



UNIVERSITÉ  
CÔTE D'AZUR

# Introduction aux Systèmes et Logiciels Embarqués

**Présentation: Stéphane Lavirotte**

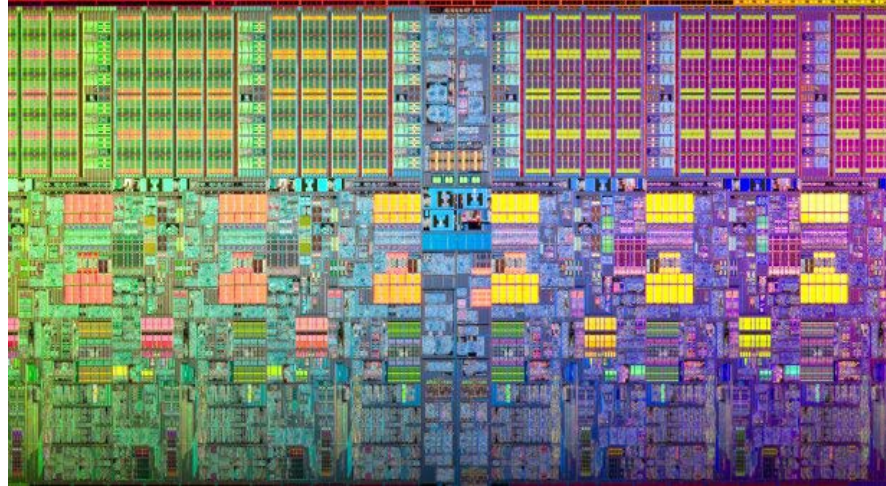
**Auteurs: ... et al\***

(\*) Cours réalisé grâce aux documents de :  
Stéphane Lavirotte

**Mail: [Stephane.Lavirotte@univ-cotedazur.fr](mailto:Stephane.Lavirotte@univ-cotedazur.fr)**

**Web: <http://stephane.lavirotte.com/>**

**Université Côte d'Azur**



# Architecture microprocesseur

Rappel de grands principes  
pour savoir ce que vous manipulez



# Caractéristiques

- ✓ **Caractéristiques d'un microprocesseur**
  - Le jeu d'instructions qu'il peut exécuter
    - Additionner, multiplier, comparer, appeler une fonction, ...
  - La complexité de son architecture
    - Nombre de transistors (finesse de gravure) – [Loi de Moore](#)
  - Le nombre de bits que le processeur peut traiter ensemble
    - 8, 12, 16, 32 ou 64 bits
  - La vitesse de l'horloge
    - Cadencer le rythme de travail du microprocesseur
- ✓ **Toutes les combinaisons sont possibles**
  - Des dizaines de familles de microprocesseurs
- ✓ **Plusieurs grandes familles**



# Jeu d'instructions : CISC vs RISC

- ✓ **Comparaison CISC - RISC:**
  - **CISC: *Complex Instruction Set Computer***
    - Faire une tâche complexe en un minimum d'instructions assembleur
    - Mise sur le matériel (hardware) pour faire des tâches de plus en plus complexe (tâches plus longues à exécuter)
    - x86, Pentium (Intel), m68k (Motorola)
  - **RISC: *Reduced Instruction Set Computer***
    - Jeu d'instructions simples qui peuvent être effectuées en un cycle
    - Mise sur le logiciel (software) pour enchaîner très vite les instructions. Plus d'instructions pour réaliser une tâche complexe
    - Alpha (DEC), PowerPC (Motorola), MIPS, Sparc
- ✓ **La différence est donc dans le jeu des instructions**
  - Donc le compilateur doit générer du code différent



# Représentation des données: Endianness

- ✓ De nombreuses données utilisent plusieurs octets
  - Entiers, flottants
- ✓ Endianness
  - L'ordre dans lequel les octets sont rangés en mémoire ou lors de la communication
- ✓ Orientation
  - Soit le nombre 0xA0B70708, soit 4 octets A0 B7 07 08
  - **Big-endian** (Motorola 680x0 donc MacOS)
    - l'octet de poids le plus fort est enregistré à l'adresse mémoire la plus petite (donc A0 dans notre exemple)
  - **Little-endian** (Intel x86 et Pentium donc Windows)
    - l'octet de poids le plus faible en premier (donc 08 dans l'exemple)
  - Certaines architectures supportent les deux modes
    - C'est le cas de ARM ou IA64 qui sont **bi-endian**
    - Le choix se fait alors logiciellement, matériellement ou les deux



