Architecture des systèmes embarqués

SE203/SE743

Guillaume Duc
guillaume.duc@telecom-paris.fr
2021–2022

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Objectifs du cours

- Découvrir l'architecture matérielle et logicielle typique d'un système embarqué
- Approfondir quelques notions utiles autour des processeurs (architecture, interruptions...)
- Comprendre les mécanismes de communication avec les périphériques
- Comprendre le composition et le démarrage d'un exécutable

TELECOM Paris

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Fonctionnalités générales d'un système embarqué

- Traitement de données
- Stockage de données
- Interaction avec le monde extérieur
  - Acquisition de données
  - Actions sur l'environnement
  - Communication

TELECOM Paris

Composants matériels d'un système embarqué

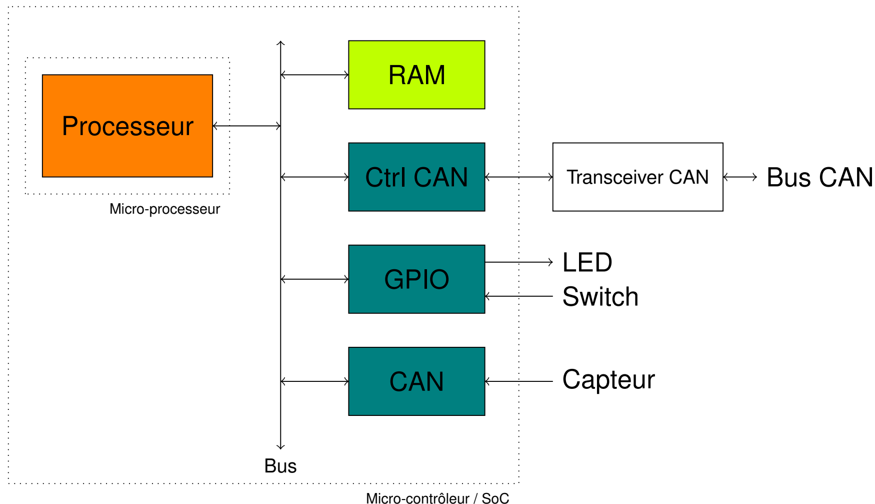
- Traitement de données : *processeur*
- Stockage de données
  - Volatile : *registres*, *cache*, *mémoire vive* (SRAM, DRAM...)
  - Non volatile : *mémoire morte* (ROM, PROM, EPROM, EEPROM, flash...), *support de stockage de masse*
- Interaction avec le monde extérieur : *périphériques*
  - De communication : contrôleur I^{2}C, SPI, CAN, ethernet, WiFi, ZigBee, LoRaWAN...
  - D'acquisition : capteur, convertisseur analogique-numérique, GPIO...
  - De sortie : convertisseur numérique-analogique, LED, pilotage de moteur...
- Interconnexion processeur / autres composants (mémoire, périphériques) : *bus* (sera raffiné par la suite)
- Composants annexes : alimentation (nous n'en parlerons pas ici, cf. SE208)...

TELECOM Paris

Un peu de vocabulaire

- *Processeur* : composant qui exécute les instructions d'un programme
- *Micro-processeur* : processeur se présentant sous la forme d'un circuit intégré
- *Micro-contrôleur* : circuit intégré unique intégrant un processeur, de la mémoire et un ensemble de périphériques
- *System sur puce* (*System on a chip, SoC*) : définition similaire à celle d'un micro-contrôleur. La différence, floue, se fait souvent sur la taille et le nombre de composants intégrés
  - Micro-contrôleur : dédié aux petits systèmes embarqués, intègre quelques périphériques
  - SoC : dédié aux systèmes plus importants, intègre souvent en plus des unités complexes dédiées pour du traitement du signal, traitement vidéo, cryprographie, etc.

Vue matérielle (très) schématique



Vocabulaire

Deux périphériques très courants

- *GPIO* (*General-Purpose Input/Output*) : patte (*pin*) d'un circuit intégré ou connecteur d'une carte électronique qui peut être utilisé comme une entrée ou une sortie numérique (configurable dynamiquement). Plus largement désigne le périphérique qui va permettre de contrôler ces entrées-sorties
- *UART* (*Universal Asynchronous Receiver Transmitter*) : périphérique gérant un lien de communication série (les différents bits d'un mot sont envoyés les uns après les autres sur un seul fil) asynchrone (sans transmission d'horloge, celle-ci est reconstituée par le récepteur en détectant le début de la transmission et en connaissant, par configuration, le débit binaire)

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Interface processeur/monde extérieur

- Le processeur communique
  - Avec des mémoires
  - Avec des périphériques
- Communication avec la mémoire (vu en ELECINF102 et INF104)
  - Type (lecture ou écriture)
  - Adresse de la case mémoire concernée
  - Donnée à écrire (sens processeur vers mémoire) ou donnée lue (sens mémoire vers processeur)

