Taille mot mémoire

La taille d'un mot mémoire est forcément un multiple de 8. C'est pourquoi nous pouvons appliquer le tableau suivant :

Nom	Symbole	Puissances binaires et valeurs en décimal	Nombre	Hexa	Ordre de grandeur SI décimal
unité	o/B	2^0 = 1	un(e)	1	10^0 = 1
kilo	ko/Ko kB/KB	2^{10} = 1 024	mille	400	10^3 = 1 000
méga	Mo/MB	2^{20} = 1 048 576	million	100000	10^6 = 1 000 000
giga	Go/GB	2^{30} = 1 073 741 824	milliard	40000000	10^9 = 1 000 000 000
téra	To/TB	2^{40} = 1 099 511 627 776	billion	10000000000	10^{12} = 1 000 000 000 000
péta	Po/PB	2^{50} = 1 125 899 906 842 624	billiard	400000000000	10^{15} = 1 000 000 000 000 000
exa	Eo/EB	2^{60} = 1 152 921 504 606 846 976	trillion	1000000000000000	10^{18} = 1 000 000 000 000 000 000

Gestion des adresses

En fonction de la taille de la mémoire nous aurons une taille d'adresses variables, le tableau suivant représente les possibilités :

Adressage	Puiss. binaire et décimal	Hexa	byte	bit
8 bits	2^8 = 256	100	256 B	2 Kb
16 bits	2^{16} = 65 536	10000	64 KB	512 Kb
32 bits	2^{32} = 4 294 967 296	100000000	4 GB	32 Gb
64 bits	2^{64} = 18 446 744 073 709 551 616	10000000000000000	16 EB	128 Eb

Calculer la mémoire

Calculer adresse de fin

$$\text{Adr.Fin} = \text{Adr.Deb} + \text{Taille} - 1$$

Calculer adresse de début

$$\text{Adr.Deb} = \text{Adr.Fin} - \text{Taille} + 1$$

Calculer la taille

$$\text{Taille} = \text{Adr.Fin} - \text{Adr.Deb} + 1$$

Autre formule

$$\text{Taille} = 1 \ll \log_2(2^n)$$

n = le nombre de bits alloué à la zone mémoire Exemple : $2\text{KB} = 2^{10} * 2^1 = 2^{11}$ donc $n = 11$

Saut inconditionnel

L'opcode pour un saut inconditionnel prends 5 bits et le reste est alloué pour donner l'adresse de la prochaine instruction à exécuter.

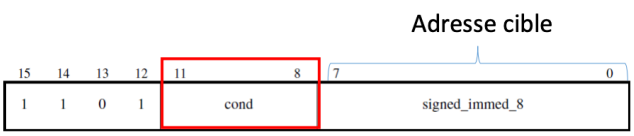
Calcul de l'adresse de saut

Pour calculer l'adresse de saut il suffit d'utiliser la formule suivante :

$$\text{Adr} = \text{PC} + \text{extension\_16bits}(\text{offset}_{11} * 2) + 4$$

Code d'instruction	Incrément
Adr	Adresse finale du saut
PC	Adresse de l'instruction courante
Extension 16 bits	Extension de l'adresse de saut en y ajoutant la valeur du bit de signe
Offset	Correspond à l'instruction moins les 5 bits de l'opcode
4	Valeur en fixe à ajouter à l'adresse de saut

Saut conditionnel



Pour calculer l'adresse de saut il suffit d'utiliser la formule suivante **attention elle est légèrement différente de celle pour le saut inconditionnel** :

$$\text{Adr} = \text{PC} + \text{extension\_16bits}(\text{offset}_8 * 2) + 4$$

Endianess

- **Big Endian**: lecture de gauche à droite
- **Little Endian**: lecture de droite à gauche

Instructions

Mnemonic	Instruction
ADC	Add with Carry
ADD	Add
AND	AND
ASR	Arithmetic Shift Right
B	Uncoditional Branch
Bxx	Conditional Branch
BIC	Bit Clear
BL	Branch and Link
BX	Branch and Exchange
CMN	Compare NOT
CMP	Compare
EOR	XOR
LDMIA	Load Multiple
LDR	Load Word
LDRB	Load Byte
LDRH	Load Halfword
LSL	Logical Shift Left
LDSB	Load Sign-Extended Byte
LDSH	Load Sign-Extended Halfword
LSR	Logical Shift Right
MOV	Move Register
MUL	Multiply
MVN	Move NOT(Register)
NEG	Negate ($* -1$)
ORR	OR
POP	Pop Register
PUSH	Push Register
ROR	Rotate Right
SBC	Subtract with Carry
STMIA	Store Multiple
STR	Store Word
STRB	Store Byte
STRH	Store Halfword
SWI	Software Interrupt
SUB	Subtract
TST	Test Bits

Décimal	Héxadécimal	Binaire
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Incrémenter le PC

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

Registres spéciaux

Registre	Objectif
R13 / R5	Stack Pointer (SP) → Stocke la position dans la pile de stockage (interruptions chaînées)
R14 / R6	Link Register (LR) →Garde l'adresse de retour (appel de fct, interruption)
R15 / R7	Program Counter (PC) →Stocke l'adresse de la prochaine instruction

Lors d'une interruption on stocke la valeur actuelle du PC dans le LR et on met la valeur de l'adresse de l'interruption dans le PC.

