Projet de Programmation Système – CS207-a – 2019

Description générale du projet principal :

Simulation de la mémoire cache d'un processeur Intel Kaby Lake

Plan de ce document

1. Contexte du projet
2. Buts du projet
3. Description générale
4. Glossaire

Contexte

Le but premier de ce projet est de vous faire développer un large programme en C sur une thématique « système ». Le cadre choisi cette année est la simulation de la hiérarchie de mémoires d'un processeur Intel Kaby Lake. Le but de ce projet **n'est pas** du tout de se substituer au cours de *Computer Architecture* que vous avez eu le semestre passé (ni même de le compléter), mais simplement de vous faire développer, en suivant une spécification pas à pas, une version simplifiée de concepts proches de ceux qui vous y ont été présentés.
Tous les concepts de base requis pour ce projet sont introduits ici de façon simple en ne supposant qu'une connaissance « utilisateur » standard d'un système informatique. En cas de doutes sur la terminologie, le glossaire en fin de document peut se révéler pratique.

Vous allez développer une version simplifiée de la mémoire cache d'un processeur. Les processeurs modernes utilisent des hiérarchies de mémoires et des « *translation lookaside buffer* » (TLB) pour accélérer le temps d'accès à l'information. Parallèlement, le systèmes d'exploitation modernes utilisent une représentation virtuelle de la mémoire physique. C'est tout cela que ce projet se propose de simuler :

- traduction d'adresses mémoires virtuelles en adresses mémoire physiques, et réciproquement ;
- simulation (et gestion) de plusieurs sortes de TLB ;
- simulation de mémoires cache hiérarchiques.

Buts et organisation du projet

Durant les 10 semaines de ce projet, vous allez devoir implémenter, graduellement morceau par morceau les composants clés mentionnés ci-dessus et décrits plus bas puis détaillés dans les sujets hebdomadaires.

Vous allez aussi devoir développer des tests complémentaires utiles pour observer et analyser le fonctionnement du système. Ces tests seront développés sous formes d'exécutables indépendants du cœur principal.

Afin de faciliter au mieux l'organisation de votre travail (dans le groupe et dans le temps), voici une représentation synthétique des différents modules logiciels impliqués dans ce projet :

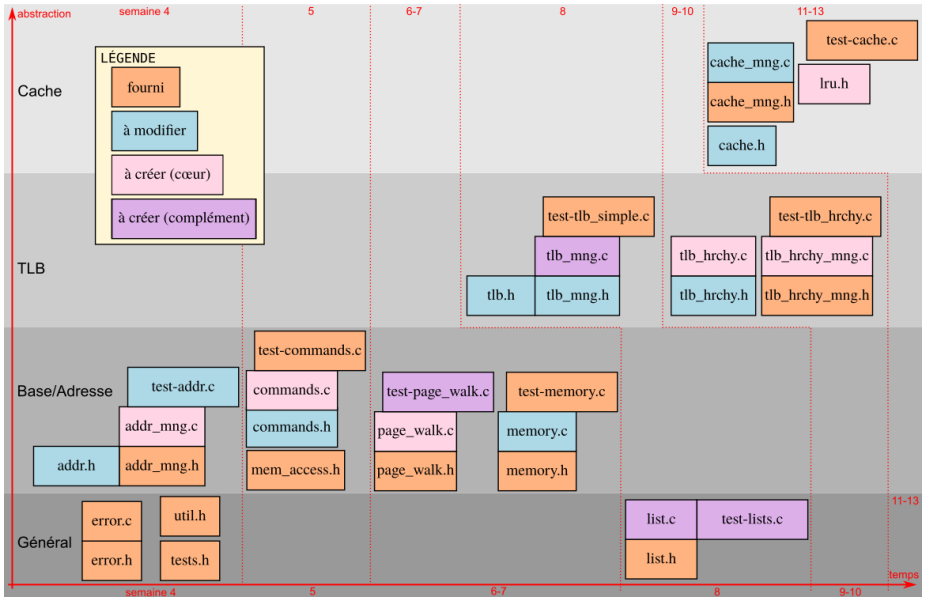

Figure 1: Modules du code du projet

Toujours pour vous organiser au mieux, veuillez également consulter la page de barème du cours (et la lire en entier !!).