Mémoires

- Historiquement, on fait la distinction entre
  - *Mémoire vive* (*Random Access Memory, RAM*)
    - Rapide, aussi bien en lecture qu'en écriture
    - Le plus souvent volatile (à quelques exceptions près : MRAM par exemple)
    - Utilisée pour stocker les données (et dans la plupart des cas le code) des applications en cours d'exécution
    - Différentes technologies : SRAM (*Static*), DRAM (*Dynamic*), SDRAM (*Synchronous Dynamic*)...
  - *Mémoire morte* (*Read-Only Memory, ROM*)
    - Généralement plus lente, en particulier en écriture (lorsqu'elle est possible)
    - Non volatile
    - Utilisée pour stocker sur le long terme le code, les données initiales, la configuration...
    - Technologies : Mask ROM, PROM (*Programmable*), EPROM (*Erasable Programmable*), EEPROM (*Electrically Erasable Programmable*), Flash...

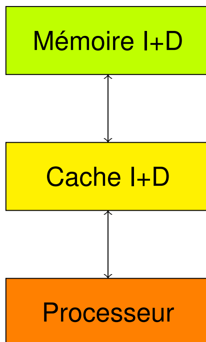
Mémoires

- Distinction historique
- Parfois floue (exemple MRAM)
- Dans la suite
  - Lorsqu'un programme est en cours d'exécution, où se trouve le code et où se trouvent les données ?
  - Comment les mémoires sont reliées au processeur ?
    - Architectures Harvard et von Neumann

TELECOM Paris

Architectures von Neumann

- Un seul chemin et un seul espace mémoire pour le code (instructions) et les données

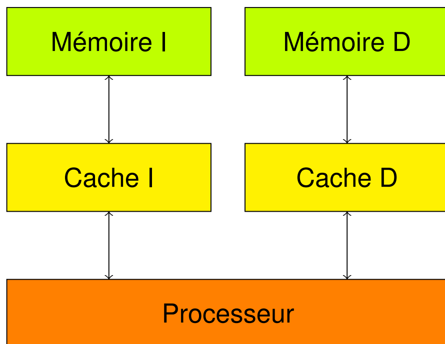


Architectures von Neumann

- Permet de traiter du code comme des données
  - Permet ainsi de charger un programme en mémoire avant de l'exécuter, de le modifier à la volée (remplacement d'une instruction par un point d'arrêt), etc.
  - Permet du code auto-modifiant (tombé en désuétude)
- Permet de traiter des données comme du code
  - Utilisé par exemple lors de la compilation juste-à-temps (*Just-in-time compilation*)
  - Inconvénient du point de vue de la sécurité (injection de code via un débordement de tampon par exemple)
- Problème de performance : goulot d'étranglement. Une instruction utilisant une donnée en mémoire (opérande ou résultat) nécessite au minimum deux cycles, un pour récupérer l'instruction et un pour manipuler la donnée en mémoire

Architectures Harvard

- Deux chemins et deux espaces mémoires distincts, un pour le code (instructions) et un pour les données

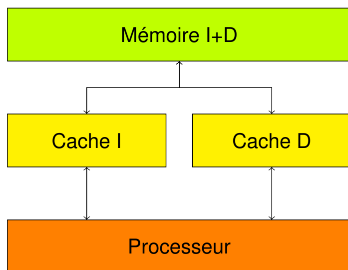


Architectures Harvard

- Plus de goulot d’étranglement
- Quelques problèmes : stockage des valeurs initiales des variables, stockage des constantes, debug...
- Architecture utilisée par plusieurs micro-contrôleurs (notamment PIC et AVR)

Architectures Harvard modifiée

- Architecture la plus répandue
- Quelques inconvénients : problème de cohérence de cache lors du traitement d'une instruction comme une donnée, goulot d'étranglement à la sortie du cache (moins critique, seulement en cas de défaut de cache)



TELECOM Paris

Interface processeur/monde extérieur

- Comment le processeur communique-t-il avec les périphériques ?
- Besoins pour cette communication
  - Transférer des informations vers le périphérique (données, commandes, configuration...)
    - Similaire à une écriture en mémoire
  - Transférer des données depuis le périphérique (données, statut...)
    - Similaire à une lecture depuis la mémoire
  - Identifier à quoi correspondent les données échangées (canal si plusieurs canaux de données, quelle partie de la configuration, statut de quel sous-système...)
    - Similaire à l'utilisation d'une adresse pour identifier la case mémoire impactée

Interface processeur/monde extérieur

- Dans les deux cas, même sémantique
  - Type (lecture ou écriture)
  - Adresse de la case mémoire/*registre* concerné
  - Donnée à écrire (sens processeur vers mémoire/périphérique) ou donnée lue (sens mémoire/périphérique vers processeur)
- Comment distinguer les différentes mémoires et les différents périphériques et savoir auquel s'adresse un transfert ?
  - L'adresse utilisée permet de connaître à quel composant (mémoire/périphérique) s'adresse un transfert
    - *Plan d'adressage mémoire* (*memory map*) décrit comment est partagé le(s) espace(s) d'adressage entre les différents composants

