le cnam

Jeu d'instruction et pipeline

NSY 104

Les jeux d'instructions

- Définitions
  - Partie de l'architecture avec laquelle le programmeur ou le concepteur de compilateur est en contact.
  - Ensemble des instructions que peut exécuter un processeur
  - Ensemble des circuits logiques câblés dans un processeur

- Exemples d'opérations
  - Arithmétique & logique (addition, division, etc.)
  - Transfert de données (chargement, rangement)
  - Contrôle (branchement, saut, appel, etc.)
  - Système (appels OS, gestion memoire virtuelle)
  - Flottant (addition, division, comparaison,etc.)
  - Décimal (addition, multiplication, etc.)
  - Chaines (comparaison, recherche, etc.)
  - Graphique (pixel, vertex, etc.)

Les jeux d'instructions

- Contraintes
  - Rétrocompatibilité
  - Environnement
    - Serveur
    - Ordinateur de bureau
    - Embarqué / enfoui
- Exemples
  - MMX
  - 3DNow!
  - SSE
  - AltiVec

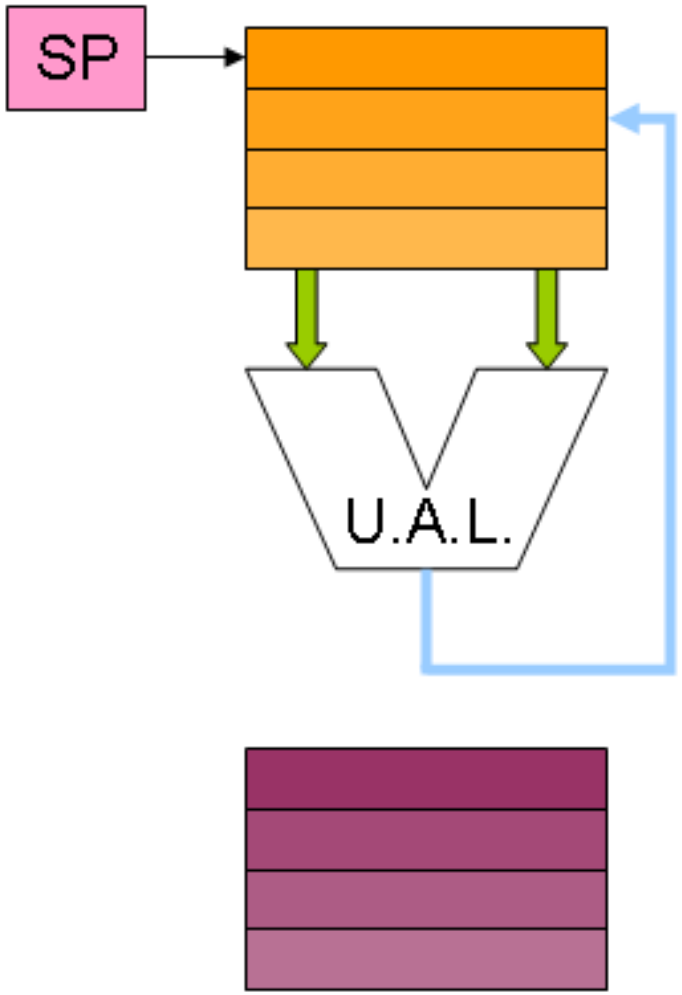
Les jeux d'instructions

- Classification basée sur le stockage interne au processeur
  - Architecture à pile
    - Opérandes au sommet de la pile
  - Architecture à accumulateur
    - Un opérande est l'accumulateur
  - Architecture à registres généraux
    - Opérandes explicites:
      - Registre-mémoire
        - Accès mémoire comme partie d'une instruction quelconque
      - Registre-registre et chargement-rangement
        - Instructions dédiées pour accéder à la mémoire (LOAD, STORE)

