**Partie 1 : Mémoire**

*Architecture de Von Neumann*

Constituer d'une mémoire centrale, et d'une unité centrale (processeur) ou il y’a 2 unité UAL (Arithmétique est logique) est unité de commande (contrôle);

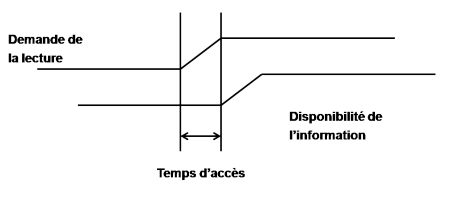
*Mémoire :* dispositif capable d'enregistrer des informations;

*Capacité:* quantité d'information qu'on peut stocker dans une mémoire;

*Volatilité:* la mémoire perd son contenu lors d'une coupure de courant;

*Mode d'accès:* soit le mode écriture ou bien le mode lecture;

*Temps d’accès:* c'est le temps qui sépare la demande et la disponibilité de l'information;

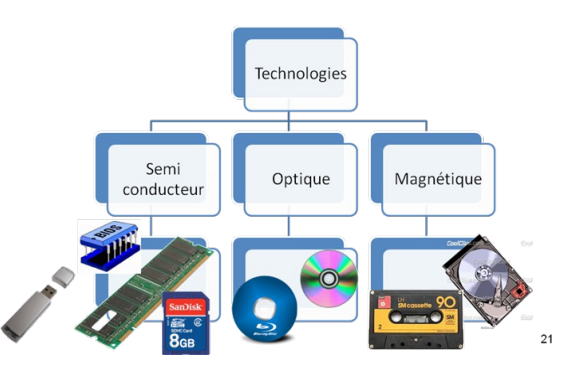


*Classification des mémoires*

*Mémoire semi-conducteur*: RAM, ROM (transistor ou condensateur)

*Mémoire magnétique:* disque dur, disquette;

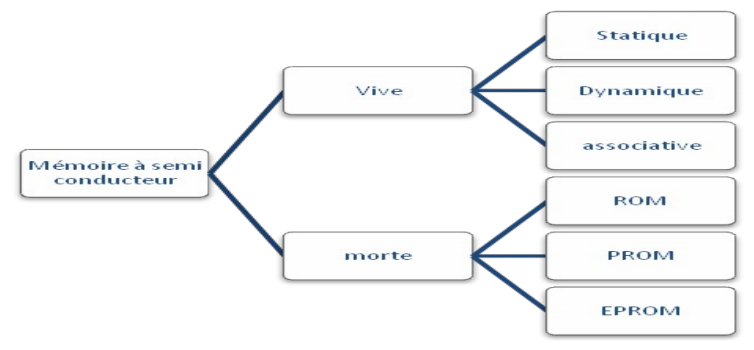
*Mémoire optique:* CD, DVD-ROM;



*Mémoire semi-conducteur:*

*Mémoire vive:* statique (bascule d), dynamique(condensateur), associative (entre les 2);

*Mémoire mort:* en lecture seul, ROM, PROM, EPROM;



*Mémoire central:*

C’est l'espace de travail du processeur, pour exécuter un programme il faut le charger dans une mémoire centrale.

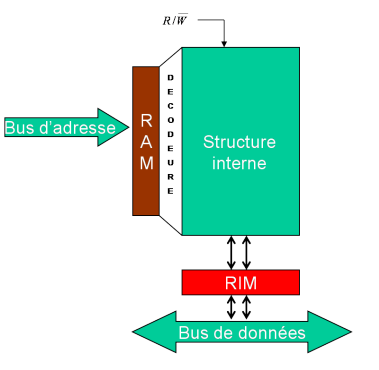
*RAM:* mémoire en accès aléatoire, le temps d'avoir une information est indépendant de sa place dans la mémoire.

