

Sistemas Operacionais Embarcados

Buildroot

Conteúdo

- Buildroot
 - Introdução
 - Requerimentos
 - Download
- Criação de um sistema Linux
 - Toolchain
 - Bootloader
 - Kernel
 - Rootfs
- Criação de um sistema mínimo Linux
- Buildroot *Hello world*
- Referências

Buildroot - Introdução

- Buildroot é uma ferramenta que simplifica e automatiza o processo de montar um sistema Linux completo para um sistema embarcado, usando compilação cruzada (*cross-compilation*)
- Outras opções são o OpenWRT Project e o Yocto Project

Buildroot - Introdução

- O Buildroot gera:
 - Uma *toolchain* de *cross-compilation*
 - O sistema de arquivos *root*
 - Uma imagem do kernel do Linux
 - Um bootloader para o sistema embarcado
- Você pode escolher executar somente a parte destes resultados que te interessar.

Buildroot - Introdução

- O Buildroot é importante no desenvolvimento de sistemas embarcados porque vários destes não utilizam processadores x86, tais como processadores PowerPC, MIPS, ARM etc.
- Além de atender a diversos processadores, ele também oferece opções prontas para diversas placas

Buildroot - Requerimentos e *Download*

- Instale as ferramentas necessárias:

```
$ sudo apt-get install build-essential libncurses5-dev bzip2  
cvs git mercurial rsync subversion
```

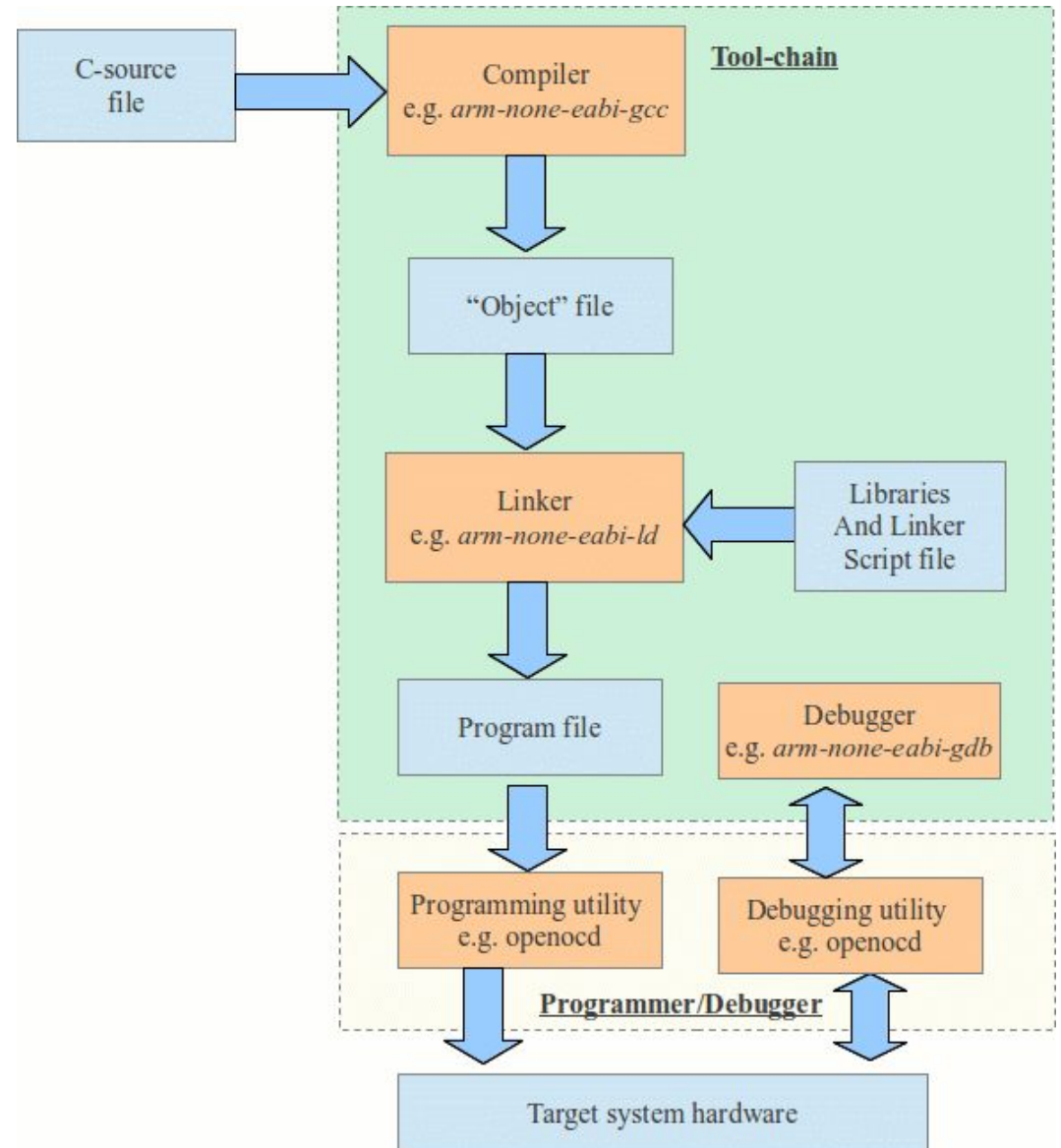
- Faça o *download* do Buildroot no repositório <https://github.com/buildroot/buildroot> ou em <http://buildroot.org/download.html>
- O código do Buildroot é atualizado a cada 3 meses, em fevereiro, maio, agosto e novembro. As versões são indicadas pelo formato YYYY.MM (por exemplo, 2013.02, 2014.08 etc.).