# Compilation Croisée

Produire du code pour un autre processeur



# Rappels sur la Compilation

## ✓ Etape 1: Pré-processeur

- Les fichiers source (.c/.cpp/...) sont lus par le **préprocesseur**
- Traite les commandes #XXX (#include, #define, #ifdef...)
- Produit un fichier source où les directives ont été exécutées

## ✓ Etape 2: Compilateur

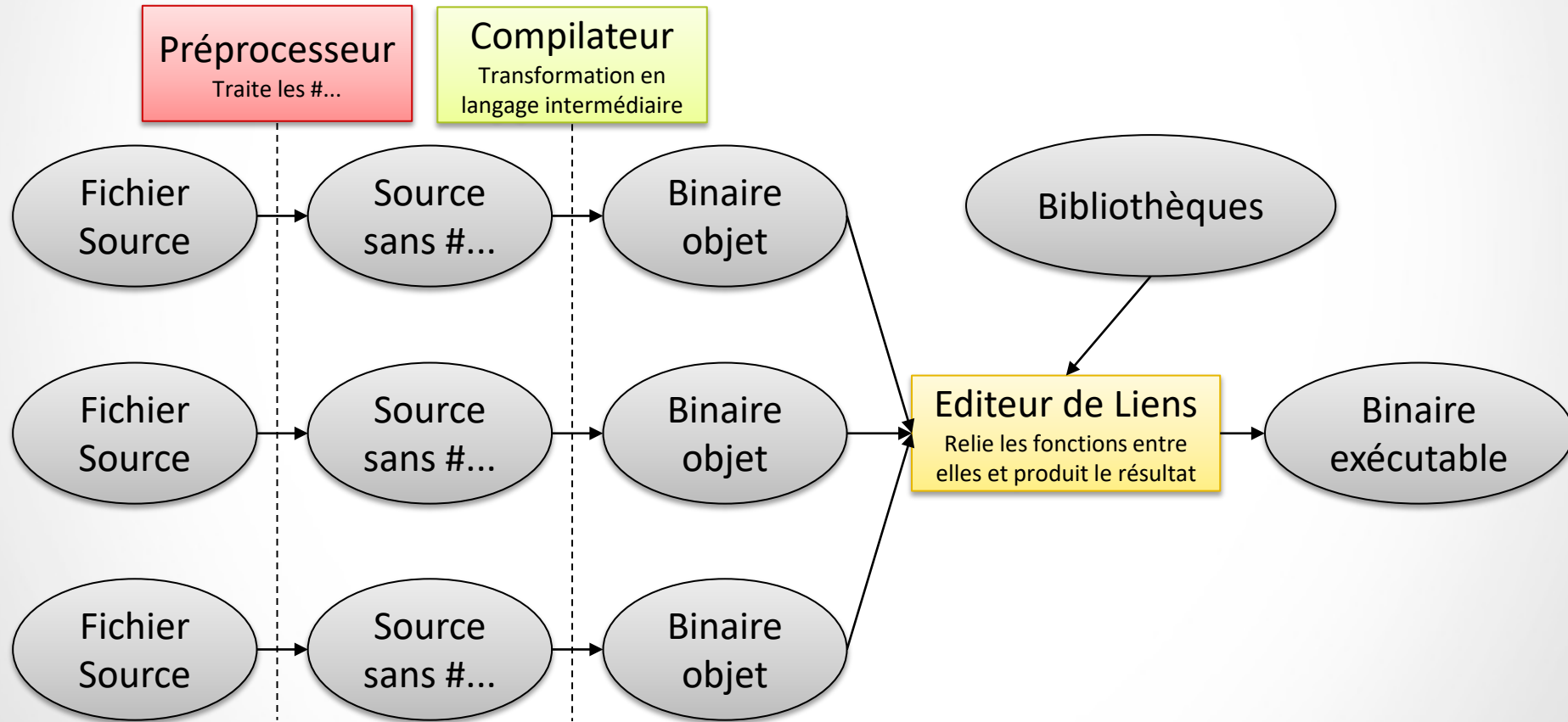
- Transforme le source en un fichier intermédiaire (.o/.obj), compilé mais non exécutable
- Le langage utilisé est spécifique au compilateur
- Pour valider votre code, le compilateur a besoin de connaître tous les symboles que vous utilisez
- En C/C++, les fichiers .h/.hpp permettent de déclarer les symboles au compilateur et donc de lui dire : « cette fonction existe, voici les paramètres qu'elles acceptent, le type de valeur qu'elle retourne »

## ✓ Etape 3: Editeur de Liens

- Prend les fichiers objets du programme pour les assembler
- Produit un exécutable ou une bibliothèque