Les jeux d'instructions

- Architecture à pile
  - Opérandes au sommet de la pile

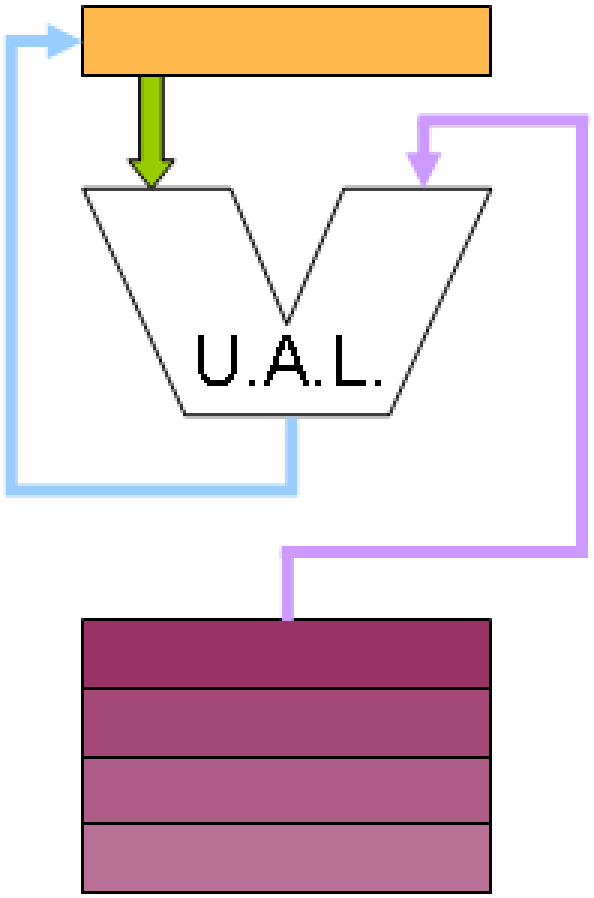
- C = A + B
  - Push A
  - Push B
  - Add
  - Pop C



Les jeux d'instructions

- Architecture à accumulateur
  - Un opérande est l'accumulateur

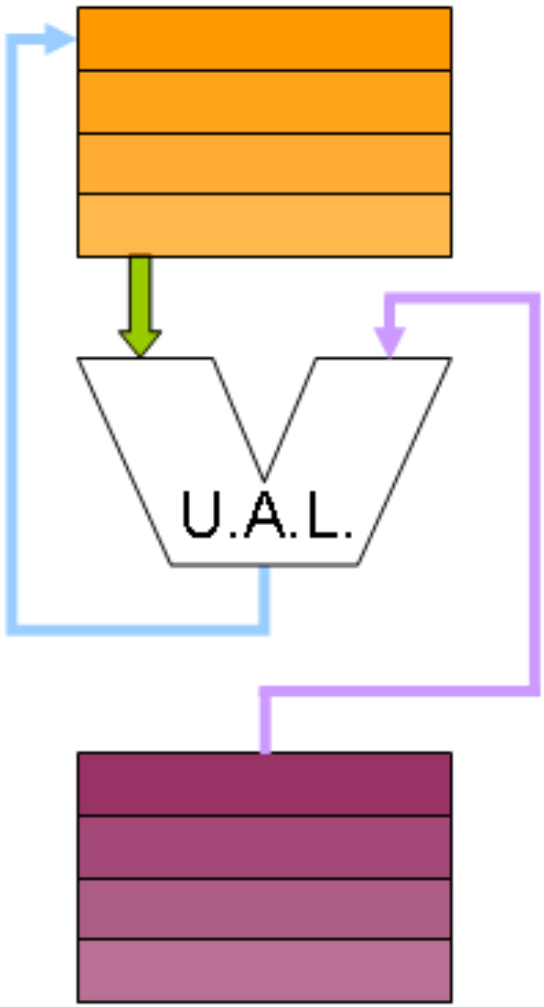
- C = A + B
  - Load A
  - Add B
  - Store C



Les jeux d'instructions

- Architecture de type registre-mémoire
  - Accès mémoire comme partie d'une instruction quelconque