Criação de um sistema Linux

- Elementos básicos:
 - *Toolchain*
 - *Bootloader*
 - *Kernel*
 - *Rootfs*

Toolchain

- Conjunto de ferramentas de desenvolvimento de *software*

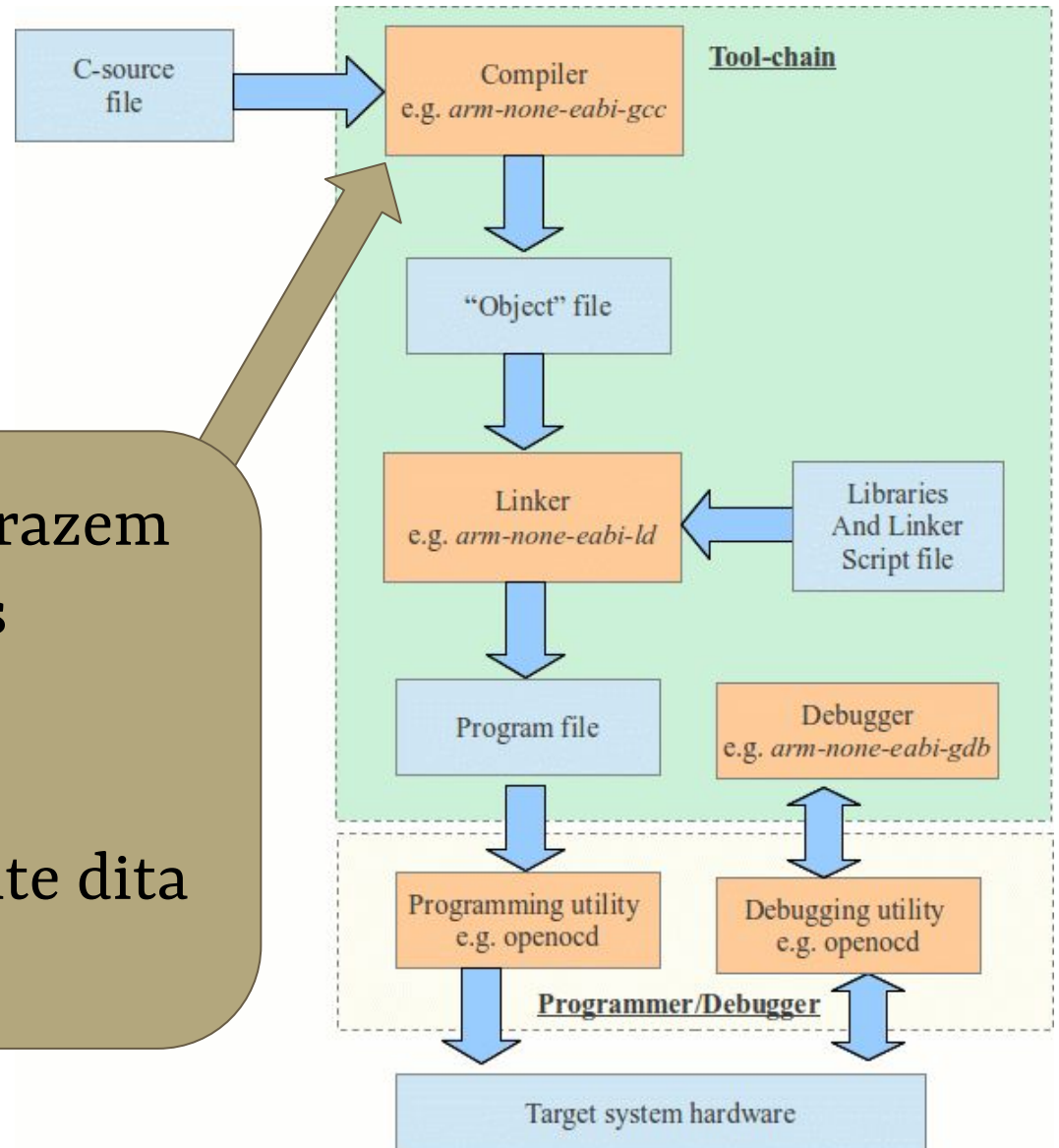


Toolchain

- Conjunto de ferramentas de desenvolvimento de *software*

Os compiladores modernos trazem na verdade 3 funcionalidades básicas:

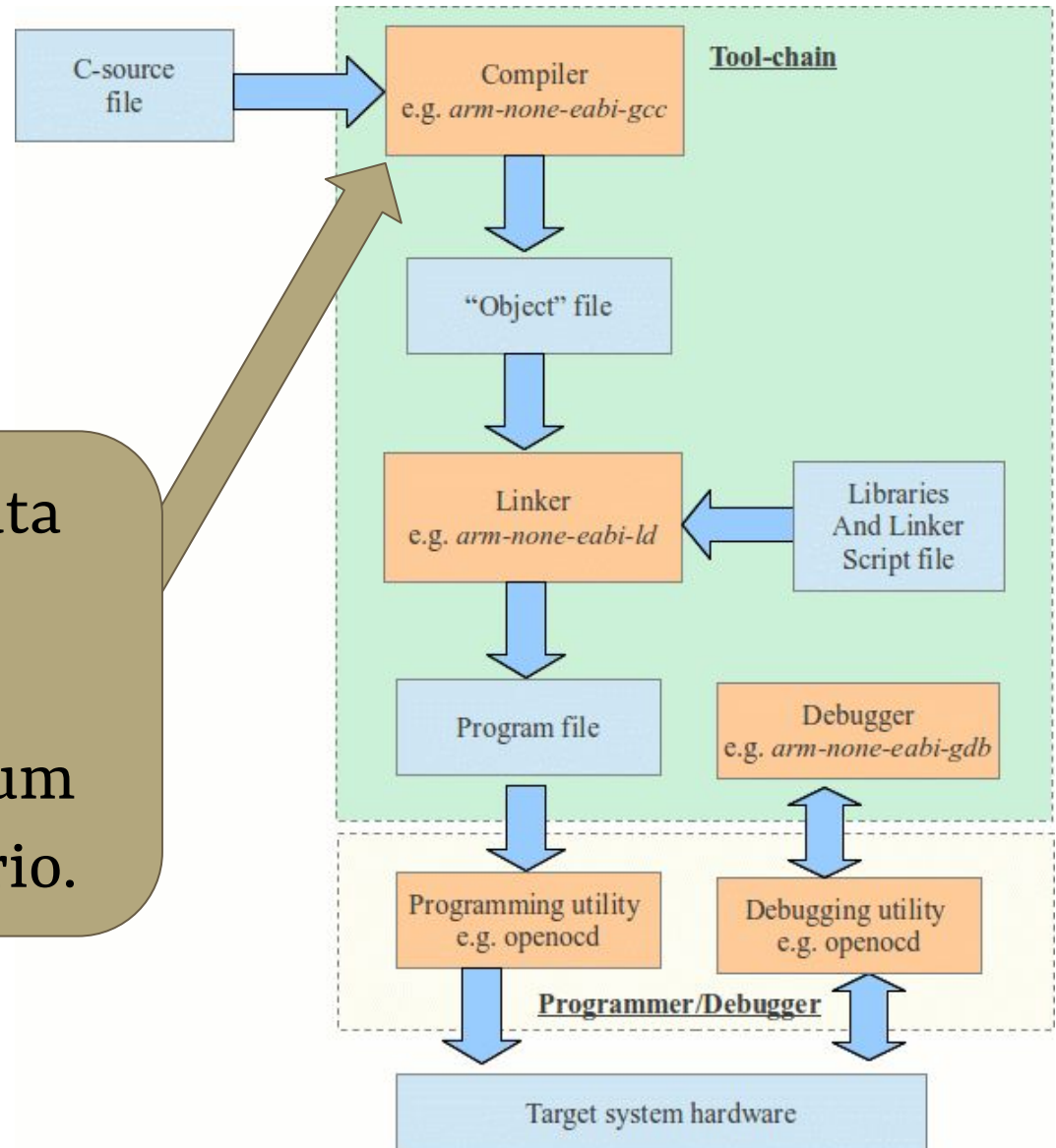
- Pré-processamento
- A compilação propriamente dita
- O *assembler*



Toolchain

- Conjunto de ferramentas de desenvolvimento de *software*

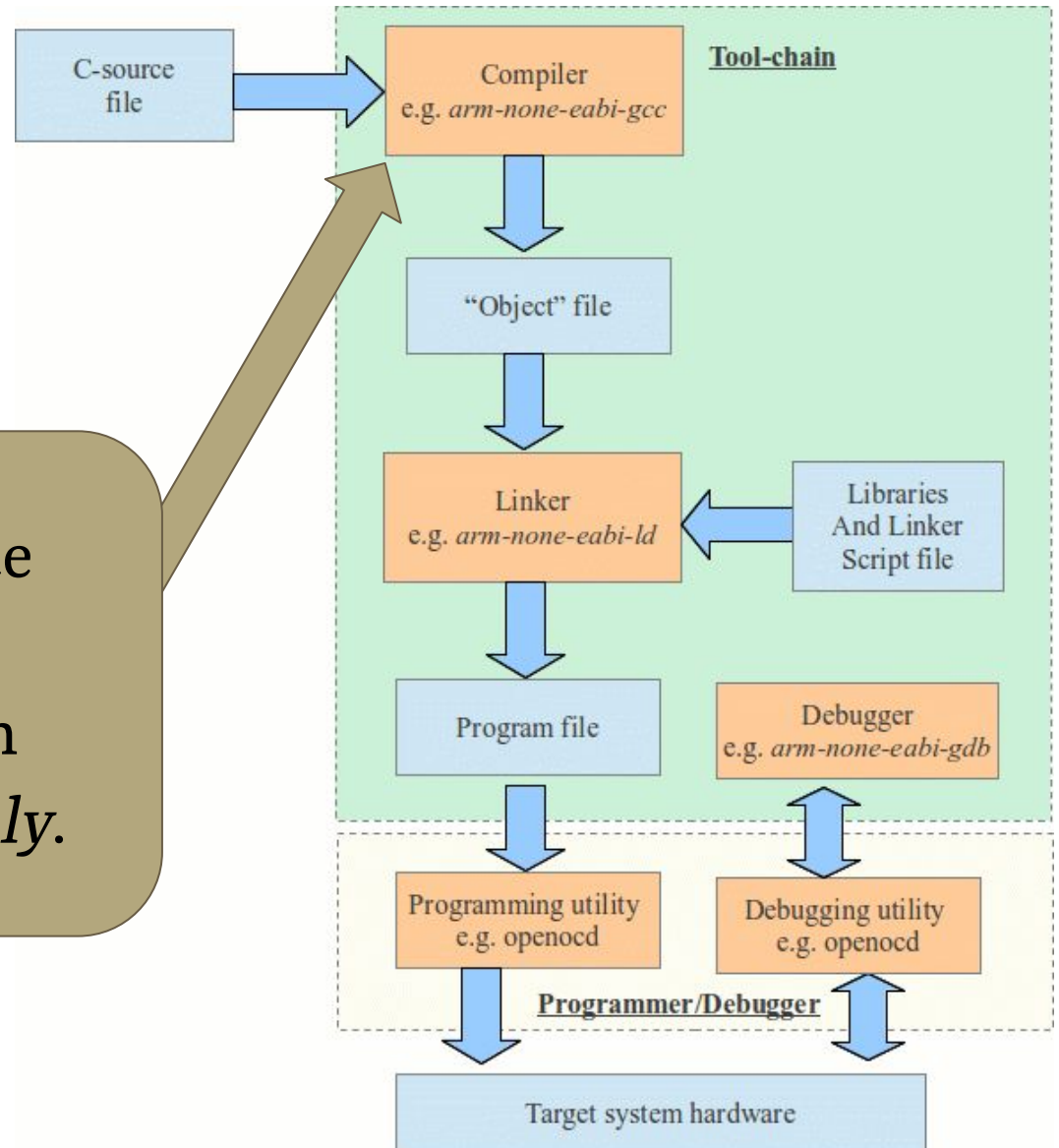
Pré-processador: trata todas as diretivas de pré-processamento (#define etc.) e gera um código C intermediário.



