouvelles commandes (dont arms, armgcc, armrun, armgdb et armddd). La commande **alias** liste l'ensemble des nouvelles commandes ainsi définies.

La commande **armgdb** exécute **arm-elf-gdb** et lance une suite de commandes adaptées à l'exécution d'un programme sur le simulateur ARM. La commande **armddd** permet de lancer l'équivalent une session graphique ddd avec le même environnement.

Ces commandes vérifient le type du fichier exécutable lancé et lancent automatiquement la séquence de commandes gdb (target, load, break, run) à votre place.

[1] Plus précisément dans chaque session shell

4.1 Test de l'environnement

Après avoir exécuté la commande **source setenvarm.csh** (voir section 3.2), allez dans le sous-répertoire **DEMO_TP** de **~/.envarm** .

Compilez le programme echo.c puis lancez l'exécution sans puis avec gdb.

```
mandelbrot> source ~/.envarm/setenvarm.csh
   ...   /* voici les deux dernières lignes du message de réponse */
   simgcc  (/opt/gnu/arm/bin/arm-elf-gcc -Wall -gdwarf-2)
   simgdb  /users/huron/waille/.envarm/simgdb
mandelbrot> cd ~/.envarm/DEMO_TP
mandelbrot> armgcc echo.c -o echo
mandelbrot> armrun echo 123 456 78
mandelbrot> armgdb --args echo 123 456 78
                  /* messages divers affichés par gdb */
...
(gdb) n
      ...         /* autant de next ou step que vous voulez */
(gdb) n
(gdb) c
                  /* echo affiche les argv */
(gqb) quit
```

5 Quelques commandes utiles de gdb/ddd

Voici quelques commandes gdb utiles :

- **Stepi** (**si** ou si 5) et **Nexti** (**ni** ou ni 5) avancent d'une (ou de 5) instruction(s) machine. **Step**(**s** ou s 2) et **Next** (**n** ou n 2) avancent d'une (ou deux) ligne(s) dans le fichier source (généralement 1 ligne de C). En cas d'appel de procédure, step détaille l'exécution du corps, et next exécute l'ensemble de la procédure.
- **Cont** (**c**) exécute en continu jusqu'au prochain point d'arrêt.
- **Finish** (**c**) exécute en continu jusqu'au retour de la procédure en cours.
- Changer le contenu d'une variable en mémoire : **set *&x = valeur**
- Passer de la base 10 à 16, puis de 16 à 10 (pour une commande set) : **set input-radix 16 ; set input-radix a**.
- Lire le contenu de 6 mots en hexadécimal à partir de l'adresse 0x1000 : **x /6x 0x1000**.
- Désassembler 3 instructions à partir de l'adresse contenue dans pc **x /3i $pc**.
- Afficher le contenu de la variable x : **printf "%d\n", x**
- Afficher les informations à chaque arrêt de l'exécution au lieu d'une seule fois : utiliser display (ou graph display sous ddd) au lieu de print.

La commande unix **info gdb** donne des détails sur gdb.

Sous ddd, on peut aussi utiliser un affichage graphique de n emplacements consécutifs **(Menu Data → Memory)**. Préciser le nombre d'éléments affichés (Examine), le format (décimal, hexa), la taille d'un élément (octet, demi-mot ou mot 32 bits) et l'adresse du premier élément(from), puis Display. Note : **10,instruction,words,&main** désassemble le début de

main. L'affichage est mis à jour à chaque arrêt de l'exécution ou à la demande (**Menu Data → refresh**). Pour supprimer un affichage, cliquer dessus puis sur **Undisp**.

6 Printf de débogage : insérés avec gdb, pas dans le programme !

La (mauvaise) méthode consistant à truffer le code du programme d'appels à printf pour tracer l'exécution et le mettre au point est à oublier. Outre la possibilité d'exécuter pas à pas un programme, et d'afficher l'arborescence des appels de procédure, gdb permet de mettre des points d'arrêts à certains endroits d'un programme.

La commande break retourne un numéro, auquel il est possible d'associer une liste de commandes gdb à effectuer automatiquement lorsque l'exécution atteint le point d'arrêt. Si la liste de commandes associée à un point d'arrêt se termine par cont, l'exécution sera automatiquement relancée jusqu'au prochain point d'arrêt.

```
(gdb) break facto.c:10
Breakpoint 4 at 0x34dc: file facto.c, line 10.
(gdb) commands 4
Type commands for when breakpoint 4 is hit, one per line.
End with a line saying just "end".
>printf "multiplication : f=%d i=%d\n",f,i
>end
(gdb) cont
Breakpoint 4, facto (n=7) at facto.c:10
10          f = f*i;
multiplication : f=1 i=1
(gdb) cont
Breakpoint 4, facto (n=7) at facto.c:10
10          f = f*i;
multiplication : f=1 i=2
(gdb) commands 4
Type commands for when breakpoint 4 is hit, one per line.
End with a line saying just "end".
>printf "multiplication : f=%d, i=%d\n", f,i
>cont
>end
(gdb) cont
Breakpoint 4, facto (n=7) at facto.c:10
10          f = f*i;
multiplication : f=1, i=1
Breakpoint 4, facto (n=7) at facto.c:10
10          f = f*i;
multiplication : f=1, i=2
...
Breakpoint 2, _shutdown () at /users/huron/waille/.envarm/SRC/reset.s:142
```