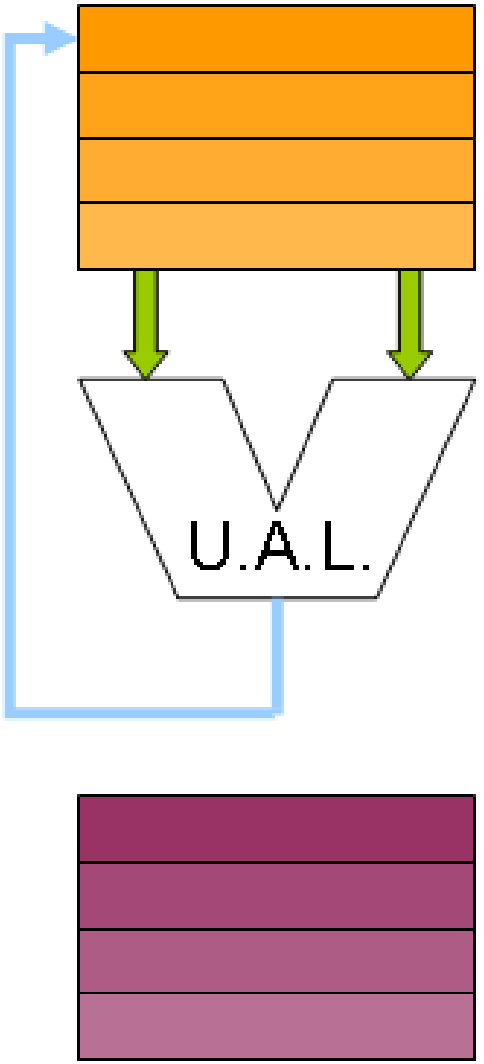
- C = A + B
  - Load R1, A
  - Add R3, R1, B
  - Store R3, C



Les jeux d'instructions

- Registre-registre
- Chargement-rangement
  - Instructions dédiées pour accéder à la mémoire

- C = A + B
  - Load R1, A
  - Load R2, B
  - Add R3, R2, R1
  - Store R3, C



Adressage des opérandes

- Les modes d'adressage

Mode d'adressage	Instruction complète	Signification
Registre	add r4, r3	[r4] <= [r4] + [r3]
Immédiat ou littéral	add r4, #3	[r4] <= [r4] + 3
Déplacement ou basé	add r4,100(r1)	[r4] <= [r4] + [100 + [r1]]
Indirect par registre	add r4, (r1)	[r4] <= [r4] + [[r1]]
Indexé	add r3, (r1 + r2)	[r3] <= [[r1] + [r2]]
Direct ou absolu	add r1, (1001)	[r1] <= [r1] + [1001]
Indirect via mémoire	add r1, @(r3)	[r1] <= [r1] + [[[r3]]]
Auto – incrémenté	add r1, (r2)+	[r1] <= [r1] + [[r2]] [r2] <= [r2] + d
Auto – décrémenté	add r1, -(r2)	[r2] <= [r2] – d [r1] <= [r1] + [[r2]]
Indexé étendu	add r1, 100(r2)[r3]	[r1] <= [r1] + [100+[r2]+[r3] *d]

d, taille d'un élément

Le processeur

- Traitement *normal* (non pipelinée) du flot d'instructions

- Traitement d'une instruction = séquence d'étapes
  - lecture, décodage, etc.
  - Lorsque le traitement se situe à une étape, les éléments responsables des autres étapes sont au repos (inutilisés)

Implémentation simple de RISC

- Chaque instruction prendrait, au plus, 5 cycles d'horloge
  - Sous ensemble:
    - Chargement/rangement
    - UAL entières
    - Branchement
- Branchement en 2 cycles
  - Peut se terminer à DI
- Rangements en 4 cycles
  - Se termine à MEM
- Autres en 5 cycles

Indicateurs de performance

- Calcul du CPI (*cycle per instruction*)
  - Distribution des types d'instruction
    - Branchements : 12%
    - Rangements : 10%

  CPI = (12x2 + 10x4 + 78x5)/100

  CPI = 4,54

- Temps d'exécution moyen
  - Cycle d'horloge x CPI
  - Exemple T = 1ns : 4,54 ns

Indicateurs de performance

L'exécution d'une instruction nécessite plusieurs temps de cycle, c'est ce que l'on appelle le CPI (Cycles per Instruction ou nombre de cycles par instruction).