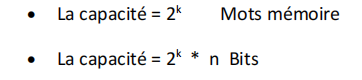
*Vue logique d'un mémoire:*

* Mémoire est un vecteur constitue des mots mémoire
* Un mot mémoire se compose de n bit (cellule mémoire)
* Chaque mot a une adresse unique qui lui représente ;
* Les adresse sont séquentielles et la taille d’adresse dépend de la capacité mémoire;

*Vue physique d'une mémoire:*

* RAM (registre d'adresse mémoire), stock l'adresse concernée par la lecture ou bien l’écriture.
* RIM (registre d'information mémoire): stock le mots mémoire dont on veut lire, ou bien écrire.
* Décodeur: sélectionner un mot mémoire, après avoir son adresse de RAM;
* R/W: commande qui sélectionne soit l’opération de lecture ou d'écriture;
* Bus d’adresses de taille k bits;
* Bus de données de taille n bits;





*Lire une information:*

* Charger l'adresse du mot qu'on veut lire dans le RAM;
* Lancer la commande R/W = 1
* L'information se trouve dans le RIM au bout d'un certain temps

*Ecrire une information:*

* Charger l'adresse mémoire dont on veut mettre l'information;
* Placer l'information qu'on veut enregistrer dans le RIM;
* Activer la commande R/W = 0, pour écrire l'information;

*Mémoire modulaire :* (éviter qu'un cpu monopolise toute la mémoire)

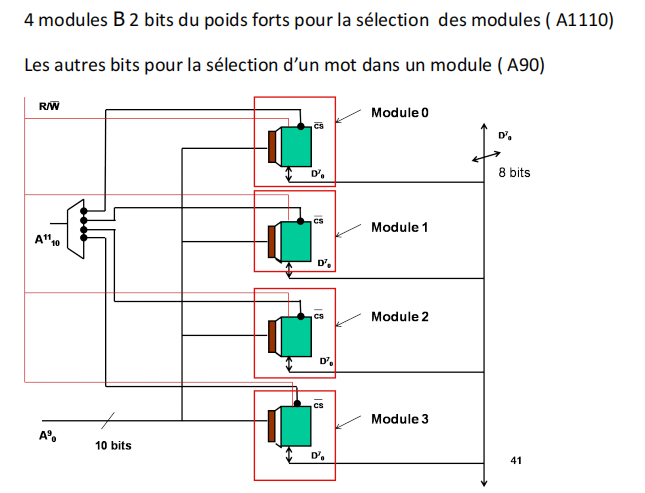
• découper la mémoire en plusieurs modules.

• Plusieurs bus permettent d’accéder simultanément (en même temps) à la MC.

• Possible d’avoir autant d’accès que de modules.

• On ne peut pas accéder simultanément à un module.

• Les bits du poids fort pour sélectionner un module.



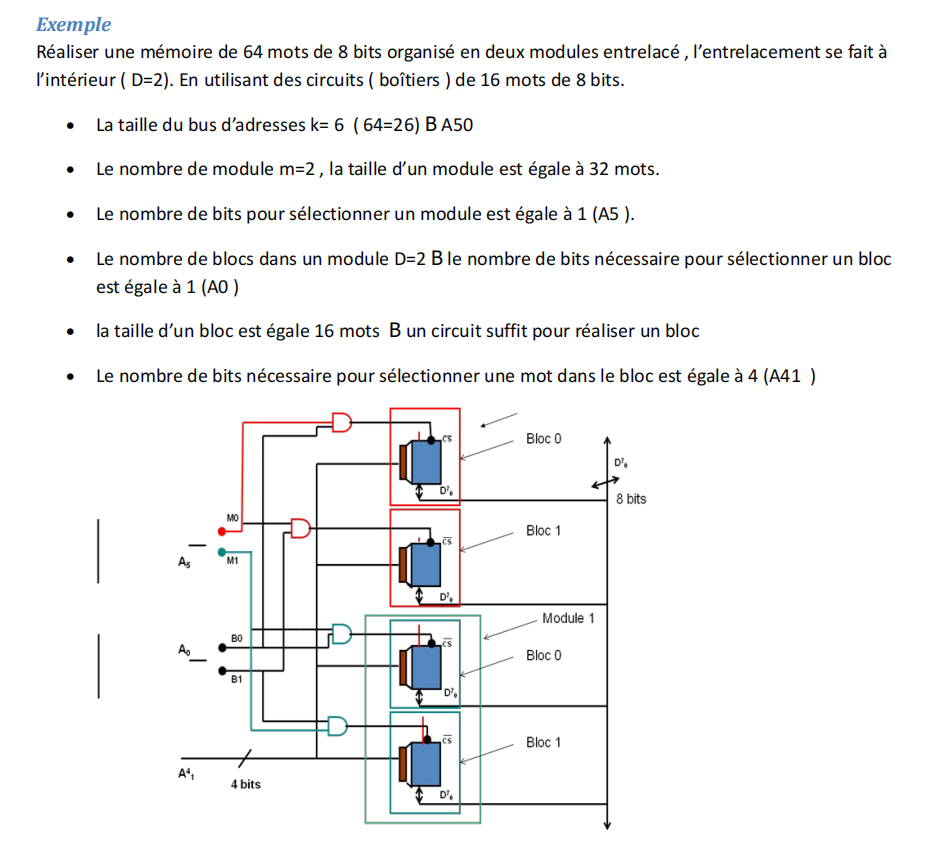
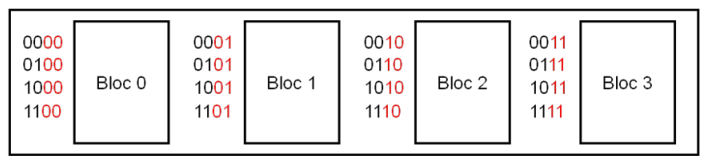
*Mémoire entrelacée:* (éviter qu'un cpu monopolise un module)