Toolchain

- Conjunto de ferramentas de desenvolvimento de *software*

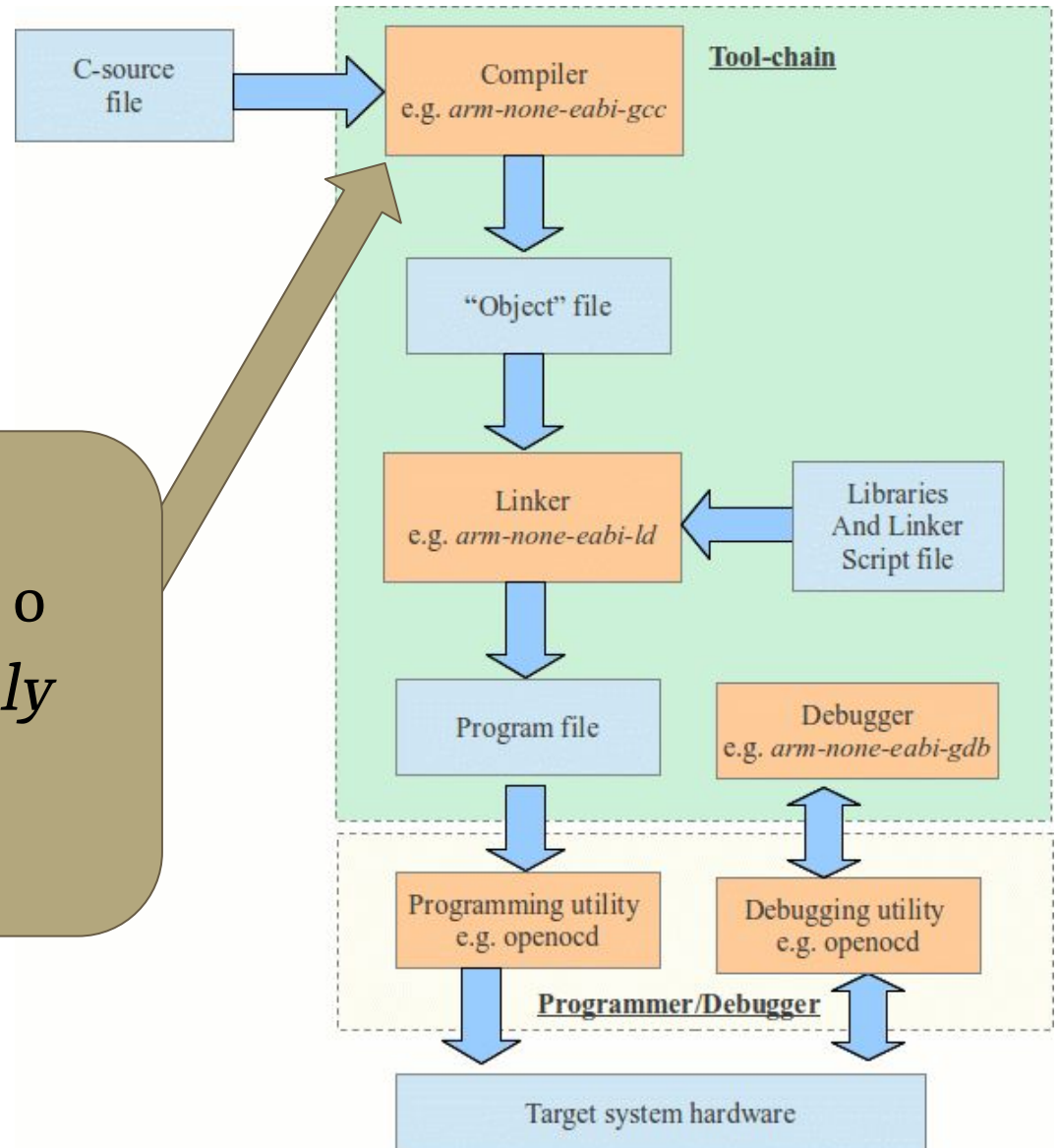
Compilador: converte este código C intermediário em um código-fonte *assembly*.