Le temps d'exécution d'un programme est alors donné par la formule suivante (si on considère que toutes les instructions ont le même CPI) :

Texec = Nins x CPI x Tcycle

avec :

- Texec : temp d'exécution du programme
- Nins : nombre d'instructions
- CPI : nombre de cycles par instructions
- Tcycle : temps de cycle

Indicateurs de performance

L'ensemble des améliorations des microprocesseurs visent à diminuer le temps d'exécution du programme.

Deux types d'améliorations sont possibles :

la première consiste à diminuer le temps de cycle, pour cela il suffit d'augmenter la fréquence de fonctionnement du processeur.

la seconde consiste à diminuer le nombre d'instructions ou diminuer le nombre de cycles par instruction. Or il semble que dans ce cas, le produit Nins x CPI reste constant :

en effet si on diminue le nombre d'instructions on crée des instructions complexes (CISC) qui nécessitent plus de cycles pour être décodées

si par contre on diminue le nombre de cycles par instruction, on crée des instructions simples (RISC) et on augmente alors le nombre d'instructions nécessaires pour réaliser le même traitement qu'une instruction CISC.

Le pipeline

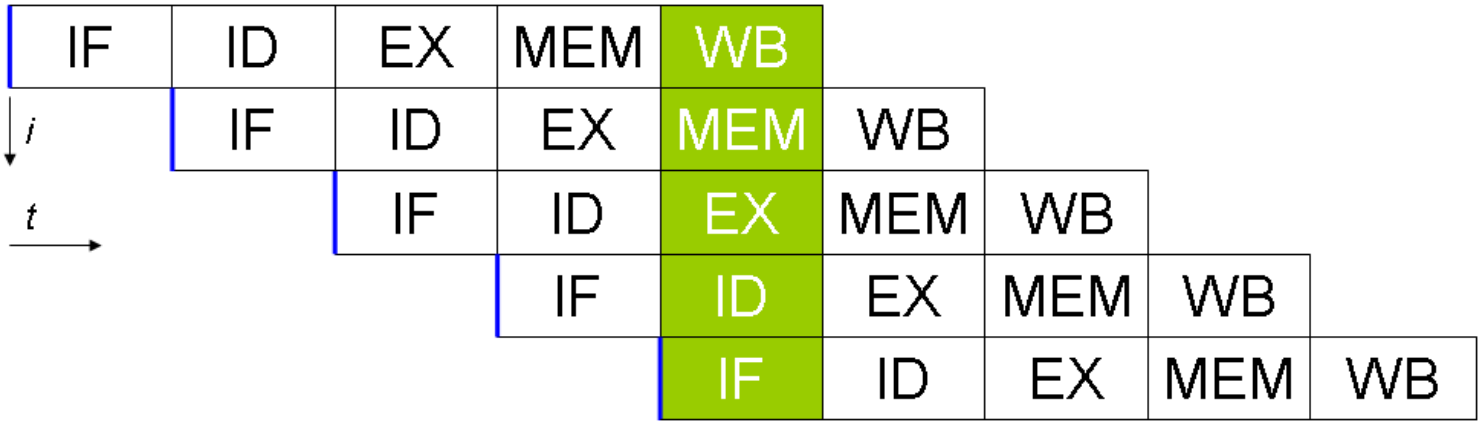
- Idée générale
  - Lancer le traitement d'une instruction avant que la précédente ne soit terminée
  - Recouvrement des instructions
  - On exploite le parallélisme entre les instructions d'un flot d'instructions séquentielles
  - Optimiser le temps d'utilisation des différents éléments du processeur.
- Découpage des instructions en sous-parties élémentaires
  - En relation avec les étapes de traitement de l'instruction
  - Définition des étages du pipeline
  - « travail à la chaine »
- Exécution des sous-parties élémentaires dans les étages correspondants du pipeline.