• Un module est divisé en plusieurs Blocs.

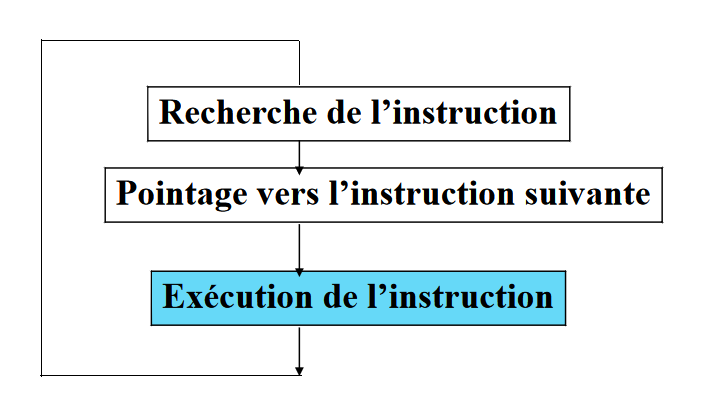
• les adresses consécutives sont placé dans des différents blocs

• Le nombre de blocs représente le degré d’entrelacement.

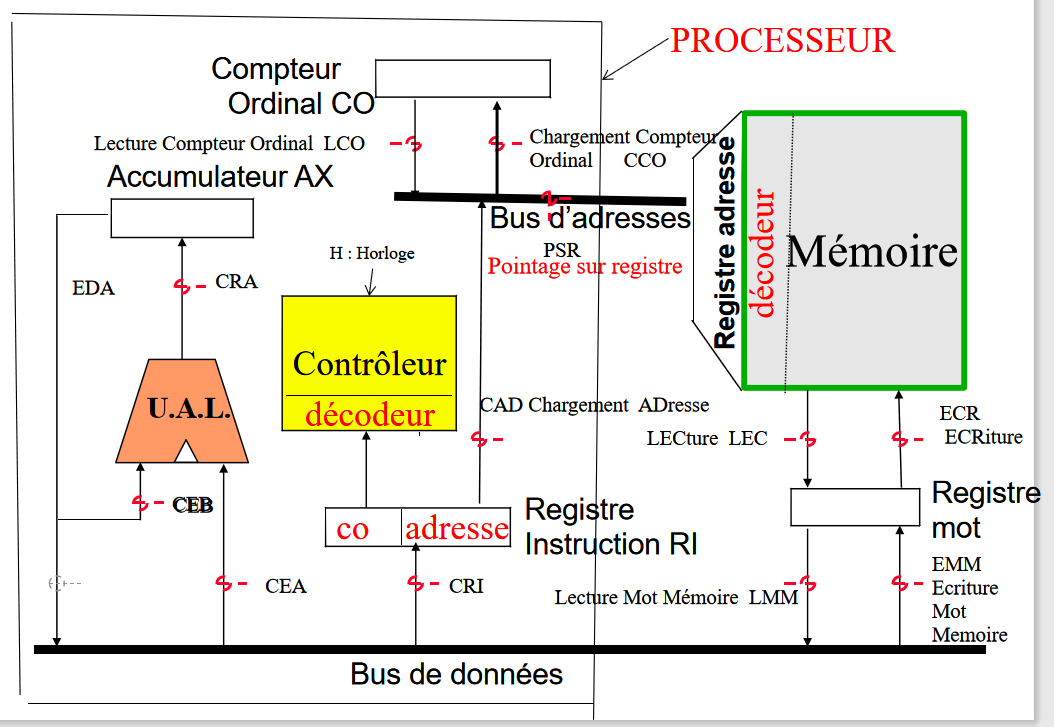
• Les bits de poids forts pour sélectionner un module, les bits du poids faible pour sélectionner un bloc dans un module. Le milieu pour sélectionner le mot.



