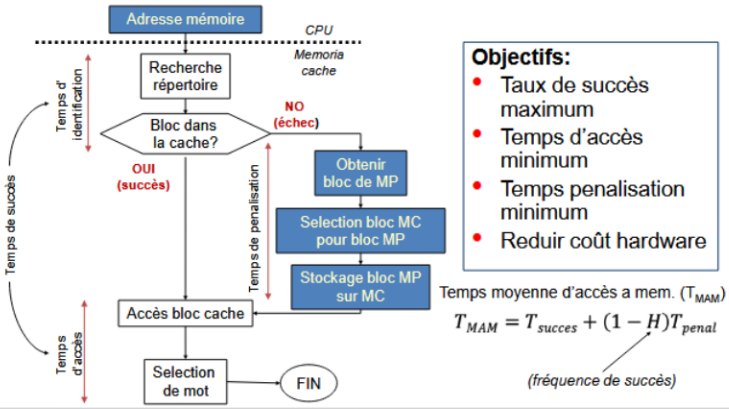
Structure dans le cache

- **Le cache est divisé en blocs (ligne) (groupe de 8-64 octets)**
- **les bloc on une étiquette unique (tag)**

• Combien de lignes dans un cache de 64KB avec des lignes de taille 64 Bytes ?
KB = 2^10 =1024 (oct) → (64 * 1024) / 64 = 1024



Objectifs:
- Taux de succès maximum
- Temps d'accès minimum
- Temps penalisation minimum
- Reduir coût hardware

Temps moyenne d'accès a mem. (T_{MAM})

$$T_{MAM} = T_{succes} + (1 - H)T_{penal}$$

(fréquence de succès)

Fréquence de succès

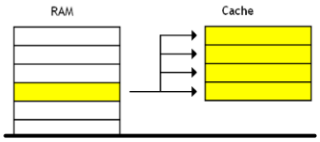
- **dépend de la taille du cache**
- **Miss penality devient du Hit time**

	Rate	Time
Cache Hit	**Hit rate =** Nombre de hits/Nombre d'accès	Temps d'accès à la cache (**hit time**)
Cache Miss	1 – (Hit rate)	Temps d'accès à la mémoire principale + temps de chargement cache (**miss penalty**)

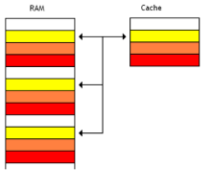
Recherche dans la cache

mots = mots dans la memoir principale
bute = trouver la ligne dont le tag correspond à l'Addr de base demander au repértoire

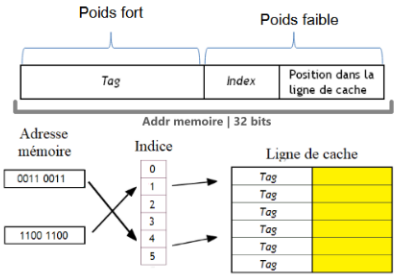
1) Complètement associative : mots peuvent être stocker n'import où dans le cache
+ : taux de hit très élevé
- : trop lent (sequentiel, toutes les lignes à regarder)



2) Associative par ensembles : mots est charger dans des ligne predifinie (toujour la meme) dans le cache
+ : get rapide, car il faut checker 1 ligne
- : conflit, taux de hit éclater

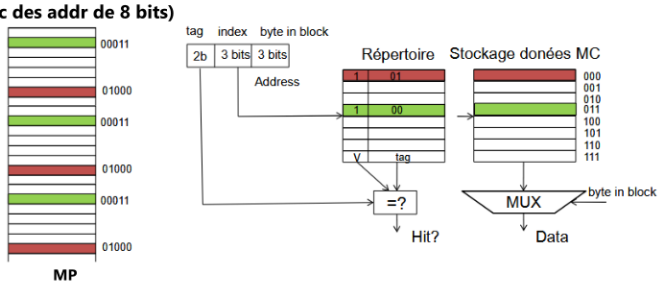


3) Associative par voies :
- Bits faible → index (qu'elle ligne) et position dans la ligne
- Bits fort → Tag



Chaque bloc tombe sur les meme cache
- Memoire principale (MP) : 256 (oct), 8 (oct) blocs → 32 blocs (Addr 5 (oct))
- Cache (MC) : 64 (oct), 8 blocs (Addr 3 (oct))

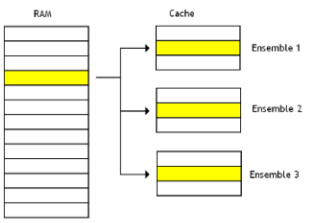
Exemple (avec des addr de 8 bits)



4) Associative par voie : Cache composé de plusieur "cache" (meme addr) accessibles en parallèle

- Chaque cache est appelé une voie
- Mots peut être stocké en N positions différentes dans le cache

- Diminution de la fréquence d'échec :
Associativité *2 = diminution de 20%
Taille cache *2 = diminution de 69%



Cache (ou antémemoire)

Cache = petit memoire plus rapide pour stocker des données que l'on vas souvent utiliser

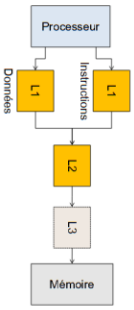
- **Composer de transistors et pas de condensateurs**
- **Pas de contrôle sur son contenue**

Localité

Localité spatiale	Localité temporelle
Le code d'un programme s'exécute toujours à l'intérieur de petites zones répétées de mémoire (des blocs correspondant à des boucles ou/et des sous-programmes). Si une adresse mémoire est utilisée, alors **les adresses proches** le seront probablement dans un futur proche.	Les blocs s'exécutent en **séquences très proches** *il y a plus de chances d'accéder à une position de mémoire utilisée il y a 10 cycles qu'à une autre utilisée il y a 10000 cycles* Si une adresse mémoire est utilisée, elle le sera probablement **à nouveau** dans un futur proche

Localité temporal: Chaque fois que le processeur prend un mot de la mémoire, il copie le mot à la cache.
- Pénalisation dans le premier accès
- Plusieurs utilisations seront plus rapides

Localité spatiale: Au lieu de copier un seul mot, on va en prendre plusieurs en même temps (un bloc)
- Accès à a[0]
- On prend aussi a[1], a[2], a[3]



Niveaux

souvent plusieur niveaux de cache (instruction et données)

Repertoire

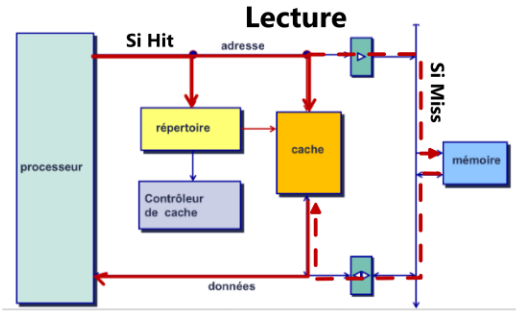
Indique si le mot accedépar le processeur setrouve dans la cache. Si oui → indique l'Addr dans le cache

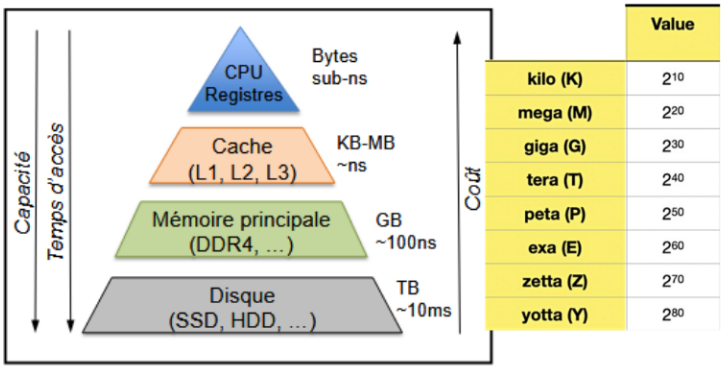
Contrôleur de la cache

Controle de la lecture/écritureen mémoire (parmi des buffers, les bloc avec les triangles)

Écriture et Lecture

En gros,
- lecture : quand ça miss ça vas chercher dans la memoire et ça écrit le resulta en meme temps que de retourné la valeur
- Ecriture : on écrit dans les 2 de toute manière mais si ça hit, on peut le retourné plus vite





	Value
kilo (K)	2^{10}
mega (M)	2^{20}
giga (G)	2^{30}
tera (T)	2^{40}
peta (P)	2^{50}
exa (E)	2^{60}
zetta (Z)	2^{70}
yotta (Y)	2^{80}

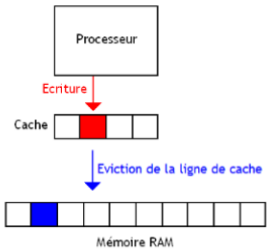
CPU à 2.6 GHz, mémoire RAM avec 10ns de latence. Le CPU veut accéder à une information mémoire.
• Après combien de temps va-t-il recevoir l'information ?
$10 * 10^{-9}$ s = **10^{-8} = 10 µs**
• Combien de cycles CPU se sont déroulés pendant ce temps ?
$10 * 10^{-9} * 2.6 * 10^{9}$ = 26 cycles

Cache ARM9

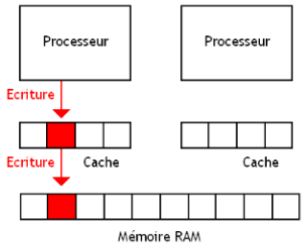
- 128KB : 4 voies associative 32KB
- ligne : 64 (oct)

Algo d'écriture

- **Write-back**



- **Write-through**



Algo de suppr

- Il faut ajouter l'information manquante à la cache
  - Si la cache est pleine, comment choisir la ligne à remplacer ?

	Fully associative	Direct Mapped	Set associative
Choix			

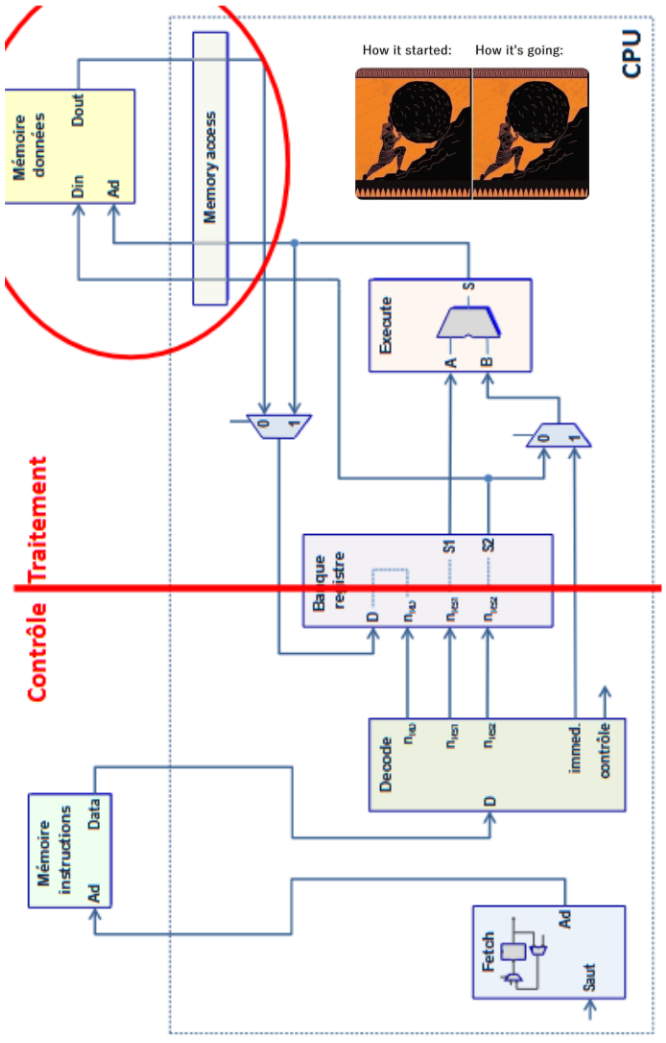
- **Politiques** de remplacement

Algo de suppr (cache miss)

- Il faut ajouter l'information manquante à la cache
  - Si la cache est pleine, comment choisir la ligne à remplacer ?