Le pipeline

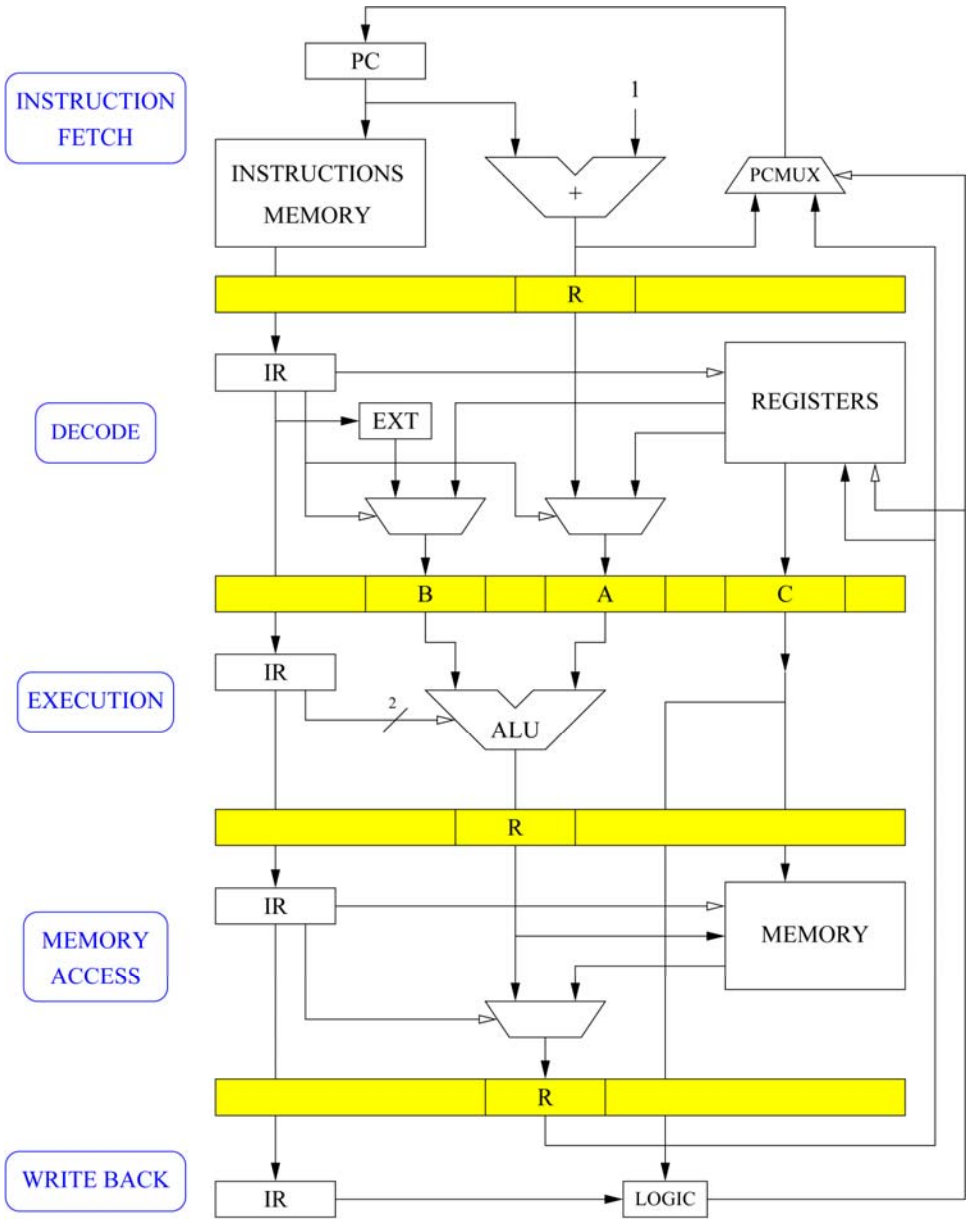
- Le temps passé par une instruction dans un étage est appelé (temps de) cycle processeur
  - La longueur d'un cycle est déterminée par l'étage le plus lent.
  - Souvent égal à un cycle d'horloge, parfois 2
- L'idéal est d'équilibrer la longueur des étages du pipeline
  - Sinon le étages les plus rapides 'attendent' les plus lents
  - