Système d'exploitation
Architecture

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard
Jean-louis.lanet@unilim.fr

Vincent Neiger --- vincent.neiger@unilim.fr

Université de Limoges
FACULTÉ DES SCIENCES ET TECHNIQUES

sources: Jean-Louis Lanet / Guillaume Bouffard

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d’exécution
- Exceptions & Interruptions
- Les différentes mémoires

Le rôle du système

Système d'exploitation :

- Programme dont le rôle est de gérer le matériel
- Permet de lancer les programmes des utilisateurs
- Plus qu'un simple moniteur et que le BIOS
- Offre une couche d'abstraction vis-à-vis du matériel
- Gestion des fichiers, de la mémoire, du réseau, du multi-tâches, de l‘IHM...

Le rôle du système

- Système d'exploitation :
  - Années 60, premiers OS sur des mainframes : traitement par lot,
  - temps partagé
  - Années 70, Unix : multi-utilisateur, réseau
  - Années 80 et 90, PC : interactivité, plug & play
  - Années 2000, OS embarqués : PDA, téléphones portables...
- Mais aussi voitures (environs 60 CPU / voiture) , appareil photo, routeur ADSL, chaîne HIFI
- Mais encore : Internet, systèmes distribués, systèmes temps réels, multiprocesseurs

Par exemple…

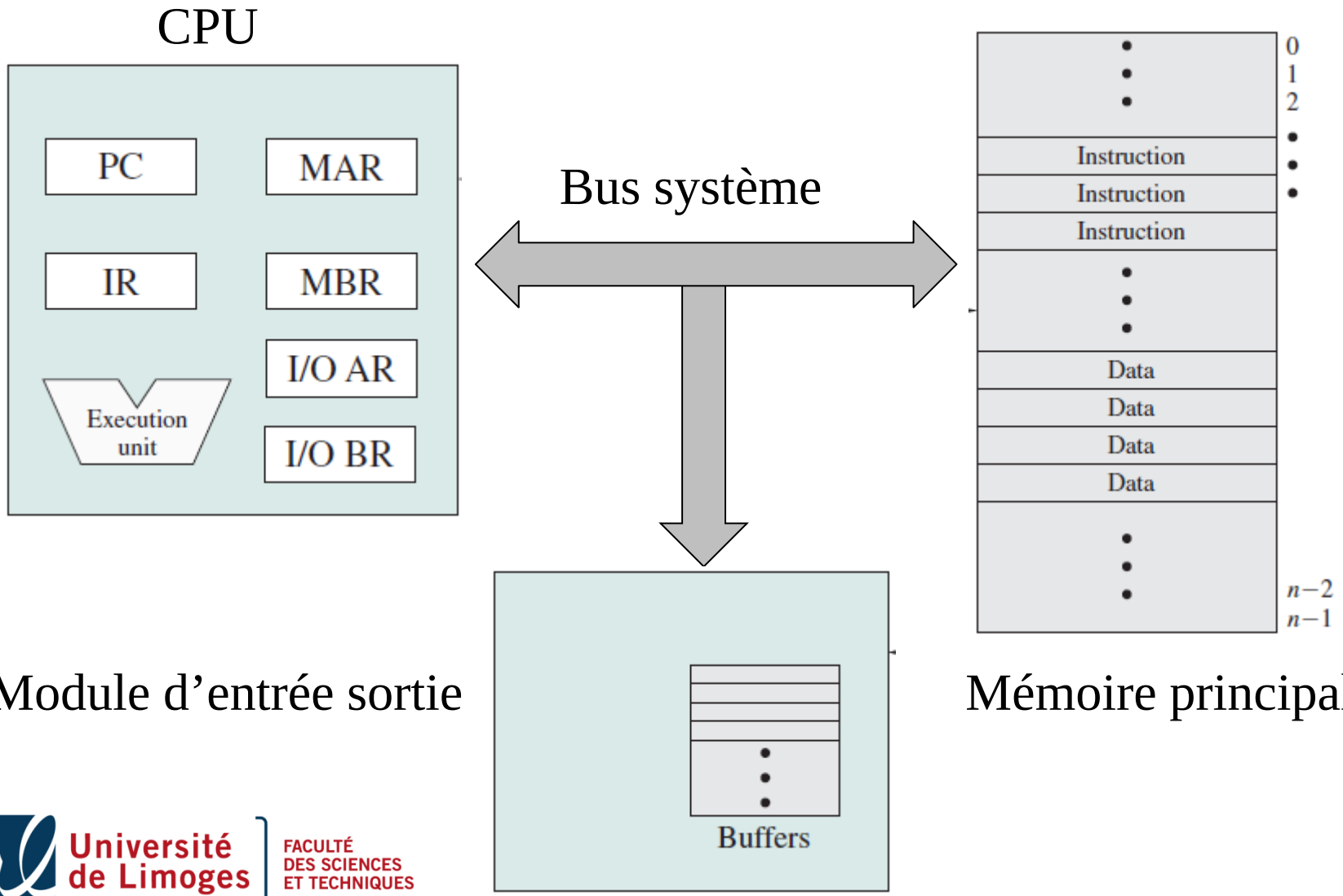
Quelques OS

- Unix , Linux
- Windows NT, 2000, XP, Vista, 07
- MAC OS X
- Windows CE, pocket PC
- Palm OS,
- RT-Linux, µCLinux
- VxWorks, pSos
- JavaOS

Les composants

- Un système est composé de trois sous systèmes: le CPU; la mémoire et les composants d'entrée sortie.
  - Le processeurs exécute les calculs généralement sur des registres internes,
  - La mémoire principale stocke les données et les programmes avant de les transférer en mémoire secondaire,
  - Les entrées sorties transfèrent les données de la mémoire principale vers les mémoires secondaires,
  - Les bus assurent les communication entre ces entités.

Composants


CPU
PC
MAR
IR
MBR
I/O AR
I/O BR
Execution unit
Bus système
0
1
2
Instruction
Instruction
Instruction
Data
Data
Data
Data
n−2
n−1
Buffers
Module d'entrée sortie
Mémoire principale

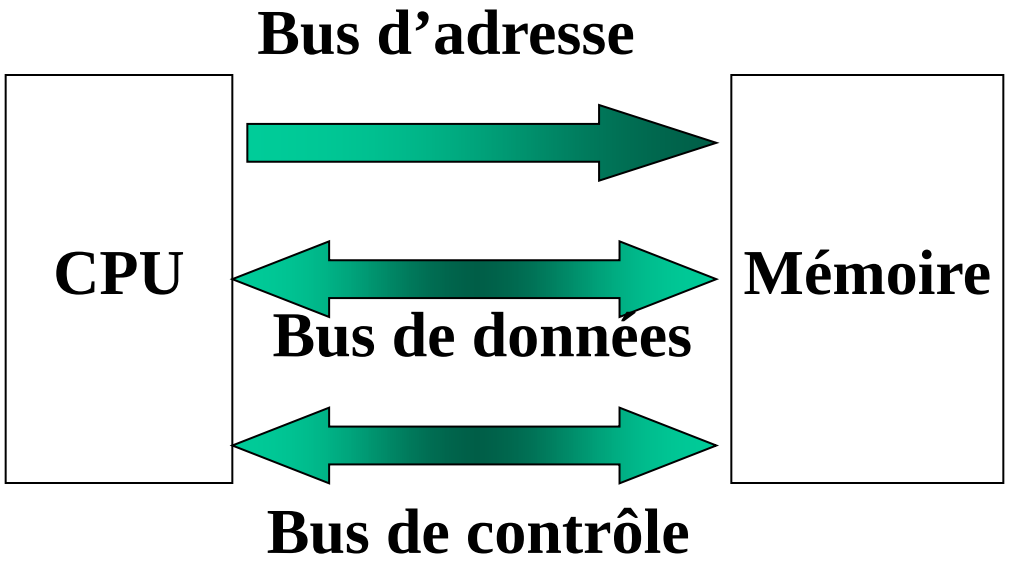
Les cœurs de microprocesseurs

- Fin des processeurs en tranche, ils sont conçu en monochip ou agrégation de chip,
- Le GPU (Graphical Processor Unit) exécutent de manière efficace des calculs sur de multiples données,
- DSP (digital Signal Processor) sont capables d'exécuter des calculs sur des flux de signaux,
- SoC (System on Chip) incluent sur la même puce des composants différents (mémoire, ligne d'IO,…)

CPU

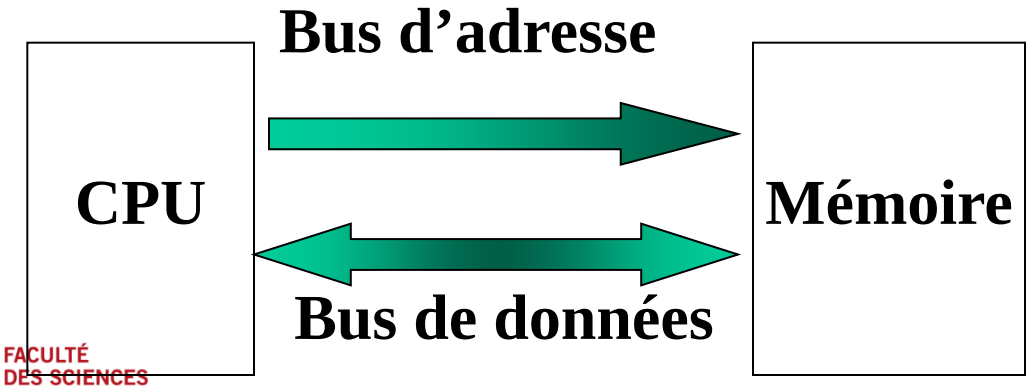
- PC: adresse prochaine instruction à exécuter
- SP: adresse du sommet de pile
- Registres généraux de calcul
- Registre (ou file) de registre de décodage d’instruction,
- Registre d'état
  - Bits de conditions (Z, LT, GT, etc.)
  - Modes d'exécution
    - Niveau de protection (user/supervisor)
    - Adressage physique/virtuel
  - Masque des interruptions

Interaction CPU / mémoire



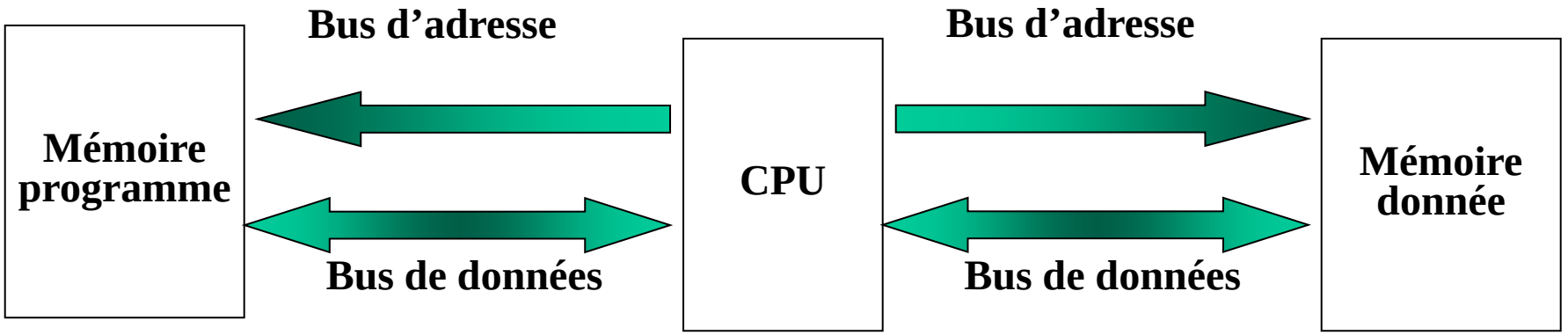
Architecture Von Neuman

- Mémoire contient instructions et données
- CPU lit les instructions depuis la mémoire
- Chargement d'un programme en mémoire
  - Instructions manipulées comme des données
- Programmes peuvent s'auto modifier.
  - Sauf certaines parties du Bios (secure bootloader)



Architecture Harvard

- Deux types de mémoire
  - Instructions et données
  - Accès en parallèle aux instruction et aux données
  - Données sont accédée plus fréquemment que les instructions,
- CPU Idem à Von Neuman
  - Ne peut pas utiliser du code qui se modifie lui-même,
  - Plus grand débit accès mémoire



Architecture

- PC: adresse prochaine instruction à exécuter
- SP: adresse du sommet de pile
- Registres généraux de calcul
- Registre d'état
  - Bits de conditions (Z, LT, GT, etc.)
  - Modes d'exécution
    - Niveau de protection (user/supervisor)
    - Adressage physique/virtuel
  - Masque des interruptions

RISC vs. CISC

- RISC (Reduced Instruction Set Computer)
  - Choisit les instructions et les modes d'adressage les plus utiles
  - Peuvent être réalisée en hardware directement
- CISC (Complex Instruction Set Computer)
  - Choisit les instructions et les modes d'adressage qui rendent l'écriture de compilateurs plus simple
  - Nécessite l'utilisation de micro-code

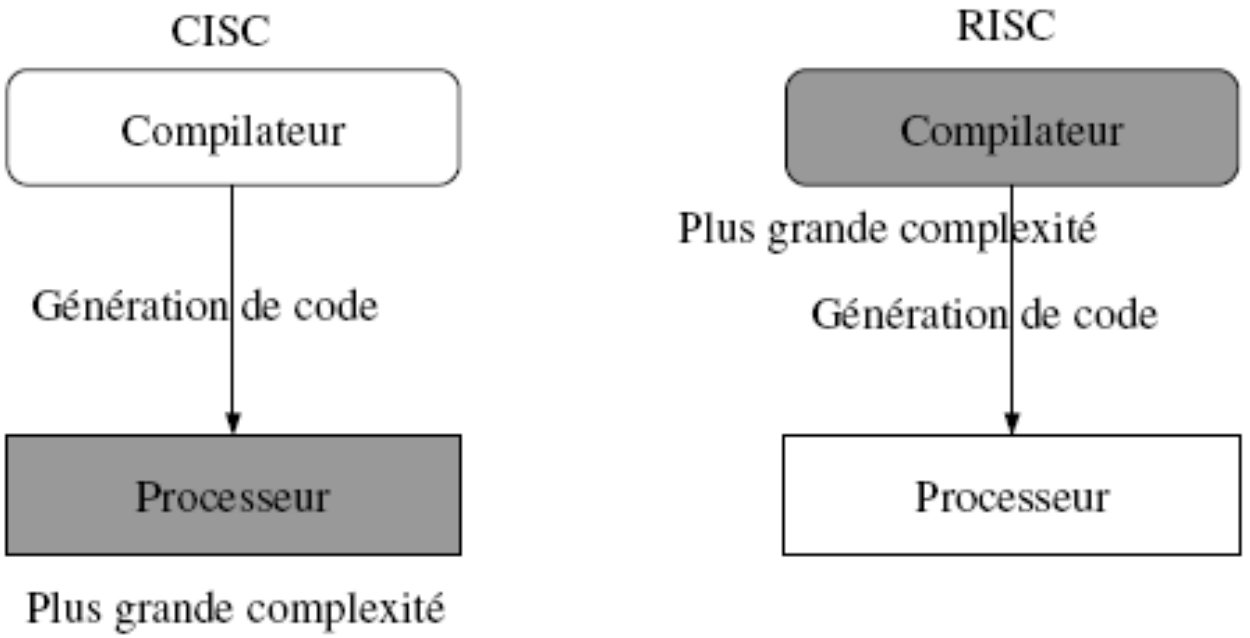
RISC vs. CISC

- Avantages CISC
  - Réduit le nombre d'instructions d'un programme
- Inconvénients CISC
  - Temps d'exécution (des instructions) plus long
  - Processeur plus complexe et donc plus lent

RISC vs. CISC

- RISC:
  - Place disponible sur la puce pour de nombreux registres généraux et des caches mémoire
  - écriture de compilateur et programmation assembleur difficile
- CISC:
  - Peu de place sur la puce: peu de registres et caches de petite taille
  - écriture de compilateur et programmation assembleur facile

RISC vs. CISC


CISC
Compilateur
Génération de code
Processeur
Plus grande complexité
RISC
Compilateur
Plus grande complexité
Génération de code
Processeur

Architecture RISC

- Nombre d'instructions réduit
  - Une instruction par cycle
  - Instructions de taille fixe => facile à pré-charger
- Pipelines
  - Traitement d'une instruction est décomposé en unités élémentaires
  - Exécution de ces unités en parallèle