TELECOM Paris

Deux cas pour les périphériques

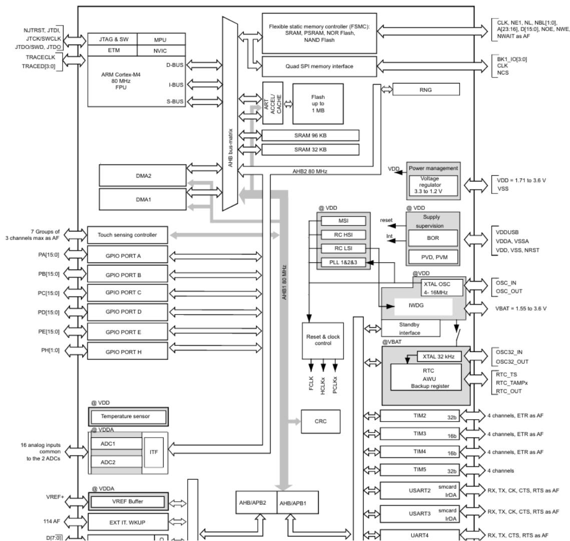
- *Memory-Mapped I/O (MMIO)*
  - Les périphériques et les mémoires partagent un unique espace d'adressage
  - Les transferts vers et depuis les périphériques (données, commandes, configuration ou statut) se réalisent de façon identique avec les transferts mémoires (ex. instructions *load/store*, modes d'adressage impliquant la mémoire, etc.)
  - On parle de *périphériques mappés en mémoire*
- *Port-Mapped I/O (PMIO)*
  - Deux espaces d'adressage distincts : un pour la mémoire et un pour les périphériques
  - Les transferts vers et depuis les périphériques se réalisent avec des instructions dédiées (ex. instructions `IN` et `OUT` du `x86`), différentes de celles utilisées pour accéder à la mémoire

MMIO vs. PMIO

- *Memory-Mapped I/O*
  - Un seul bus entre le processeur et le monde extérieur
  - Toutes les instructions qui manipulent des données en mémoire peuvent manipuler des données en provenance de périphériques
- *Port-Mapped I/O*
  - Pas de réduction de l'espace d'adressage utilisable par de la mémoire (moins un problème avec les architecture 32/64 bits)
  - Contrôle d'accès souvent en tout ou rien (mode superviseur)
  - Instructions spécifiques souvent plus frustres (exemple architecture `x86` : `IN` et `OUT` ne peuvent utiliser que le registre `(E)AX` comme source ou destination du transfert)
- Les périphériques MMIO sont les plus courants dans l'embarqué, on ne considérera donc que ce cas dans la suite

Un seul bus ?

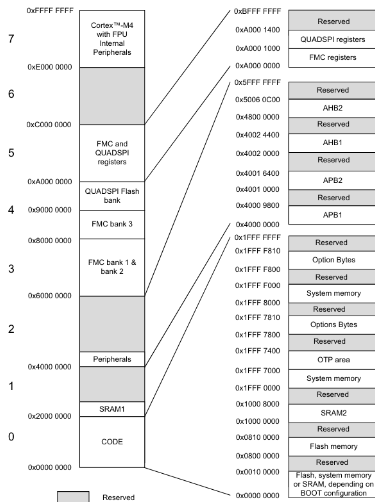
Exemple : STM32L475xx (source image : STMicroelectronics)



TELECOM Paris

Plan d'adressage mémoire

Exemple : STM32L475xx (source image : STMicroelectronics)



Plan d'adressage mémoire (détails)

Exemple : STM32L475xx (source image : STMicroelectronics)

Bus	Boundary address	Size (bytes)	Peripheral
AHB3	0xA000 1000 - 0xA000 13FF	1 KB	QUADSPI
	0xA000 0000 - 0xA000 0FFF	4 KB	FMC
AHB2	0x5006 0800 - 0x5006 0BFF	1 KB	RNG
	0x5004 0400 - 0x5006 07FF	129 KB	Reserved
	0x5004 0000 - 0x5004 03FF	1 KB	ADC
	0x5000 0000 - 0x5003 FFFF	16 KB	OTG_FS
	0x4800 2000 - 0x4FFF FFFF	~127 MB	Reserved
	0x4800 1C00 - 0x4800 1FFF	1 KB	GPIOH
	0x4800 1800 - 0x4800 1BFF	1 KB	GPIOG
	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF
	0x4800 1000 - 0x4800 13FF	1 KB	GPIOE
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA
-	0x4002 4400 - 0x47FF FFFF	~127 MB	Reserved
AHB1	0x4002 4000 - 0x4002 43FF	1 KB	TSC
	0x4002 3400 - 0x4002 3FFF	1 KB	Reserved
	0x4002 3000 - 0x4002 33FF	1 KB	CRC
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH registers
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved
	0x4002 1000 - 0x4002 13FF	1 KB	RCC
	0x4002 0800 - 0x4002 0FFF	2 KB	Reserved
	0x4002 0400 - 0x4002 07FF	1 KB	DMA2
	0x4002 0000 - 0x4002 03FF	1 KB	DMA1

TELECOM Paris

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Modes de fonctionnement

- La très vaste majorité des processeurs (sauf les plus simples) offrent plusieurs *modes de fonctionnement* (appelés aussi *niveaux de privilèges*, *anneaux de protection*...)
- En général au moins deux
  - Superviseur
  - Utilisateur
- À chacun de ces modes sont associés
  - Certains privilèges (autorisation ou non d'exécuter certaines instructions, d'accéder à certains registres, etc.)
  - Des ressources propres (pointeur de pile, compteur ordinal...)

Modes de fonctionnement

- Ces modes permettent d'assurer certaines propriétés de sécurité et de sûreté
  - Notamment en isolant le système d'exploitation vis-à-vis des applications
- Le passage d'un mode à l'autre s'effectue
  - Suite au déclenchement d'une interruption ou d'une exception
  - Suite à l'exécution de certaines instructions (exemple `svc` (*Supervisor Call*) sur ARM)
  - En écrivant dans des registres de configuration du processeur

TELECOM Paris

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Exceptions

- Permet au processeur d'interrompre l'exécution du programme courant pour pouvoir exécuter un traitement particulier en réponse à un événement
- Vocabulaire : *interruption* et *exception* sont utilisées de façon non uniforme dans la littérature technique. Nous préférerons le terme exception pour parler du mécanisme générique (Hennessy & Patterson)
- Source des exceptions
  - Un périphérique (via une *interrupt request, IRQ*)
  - Suite à une anomalie : division par zéro, accès mémoire invalide, instruction invalide...
  - Le logiciel (on parle souvent d'*interruption logicielle*) via une instruction particulière (utilisé le plus souvent pour implémenter des appels systèmes ou pour du débogage)

TELECOM Paris

Typologie

- *Synchrone* : l'exception se produit au même endroit chaque fois que le programme est exécuté (avec les mêmes données et la même allocation mémoire)
  - Exemples : division par zéro, appel système, déréférencement de pointeur invalide...
- *Asynchrone* : pas de lien direct avec l'exécution d'une instruction
  - Exemples : interruptions en provenance d'un périphérique, reset...

Gestionnaire d’interruption

- *Gestionnaire d’interruption* (*Interrupt Handler* ou *Interrupt Service Routine, ISR*) : bloc de code à exécuter en réponse à une exception donnée
- *Table des vecteurs d’interruption* (*Interrupt Vector Table*) : structure de données associant les exception et les gestionnaires correspondants (plus précisément l’adresse de la première instruction du gestionnaire, appelé *vecteur d’interruption*)

Contrôleur d'interruption

- Le *contrôleur d'interruption* est un module matériel qui reçoit les différentes requêtes d'exceptions (plusieurs peuvent survenir au même moment)
- Il les traite (priorités fixes ou configurables, masquage configurable, etc.) pour générer le numéro de l'exception à traiter en premier (s'il y en a une)

Déroulement d'une exception

- Le processeur exécute un programme
- Une ou plusieurs exception surviennent, le contrôleur d'interruption en sélectionne une
- Le processeur arrête l'exécution du programme en faisant en sorte que toutes les instructions avant un certain point soient totalement exécutées et aucune après ce point
- Le processeur sauvegarde ce point de reprise et quelques autres informations
- Le processeur consulte la table des vecteurs d'interruption pour connaître l'adresse du gestionnaire
- Le processeur débute l'exécution de ce gestionnaire

Déroulement d'une exception

- En général, pendant le traitement d'une exception, les exceptions sont désactivées
  - Le traitement d'une exception doit donc être court (risque de trop retarder le traitement d'une exception plus importante, risque de perdre des données si les périphériques ont un petit tampon...)
- Certaines architectures autorisent les exceptions imbriquées (*nested interrupt*)
- À la fin du traitement, l'exécution initiale reprend (en cas d'exception causée par l'exécution fautive d'une instruction, celle-ci peut être réexécutée)

TELECOM Paris

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Forte diversité au niveau logiciel

- Sans système d'exploitation : approche carte nue (*bare metal*)
- Avec système d'exploitation (voir plus : hyperviseur...)