Puissance de 2

Puissance de 2	Résultat
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512

Réels

Standard IEEE 754 (virgule flottante)

Binary8 (quarter precision)
- nombre de bits : 8
- exposant sur 4 bits
- mantisse sur 3 bits
- biais : 7 ($2^{4-1}-1$)

Binary16 (half precision)
- nombre de bits : 16
- exposant sur 5 bits
- mantisse sur 10 bits
- biais : 15 ($2^{5-1}-1$)

Binary32 (simple precision)
- nombre de bits : 32
- exposant sur 8 bits
- mantisse sur 23 bits
- biais : 127 ($2^{8-1}-1$)

Décimal vers binaire (float)

1. Partie entière : conversion de la partie entière en binaire
2. Partie décimale : conversion de la partie décimale en binaire en utilisant le tableau ci-dessous

Binaire	Valeur
0.1	$2^{-1} = \frac{1}{2^1} = 0.5$
0.01	$2^{-2} = \frac{1}{2^2} = 0.25$
0.001	$2^{-3} = 0.125$
0.0001	$2^{-4} = 0.0625$
0.00001	$2^{-5} = 0.03125$
0.000001	$2^{-6} = 0.015625$

Binaire IEEE754 vers décimal (ex sur 32 bits)

1. Signe : le premier bit est le signe
2. Exposant : les 8 bits suivants sont l'exposant
3. Mantisse : les 23 bits suivants sont la mantisse

Règles importantes

- Si l'exposant est à 0 ou hors de la plage $2^{k-1}-1$ à 2^{k-1} pour k le nombre de bits de l'exposant, alors le nombre est un nombre non-normalisé.

Pipeline

Découpage des instructions

Découpage d'une instruction
- **Fetch** : Recherche d'instruction
- **Decode** : Décodage instruction & lecture registres opérandes
- **Execute** : Exécution de l'opération / calcul adresse mémoire
- **Memory** : Accès mémoire / écriture PC de l'adresse de saut
- **Write back** : Écriture dans un registre du résultat de l'opération

Calcul de métriques

Formule séquentielle vs pipeline

$$T_e = \text{temps de passage à chaque étape}$$
$$T_p = n * T_e = \text{temps de passage pour n étapes}$$
$$T_t = m * T_p = \text{temps total pour m personnes}$$
$$T_t = n * m * T_e$$

Formule pipeline

$$T_t = n * T_e + (m-1) * T_e$$

Nombre de cycles d'horloges

$$\frac{T_t}{T_e} = n + m - 1$$

Latence

Temps écoulé entre le début et la fin d'exécution de la tâche (instruction).

$$n * \frac{1}{\text{Frequence (temps de cycle)}}$$

Exemple

Frequence = 1 GHz / 1 instruction par nanoseconde / 10^{-9} n = nombre d'étapes = 5

$$\text{Latence} = 5 * \frac{1}{10^9} = \frac{5}{10^9} = 5 * 10^{-9} = 5 \text{ ns}$$

Débit

Nombre d'opérations (tâches, instructions) exécutées par unités de temps.

Sans pipeline

$$\frac{1}{\text{Latence}}$$

Avec pipeline

$$\frac{1}{\text{Frequence (temps de cycle)}}$$

IPC

Instructions Per Cycle : nombre d'instructions exécutées par cycle d'horloge.

$$\text{IPC} = \frac{1}{\text{ratio instruction sans arret} * 1 + \text{ratio instruction avec arret} * (\text{nb opération})}$$

Accélération

Accélération: nombre de fois plus vite qu'en séquentiel.

$$A = \frac{T_s}{T_p} = \frac{m * n * T_e}{(n+m-1) * T_e} = \frac{m*n}{n+m-1} \sim n \quad \text{(pour m très grand)}$$

- T_t : temps total
- m : nombre d'instructions fournies au pipeline
  - **Attention**: si le nombre d'instruction est **très grand**, il sera égal au nombre d'étages du pipeline.
- n : nombre d'étages du pipeline (MIPS, ARM = 5)
- T_e : temps de cycle d'horloge ($= \frac{1}{\text{Fréquence horloge}}$)