Architecture RISC

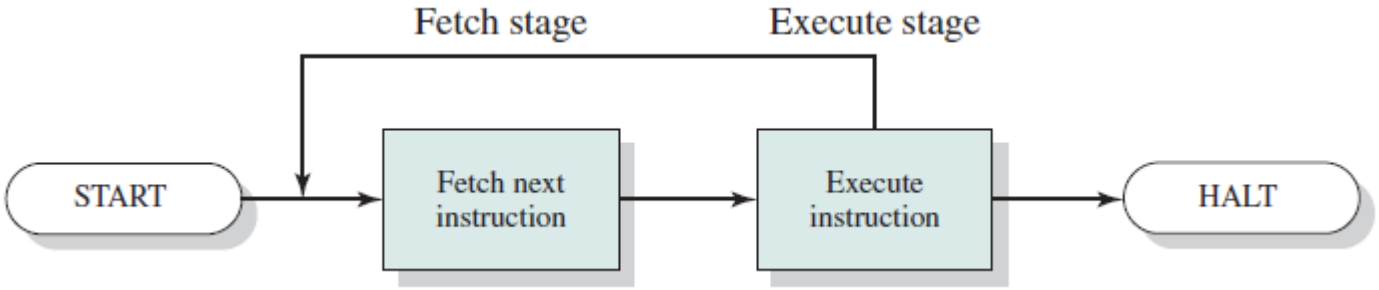
- Registres
  - Grand nombre peu spécialisés (adresses ou données)
- "Load-Store"
  - processeur opère seulement sur les données des registres
  - Instructions spécifiques pour lire un mot mémoire dans un registre, et pour écrire un registre en mémoire.

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d’exécution
- Exceptions & Interruptions
- Les différentes mémoires

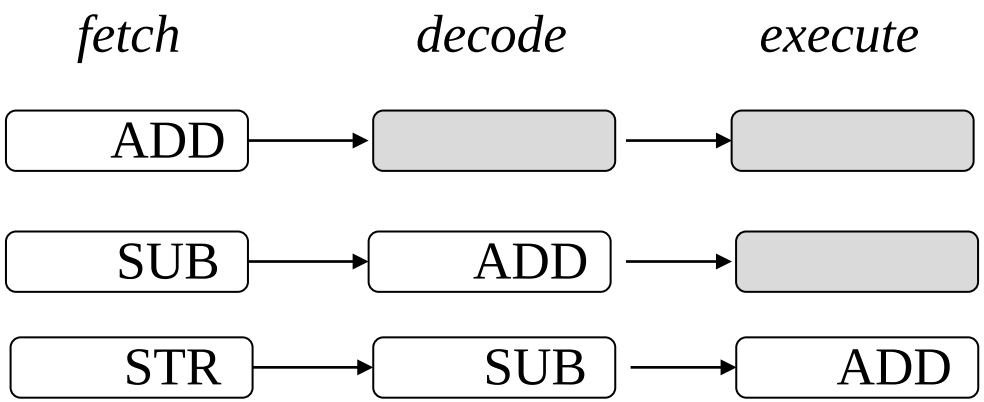
Exécution d'une instruction

- Cycle d'exécution:
  - Lecture (*fetch-stage*) des instructions de la mémoire,
  - Exécution (*execution-stage*) de l'instruction.
  - Le PC est incrémenté après le fetch,
  - L'instruction est chargée dans le registre d'instruction (IR) et il est décodé,
  - Quatre type d'instruction : Transfert mémoire-registre, Transfert processeur entrée sortie, Calcul de donnée ou Control



RISC et pipe line

- Exemple du ARM7 : 3 niveaux
  - Fetch: charge une instruction depuis la mémoire
  - Décode: identifie l'instruction à exécuter
  - Exécute: exécute l'instruction et écrit le résultat dans le registre



Pipeline

- Instructions de branchement
  - vidage (***flush***) du pipeline
  - **Pourquoi** ?

Pipeline et PC

- Le PC (au niveau *execute*) pointe sur l'adresse de l'instruction courante + n octets
  - Important quand on calcul un offset relatif,
  - Exemple
    - 0x8000 LDR pc,[pc,#0]
    - 0x8004 NOP
    - 0x800D DCD JumpAddress

- Quelle adresse pointe le PC lorsque s'exécute le LDR ?

Pipeline et PC

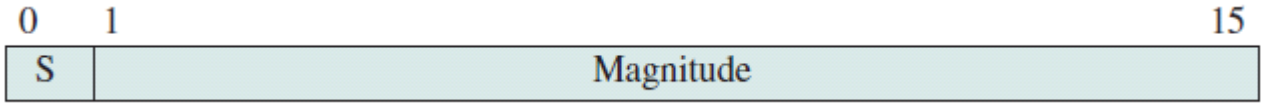
- Le PC (au niveau *execute*) pointe sur l'adresse de l'instruction courante + octets
  - Important quand on calcul un offset relatif,
  - Exemple
    - 0x8000 LDR pc,[pc,#0]
    - 0x8004 NOP
    - 0x800D DCD JumpAddress

PC+8 : 0x8008

Exécution d'une instruction

- Soit un processeur 16 bits imaginaire,
  - Des registres : PC, MAR (Memory Adress Register, IR (Instruction Register) et AC un accumulateur
  - Au moins trois opcodes
    - 0001 charge AC par un contenu mémoire
    - 0010 stocke AC dans la mémoire
    - 0101 additionne dans AC depuis la mémoire
- Un format d'instruction

- Le format d'un entier



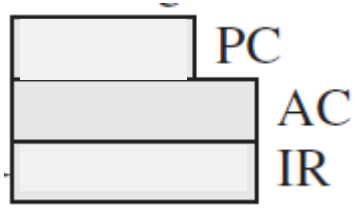
Exécution d'une instruction

- Soit le programme suivant situé en 0x300:
  - 1940, 5941 et 2941

Mémoire programme

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1

Registres du processeur



Mémoire donnée

940	0	0	0	3
941	0	0	0	2

Exécution d'une instruction (1)

Étape de *Fetch*

Mémoire programme

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1

Mémoire donnée

940	0	0	0	3
941	0	0	0	2

Registres du processeur

3	0	0		PC
				AC
1	9	4	0	IR

Étape de *Execute*

Mémoire programme

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1

Mémoire donnée

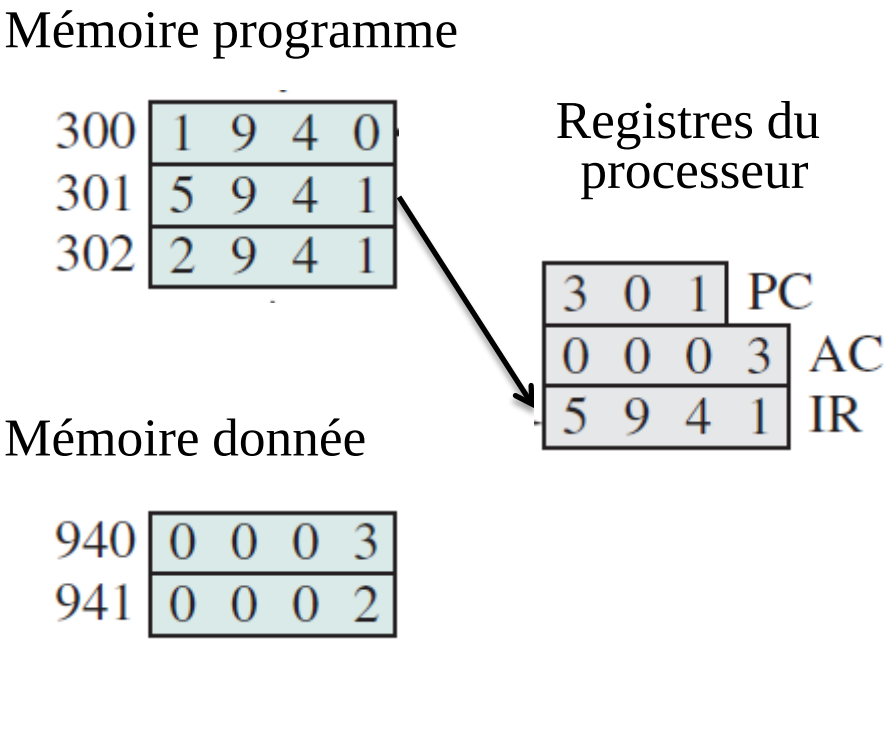
940	0	0	0	3
941	0	0	0	2

Registres du processeur

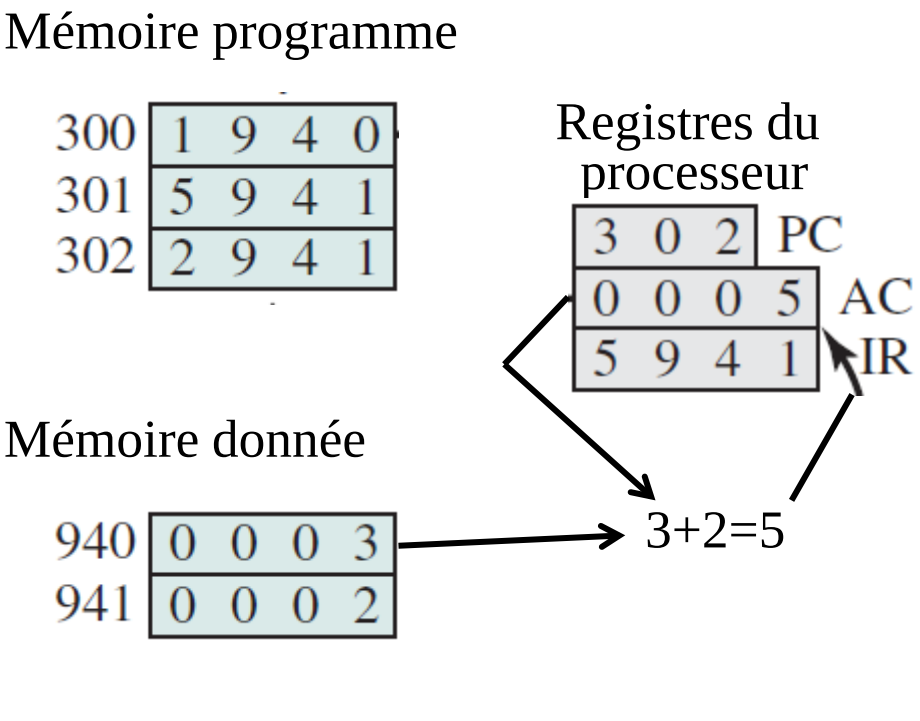
3	0	1		PC
0	0	0	3	AC
1	9	4	0	IR

Exécution d'une instruction (2)

Étape de *Fetch*



Étape de *Execute*



Exécution d'une instruction (3)

Étape de *Fetch*

Mémoire programme

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1

Registres du processeur

3	0	2		PC
0	0	0	5	AC
2	9	4	1	IR

Mémoire donnée

940	0	0	0	3
941	0	0	0	2

Étape de *Execute*

Mémoire programme

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1

Registres du processeur

3	0	3		PC
0	0	0	5	AC
2	9	4	1	IR

Mémoire donnée

940	0	0	0	3
941	0	0	0	5

Exercice

- Supposons que le processeur dispose de deux instructions d'accès I/O
  - 0011 Charge AC depuis I/O
  - 0111 Stocke AC dans I/O
- Dans ce cas, l'adresse sous 12 bits identifie un périphérique particulier. Montrez l'exécution du programme suivant en utilisant le format précédent:
  - Charge AC depuis le périphérique 5.
  - Additionne son contenu avec la mémoire 940.
  - Stocke le résultat dans le périphérique 6.
- Hypothèse la valeur dans le périphérique 5 est 3 et en 940 est stockée la valeur 2.

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d’exécution
- Exceptions & Interruptions
- Les différentes mémoires

Mode Superviseur

- Accès aux ressources protégées
  - Registres d'I/O, de gestion de la mémoire
  - Registre contenant le mode d'exécution
  - Toute la mémoire physique, tous les périphériques
- Droit d'exécuter les instructions *privilégiées*
  - Masquage des interruptions
  - Changement de mode d'exécution
  - Instructions d'I/O
- Processeur démarre en mode superviseur

Mode Utilisateur

- Droits restreints
  - Accès aux registres généraux
  - Exécution des instructions standard
- Passage superviseur -> utilisateur
  - Contrôlé par OS
  - Lancement d'un programme
- Passage user -> superviseur
  - Instruction *svc* (supervisor call)
  - Opération invalide ou non autorisée =>exceptions

Supervisor call

- Utilisé pour invoquer un service du système
  - service id. (open, write, ...) dans un registre
  - service args (pathname, fd, ...) dans registre(s) ou dans la pile utilisateur
  - résultat / code erreur dans registre(s)
- Synchrone avec (processus / thread) appelant
  - relatif au contexte système appelant
- Appels système fournis par fonctions de bibliothèque du langage (libc)

Modèle du programmeur ARM

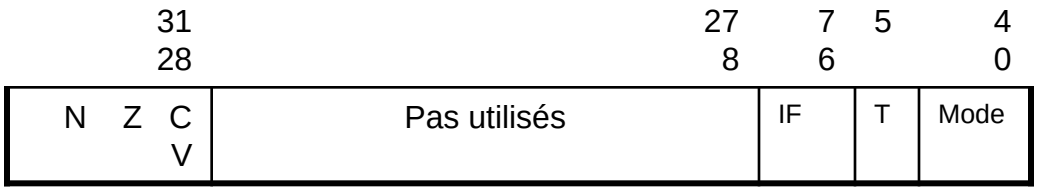
- 1 mode utilisateur non privilégié
- 5 modes système, privilégiés :
  - Interrupt (standard) _irq
  - Fast interrupt _fiq
  - Abort _abt
  - Indéfini _und
  - Software interrupt _svc
  - + tâches Système

Des registres (mode User)

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15

- 16 Registres de 32 bits
- R0-R10, R12 : généraux
- R11 (fp) Frame pointer
- R13 (sp) Stack pointer
- R14 (lr) Link register
- R15 (pc) Program counter

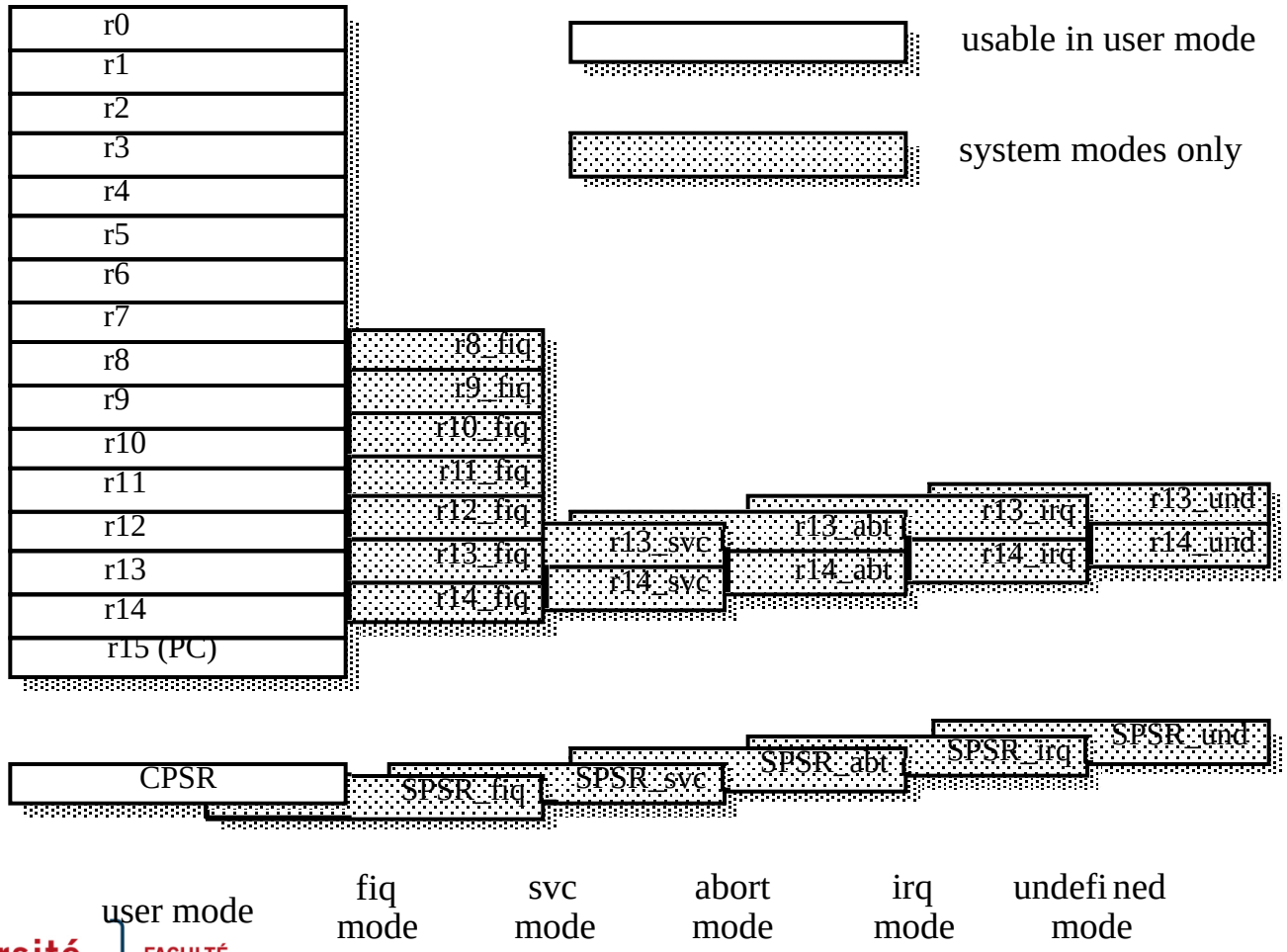
- Current Program Status Register



Modèle du programmeur ARM

- Associés aux divers modes de travail, des registres différents sont disponibles :
- **r13** (stack pointer) et **r14** (link register) sont distincts dans tous les modes
- **r8 à r12** sont distincts en mode **fiq,** ils ont été prévus pour réaliser une fonction DMA par logiciel sans variables en mémoires
- Les autres registres sont visibles dans tous les modes

Registres


usable in user mode
system modes only
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15 (PC)
r8_fiq
r9_fiq
r10_fiq
r11_fiq
r12_fiq
r13_fiq
r14_fiq
r13_svc
r14_svc
r13_abt
r14_abt
r13_irq
r14_irq
r13_und
r14_und
CPSR
SPSR_fiq
SPSR_svc
SPSR_abt
SPSR_irq
SPSR_und
user mode
fiq mode
svc mode
abort mode
irq mode
undefined mode

Modèle du programmeur ARM

- Les registres **SPSR_**xxx :
  - **S**aved **P**rogram **S**tatus **R**egister
  - Utilisés lors des appels des exceptions :
  - Copie local du **CPSR** (**C**urrent **PSR**) à restaurer en fin de procédure
  - Le nouveau mode dépend de la source de changement de mode

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d’exécution
- Exceptions & Interruptions
- Les différentes mémoires

Les interruptions

- Mécanisme pour optimiser le traitement par le processeur,
  - Les I/O sont toujours plus lentes que le processeur,
  - Un PC à 1 Ghz exécute environ 109 instruction à la seconde,
  - Un disque dur tourne à 7200 tr/mn soit une demi rotation toutes les 4 ms.

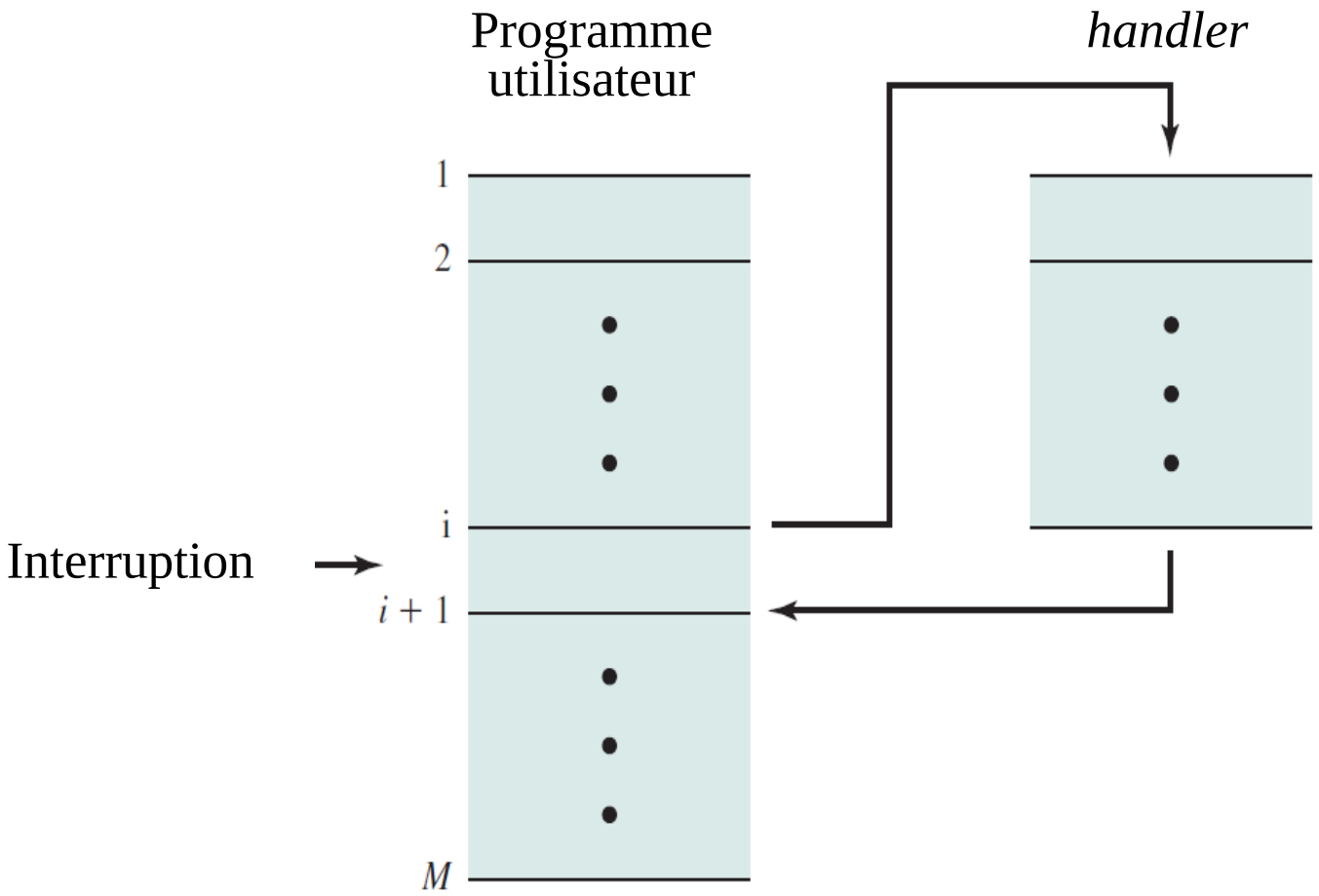
Interruptions

- Événement déclenché par composant matériel
  - Fin d'entrée/sortie d'un périphérique
  - Échéance de temps terminée
- Signal envoyé au processeur de l'extérieur
  - Par l'intermédiaire d'un PIC
  - Partageable ou non
- Événement asynchrone
  - Indépendant du programme courant
  - Masquable par le système

Exceptions

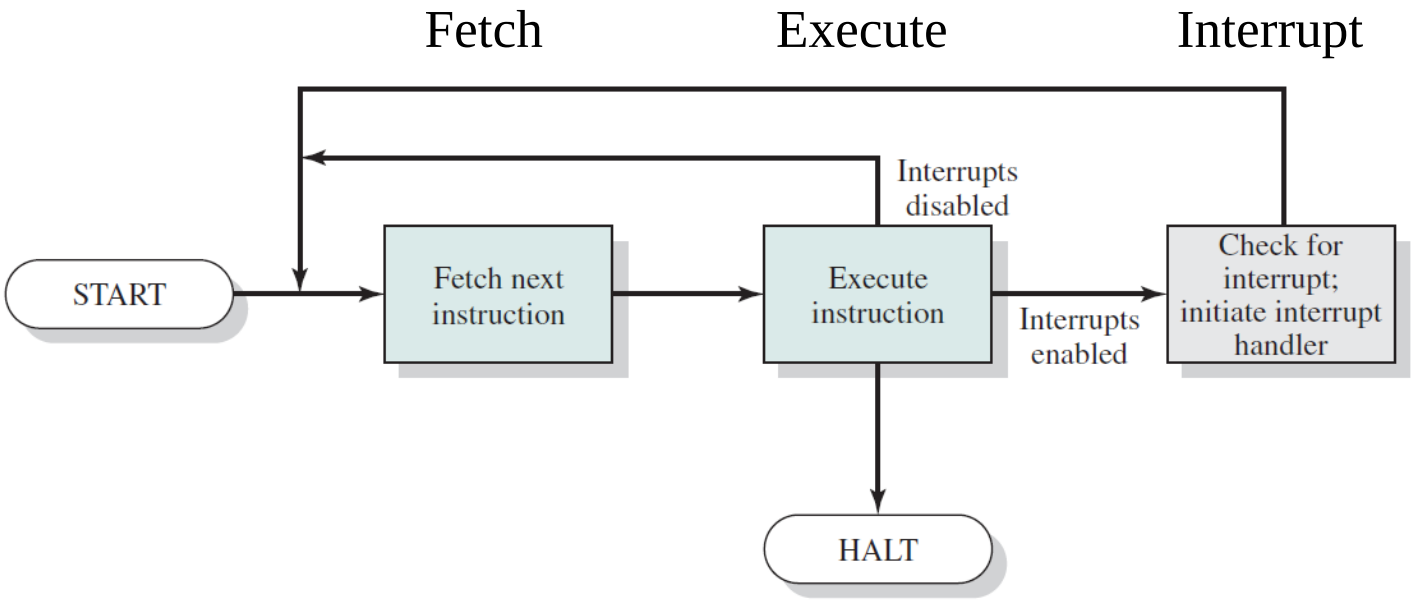
- Provoquée par une condition exceptionnelle lors de l'exécution de l'instruction courante
  - Adresse mémoire invalide
  - Instruction non autorisée ou invalide
  - Division par zéro
- Synchrone avec le programme exécuté
  - Détectée par le CPU
  - Traitée dans le contexte du programme courant

Transfert de contrôle vers un *handler*



Conséquence sur le cycle

- Rajout d'une évaluation des condition dans le cycle d'exécution,
- Si présence met à jour son registre d'état,



Traitement Exceptions/Interruptions (1)

- Processeur suspend l'exécution en cours
- Sauvegarde état courant
  - PC, SP, registre d'état
  - Dans des registres dédiés ou dans la pile superviseur
- Ou utilise des *shadow registers*
  - 2 jeux de registres, un par mode d'exécution

Traitement Exceptions/Interruptions (2)

- Charge nouveau contexte d'exécution :
  - Registre d'état avec mode « superviseur »
  - PC avec adresse mémoire spécifique
- Opérations de sauvegarde et de chargement
- Non-interruptible
  - Sinon état non cohérent
  - Sauf par NMI, sur PowerPC par exemple
- Si exception durant opérations (« double faute»)
  - => arrêt CPU ou traitement spécial (Intel)

Exceptions/Interruptions "Vector Table"

- Adresse de branchement
  - Dépend de l'exception/interruption
- Vers une table de vecteurs localisée
  - à adresse prédéfinie (0x00000000 par exemple)
  - dans un registre (IDTR sur Intel)
- Contient une instruction de branchement
- Table de vecteurs initialisée par le système

Table des vecteurs

- La table de vecteur contient une instruction et non pas une adresse,
- En général une instruction de branchement sur une fonction appropriée

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Exceptions ARM, entrée

- Lors d'une exception **le processeur** :
  - Termine l'instruction courante
  - Change le mode selon l'exception
  - Sauve l'adresse de l'instruction qui suit dans r14 (**pc → lr**, ***link register***)
  - Copie **CPSR → SPSR** du nouveau mode
  - Interdit interruption IRQ: 1→ CPSR:bit I(7)
  - Si exception FIQ, interdit FIQ: 1 → CPSR:bit F(6)
  - Met adresse vecteur → pc (r15)
  - **La recherche de la source d'interruption matérielle est à réaliser par logiciel** !

Exceptions ARM, sortie

- Lors de la fin d'une exception **la routine d'exception** :
  - Restaure les éventuels registres modifiés depuis la pile
  - **SPSR → CPSR**
  - Récupère l'adresse de retour dans r14 **(lr → pc)**

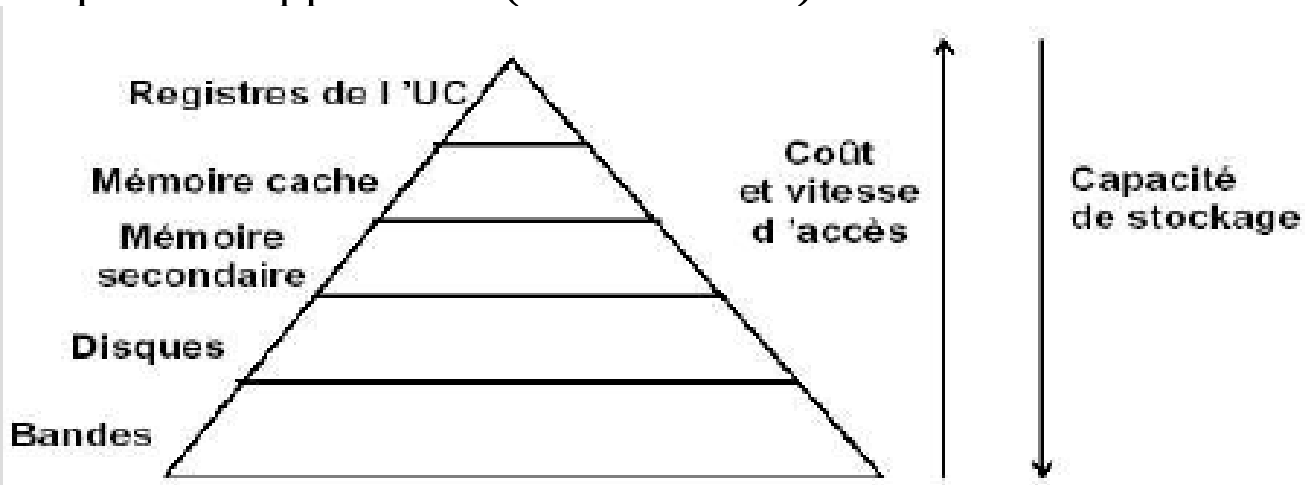
Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d’exécution
- Exceptions & Interruptions
- Les différentes mémoires

Gestion de la mémoire

Le matériel (données AMD Athlon 64):

- Les registres du processeurs – (tps d'accès moyen à un mot 64 bits : 0,3 ns)
- Cache de niveau 1 : à l'intérieur du processeur - 64 ko (1 ns)
- Cache de niveau 2 : à l'intérieur – SRAM de de 512 ko
- Cache de niveau 3 : à l'extérieur – SRAM de de 2Mo
- Mémoire principale : SDRAM – jusqu'à 4Go (de 5 ns à 100 ns))
- Disque dur : qq 100 Go (de 1us à 1ms)



Rôle de la mémoire

- Faible nombre de registres : on ne peut presque rien faire sans la mémoire.
- A part des calculs numériques, les algorithmes parcourent des structures de données complexes représentées en mémoire.
- Il y a alors des accès mémoires toutes les 2-5 instructions...
- Goulot d'étranglement
  - Les opérations mémoires sont plus lentes que les instructions des processeurs (croissance plus rapides).
  - Plusieurs niveaux de caches.

Des temps d’accès

- Niveau 1 : 1 ko tps d’accès 0.1 ms
- Niveau 2 : 100 ko tps d’accès 1ms
- Si 95% des accès mémoire sont dans le cache quel est le temps d’accès moyen d’un octet ?

Les types de mémoires

- Deux catégories :
  - volatile : elles doivent être alimentées en électricité pour conserver les informations (RAM et dérivés)
  - non-volatile : elles conservent les informations même non alimentées (ROM, EEPROM, Flash, etc.)
- La ROM
  - Non modifiable, pas cher, très miniaturisable,
  - Utilisé dans les systèmes embarqués et divers composants d'un ordinateur ne nécessitant pas de mise à jour,
- L'EEPROM
  - Electrically Erasable Programmable Read-Only Memory
  - Effaçable, utilisée dans ?

Les types de mémoires

- Deux catégories :
  - volatile : elles doivent être alimentées en électricité pour conserver les informations (RAM et dérivés)
  - non-volatile : elles conservent les informations même non alimentées (ROM, EEPROM, Flash, etc.)
- La Feram
  - Quel type ?

Les types de mémoires

- Deux catégories :
  - volatile : elles doivent être alimentées en électricité pour conserver les informations (RAM et dérivés)
  - non-volatile : elles conservent les informations même non alimentées (ROM, EEPROM, Flash, etc.)
- La Feram
  - Quel type ?
- La FLASH Nand
  - Quel type ?
  - Accès ?

Une mémoire série…

- Accès par un bus série, et dé-sérialisation en interne,
- Mémoire RAM tampon pour la dé-sérialisation,
  - Accès par shadowing uniquement,
  - Le contrôleur gère la copie de la RAM vers la Nand,
- Pages :
  - Unité minimale de lecture (perte d'efficacité pour les petits fichiers)
  - Taille des pages 4Ko + overhead,
- Bloc
  - Unité minimale d'écriture,
  - Un bloc contient 32, 64 ou 128 pages,
  - Ecriture très lente !!!

Rôle de la mémoire

- La sécurité passe par la mémoire
  - Au minimum : zone système et zone utilisateur.
  - Si possibles, protéger des zones plus fines parmi les zones utilisateurs.
- La modularité passe par la mémoire
  - Avoir plusieurs programmes chargés en parallèle.
  - Partageant du code.
- Besoin de support matériel
  - Pour pouvoir faire les vérifications de sécurité.
  - Pour les faire efficacement.
  - Pour permettre et faciliter le partage de code.

Concept de base en gestion mémoire

Manipulation symbolique implicite des adresse : variables et références

- Localisation (adresse) masquée par le système qui gère la mémoire,

L'abstraction mémoire

- Chaque processus voit la mémoire comme si elle était tout entière à lui seul avec un espace d'adressage énorme.
- C'est un espace linéaire organisé en régions contiguës protégées (séparées par des trous).
- Le matériel et le logiciel permettent :
  - De garantir la sécurité (protection mémoire).
  - Un partage de la mémoire entre processus.
  - Sans surcoût au runtime ; au contraire. . .
- Les adresses virtuelles sont traduites en adresses physiques par le **Memory Management Unit** (unité spécialisée du processeur)

Le cache

- Le processeur accède lors de chaque instruction à la mémoire même si il ne manipule pas de donnée pourquoi ?

Cache et localité

- Soit le code suivant :

```
for (i=0; i<20; i++)
  for (j=0; j<10; j++)
    a[i]= a[i] * j
```

  - Donnez un exemple de localité spatiale dans ce code
  - Donnez un exemple de localité temporelle dans ce code

Transferts de Données

- Par interruption,
- Indirects à travers des registres
  - Simple mais pas efficace
  - Composants lents (clavier, souris, etc.)
- Par accès direct en mémoire (**DMA**)
  - Données copiées par composant dans les buffers du driver
  - Efficace (copie en parallèle avec CPU)
  - Composants rapides (disques, ethernet, USB, etc.)
  - Complexe
    - Buffers pas dans cache / « bus snooping » (Intel)
    - Continue après arrêt brutal puis re-démarrage à chaud

DMA

- Lorsque le processeur veut lire / écrire un bloc de donnée il envoi un signal au DMA comprenant,
  - Le type d'opération (R,W)
  - L'adresse du composant d'IO impliqué (usb, graphique,..)
  - Le début de l'adresse à lire ou écrire,
  - Le nombre de mots concernés.
- Le processeur continue son exécution, le DMA transfert les données.
  - Transfert mot par mot, en gérant le bus de données,
  - En cas de compétition : ajout de *t_wait* au processeur,
  - Signale la fin par une interruption au processeur.

DMA

- Un module de DMA transfert des caractères vers la mémoire principale à partir d'un périphérique transmettant à 9600 bits par seconde.
- Le processeur peut *fetcher* des instruction à 1 million d'instructions par seconde,
- Quel est le ralentissement du processeur du au DMA ?

Données globales / locales

```
extern char c_array[];
    /* global public to all files */
static int i_array[2048];
    /* global private to file */
int /* return value */
func (int i, char* name) {
    /* call arguments */
int* ptr;
    /* local to func() */
struct example_t ex;
    /* local to func() */
static char c_pfgd[100];
    /* global private to func() */
```

La pile

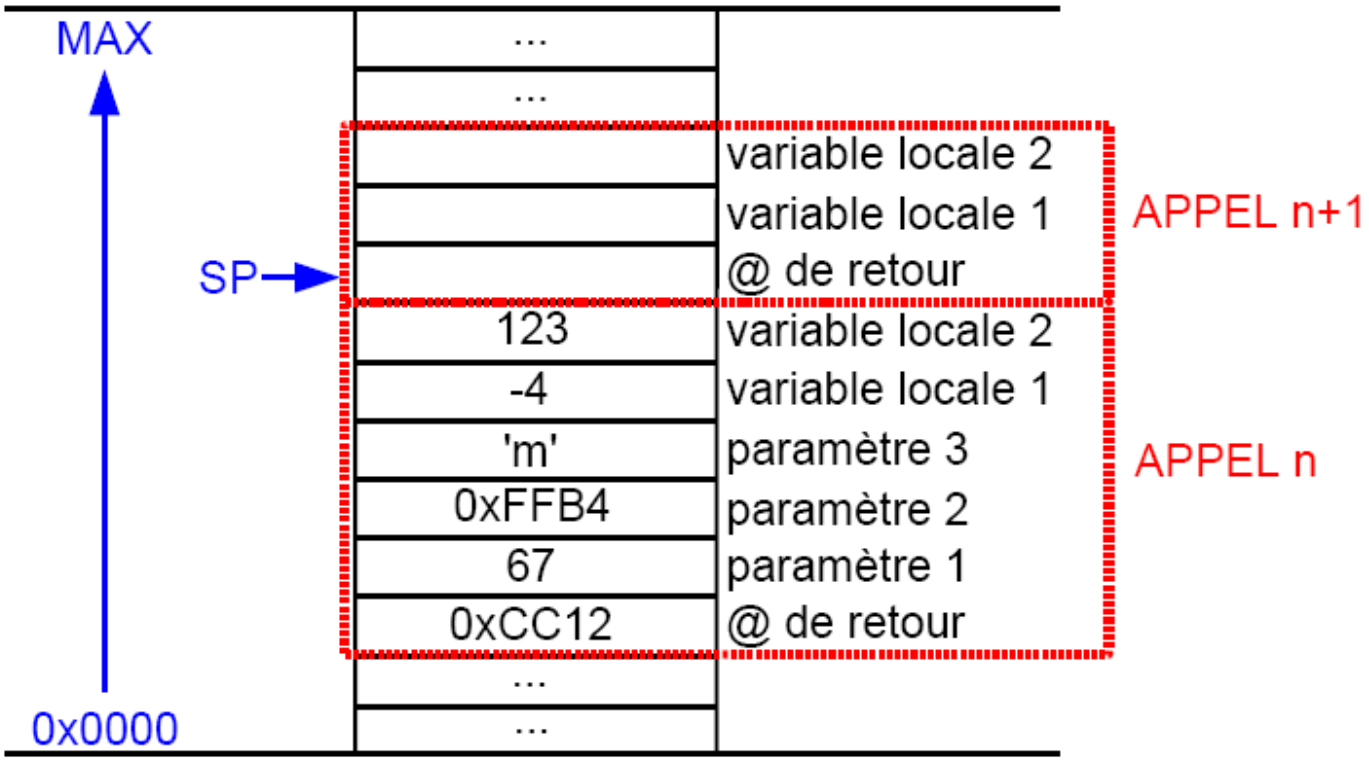
Zone « dans laquelle s'exécute le programme »

- paramètres, variables locale, adresse et valeur de retour de fonction / routine / méthode
- croît à chaque appel, décroît à chaque retour

Automatique: géré par l'environnement d'exécution (*runtime*)

+/- invisible du programme(ur)

La pile

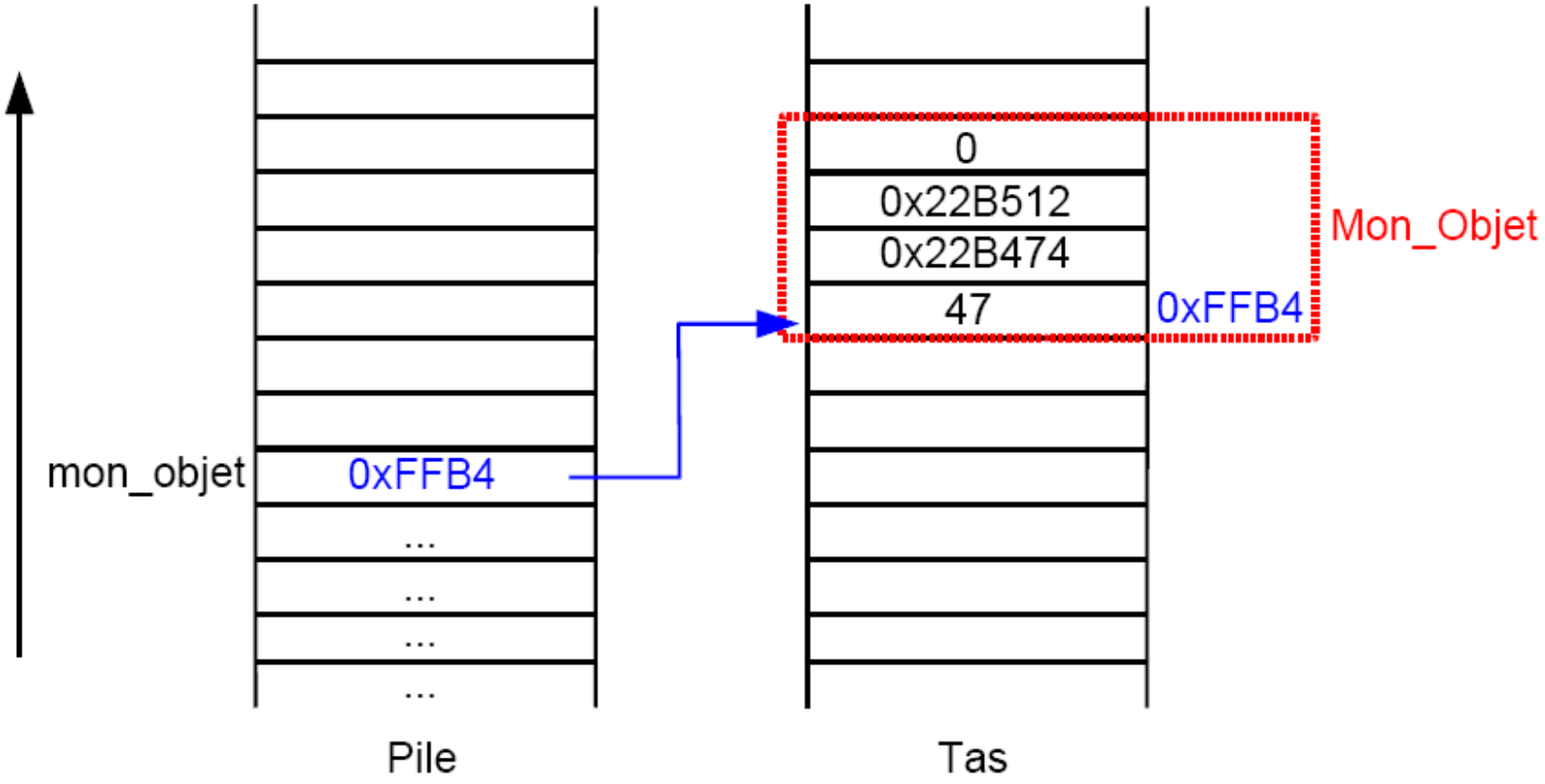


- NB: Ici, pile croissante avec adresses croissantes.
- En pratique, la pile croît souvent vers 0x0000.

Le tas

- Zone où le programme(ur) alloue toutes ses données qui ne sont pas en pile
- Allocation explicite: malloc C
  - `MonObjet*mon_objet=(MonObjet*)malloc(sizeof(MonObjet));`
- Allocation « implicite »: new en Java, C++...
  - `MonObjet mon_objet = new MonObjet();`

Le tas

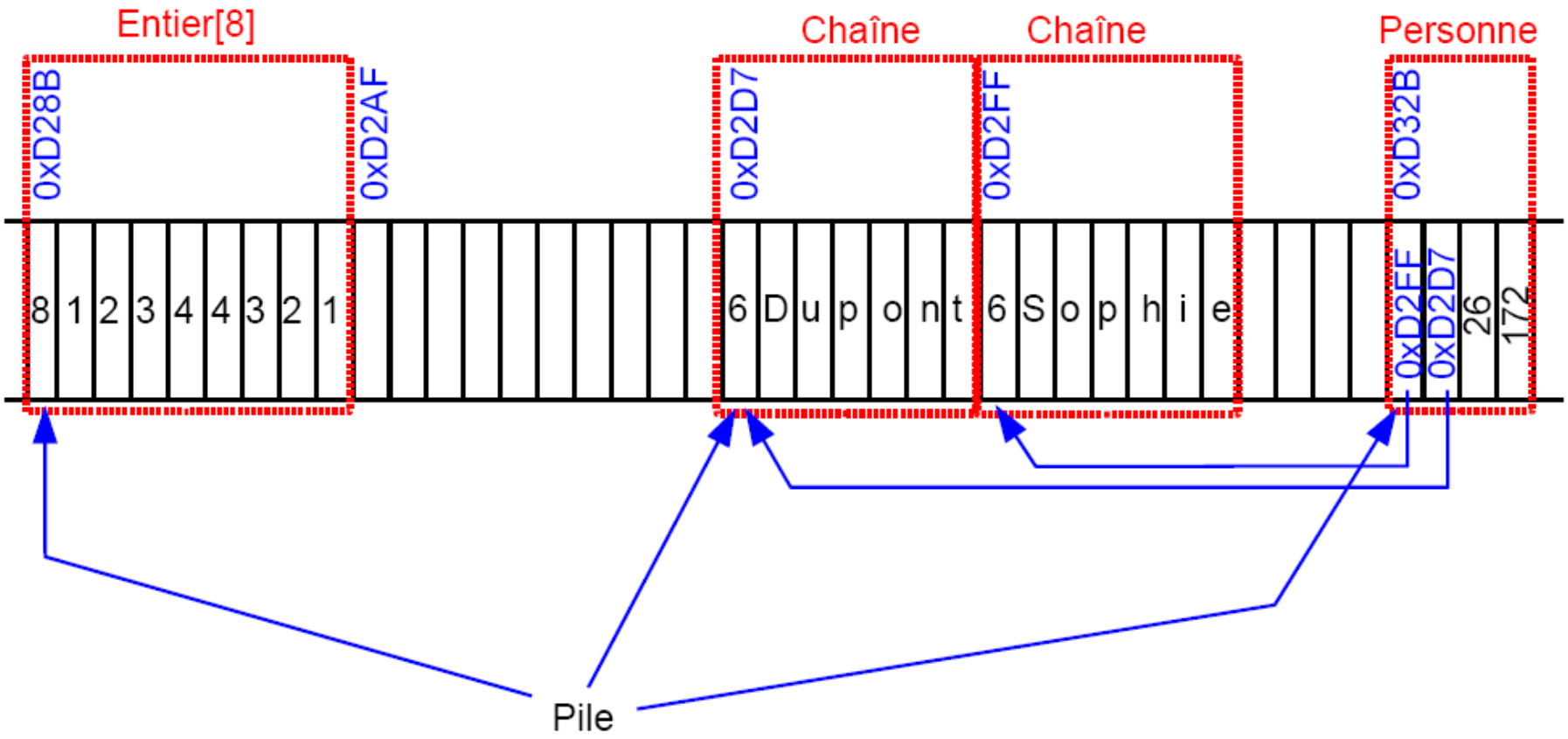

mon_objet
0xFFB4
...
...
...
...
Pile
0
0x22B512
0x22B474
47
0xFFB4
Mon_Objet
Tas

Le tas

Zone « désordonnée », contrairement à la pile (d'où les noms...)

La gestion mémoire concerne principalement le tas.

Le tas


Entier[8]
Chaîne
Chaîne
Personne
0xD28B
0xD2AF
0xD2D7
0xD2FF
0xD32B
8 1 2 3 4 4 3 2 1
6 D u p o n t
6 S o p h i e
0xD2FF
0xD2D7
26
172
Pile

Appels système

- Quelles sont les trois méthodes pour passer des paramètres au SE ?

Le processus de démarrage

- Les mémoires du processeur et la mémoire centrale est volatile
- Il faut donc un programme stocké dans une mémoire non volatile
- BIOS : Basic Input Output System
- Lors du démarrage le processeur met automatiquement son PC à l'adresse du BIOS

Démarrage

- Le BIOS effectue les opérations suivantes:
  - phase de test
  - chargement de la configuration depuis la mémoire du BIOS
  - le BIOS cherche un périphérique de stockage bootable
  - chargement des 512 premiers octets de ce périphérique contenant le bootstrap
  - Comment continuer le processus de boot ?

Processus de démarrage

- Le noyau est à son tour chargé par le *bootloader*
- Il configure le processeur:
  - la table des interruptions
  - le contrôleur de mémoire (MMU)

Any question ?