Système d'exploitation

- Un système d'exploitation offre un certain nombre de services aux applications
  - Abstraction du matériel (*couche d'abstraction du matériel*, *Hardware Abstraction Layer, HAL*)
  - Partage des différentes ressources (temps processeur, mémoire, périphériques, fichiers...)
  - Communication et synchronisation entre processus (sémaphores...)
- Il charge les exécutables et contrôle leur exécution
- Il est aussi responsable de l'initialisation du support d'exécution et des périphériques (conjointement avec le chargeur d'amorçage)
- Enfin, il traite les différentes exceptions qui surviennent (en installant ses gestionnaires d'interruption)

Système d'exploitation pour l'embarqué

- Il peut se présenter sous la forme d'une bibliothèque directement liée avec l'application principale
  - Les appels aux services du système d'exploitation se font via des appels de fonctions classiques
  - Exemples : ChibiOS/RT, FreeRTOS...
- Il peut aussi de présenter comme un code indépendant (comme sur un ordinateur plus « classique »)
  - Les appels aux services se font alors via des appels systèmes (interruptions logicielles)

Systèmes d'exploitation pour l'embarqué

Exemples

- Généralistes : Linux, *BSD (FreeBSD, NetBSD...), Windows...
- Dédiés
  - Windows IoT (ex. Windows Embedded, propriétaire), QNX (propriétaire), iOS (propriétaire), Android (*open source*)...
  - Temps réel : VxWorks (propriétaire), FreeRTOS (*open source*), ChibiOS/RT (*open source*), INTEGRITY (propriétaire, classifié), RTX (propriétaire)...

L’autre extrémité : *bare metal*

- Pas de système d’exploitation
- Un seul exécutable qui doit tout faire
  - Initialisation du matériel et des périphériques
  - Fonctionnalités principales
  - Éventuellement : mise à jour et débogage

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Anatomie d'un exécutable

Code

- Ensemble des instructions du programme
- Non modifiable
- Stockage : En mémoire non volatile (ROM, flash, stockage de masse...)
- Exécution
  - Depuis la RAM
    - Exécution plus rapide mais nécessite une recopie au démarrage et éventuellement des mécanismes de protection pour s'assurer que le code est en lecture seule
  - Directement depuis la mémoire non volatile : *eXecute In Place* (XIP)
    - Pas toujours possible (stockage de masse)
    - Pose certains problèmes : parfois plus lent, point d'arrêts...
  - Combinaison possible

Anatomie d'un exécutable

Données allouées à la compilation (statiques)

- Non modifiables (constantes)
  - Stockage (valeur) : en mémoire non volatile
  - Exécution : en RAM (nécessite une recopie et un mécanisme de protection) ou en mémoire non volatile (si possible)
- Modifiables (variables) avec valeur initiale différente de 0
  - Stockage (valeur initiale) : en mémoire non volatile
  - Exécution : nécessairement en RAM (nécessite une recopie)
- Modifiables (variables) avec valeur initiale égale à 0
  - Stockage : non nécessaire (optimisation). Elles sont regroupées et ne sont stockées que l'adresse initiale et la taille de cette zone
  - Exécution : nécessairement en RAM dans une zone initialisée à 0

Anatomie d’un exécutable

Données allouées dynamiquement

- Pile (*Stack*)
  - Stockage : absente
  - Exécution : en RAM (vide ou presque au démarrage du programme)
- Tas (*Heap*)
  - Stockage : absent
  - Exécution : en RAM (vide au démarrage)

Anatomie d'un exécutable

Sections

- Les exécutables sont divisés en plusieurs *sections*
  - `text` : le code (stockage : ROM, exécution : ROM ou RAM)
  - `bss` : données modifiables non initialisées ou initialisées à zéro (non stockée, exécution : RAM)
  - `data` : données modifiables initialisées (stockage : ROM, exécution : RAM)
  - `rodata` : constantes (stockage : ROM, exécution : ROM ou RAM)
- Sera complété dans un cours ultérieur

TELECOM Paris

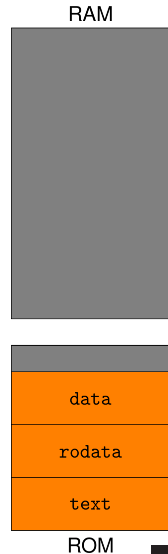
À quelles adresses ?

- C'est lors de l'étape d'édition des liens que les adresses des différentes sections sont déterminées
- Plusieurs cas possibles
  - *Code fixe* : il est prévu pour être placé et exécuté à une adresse précise fixe
  - *Code relogeable* : le code est prévu pour être facilement modifié par le chargeur ou lui-même en fonction de l'adresse à laquelle il a été placé
  - *Code indépendant de sa position* (*Position Independant Code*, PIC) : code prévu pour être exécuté, sans modification, à partir de n'importe quelle adresse

Démarrage d'un exécutable

Cas système bare metal

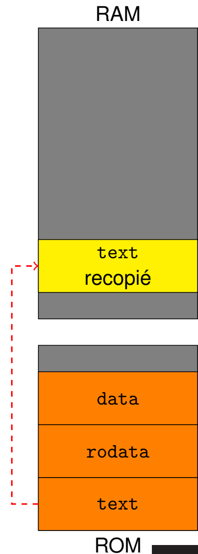
- Exécutable en ROM



Démarrage d'un exécutable

Cas système bare metal

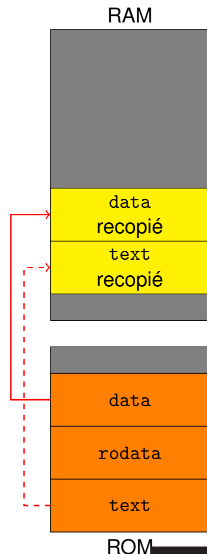
- Exécutable en ROM
- Éventuellement recopie du code en RAM
  - Par du code externe (bootloader)
  - Par le code lui même (`crt0.s`)



Démarrage d'un exécutable

Cas système bare metal

- Exécutable en ROM
- Éventuellement recopie du code en RAM
  - Par du code externe (bootloader)
  - Par le code lui même (`crt0.s`)
- Recopie de `data` en RAM par le `crt0.s`

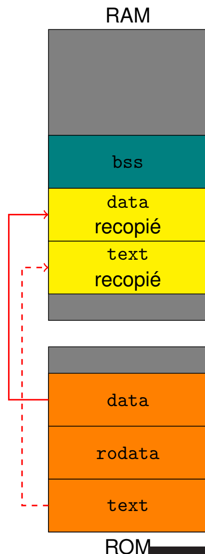


TELECOM Paris

Démarrage d'un exécutable

Cas système bare metal

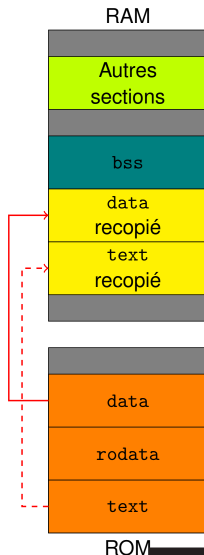
- Exécutable en ROM
- Éventuellement recopie du code en RAM
  - Par du code externe (bootloader)
  - Par le code lui même (`crt0.s`)
- Recopie de `data` en RAM par le `crt0.s`
- Création et initialisation à 0 de la `bss` par le `crt0.s`



Démarrage d'un exécutable

Cas système bare metal

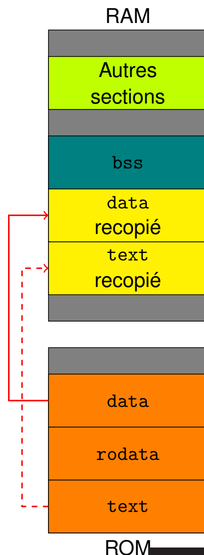
- Exécutable en ROM
- Éventuellement recopie du code en RAM
  - Par du code externe (bootloader)
  - Par le code lui même (`crt0.s`)
- Recopie de `data` en RAM par le `crt0.s`
- Création et initialisation à 0 de la `bss` par le `crt0.s`
- Création de la pile et éventuellement d'autres sections par `crt0.s`



Démarrage d'un exécutable

Cas système bare metal

- Exécutable en ROM
- Éventuellement recopie du code en RAM
  - Par du code externe (bootloader)
  - Par le code lui même (`crt0.s`)
- Recopie de `data` en RAM par le `crt0.s`
- Création et initialisation à 0 de la `bss` par le `crt0.s`
- Création de la pile et éventuellement d'autres sections par `crt0.s`
- Appel de `main()`



TELECOM Paris

Démarrage d'un exécutable

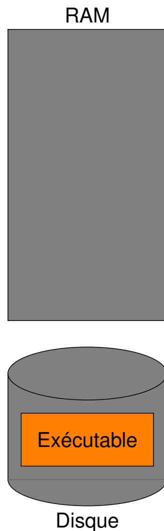
Adresses

- Si on considère la section `text` ou `data`, on constate qu'elles ont deux adresses
  - Une avant recopie (en ROM) : la *LMA*
  - Une après recopie (en RAM) : la *VMA*

Démarrage d'un exécutable

Cas OS + stockage de masse

- Exécutable sur stockage de masse non mappé en mémoire

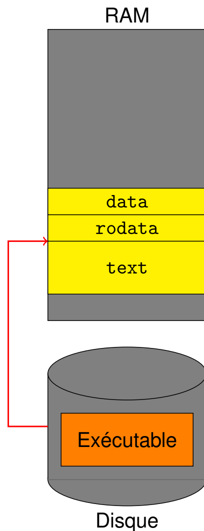


TELECOM Paris

Démarrage d'un exécutable

Cas OS + stockage de masse

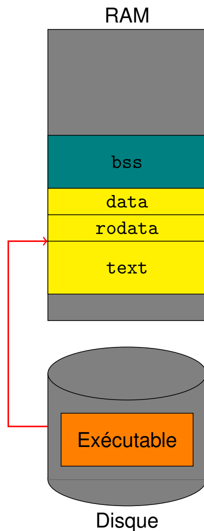
- Exécutable sur stockage de masse non mappé en mémoire
- Le chargeur (*loader*) du système d'exploitation charge les sections importantes de l'exécutable en mémoire (soit par copie complète, soit plus intelligemment)