Sans fowarding

	1	2	3	4	5	6
ADD R1, R1, 1	IF	ID	EX	MA	WB	
ADD R2, R1, R0		IF	ID	EX	MA	WB
ADD R2, R1, R0		IF				ID

Avec Fowarding

	1	2	3	4	5	6
ADD R1, R1, 1	IF	ID	EX	MA	WB	
ADD R2, R1, R0		IF	ID	EX	MA	WB

Pipelining aléas

Résolution d'aléas

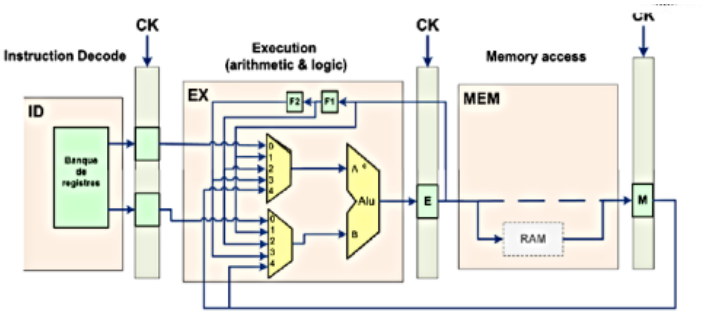
Arrêt de pipeline (hardware/software)
Hardware : Arrêter le pipeline (stall/break) -> dupliquer les opérations bloquées.
Software : Insérer des NOPs (no operation).

Sans forwarding

Les règles de gestion des aléas sont les suivantes :
- Si l'instruction dépend d'une **valeur calculée par une instruction précédente** (RAW) nous devons attendre que l'opération **write-back soit terminée**.
- Dans le cas ou nous avons des dépendances de données (WAW ou WAR) cela n'impacte pas le pipeline.

Attention dans le cas d'aléas structurels, nous ne pouvons pas faire d'opération **Memory Access (M)** et **Fetch (F)** en même temps.

Avec forwarding



Les règles de gestion des aléas sont les suivantes :
- Si l'instruction suivante dépend d'une valeur calculée par une instruction précédente, la valeur sera directement disponible dans le bloc **Execute**.
- Si un **LOAD** est fait, la valeur sera accesible directement après le bloc **Memory** dans le bloc **Execute**, donc pas besoin d'attendre jusqu'au bloc **Write Back**.

		Dep.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
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																			
4	ADD r3,r5,r6	2					F	D	E	M	W																		
5	SUB r2,r6,#1							F	D	E	M	W																	
6	SUB r4,r3,#5	4							F	D	E	M	W																
7	ADD r3,r2,r4	5,6								F	D	E	M	W															
8	LDR r2,[r7]										F	D	E	M	W														
9	ORR r4,r2,r1	3,8										F	D	D	E	M	W												
10	SUB r7,r3,#9	4,7											F	F	D	E	M	W											

Taxonomie de Flynn

Classification basée sur les notions de flot de contrôle
- 2 premières lettres pour les instructions, I voulant dire Instruction, S - Single et M - Multiple
- 2 dernières lettres pour le flot de données avec D voulant dire Data, S – Single et M - Multiple

	Single Data	**Multiple Data**
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

SISD

Machine SISD (Single Instruction Single Data) = Machine de «Von Neuman».
- Une seule instruction exécutée et une seule donnée (simple, non-structurée) traitée.

SIMD

Plusieurs types de machine SIMD : parallèle ou systolique.
- En général l'exécution en parallèle de la même instruction se fait en même temps sur des processeurs différents (parallélisme de donnée synchrone).

MISD

Exécution de plusieurs instructions en même temps sur la même donnée.

MIMD

Machines multi-processeurs où chaque processeur exécute son code de manière asynchrone et indépendante.
- S'assurer la cohérence des données,
  - Synchroniser les processeurs entre eux, les techniques de synchronisation dépendent de l'organisation de la mémoire.

Memoire cache

Calcul accès mémoire

Calcul temps moyenne d'accès

Temps moyenne d'accès a mem = T_{MAM}

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

$H =$ fréquence de succès

Fréquence de succès

- Dépend de la taille de la cache et des politiques de placement
- Miss penalty >> 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 le cache

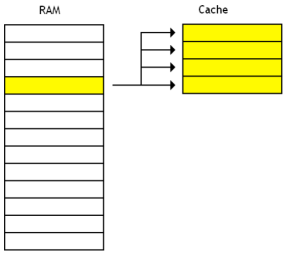
Consiste à trouver quelle est la ligne de cache dont le tag correspond à l'adresse de base demandée au répertoire.