**Partie 1 : La pile**

*Etapes d'exécution d'un programme.* 

Le compteur ordinal contient l'adresse de l'instruction a exécuter, il le passe au registre adresse RAM, l'instruction se charge dans le registre d'instruction à partir du registre mot RIM, si cette instruction contient un adresse mémoire d'une information dont on a besoin, l'unité de contrôle passe un signal pour piloter les bus et de mettre l'adresse dans le RAM , l'information et disponible dans le RIM, si c'est une opération de calcul on passe cette information a UAL et on enregistre le résultat dans accumulateur, si c'est une instruction d'écriture, on déplace l'information de l'accumulateur vers le registre RIM, et on charge l'adresse ou on veut mettre l'information dans le RAM, et on incrémente compteur ordinal pour passer à l'instruction suivantes.



*Fonction call & ret*

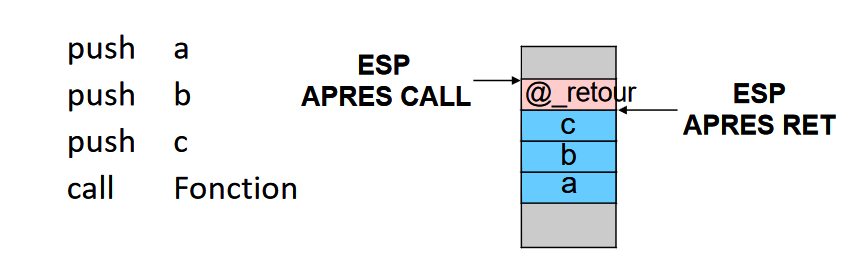
On est dans la fonction principale main, le compteur ordinal contient l'adresse de l’instruction á exécuter, lorsqu’il se trouve avec l’appel call, cette dernière push l'adresse de l'instruction suivante du programme principal dans la pile, et met l'adresse de la 1 ère instruction de la fonction á exécuter dans le compteur ordinal, á la fin d’exécution de cette fonction on se trouve avec l’instruction ret qui va dépiler de sa part l’adresse de retour de programme principal dans le compteur ordinale.

*Passage des paramètres á la pile*

On passe les paramètres un par un dans la pile, et on empile &return,

Esp point sur &return, on peut appeler parm1 par [esp + 4] --> 4 est la taille d'@ de return

Le paramètre suivant c'est [esp + 4 + taille(para1)]



*Passage des variables local á la pile*

On empile ebp pour avoir un point fixe pour l'appel des paramètres,

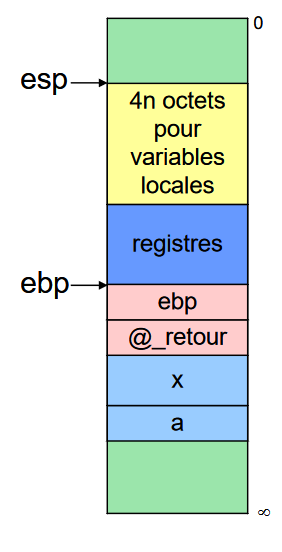
&return, epb;4 + 4; on empile les variables locales

1ere variable local [ebp - taille(var1)]

2eme variable local [ebp - taille (var1 + var2)]

1ere paramètre [ebp + 8];

2eme paramètre [ebp + 8 + taille(param1)];