Démarrage d'un exécutable

Cas OS + stockage de masse

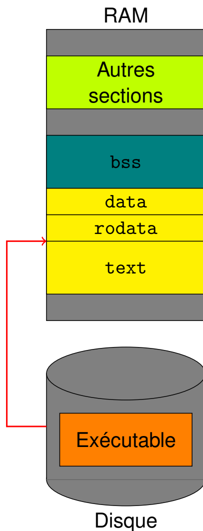
- Exécutable sur stockage de masse non mappé en mémoire
- Le chargeur (*loader*) du système d'exploitation charge les sections importantes de l'exécutable en mémoire (soit par copie complète, soit plus intelligemment)
- Création et initialisation à 0 de la `bss` par le chargeur



Démarrage d'un exécutable

Cas OS + stockage de masse

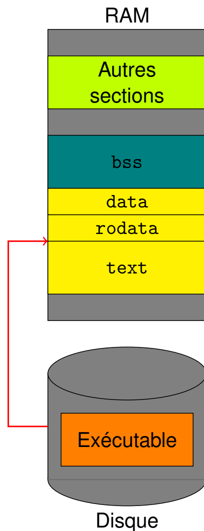
- Exécutable sur stockage de masse non mappé en mémoire
- Le chargeur (*loader*) du système d'exploitation charge les sections importantes de l'exécutable en mémoire (soit par copie complète, soit plus intelligemment)
- Création et initialisation à 0 de la `bss` par le chargeur
- Création de la pile et éventuellement d'autres sections par le chargeur



Démarrage d'un exécutable

Cas OS + stockage de masse

- Exécutable sur stockage de masse non mappé en mémoire
- Le chargeur (*loader*) du système d'exploitation charge les sections importantes de l'exécutable en mémoire (soit par copie complète, soit plus intelligemment)
- Création et initialisation à 0 de la `bss` par le chargeur
- Création de la pile et éventuellement d'autres sections par le chargeur
- Appel de `main()`



TELECOM Paris

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Démarrage

- L'initialisation du matériel peut être assez complexe en fonction du système
  - Configuration du processeur (modes...)
  - Initialisation de la RAM (configuration du contrôleur RAM, mise en place de la pile...)
  - Initialisation des périphériques (configuration de l'arbre d'horloge, des PLL, des différents contrôleurs de bus...)
  - Configuration du cache, de la MPU/MMU
  - Configuration du contrôleur d'interruption...

Chargeur d’amorçage

- La responsabilité de l’initialisation du matériel lors du démarrage peut être déléguée à un programme dédié appelé *chargeur d’amorçage* (*bootloader*)
- Il a alors le premier programme à s’exécuter et se charge de :
  - L’initialisation du matériel (au moins le strict minimum pour qu’un programme puisse s’exécuter correctement)
  - La préparation d’un environnement d’exécution correct (pile...)
  - Le lancement du système d’exploitation ou le programme principal
  - Éventuellement la possibilité de mettre à jour le programme principal ou de faire du debug (possibilité d’interactivité avec l’utilisateur)

Exemples de chargeur d'amorçage

- Das U-Boot (très utilisé dans l'embarqué)
- GRUB (PC)
- Windows Boot Manager (PC)

Exemple : Démarrage de Linux

Souce : `https://www.kernel.org/doc/Documentation/arm/Booting`

- Vu du noyau Linux, voici ce qui doit être fait par le bootloader sur un système à base d'ARM
  - Initialisation de toute la RAM de la plate-forme
  - Initialisation d'un port série (recommandé)
  - Détection du type de machine (obligatoire sauf support des arbres de périphérique) pour le passer au noyau
    - Utilisé pour sélectionner le code spécifique pour l'initialisation de la plate-forme dans le noyau
  - Préparer soit la structure ARM Tags (ATAG) soit l'arbre des périphériques pour la passer au noyau
  - Charger l'image disque mémoire initiale (optionnel)
  - Appeler l'image du noyau

Plan

Introduction

Architecture matérielle

Compléments sur les processeurs
- Interface processeur/monde extérieur
- Modes
- Interruptions & Exceptions

Architecture logicielle
- Système d'exploitation
- Composition et démarrage d'un exécutable
- Chargeur d'amorçage
- Débogage

Pourquoi déboguer ?