3 stratégies (cablées, non logicielles) :

1. Fully Associative – Complètement associative
2. Direct Mapped – Associative par ensembles
3. Set Associative – Associative par voies

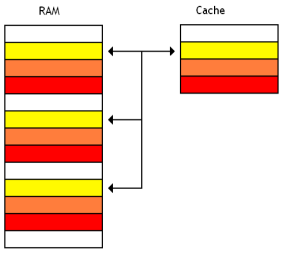
Fully Associative

- un mot de la mémoire principale peut être stocké n'importe où dans la cache
- Avantages : taux de succès très élevé (pas de conflit)
- Désavantages : trop lent (séquentiel, toutes les lignes à regarder)



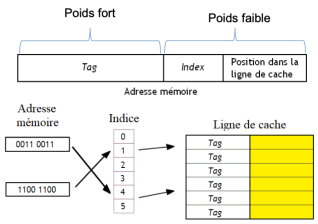
Direct Mapped

- un mot de la mémoire principale est chargé dans une ligne de cache prédéfinie (toujours la même)
- Avantages : accès rapide, car il faut vérifier qu'une ligne
- Désavantages : défaut par conflit, taux de succès mauvais



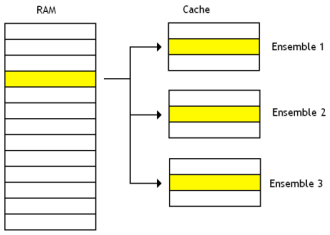
L'adresse physique (32 bits) est divisée en deux parties :

- Bits de poids faible permettent de spécifier un index, qui indique à quelle ligne de la cache l'information se trouve, ainsi que la position dans la ligne
- Bits de poids fort forment un tag, qui est à comparer avec la valeur stockée dans le répertoire
  - Bit de validité inclus

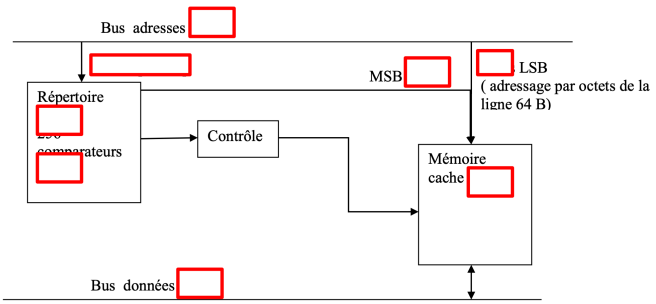


Set Associative

- Caches composés de plusieurs «caches» directement adressés accessibles en parallèle
- Chaque cache est appelé une voie
- Un mot de la mémoire peut être stocké en N positions différentes de la cache
- Diminution de la fréquence d'échec:
- Associativité $* 2$ = diminution de 20%
- Taille cache $* 2$ = diminution de 69%

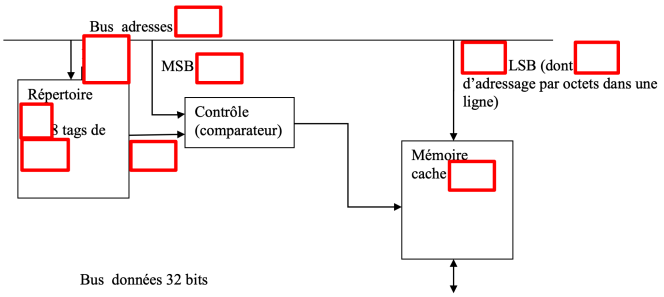


Map avec Fully associative



1. Nbr bits mémoire cache donné dans la consigne
2. Calculer le nbr de bits d'une ligne de la mémoire
3. LSB = puissance de 2 calculée de la taille de la ligne
4. MSB = puissance de 2 calculée par -> $\frac{\text{taille mémoire}}{\text{taille ligne}}$
5. Cacluler le nombre de répertoire -> $\frac{\text{taille mémoire}}{\text{taille ligne}}$
6. Comparateur et Tag = $\text{taille du bus} - \text{LSB}$

Map avec Direct Mapped



1. Nbr bits mémoire cache donné dans la consigne
2. Calculer le nbr de bits d'une ligne de la mémoire
3. Adressage par octet dans une ligne = puissance de 2 calculée de la taille de la ligne
4. Cacluler le nombre de tag dans les repertoires -> $\frac{\text{taille mémoire}}{\text{taille ligne}}$
5. MSB et Tag = $\text{taille du bus} - \text{LSB}$

6 Index = $\text{LSB} -$ puissance de 2 calculée de la taille de la ligne

Attention: si c'est une mémoire cache à voie multiple, il faut multiplier le nombre de tag par le nombre de voies.