Description générale

Nous décrivons ici de façon générale les principaux concepts et structures de données que ce projet nécessitera. Leurs détails d'implémentation seront précisés plus tard lorsque nécessaire dans le sujet hebdomadaire correspondant.

Mémoires hiérarchiques

Les processeurs 64-bits modernes tels que l'i7 Kaby Lake d'Intel utilisent 48-bits d'adressage virtuel et peuvent ainsi adresser un espace de 256 Tio de mémoire virtuelle. La hiérarchie de mémoire utilisée par de tels processeurs est composées de plusieurs niveaux de TLB et plusieurs niveaux de cache. Dans ce projet, nous allons considérer un système composé de :

- une hiérarchie de TLBs à deux niveaux :

1. deux TLBs de niveau 1 : un pour les instructions et un pour les données ;
2. un TLB commun (instructions et données) de niveau 2 ;

- une hiérarchie de mémoires cache à deux niveaux :
  1. deux mémoires cache de niveau 1 : une pour les instructions et une pour les données ;
  2. une mémoire cache commune (instructions et données) de niveau 2 ;

comme illustré sur la figure suivante :

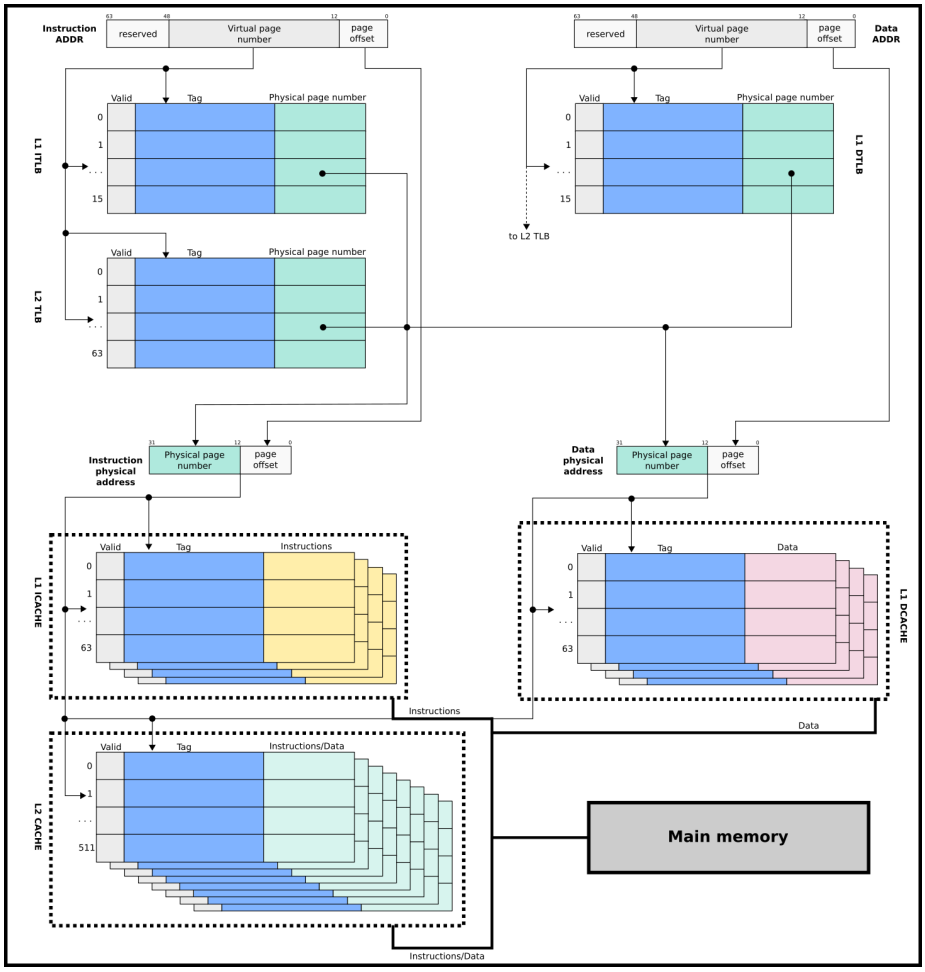

Figure 2: Illustration du modèle mémoire utilisé dans ce projet : 3 TLBs et 3 mémoires cache

Traduction d'adresses mémoire virtuelles

Bien que les processeurs Intel 64-bits aient des adresses virtuelles de, bien sûr, 64 bits, seuls les 48 bits de poids faibles sont effectivement utilisés (pour le moment) :

- les bits de 0 à 11 (c.-à-d. les 12 derniers bits, les 12 bits de poids faible) représentent l'*offset* dans la page, c.-à-d. la position par rapport au début de la page de mémoire physique (plus de détails plus tard) ;
- les bits 12 à 47 sont utilisés pour traduire, de façon hiérarchisée (arbre) l'adresse virtuelle en l'adresse d'une page de mémoire physique.

Cette traduction hiérarchisée est composée de 4 niveaux, nommés de haut (plus abstrait) en bas (adresse physique) :

- **PGD**, Page Global Directory ;
- **PUD**, Page Upper Directory,
- **PMD** Page Middle Directory,
- **PTE**, Page Table Entries.

Les bits 12 à 47 de l'adresse virtuelle s'interprètent alors comme suit (voir la figure ci-dessous), par tranches de 9 bits :

- les bits 12 à 20 représentent l'*offset* de l'entrée correspondante dans la page table PTE ; c.-à-d. sa position par rapport au début de PTE (ou encore, si l'on considère PTE comme un tableau, simplement son index dans ce tableau) ;
- les bits 21 à 29 représentent l'*offset* dans le page middle directory PMD ;
- les bits 30 à 38 représentent l'*offset* dans le page upper directory PUG ;
- et enfin les bits 39 à 47 représentent l'*offset* dans le page global directory PGD.

La taille d'une page physique est de 4 Kio et les offsets indexent des octets (et non pas des mots du processeur). Tous ces détails seront bien sûr reprécisés lorsque nécessaire.

Voici une illustration de ces conventions :

Page directories et pages d'instructions/de données

Normalement, chaque processus a un pointeur vers son propre PGD. Nous allons cependant simplifier dans ce projet et supposer que nous n'avons qu'un seul processus et donc un seul PGD.

Chaque « annuaire » (directory) est en fait un tableau de mots de 32-bits ; chacun de ces mots étant l'adresse physique du début d'une page de mémoire physique. Comme représenté ci-dessus, cette page de mémoire physique est elle-même soit un autre « annuaire » (directory), soit la page de mémoire physique finalement visée contenant les instructions ou les données recherchées :

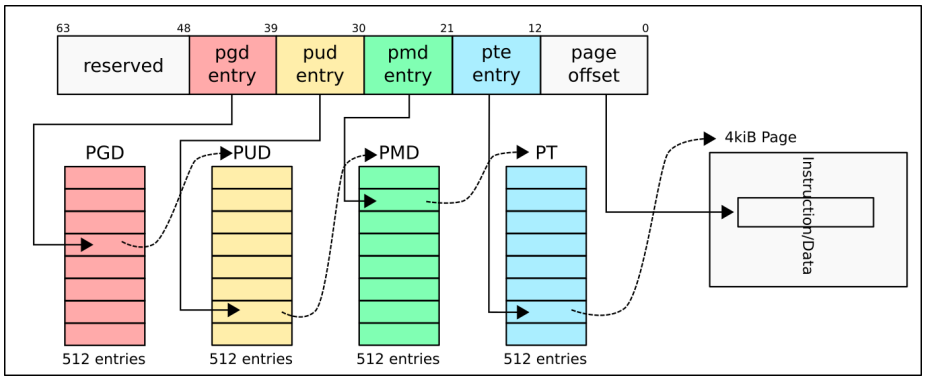

Figure 3: Utilisation des 48 bits de l'adresse virtuelle

- chaque entrée du PGD pointe vers le début d'une page physique contenant un PUD ;
- chaque entrée du PUD pointe vers le début d'une page contenant un PMD ;
- chaque entrée du PMD pointe vers le début d'une page contenant une PTE ;
- chaque entrée de la PTE pointe vers le début d'une page contenant les données ou les instructions recherchées en mémoire.

Bien sûr, toutes ces adresses, pointant vers des débuts de page, sont donc « alignées » sur 4Kio (c.-à-d. des multiples de 4Ki).

TLBs

Des traductions d'adresses virtuelles vers adresses physiques étant sans arrêt effectuées, il est au moins aussi important, sinon plus, de garder en cache de telles associations afin d'accélérer le processus d'accès à la mémoire. C'est justement le rôle des TLBs (« *translation lookahead buffers* »). Dans ce projet, tous les TLBs seront « *direct mapped* », c.-à-d. que chaque page virtuelle a sa propre entrée dans le TLB ; autrement dit, une partie des bits de l'adresse virtuelle sert également à indexer directement le TLB.

Dans ce projet, nous considérerons la hiérarchie de TLBs à deux niveaux suivante :

- deux TLB de niveau 1 (« *Level-1 TLB* ») : un pour les instructions (ITLB) et un pour les données (DTLB), contenant 16 entrées chacun ;
- un seul TLB de niveau 2 (« *Level-2 TLB* ») utilisé à la fois pour les accès aux instructions et les accès aux données, contenant 64 entrées. Ce TLB partagé sera « *inclusive* » : en cas d'échec (« *miss* ») avec le ITLB (resp. la DTLB), l'information est alors recherchée dans le TLB niveau 2 ; si l'information s'y trouve, alors cette information est **recopiée** dans le ITLB (resp. DTLB) niveau 1 ; sinon, cette information est créée (mécanisme de traduction d'adresse virtuelle en adresse physique expliqué précédemment)

puis insérée **à la fois** dans le ITLB (resp. DTLB) niveau 1 **et** dans le TLB partagée niveau 2.

Pour résumer, nous aurons dans ce projet :

- un ITLB niveau 1, direct-mapped, à 16 entrées ;
- un DTLB niveau 1, direct-mapped, à 16 entrées ;
- un TLB niveau 2 partagé, inclusive, direct-mapped, à 64 entrées.

Mémoires cache

Toutes les mémoires cache (pour instructions ou pour données ; les mémoires cache pour adresses étant nommées « TLBs » et traitées dans la section précédente) utilisée dans ce projet sont « *set associative* » et sont adressées via une adresse physique. De plus, même si par souci de généralité de conception on prévoira la possibilité de changer la politique de remplacement de ces mémoires cache, la seule politique de remplacement demandée dans ce projet sera la « *least recently used* » (LRU).

La hiérarchie de mémoires cache considérée dans ce projet sera organisée de la façon suivante :

- deux mémoires cache niveau 1 séparées : une pour les instructions (ICache) et une pour les données (DCache) ; chacune de ces deux mémoires cache sera « *4-way set associative* », avec 64 lignes de 4 mots par ligne ;
- une mémoire cache niveau 2, partagée (à la fois pour les instructions et pour les données) ; ce sera une mémoire « *8-way set associative* », avec 512 lignes de 4 mots par ligne ; cette mémoire cache partagée sera **exclusive** : elle ne contient que des données/instructions qui ne sont pas présentes dans les mémoires cache de niveau 1 et sera utilisée comme une « **victim cache** ».

Prenons un exemple : supposons que le processeur demande l'accès à un bloc de données. Trois situations peuvent se produire :

1. le bloc demandé est dans la cache niveau 1 (DCache), ce que l'on appelle un « *L1 hit* » ; il est alors simplement retourné (depuis cette cache) au processeur ;
2. le bloc demandé n'est pas dans la DCache, mais se trouve dans la cache partagée niveau 2 (« *L2 hit* ») ; ce bloc est alors **recopié** dans la DCache et **invalidé** dans la cache niveau_2 (et retournée au processeur, bien sûr !) ; si son insertion dans la DCache provoque l'éjection d'un autre bloc de celle-ci, ce bloc éjecté de la DCache est inséré dans la cache partagée niveau 2 ; la seule façon d'ajouter des informations à la cache partagée niveau 2 est par l'éjection de blocs des mémoires cache niveau 1 ;
3. le bloc demandé n'est ni dans la DCache, ni dans la cache partagée niveau 2 : il est alors recherché directement en mémoire et inséré dans la DCache ; de même que ci-dessus, si cette insertion provoque l'éjection d'un autre bloc, ce bloc éjecté est inséré dans la cache partagée niveau 2.

Pour résumer, nous aurons dans ce projet :

- une mémoire cache niveau 1 pour les instructions : 4-way set associative, 64 lignes par « *way* », 4 mots par ligne ;
- une mémoire cache niveau 1 pour les données : mêmes caractéristiques que celle pour les instructions ;
- une mémoire cache niveau 2, partagée, exclusive et « *victim cache* » ; 8-way set associative, 512 lignes par « *way* », 4 mots par ligne.

Glossaire

NOTE : Vous avez cherché un terme dans ce glossaire et ne l'avez pas trouvé ? Signalez-le nous et nous l'ajouterons.

- **DCache** : mémoire cache de niveau 1 pour les données ;
- **DTLB** : TLB de niveau 1 pour des traductions d'adresses virtuelles de données ;
- **ICache** : mémoire cache de niveau 1 pour les instructions ;
- **ITLB** : TLB de niveau 1 pour des traductions d'adresses virtuelles d'instructions ;
- **L1 TLB** : TLB de niveau 1, pour des traductions d'adresses virtuelles soit (exclusivement) de données (DTLB), soit d'instructions (ITLB) ;
- **L1 cache** : mémoire cache de niveau 1, soit (exclusivement) pour des données (DCache), soit pour des instructions (ICache) ;
- **L1 hit** : contraire de « *L1 miss* » ; on a trouvé l'information recherchée dans une mémoire cache (ou TLB) de niveau 1 ;
- **L1 miss** : contraire de « *L1 hit* » ; on n'a pas trouvé l'information recherchée dans une mémoire cache (ou TLB) de niveau 1 ;
- **L2 TLB** : TLB de niveau 2, partagée entre adresses virtuelle d'instructions et adresses virtuelles de données ;
- **L2 cache** : mémoire cache de niveau 2, partagée entre instructions et données ;
- **L2 hit** : similaire au « *L1 hit* », mais pour une mémoire cache (ou TLB) de niveau 2 ;
- **L2 miss** : similaire au « *L1 miss* », mais pour une mémoire cache (ou TLB) de niveau 2 ;
- **LRU** : « *Least Recently Used* », politique de remplacement d'une mémoire cache, consistant à remplacer le bloc utilisé le moins récemment par le bloc à insérer ;
- **PGD** : « *Page Global Directory* », la première moitié (2 Kio) d'une page (4Kio) contenant des adresses physiques de débuts de pages de PUD ;
- **PMD** : « *Page Middle Directory* », la première moitié (2 Kio) d'une page (4Kio) contenant des adresses physiques de débuts de pages de PTE ;
- **PTE** : « *Page Table Entries* », la première moitié (2 Kio) d'une page (4Kio) contenant des adresses physiques de début de page de mémoire physique contenant des données ou des instructions ;

- **PUD** : « *Page Upper Directory* », une page (4 Kio) contenant des adresses physiques de débuts de pages de PMD ;
- **Page offset** : position d'un octet de données/d'instructions par rapport au début de la page ; le *page offset* est donné par les 12 derniers bits (bits de poids faibles) de l'adresse virtuelle ;
- **TLB**: translation lookahead buffers: mémoire cache pour les traductions d'adresses virtuelles (vers adresses physiques) ;
- **direct map** : une partie de l'adresse physique est utilisé *directement* comme l'index de bloc de cache à acceder ;
- **exclusive L2 cache** : l'information présente dans la L1-cache est supprimée de la L2-cache ;
- **full associative** : le choix de bloc de cache à acceder est libre (le contraire de direct mapped) si, dans le cache, il y a encore des blocs invalides. Si tous les blocs de cache sont valides, alors le choix de bloc de cache à éjecter (et ensuite remplacer par an autre) est déterminée par la politique de remplacement ;
- **inclusive L2 cache** : l'information est présente à la fois dans la L1-cache et dans la L2-cache ;
- **set associative** : une solution intermédiaire entre direct mapped et fully associative. Par example, une 4-way set associative cache est composée par 4 direct mapped caches (4 ways), une 8-way set associative cache est composée par 8 direct mapped caches (8 ways). Par consequence, une tranche de l'adresse physique est utilisé *directement* comme l'index de bloc de cache à acceder, mais le choix de *way* est soit libre (si il y a des blocs au même indices toujours invalides) soi déterminée par la politique de remplacement ;