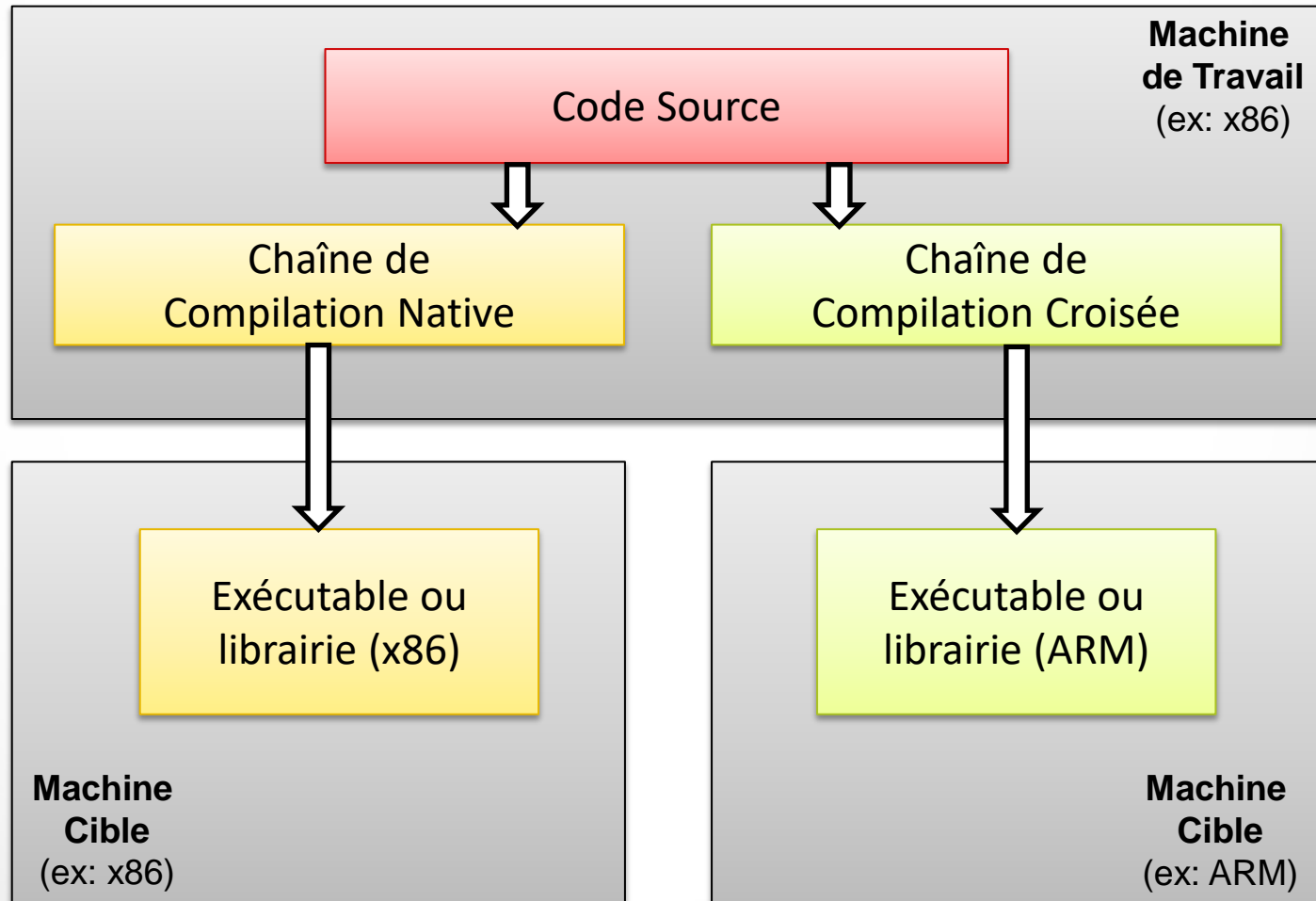
# Etapes de la Compilation de Code Source







# Compilation Croisée




# Compilation pour une Autre Architecture

## ✓ Compilation Croisée

- Plus rapide sur la station de travail que sur le système cible
- Plus facile d'avoir un environnement de développement sur la station de travail que sur la cible
  - Problème de place sur la cible
  - Problème de ressources sur la machine cible (écran, taille, clavier, souris, ...)
- Outils de compilation croisée préfixés par le nom de l'architecture
  - ABI: Application Binary Interface
  - EABI: Embedded Application Binary Interface
- Par exemple:
  - `arm-unknown-linux-uclibcgnueabi-ar`
  - `arm-unknown-linux-uclibcgnueabi-gcc`
  - `arm-unknown-linux-uclibcgnueabi-ld ...`