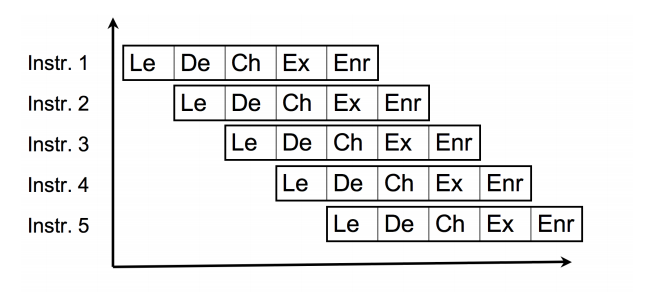


**Pipe-line**

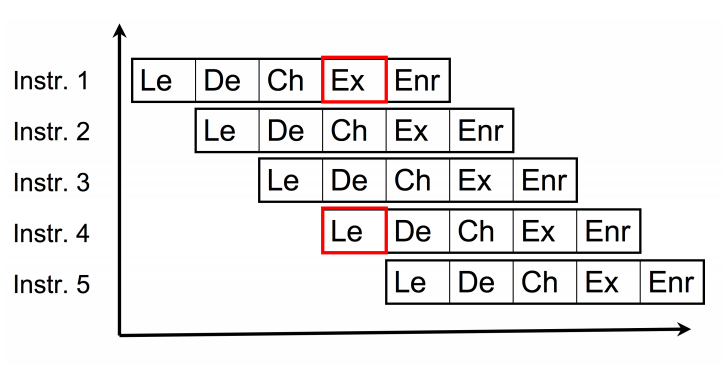
Avec les pipelines on peut avoir multiple instruction s’exécutent simultanément;

*Pipelines :*

l'[exécution](https://fr.wikipedia.org/wiki/Ex%C3%A9cution_(informatique)) des [instructions](https://fr.wikipedia.org/wiki/Instruction_(informatique)) est découpée en plusieurs étapes, le processeur peut commencer à exécuter une nouvelle instruction sans attendre que la précédente soit terminée, ce qui améliore le temps d’exécution

* Étape 1 -> LE: c'est l’étape qui sépare la lecture de l'@ d'instruction de EP, et d'avoir l'information de l'instruction dans le RIM;
* Étape 2 -> DE: copie l'information de l'instruction du RIM vers le registre d'instruction puis décodage de l’opération (connaitre le type de l'instruction);
* Étape 3 -> CH: chargement des registres de travails selon le type de l'instruction qu'on a dans l'unité de calcul;
* Étape 4 -> EX: exécution de l’instruction selon son type;
* Étape 5 -> ENR: enregistrement des résultats dans le registre destination;
* 

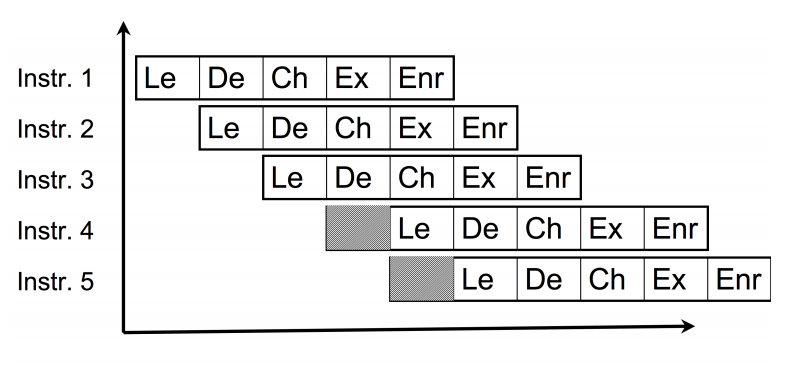
*Les aléas :*



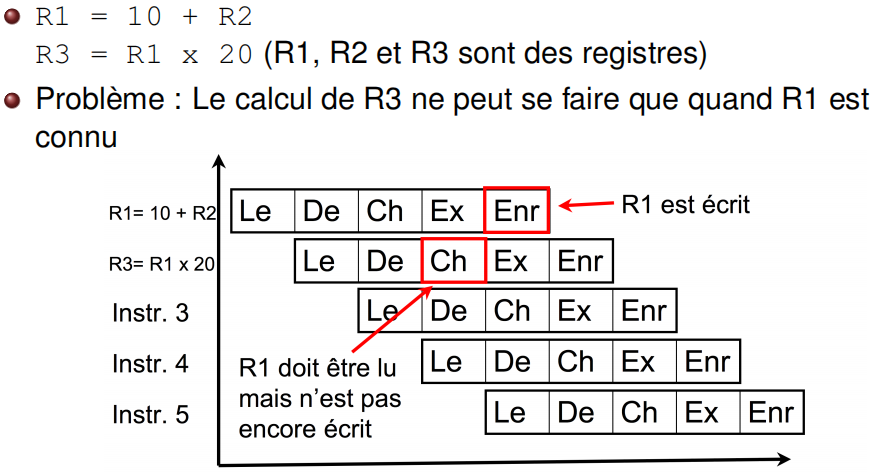
*Aléas structurel:* en cas d'instruction de type lecture/écriture, l’accès en mémoire dans les étapes lecture et exécution demande le travail avec la même unité d'accès mémoire, ce qui crée un conflit;

*Solution:*

* Suspendre lecture ou bien l’exécution en attend que l’autre finie son travail (décalage);
* Ajouter d'autre unité d’accès mémoire dans le processeur;



*Aléas de données:*



C'est quand une instruction besoin d’une donnée qui n'est pas actuellement disponible, ce qui veut dire que l'instruction dépend du résultat d'une instruction précédente qui n’a pas encore fini son travail par exemple:

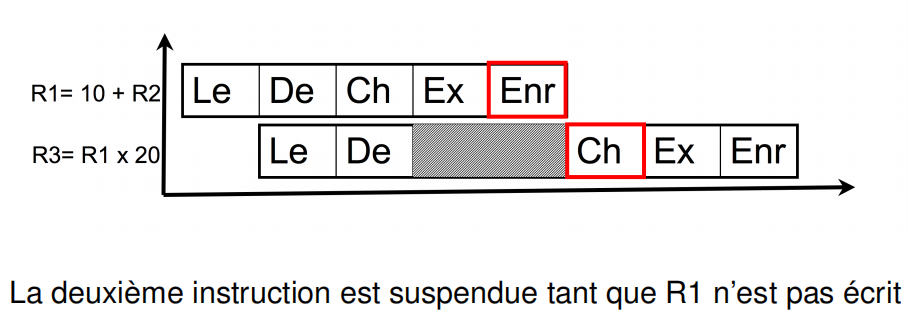
Inst 1: R1 = 10 + R2

Inst 2: R2 = R1 x 20

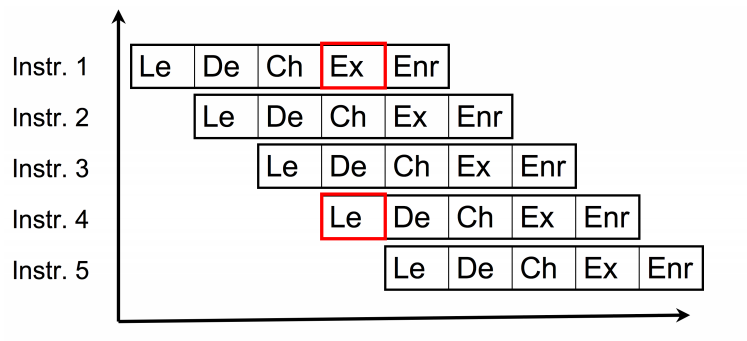
Ici instruction 2 besoin que R1 soit déjà calculer, l'étape chargement(inst2) dépasse l'étape enregistrement(int1);

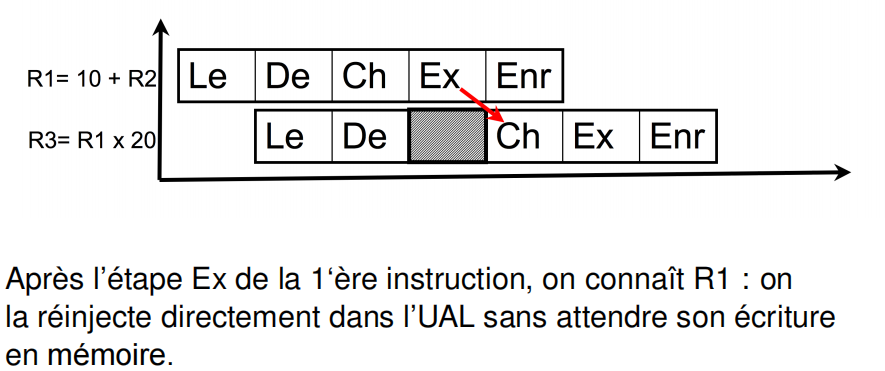
*Solution:*

* Suspension de l'étape CH(int2) tant que ENR(int1) n'est pas terminer;



* Ordonnancement: changer l'ordre de l'exécution des instruction (mettre l'inst2 à la fin) deux types d’ré ordonnancement : Logiciel (compilateur) Matériel (fait par le processeur);



* Court-circuit: âpres l'étape EXE(int1) on connait R1, en la réinjecte directement dans registre A ou B de l’UAL (là où on a étape CH(int2)), sans attendre la terminaison de l'étape ENR(int1);
* 

*Aléas de contrôles*

Inst 1: R1 = 10 + R2

Inst 2: if (R1 > 30) then R3 = 10 + 1 ; else R3 = 20 + 1

R1 n'est pas connue, alors l’instruction suivante reste inconnu, donc on aura un problème dans l'étape Lecture de inst2;

*Solution:*

* Solution avec attendre: on attend que ENR(ins1) soit terminer, et on passe à LE(inst2);
* Réordonnancer le code : pas toujours suffisant et possible;
* Prédire la valeur de R1 et commencer le calcul selon cette prédiction

On prend le chemin(branche) le plus souvent atteint pour cela on utilise:

BTB: tampon des branches(chemin) cible, contient les adresses des instructions à suivre;

BHT: table qui mémorise l’historique des choix de branchements, dépend de l’algorithme utilisé par exemple 2 bits associés à chaque branche

00 : branchement jamais pris jusqu’à présent

01 : branchement parfois pris jusqu’à présent

10 : branchement souvent pris jusqu’à présent

11 : branchement toujours pris jusqu’à présent

**Cache mémoire**

Cpu cherche tjrs l'instruction dans le cache, si elle n’existe pas il va la chercher dans la MC, car l'accès au cache est plus rapide que l’accès à la MC

*Cache á correspondance direct:*

Se fait avec la correspondance suivante:

@mot en MC modulo la taille de cache. Par exemple prenant le cache de 8 blocs;

Adresse 1, 9, 17, 25 en MC correspondant au bloc numéro 1 du cache;

Pour faire la correspondance inverse (cache vers MC) en utilise les étiquètes:

Index correspond au numéro du bloc dans le cache;

Etiquète : pour compléter l’@ afin qu’elle correspond á celle de la MC;

*Pour bloc 1 :*

Index = 001, étiquète = 00 ---> @ MC 00 001 :1

Index = 001, étiquète = 01 ---> @ MC 01 001 :9

Index = 001, étiquète = 10 ---> @ MC 00 001 :17

Index = 001, étiquète = 11 ---> @ MC 00 001 :25

*Cache totalement associatifs:*

En place le mot mémoire á n'importe quel bloc du cache;

*Cache associatifs par ensemble:*

On divise le cache en n ensembles, pour placer un mot dans un ensemble, en utilise le principe du cache á correspondance direct

@mot modulo n nombre d'ensemble, ensuite en place le mot á n'import bloc de l'ensemble, en général l'adresse se présent pour le cache comme ce ci exemple d'@ 12 bits.

Offset: si le mot mémoire se compose de 8 octet, offset prendra donc 3 bits pour le représenté

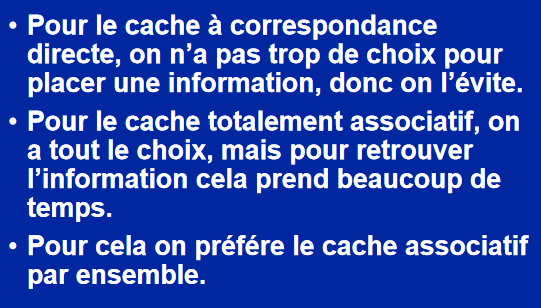
Index: c'est le cache contient 16 entrées, index prendra 4 bits pour les représentés.

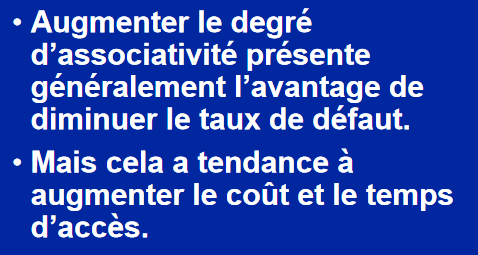
Étiquette: prendra les bits qui reste de la taille d'adresse.

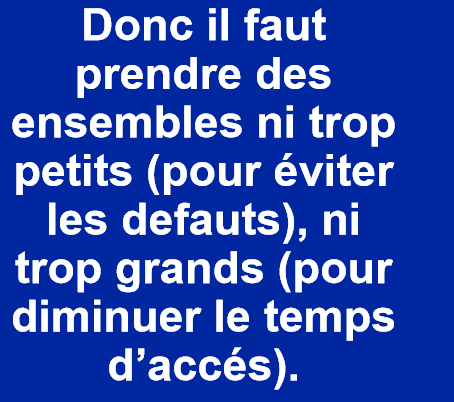
0 0 0 0 0|00 00| 0 0 0

Etiquette |index |offset

5 4 3







*Recherche d'un @ dans le cache:*

Le processeur prend l'index pour savoir l'entrée du cache á charger, si l’entrée est vide ou la comparaison avec l'étiquète n'est pas identique, il se produit un défaut du cache, le processeur va donc chercher @ dans la MC, et la mettre dans le bloc du cache qui lui correspondant, sinon il prend directement l'information du cache.

00: |000|00 mots 0

04: |001|00 mots 1

08: |010|00 mots 2

12: |011|00 mots 3

16: |100|00 mots 4

20: |101|00 mots 5

24: |110|00 mots 6

28: |111|00 mots 7

3 premier bits c'est l'index;

2 dernier bits le numéro de l'octet dans le mots;

**Language assembleur**

CMP A,B

JE (JNE):jump if (NO)Equal

JG (JGE) (JNGE):jump if (not)(equal) A great than B

JL (JLE) (JNLE):jump if (not)(equal) A less than B

example de la bouble for

for (cx=0; cx<5; cx++) ax = ax + cx;

MOV AX,0 ; initialisation de AX et CX à 0

MOV CX,0 ; CX est le compteur de boucle

for:

CMP CX, 5 ; compare CX à 5 JGE endfor ; si CX >= 5, on sort de la boucle

ADD AX, CX ; fait le calcul

INC CX ; CX est incrémenté de 1

JMP for ; on reprend au début de la boucle endfor:

ja jump above (x > y non-signé ) CF = 0 & ZF = 0

jna jump not above = jbe

jae jump above or equal CF = 0

jnae jump not above or equal = jb

jb jump below (x < y non-signé) CF = 1

jnb jump not below = jae

jbe jump below or equal CF = 1 | ZF = 1

jnbe jump not below or equal = ja

\*loop decriment CX tanque CX>0

for (cx=5; cx>0; cx--) ax = ax + cx

MOV AX,0 MOV CX,5 ; CX est le compteur de boucle

for:

ADD AX,CX ; fait le calcul

LOOP for ; décrémente CX. Si CX > 0 fait ; le saut à for

Execution d'un programme:

LEC:lecture mot memoire de la memoire vers registre mot.(ECR)

LMM:lecture mot memoire de Registre mots vers bus de donnes.(EMM)

LCO:lecture compteur ordinal,transfert le contenu de CO dans bus adresse.(CCO:chargement)

PSR:pointage sur registre,copie l'@ de bus d'adresse vers registre d'adresse.

CRI:charger registre d'instruction,mettre le mot memoire dans le registre d'instruction.

CAD:chargement adresse,copie l'@ de Registre d'istruction vers Registre d'adresse.

CEA:chargement entree A de l'UAL aupres de bus de donne

CEA:chargement entree B de l'UAL,aupres de bus de donne ou Registre accumelateur.

CRA:chargement registre Accumelateur,enregistre le resultat de l'UAL.

EDA:enrgistrement donne accumelateur.soit dans UAL ou dans la memoire.

rechere d'instruction,pointage vers instruction suivant,execution d'instruction.