- Sauf cas très particuliers (vieillissement au point que le processeur ne fonctionne plus correctement, problème d'alimentation, rayonnement cosmique...), le processeur ne fait qu'exécuter le programme que vous lui avez donné
  - Il ne « plante » pas de sa propre initiative, c'est vous qui lui avez dit de le faire (probablement sans le vouloir)
- Donc s'il ne fait pas ce que vous attendez de lui, c'est que le programme que vous avez écrit ne fait pas ce que vous vouliez
  - Un bogue dans le compilateur, bien que possible, est assez rare, donc commencez toujours par regarder ce que vous avez écrit
- Il va donc être nécessaire de le déboguer

Le débogage dans SE203

- Dans SE203, votre premier réflexe sera de déboguer
  - En exécutant le programme instruction assembleur par instruction assembleur
  - En examinant à chaque instruction le contenu des registres (processeur et/ou périphérique qui vous intéresse)
  - En examinant à chaque instruction le contenu de la mémoire

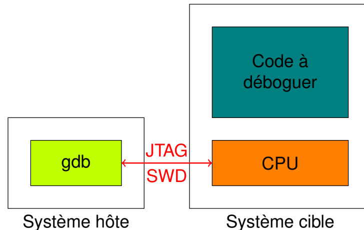
Localisation du débogueur

- Débogueur local
  - Le débogueur s'exécute sur le même système que l'application à déboguer
  - Il utilise les ressources du système (I/O, réseau) pour interagir avec l'utilisateur
  - Nécessite que les ressources du systèmes soient suffisantes pour cette interaction (exclut les systèmes les plus modestes)
  - Permet de déboguer une application mais pas le système d'exploitation (ou difficilement)

Localisation du débogueur

- Débogueur distant
  - Le débogueur s'exécute sur un système différent que l'application à déboguer
  - Il peut ainsi bénéficier de toutes les ressources d'un autre système pour interagir avec l'utilisateur
  - Nécessite un mécanisme de communication entre le système hébergeant le débogueur et le système à déboguer
    - Le logiciel à déboguer peut coopérer
    - Ou non (nécessite alors d'utiliser un mécanisme de débogage matériel : JTAG, SWD...)

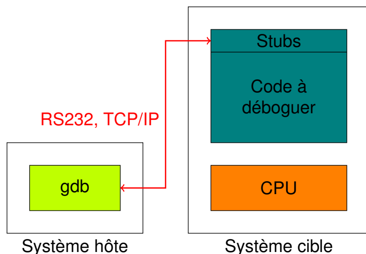
Débogage distant, sans coopération



- Débogage « bas niveau » (directement au niveau du processeur)
- Pas besoin de modifier l'application à déboguer (sans coopération), non intrusif
- Nécessite du matériel dédié pour gérer le JTAG/SWD...

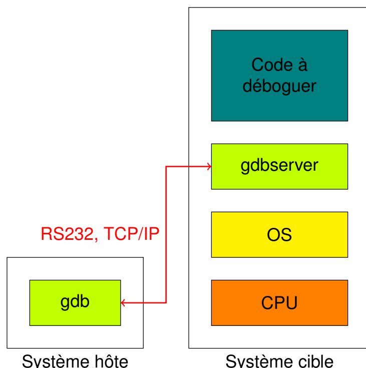
TELECOM Paris

Débogage distant, avec coopération



- Du code (*stubs*) est ajouté à l'application à déboguer pour assurer la communication avec le débogueur (via une liaison série, TCP/IP...)
- Pas de matériel spécialisé mais intrusif (l'application à deboguer est modifiée)

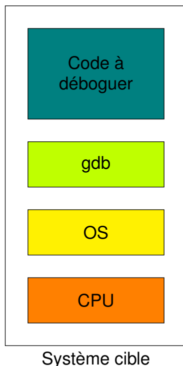
Débogage distant, avec gdbserver



- Liens standards, peu intrusif, facile à mettre en place
- Nécessite un système d'exploitation

TELECOM Paris

Débogage local



- Peu intrusif, facile à mettre en place
- Nécessite un système d'exploitation et un moyen d'interagir avec gdb (interface homme-machine ou accès réseau)

Points d’arrêt (*breakpoints*)

- Points d’arrêt logiciels
  - L’instruction ciblée est remplacée par le débogueur par une instruction déclenchant une exception (instruction invalide, instruction explicite pour le débogage...)
  - Lorsque l’exécution arrive sur cette instruction, le débogueur peut ainsi reprendre la main et remettre en place l’instruction d’origine pour poursuivre l’exécution
  - Problème lorsque le code est stocké dans une mémoire où l’écriture est compliquée (flash, EEPROM...)
- Points d’arrêt matériels
  - Certains processeurs intègrent des registres dédiés permettant de stocker des conditions d’arrêt (compteur ordinal égal à une certaine valeur par exemple)
  - À chaque cycle, ces conditions sont vérifiées et si une est vérifiée, le processeur s’arrête pour pouvoir être débogué
  - Néanmoins, ces registres sont en nombre limité