## Compilation Croisée du Noyau

- ✓ L'architecture du CPU et l'outil de compilation croisée sont défini dans le Makefile de premier niveau
  - Défini les variables `ARCH` et `CROSS_COMPILE`
- ✓ Le Makefile définit:
  - `CC=$(CROSS_COMPILE)gcc`
- ✓ La solution pour cross-compiler est de redéfinir les variables:
  - Exemple pour une architecture ARM
    - Pour le compilateur `arm-unknown-linux-uclibcgnueabi-gcc`
    - `ARCH=arm`
    - `CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi-` 
- ✓ Ou redéfinir ces variables:
  - Dans le Makefile de premier niveau (non recommandé!)
  - OU sur la ligne de commande à l'appel de la commande `make`
    - Attention à bien se souvenir des paramètres d'appel (`make ARCH=...`)
    - Attention à bien ajouter les variables à chaque appel à `make` !
  - OU Redéfinition des variables d'environnement
- ✓ Puis, ajouter la chaîne de compilation croisée au `PATH`

# Exemple: Compilation Croisée de BusyBox

- ✓ **Même approche que pour la cross-compilation du noyau**
  - Soit définir les variables à l'appel de la commande make
    - `make ARCH=... CROSS_COMPILE=...`
  - Soit définir les variables d'environnement dans le terminal
    - `export ARCH=arm`
    - `export CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi-`
  - Et ajouter le cross compilateur dans le chemin d'exécution
    - `export PATH=$PATH:/opt/x-tools/...`
- ✓ **Compilation et Installation de BusyBox**
  - Soit `make ARCH=... CROSS_COMPILE=... install`
  - Soit `make install` (si variables d'environnement redéfinies)
- ✓ **Il ne reste plus qu'à copier les fichiers sur le fs cible**
  - `cp -a _install/* /mnt/`



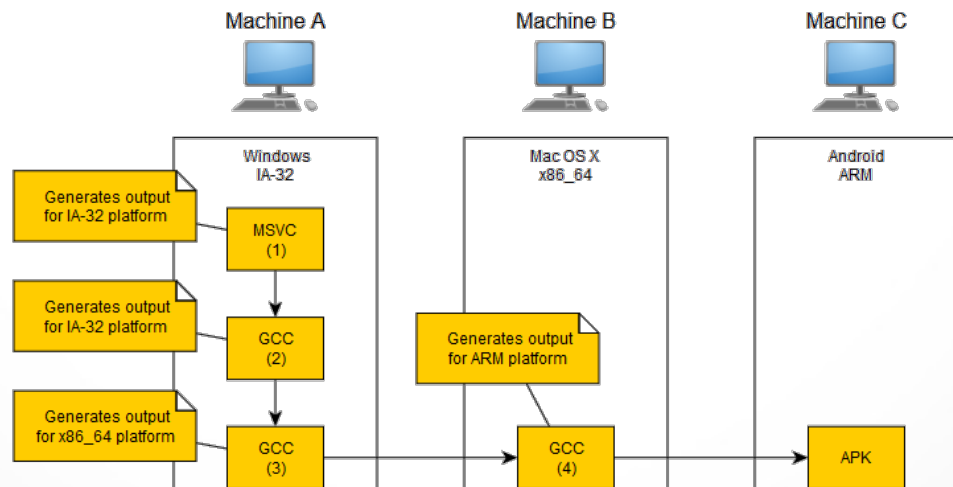
# Compilation Croisée : Critères de Choix

Bien choisir sa chaine de compilation  
croisée



# Terminologie: Canadian Cross

- ✓ **Trois machines sont à distinguer pour la création d'une chaîne de compilation croisée:**
  - La machine de construction (« build »)
    - La machine où sera construite la machine de compilation
  - La machine hôte (« host »)
    - La machine sur laquelle sera exécutée la compilation
  - La machine cible (« target »)
    - La machine pour laquelle les binaires sont créés



- ✓ **Les machines de construction et hôte sont souvent les mêmes**

# Eléments d'une Chaîne de Compilation Croisée

- ✓ **Le(s) compilateur(s)**
  - GCC est l'outil pour Linux embarqué
  - Support pour C, C++, Java, Fortran, Objective-C, Ada
- ✓ **Les « Binutils »**
  - Les utilitaires importants comme par exemple:
    - `as`: qui transforme le code généré par le compilateur en binaire
    - `ld`: qui regroupe les codes objets en bibliothèques ou exécutable
- ✓ **Le débogueur (optionnel)**
  - Pour déboguer les applications s'exécutant sur la cible
- ✓ **La bibliothèque C**
  - `glibc` ou `μClibc-ng`, ...
- ✓ **Les entêtes du noyau (API) pour compiler la libC**



# Exemple d'incompatibilités gcc / binutils / glibc / noyau

- ✓ **Tableau Récapitulatif**
  - Des version de gcc / glibc / binutils / noyau
  - En fonction des architectures
- ✓ **Pour vous aider à faire un choix...**



- ✓ <http://kegel.com/crosstool/crosstool-0.43/buildlogs/>





# Choix d'une Chaîne de Compilation Croisée

- ✓ Trouver une chaîne de compilation croisée est une activité difficile
  - Nombreux composants à compiler (le compilateur y compris)
  - Choix à faire
    - Version du compilateur, du noyau, de la librairie C, des outils, ...
  - Nombreux détails à connaître. Etre familier:
    - Des versions de `gcc`, de leurs différences et patch pour votre architecture
    - De la configuration et la compilation (noyau, outils, ...)
  - Etre sûr que la chaîne correspond à ces besoins
    - CPU, little ou big endian, version des librairies, des outils, ...
  - 26 pages de HowTo pour mettre en place une chaîne pour ARM:
    - <http://simplemachines.it/doc/toolchHOWTO.pdf>

# Chaînes de Compilation Croisée Prêtes à l'Emploi

## ✓ ARM

### – Sourcery CodeBench:

- Supporte aussi les architectures PowerPC, ColdFire, ...
- Disponible pour plateforme EABI, GNU/Linux, Windows
- <http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/overview/>

### – Linaro:

- Supporte les architecture ARM récentes: Cortex A8, A9, ...
- <http://www.linaro.org/>

### – DENX ELDK

- Environnement de développement pour embarqué et RT
- Supporte aussi PowerPC, MIPS, ...
- <http://www.denx.de/wiki/DULG/ELDK>

## ✓ MIPS

- <http://www.linux-mips.org/wiki/Toolchains>

# Chaîne de Compilation Croisée à Construire

- ✓ **Cross-compileur uniquement**
  - Crosstool-ng
    - Supporte de nombreuses architectures
      - Alpha, ARM, microblaze, MIPS, OpenRISC, PowerPC, x86, ...
    - <http://crosstool-ng.org/>
- ✓ **Cross-compileur et une création d'un FS**
  - Buildroot
    - Système de configuration identique au noyau Linux
    - Peut inclure de nombreux applicatifs dans l'image construite
    - <http://buildroot.org/>
  - OpenADK
    - Similaire à Buildroot
    - <http://www.openadk.org/>



# Crosstool-ng: Construction

- ✓ **Constructeur de Chaîne de Compilation croisée**
- ✓ **<http://crosstool-ng.org/>**
- ✓ **A Partir des sources de crosstool-ng**
  - `./configure --prefix=/... ou --enable-local`
  - `make ou make install`
- ✓ **Génère l'outil ct-ng qui permet de généré la configuration et la compilation**
  - `ct-ng help`
  - `ct-ng list-samples`
  - `ct-ng arm-unknown-linux-uclibcgnueabi`
  - `ct-ng menuconfig`
  - `ct-ng build`
- ✓ **Et vous avez maintenant le temps de prendre un café**



# Crosstool-ng: Utilisation

- ✓ **Vous disposez maintenant d'une chaîne de compilation croisée pour ARM**
- ✓ **Ajouter le dossier à votre variable PATH:**
  - `PATH=$PATH:chemin_vers_le_bin_de_la_toolchain`
- ✓ **Compilation d'un programme C pour tester:**
  - **Pour un exécutable lié dynamiquement**
    - `arm-unknown-linux-uclibcgnueabi-gcc hello.c -o hello`
  - **Pour un exécutable lié statiquement**
    - `arm-unknown-linux-uclibcgnueabi-gcc -static hello.c -o hellostatic`
  - **Pour visualiser le type d'exécutable**
    - `file hello`
- ✓ **Maintenant vous pouvez compiler votre noyau et les outils systèmes pour une autre architecture !**



# Conclusion

- ✓ **Construire une chaîne par soi-même**
  - Difficile et long à maîtriser
  - A éviter!
- ✓ **Chaîne prête à l'emploi**
  - Disponible pour de nombreuses plates-formes...
  - ...Mais pas toutes
- ✓ **Outils pour la construction de chaînes de compilation**
  - [Crosstool-ng](#) et [Buildroot](#)
  - Simplifie la création pour des besoins précis
  - Fournit une souplesse dans le choix des éléments
- ✓ **Ressource utile: [http://elinux.org/Tool\\_Chain](http://elinux.org/Tool_Chain)**