	Fully associative	Direct Mapped	Set associative
Choix	Trop de choix	Pas le choix	Un peu de choix

- **Politiques** de remplacement
  - **Aléatoire** : on choisit une ligne au hasard
  - **FIFO** : premier entré – premier sorti
  - **Least Recently Used (LRU)** : on supprime une ligne qui n'a pas été utilisée depuis longtemps



Cache Exo methods

Get memorie require 3 (ws)(wait state) et (hr)(hit rate) de 90%

- nombre moyen de waits state = 10% * 3 = 0,3 ws par cycle memoire

si donné en cache → 0 ws

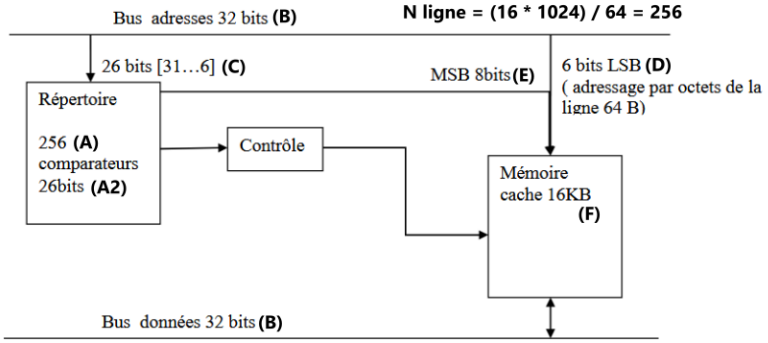
Intruction require 2 cycle sans (ws) et 5 cycle avec | (hr) = 80%

10 instruction → (10 * 0.8 * 2) + (10 * (1 - 0.8) * 5) = 16 + 26 → 26 cycle dont 16 avec cache

Intruction require 2 cycle sans (ws) et 5 cycle avec | (hr) = 80%

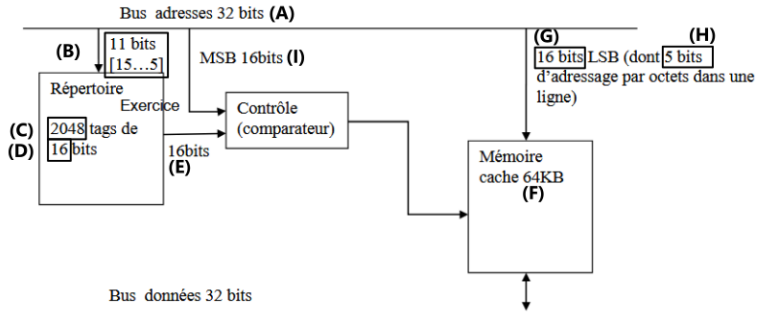
10 instruction → (10 * 0.8 * 2) + (10 * (1 - 0.8) * 5) = 16 + 26 → 26 cycle dont 16 avec cache

COMPLÈTEMENT ASSOCIATIVE : Bus (Addr et donné) du procésseur sont de 32 bits.
Cache de 16KB (Ligne de 64 (oct))



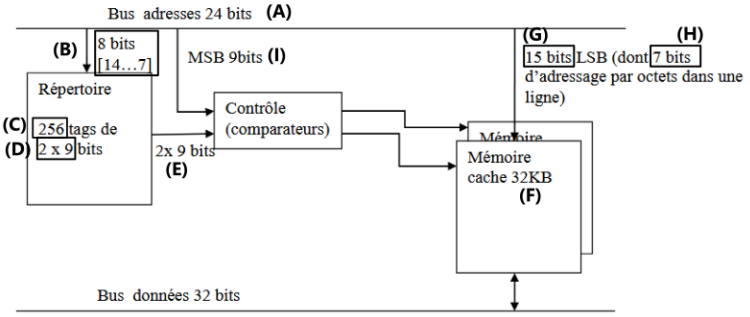
N ligne = (16 * 1024) / 64 = 256
A) 1 comp par ligne → N ligne
A2) on a besoin 26 bits à cause de (C)
B) bus de 32 bits (donné)
C) 32 - 6 (raison (D)) = 26 bits → [31...6]
D) offset → 64 = 2^6 → besoin de 6 bits pour les décrire
E) N ligne = 256 = 2^8 → besoin de 8 b pour trouver la ligne
F) donner dans la donné

ASSOCIATIVE PAR ENSEMBLES : Bus (Addr et donné) du procésseur sont de 32 bits.
Cache de 64KB (Ligne de 32 (oct))



N ligne = (64 * 1024) / 32 = 2048
A) donner
B) (32-16) - 5 (du à (H)) = 11 bits
C) (16-5) = 11 → 2^11 = 2048 = N Ligne → besoin de 16 bits au totale
D,E) Meme logic que (C)
F)donner
G) N ligne = 2048 → 11 bits → 11 + 5 (du à (H)) = 16
H) ligne de 32(oct)=2^5 → 5 bits
I) 32 - 16 = 16

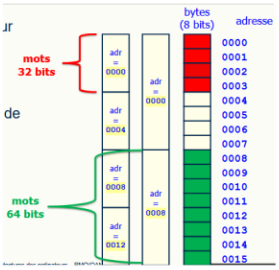
ASSOCIATIVE À 2 VOIES : Bus Addr de 24 bits et donné du procésseur sont de 32 bits.
Cache de 32KB par voie (Ligne de 128 (oct))



N ligne = (32 * 1024) / 128 = 256
A) donner
B) 15 (du à (G)) - 7 = 8 bits
C) N ligne
D,E) du à (I) et au fait qu'il y au 2 voie → 2*9
F)donner
G) N ligne = 256 → 8 bits → 8 + 7 (du à (H)) = 15
H) ligne de 128(oct)=2^7 → 7 bits
I) 24 - 15 = 9

Memoire

- Addr est stockée dans un mot
- N bits d'un mot = taille max de la memoire→mots = 32 bits → 2^32 = 4GB



Décode d'Addr memoire

2^32 = 4MW (4 Mega mots)

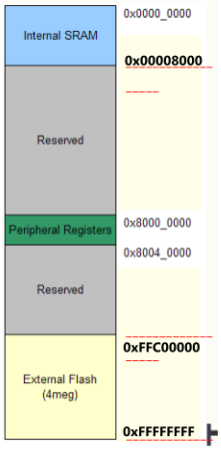
Adr = (N * nb_byte)/(taille de mots)

	0	1	2	3	4	5	6	7
123A0000	93	A4	77	1E	FB	86	79	83
123A0008	76	30	DE	BD	D9	78	CF	2F
123A0010	D6	44	94	4E	D4	4C	EA	02
123A0018	29	FA	87	2D	8A	EB	59	5B
123A0020	A8	65	3F	40	3A	0D	48	EE
123A0028	21	BA	DB	9B	1B	89	6F	47
123A0030	62	D6	80	A8	D1	9D	50	EE
123A0038	80	98	D2	CB	D9	6B	5C	21

Mémoires volatiles (vives)	Mémoires non-volatiles (mortes)
- informations **perdues** à la mise hors tension - lecture et écriture en cours d'utilisation	- informations **conservées** à la mise hors tension - lecture en cours d'utilisation - écriture (« programmation ») durant la fabrication de la mémoire, ou sur la carte (in situ). - modification du contenu durant le fonctionnement du système, implique un effacement!
RAM, SRAM, DRAM	EPROM, Flash memory, PROM, FeRAM, EEPROM, ROM, PRAM, E2PROM, MRAM

Memory Map

Exemple :
32 bits d'adressage
SRAM = 32KB = 2^15 = 0111 1111 1111 1111
= 0x00007FFF
Interal Flash = 4MB = 2^22 = 0x007FFFFF
Total mem → 2^32 = 4GB → 0x3FFFFF
→0xFFFFFFFF - 0x3FFFFF = 0xFFC00000
SRAM → 0x00000000 → 0x00007FFF +1

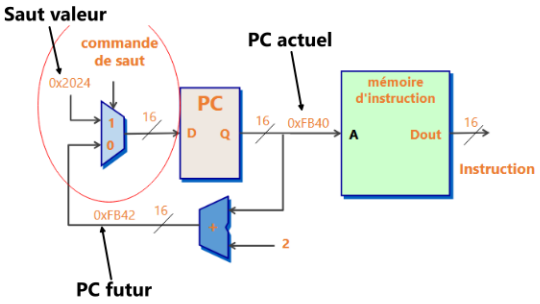


Endianness

Fetch

- PC (Programme compteur): incrément chaque instru sauf si saut

Code d'instruction	Incrément
8 bits == 1 byte	1
16 bits == 2 bytes	2
32 bits == 4 bytes	4



Saut inconditionnel

- Adr = PC + extension_16bits(offset(11) x 2) + 4

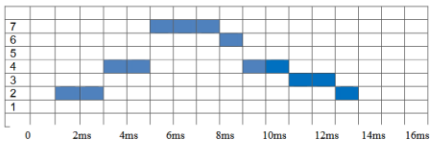
Exemple : 0xE00F → 1110 0 000 0000 1111
offset(11) x 2 → 000 0001 1110
Extention 16 bits → 0000 0000 0001 1110
+4 → 0000 0000 0010 0010 = v
PC actuel = 0000 0000 0000 1110
v + PC actuel → 0000 0000 0011 0000 → 0x0030

Saut conditionnel

- Adr = PC + extension_16bits(offset 8 x 2) + 4

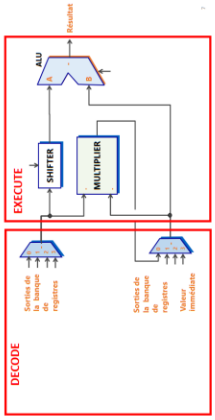
Interuption

2) t = 1ms | d = 3ms
4) t = 3ms | d = 4ms
3) t = 4ms | d = 2ms →
7) t = 5ms | d = 3ms
6) t = 6ms | d = 1ms

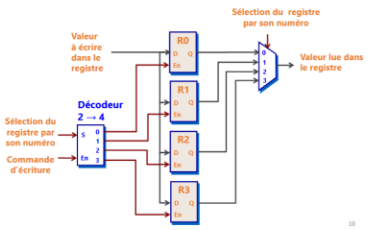


Execute

- **Logical shift** : multiplication ou division par une puissance de 2
  - ajout de 0 à droite ou à gauche
- **Arithmetic shift** : multiplication ou division par une puissance de 2 sur des nombres signés
  - ajout de 0 à droite, extension de signe à gauche
- **Rotation** : report des bits retirés d'un coté vers l'autre coté

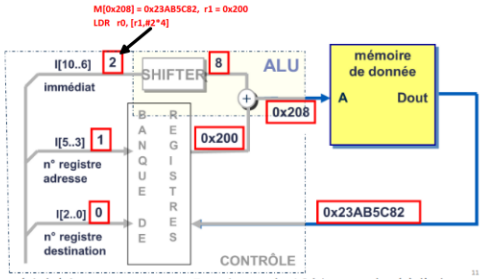


Bank de registre



- SP (Stack pointer) : Stocke la position dans lapile de stockage
- LR (Link Register) : Garde l'adresse de retour(appel de fct)

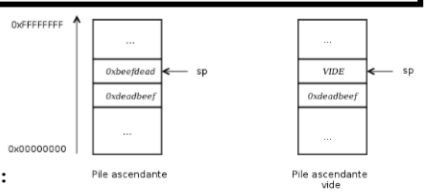
MEMORY ACCESS



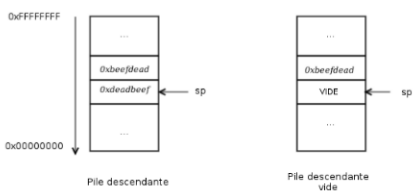
- Un périphérique comprenant 5 registres de 16 bits est placé à l'adresse. 0xF230 sur le bus d'un processeur. Le bus d'adresses est un bus de 16 bits. Donnez les adresses des 5 registres (*Incrément de 2 car 16 bits*) : *0xF230, 0xF232, 0xF234, 0xF236, 0xF238*

Pile

- Pile ascendante :
SP init sur l'Addr du bas de la pile

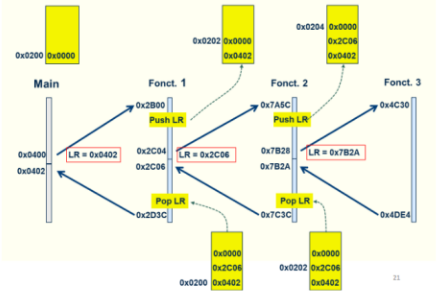


- Pile descendante :
SP init sur l'Addr du haut de la pile



Utilisation Pile

- stockage des Addr de retour pour les appl de fonction et interuption
- Save de registre pour les contexte
- Save de variable locales



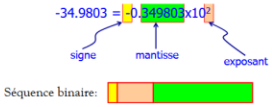
Incrémentation

Pré = ++i | Move SP → action

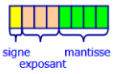
Post = i++ | action → move SP

Float

Structure d'un float



Séquence binaire:



Decimal → Binaire

0.375 = ?

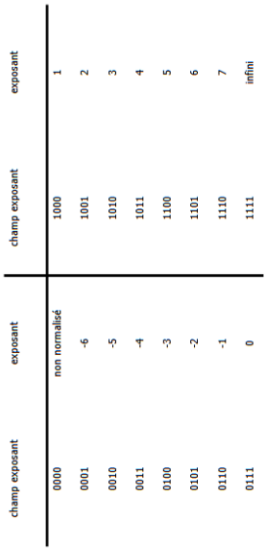
0.375x2 = 0,75
0.75x2 = 1,5
0.5x2 = 1,0

0.375 = 0.011

0.3 = ?

0.3 x 2 = 0,6
0.6 x 2 = 1,2
0.2 x 2 = 0,4
0.4 x 2 = 0,8
0.8 x 2 = 1,6
0.6 x 2 = 1,2
...

0.3 = 0.01001[1001]= 0.0[1001]



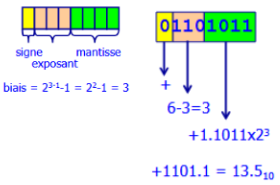
Représentaton avec biais

Champ exposant = exposant + biais

Typiquement "2^(k-1)-1"

avec k=nombre de bits duchamp de l'exposant

Exemple :



Représentaton non normaliser

Pour les nombre très petit proche de 0

+0.0=0 00...00

-0.0=1 00...00

Dans ce cas l'exposant = 1 - biais

Exemple : +0.001=+0.10*2^-2=0 000 10

Standard IEEE 754

- normaliser : biais = 127
- Non-normaliser : 1-biais =-126

Pipline

	Single Data	Multiple Data
Single Instruction	SISD — ADD R1, R2, R3	SIMD — Traitement vectoriel
Multiple Instruction	MIMD — Rarement utilisé (traitement du signal)	MIMD — Processeur multi-coeurs

Aléas

Dépendances

- Structurel : Accès aux meme ressources
- Données : Utilisation des même registres
- Contrôle : Décision de branchement

Dépendances de données

- RAW (read after write) = ADD R1, R2, R3
  SUB R5, R1, #2
- WAR (write after read) = ADD R1, R2, R3
  SUB R2, R4, #2
- WAW (write after write) = ADD R1, R2, R3
  AND R5, R1, R2
  SUB R1, R4, #2

Stop pipeline

	1	2	3	4	5	6	7	8	9	10
ADD R1, R2, R3	IF	ID	EX	MEM	WB					
SUB R5, R1, #2		IF	ID	ID	ID	ID	EX	MEM	WB	
ADD R1, R2, R3	IF	ID	EX	MEM	WB					
NOP		IF	ID	EX	MEM	WB				
NOP			IF	ID	EX	MEM	WB			
NOP				IF	ID	EX	MEM	WB		
SUB R5, R1, #2					IF	ID	EX	MEM	WB	

Forwarding

Cours ARO 2025

Exemple de pipeline avec bypass

r4 = 0x12, r3 = 0x28B2,
M[0x28BA] = 0xABCD

Récupération de la valeur pour r0

16:	1C9A	add	r2, r3, #2
18:	18A5	add	r5, r4, r2
1a:	88D0	ldrh	r0, [r2, #6]
1c:	1D01	add	r1, r0, #4

Arrêt du pipeline

	PC	16	18	1A	1C	1E	20	22	24	26	28	2A	2B
	FETCH (IR)		1C9A	18A5	88D0	1D01							
DECODE	Rs1			28B2	12	r2=rF1=28B4	r0=?	r0=rM = ABCD					
	Rs2			-	2=rE=28B4	-	-	-					
	Rd			r2	r5	r0	r1	r1					
	#imm			2	-	6	4	4					
	EXECUTE				28B4	28C6	28BA	28BA	ABD1				
	MEMORY					28B4	28C6	ABCD(rd)	ABCD	ABD1			
	WRITE BACK						r2=28B4	r5=28C6	r0=ABCD	ABCD	r1=ABD1		
	REG bypass E				28B4	28C6	(28BA)	(28BA)	ABD1				
	REG bypass F1					28B4	28C6	(28BA)	(28BA)	ABD1			
	REG bypass F2						28B4	28C6	(28BA)	(28BA)	ABD1		
	REG M					28B4	28C6	ABCD	ABCD	ABD1			

Remarque pour l'arrêt en jaune

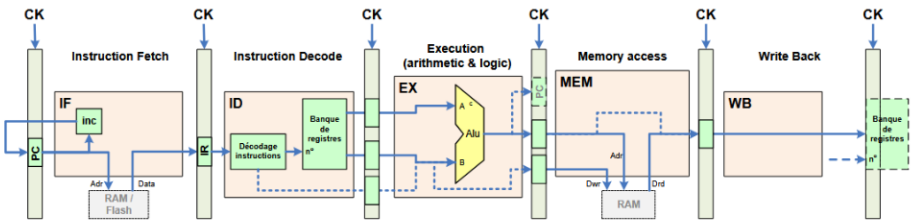
On ne connait as encore la valeur de R0 à reprendre car le Memory Acces n'a pas été fait

Maintien de la valeur dans E car le bloc execute est en arret

Maintien de la valeur dans Memory Acces car le bloc est en arret

Maintien de la valeur dans Write Back car le bloc est en arret

1	LDR r1,[r5]	F	D	E	M	W	
2	ADD r5,r1,1	1	F	D	D	E	M	W
3	LDR r1,[r6]			F	F	D	E	M	W



Pipline MATH

Base

- T_e = temps de passage à chaque étape
- T_p = n*T_e = temps pour n étapes

Calcul de performances

- Latence : Temps entre le debut et la fin de l'exec
- Débit : nombre d'op executer par unités de temps

$$D = \frac{m}{(n+m-1)*T_e} \approx \left|\frac{1}{T_e}\right|$$

-Accélération : n fois plus rapide que le séquentiel

$$A = \frac{T_{seq}}{T_{plp}} = \frac{m*n*T_e}{(n+m-1)*T_e} = \frac{m*n}{n+m-1} \sim n \text{ (pour m très grand)}$$

- IPC (nombre d'instru par cycle)

Exemple : stop 3 cycle pour 20% des instru

IPC = 1/ (0.8 x 1 + 0.2 x 4) = 0.625