Toolchain

- Conjunto de ferramentas de desenvolvimento de *software*

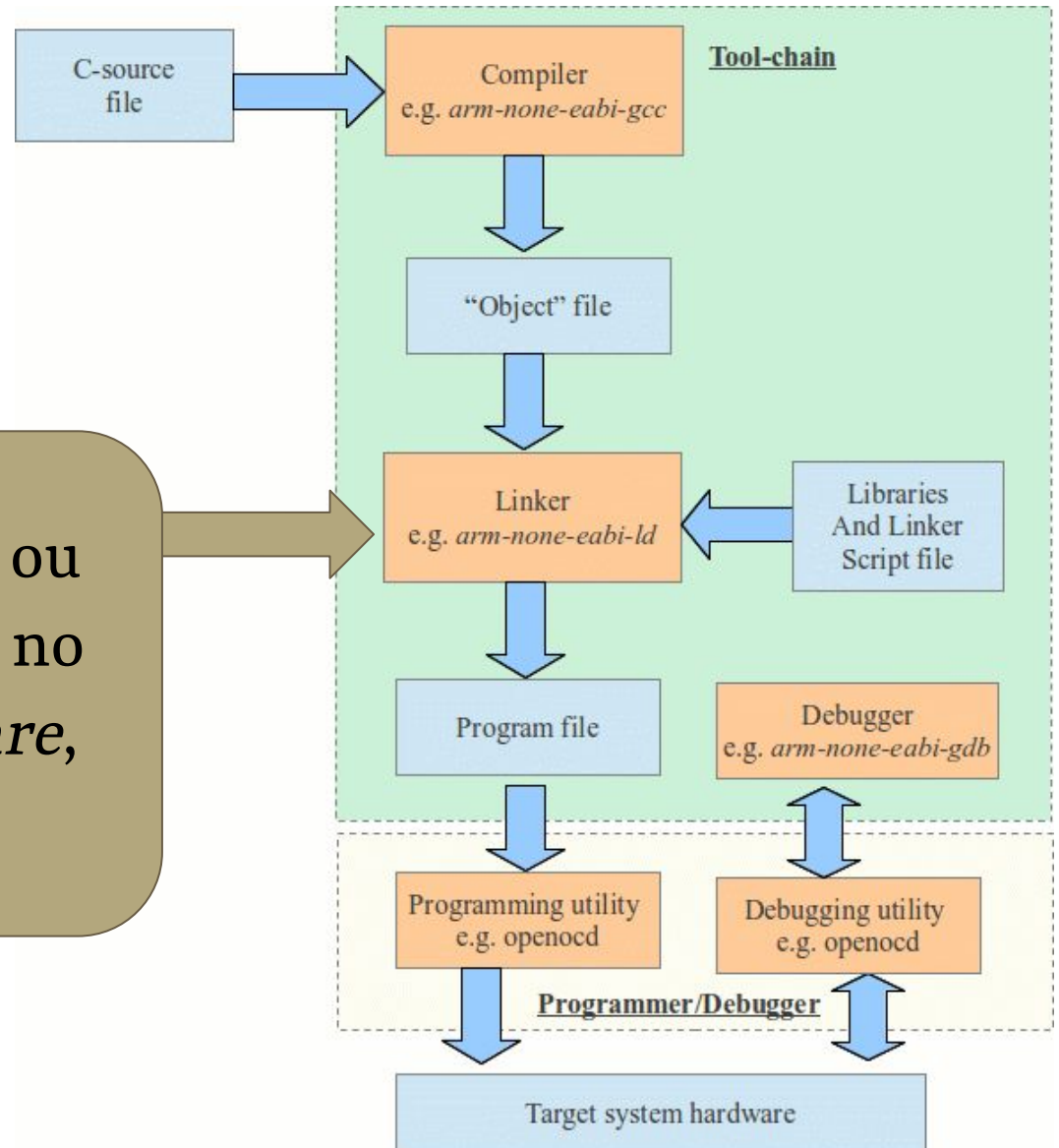
Assembler: converte o código-fonte *assembly* em arquivo objeto



Toolchain

- Conjunto de ferramentas de desenvolvimento de *software*

Linker: converte um ou mais arquivos objeto no binário final (*firmware*, aplicação, etc).



Toolchain

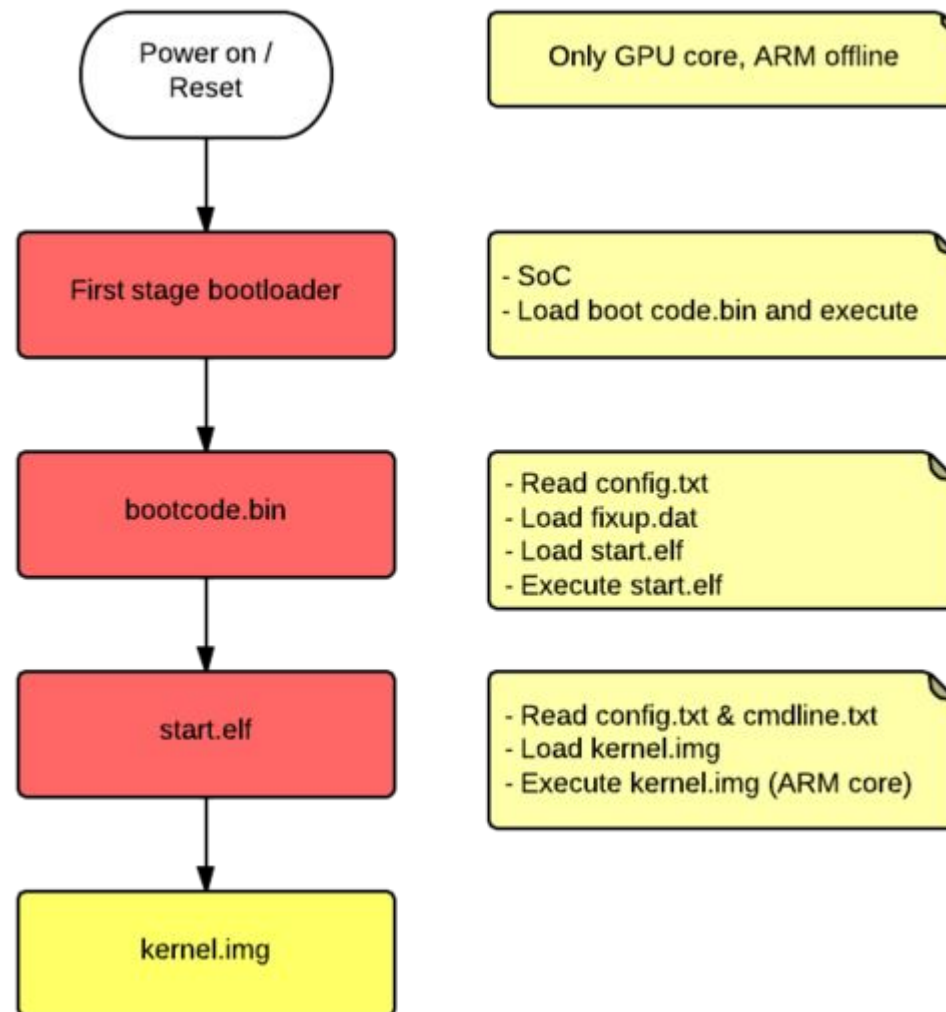
- Uma *toolchain* pode ser:
 - Nativa (*native toolchain*): compila código para ser executado em um processador da mesma família
 - De compilação cruzada (*cross-compiling toolchain*): compila código para ser executado em outra família de processadores, economizando recursos no dispositivo-alvo (por exemplo, evitando a instalação do GCC no Raspberry Pi).
- A *toolchain* de compilação cruzada para o Raspberry Pi está disponível em <https://github.com/raspberrypi/tools>, mas não se preocupe em baixa-la agora.

Bootloader

- Quando uma CPU é energizada inicialmente, ela não possui código para executar na sua memória principal. O *bootloader* é um sistema que fornece este código inicial.
- O termo *boot* vem de *bootstrap load*, que por sua vez vem da frase "*to pull oneself up by one's bootstraps*", ou "se levantar pelas próprias alças das botas". É uma variante do paradoxo do ovo e da galinha, implicando que é impossível a CPU iniciar sua execução sem código para iniciar sua execução. Por isso, é necessário algum *hardware* externo ou *firmware* para iniciar o processo de *boot*.

Bootloader


- O processo de boot no Raspberry Pi segue estes passos:



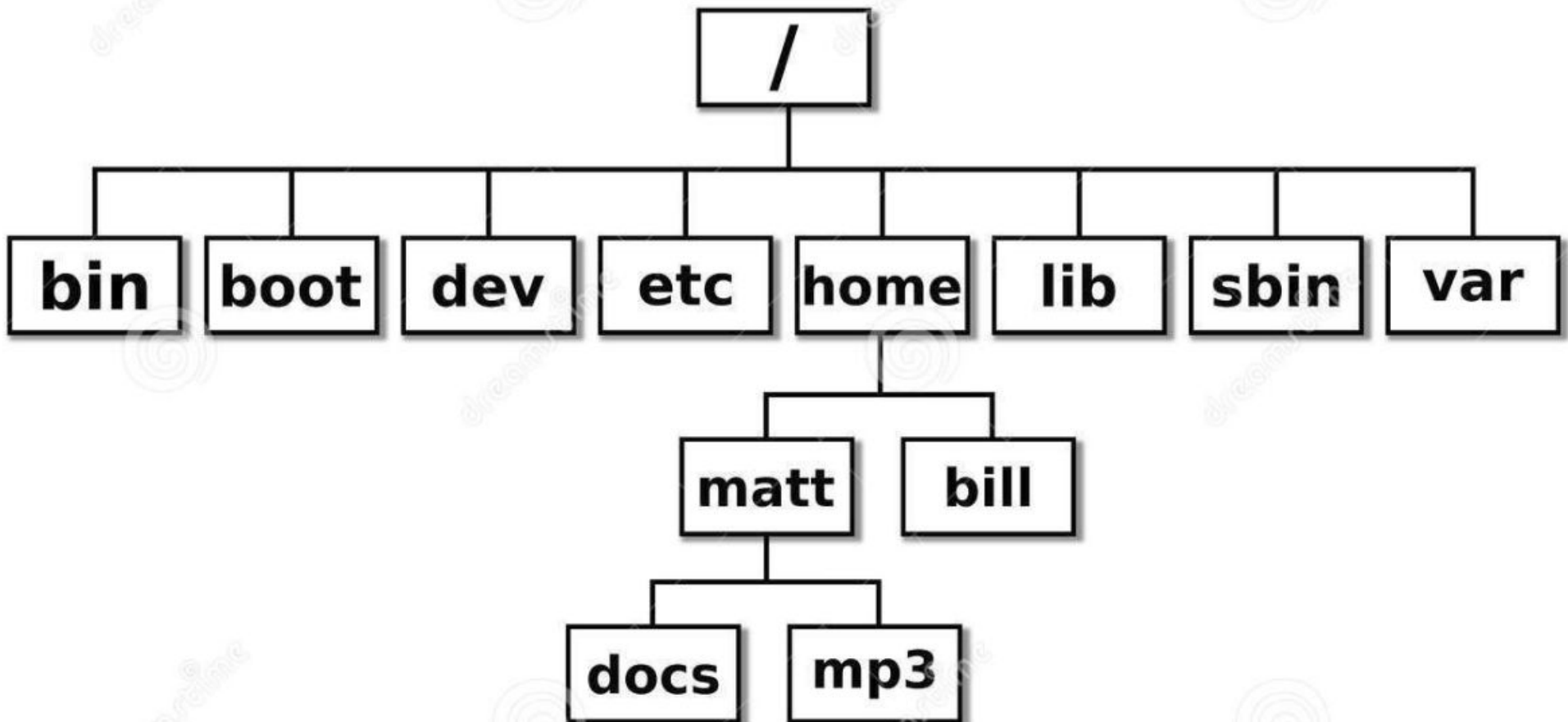
Kernel

- O *kernel* é o núcleo do sistema operacional, servindo de ponte entre aplicativos e o *hardware* do computador, o que inclui:
 - CPU: ordem de execução dos processos etc.
 - Memória: localização e acesso de dispositivos de memória para os aplicativos (memória RAM, cache etc.).
 - Dispositivos de entrada e saída: teclados, mouses, HDs, impressoras, telas e adaptadores de rede, dentre outros.
- O *kernel* também gerencia recursos de aplicativos, tais como a memória utilizada e a identificação de processos e *threads*, e realiza o sincronismo e a comunicação entre eles.


Rootfs

- O Linux diz respeito somente ao *kernel* do sistema operacional. Para completar este, são necessários aplicativos, bibliotecas e um sistema de arquivos.
- O sistema de arquivos é montado pelo *kernel* quando este termina seu processo de inicialização. O *kernel* também procura e executa uma aplicação que configura e inicializa o restante do sistema operacional (geralmente o processo *init*).
- *Rootfs* significa *root file system*, o sistema de arquivos típico de sistemas operacionais baseados em kernel Linux. Ao montar o *rootfs*, o acesso aos dados de diretórios e arquivos é disponibilizado a partir do diretório  do sistema.

Rootfs



Rootfs

- Para montar o *rootfs*, o *kernel* verifica durante o *boot* do sistema qual partição corresponde à pasta  e qual sistema de arquivos deve ser utilizado (ext2, ext3, ext4, ntfs etc.) para esta partição.
- Diferentes tipos de sistemas de arquivos existem para atenderem a objetivos específicos: melhor performance, maior segurança, menor uso de espaço em disco etc.

Rootfs

- Alguns sistemas de arquivo, como o ext3 e o btrfs, possuem *journaling*, em que todas as alterações em disco são gravadas em um *journal*, uma espécie de log, antes de escrever em um arquivo. Após a atualização no arquivo, a entrada no *journal* é removida. Se houver um *crash* do sistema, ou um *reboot* inesperado, as entradas completas não-apagadas no *journal* podem ser lidas e recuperadas.

Rootfs

- Sistemas de arquivo comprimidos, como o CramFS e o SquashFS, armazenam os dados de forma comprimida no dispositivo de armazenamento, economizando espaço em disco.
- Sistemas de arquivo voláteis, como o ramfs e o tmpfs, permitem manter um diretório no sistema montado em RAM, para maior velocidade de escrita e leitura. Estes dados são perdidos após o reboot do sistema.
- O NFS (*Network Filesystem*) permite a montagem de um sistema de arquivo pela rede, acelerando o processo de desenvolvimento de aplicações embarcadas por exemplo.

Criação de um sistema mínimo Linux

- Para baixar o Buildroot, execute:

```
$ mkdir ~/exemplos_buildroot
$ cd ~/exemplos_buildroot
$ git clone https://github.com/buildroot/buildroot.git
buildroot
$ cd buildroot
```

Criação de um sistema mínimo Linux

- O Buildroot possui configurações pré-definidas para uma série de placas.
- Execute `make list-defconfigs` para ver uma lista destas placas. Como são muitas, execute `make list-defconfigs | grep raspberry` para listar somente as configurações para placas Raspberry Pi:

```
raspberrypi0_defconfig      - Build for raspberrypi0
raspberrypi0w_defconfig     - Build for raspberrypi0w
raspberrypi2_defconfig      - Build for raspberrypi2
raspberrypi3_64_defconfig   - Build for raspberrypi3_64
raspberrypi3_defconfig      - Build for raspberrypi3
raspberrypi3_qt5we_defconfig - Build for raspberrypi3_qt5we
raspberrypi_defconfig       - Build for raspberrypi
```


Criação de um sistema mínimo Linux

- A partir daí, escolha a configuração para a sua placa. Por exemplo, para o Raspberry Pi 0W, execute `make raspberrypi0w_defconfig`. Assim, será criado um Makefile necessário para compilar todo um sistema Linux mínimo para esta placa.
- Executando `make xconfig`, surge uma interface gráfica para a definição de diversas opções de compilação.
- Execute então `make` e aguarde cerca de uma hora.

Criação de um sistema mínimo Linux

- Se não ocorrerem erros, o arquivo `~/exemplos_buildroot/buildroot/output/images/sdcard.img` conterá a imagem do sistema operacional mínimo. Instale esta imagem em um cartão SD, assim como você fez para instalar o Raspbian / Raspberry Pi OS.
- Com este sistema instalado, é necessário usar o Raspberry Pi com teclado e tela. Repare que o *boot* do sistema é bem mais rápido do que o de um sistema operacional regular, como o Raspbian / Raspberry Pi OS.
- Para fazer o *login*, utilize o usuário *root* com senha em branco. Digite `ls /bin` e `ls /usr/bin` para ver os poucos programas já instalados (`sudo`, por exemplo, não está presente), e execute `poweroff` para desligar o Raspberry Pi.

Buildroot *Hello World*

- Neste exemplo, criaremos um sistema operacional semelhante ao anterior, mas com um programa *Hello World* já compilado e executado durante o *boot*. Execute

```
$ cd ~/exemplos_buildroot
$ mkdir hello_buildroot
$ cd hello_buildroot
$
GIT_PATH1=https://raw.githubusercontent.com/DiogoCaetanoGarcia/Sistemas_Embarcados/master/Code/25_Buildroot
$ wget ${GIT_PATH}/hello_buildroot/Config.in
$ wget ${GIT_PATH}/hello_buildroot/Makefile
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot-init
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.c
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.mk
```

Buildroot *Hello World*

- `hello_buildroot.c` é o código do *Hello World*
- `Makefile` é o arquivo para compilação do *Hello World*

```
$ cd ~/exemplos_buildroot
$ mkdir hello_buildroot
$ cd hello_buildroot
$
GIT_PATH1=https://raw.githubusercontent.com/DiogoCaetanoGarcia/Sistemas_Embarcados/master/Code/25_Buildroot
$ wget ${GIT_PATH}/hello_buildroot/Config.in
$ wget ${GIT_PATH}/hello_buildroot/Makefile
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot-init
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.c
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.mk
```

Buildroot *Hello World*

- `hello_buildroot-init` é um *script* a ser instalado na pasta `/etc/init.d`, garantindo a execução do programa no *boot* do sistema.
- `Config.in` contém descrições de opções deste programa.

```
$ cd ~/exemplos_buildroot
$ mkdir hello_buildroot
$ cd hello_buildroot
$
GIT_PATH=https://raw.githubusercontent.com/DiogoCaetanoGarcia/Sistemas_Embarcados/master/Code/25_Buildroot
$ wget ${GIT_PATH}/hello_buildroot/Config.in
$ wget ${GIT_PATH}/hello_buildroot/Makefile
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot-init
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.c
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.mk
```

Buildroot *Hello World*

- `hello_buildroot.mk` indica o caminho do código do *Hello World* para o Buildroot, como compila-lo, aonde instala-lo (`/usr/bin`), e aonde instalar o *script* `hello_buildroot-init`.

```
$ cd ~/exemplos_buildroot
$ mkdir hello_buildroot
$ cd hello_buildroot
$
GIT_PATH1=https://raw.githubusercontent.com/DiogoCaetanoGarcia/Sistemas_Embarcados/master/Code/25_Buildroot
$ wget ${GIT_PATH}/hello_buildroot/Config.in
$ wget ${GIT_PATH}/hello_buildroot/Makefile
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot-init
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.c
$ wget ${GIT_PATH}/hello_buildroot/hello_buildroot.mk
```

Buildroot *Hello World*

- Execute:

```
$ BR_HOME=~/exemplos_buildroot
$ cd ${BR_HOME}/buildroot/packages
$ mkdir hello_buildroot
$ cd hello_buildroot
$ cp ${BR_HOME}/hello_buildroot/Config.in .
$ cp ${BR_HOME}/hello_buildroot/hello_buildroot.mk .
$ cd ..
```

- Agora, abra o arquivo `${BR_HOME}/buildroot/packages/Config.in`, procure pelo texto menu "Miscellaneous" e insira a linha `source package/hello_buildroot/Config.in` mantendo a ordem alfabética dos comandos abaixo da linha menu "Miscellaneous".

Buildroot *Hello World*

- Execute:

```
$ cd ..  
$ make xconfig
```

para abrir a interface gráfica de personalização da compilação do sistema operacional. Marque as seguintes opções:

- System Configuration ==> Run a getty (login prompt) after boot
- Target packages ==> Miscellaneous ==> hello_buildroot
- Target packages ==> BusyBox ==> Show packages that are also provided by busybox

e salve estas configurações. Execute `make` para criar uma nova imagem do sistema operacional, e a instale em um cartão SD.

Buildroot *Hello World*

- Depois de ligar o Raspberry Pi com o novo sistema operacional, faça *login* como *root* sem senha, e execute `grep Hello /var/log/message` para ver as mensagens deixadas durante o boot do sistema pelo *script* `hello_buildroot-init`.
- Execute `ps | grep hello` para ver que o programa *Hello World* está em execução.
- Execute `hello_buildroot` para ver o programa *Hello World* em execução, e aperte CONTROL-C para sair.

Referências

Manual Buildroot

- https://github.com/DiogoCaetanoGarcia/Sistemas_Embarcados/blob/master/Refs/Buildroot/Buildroot_Manual.pdf
- <https://sergioprado.org/como-desenvolver-um-sistema-linux-do-zero-para-a-raspberry-pi/>

Referências

Toolchains, Bootloader, Kernel e Rootfs

- <https://sergioprado.org/desmistificando-toolchains-em-linux-embarcado/>
- <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/bootflow.md>
- <https://quorten.github.io/quorten-blog1/blog/2018/09/24/rpi-otp-doc>
- [https://en.wikipedia.org/wiki/Kernel_\(operating_system\)](https://en.wikipedia.org/wiki/Kernel_(operating_system))
- <https://sergioprado.org/sistemas-de-arquivo-em-linux-embarcado-parte-1/>

Referências

Buildroot *Hello World*

- http://www.chip-community.org/index.php/Startup_Program_with_Buildroot
- <https://www.cyberciti.biz/tips/howto-linux-unix-write-to-syslog.html>
- <https://unix.stackexchange.com/questions/108281/how-long-system-has-been-awake-running-since-restart>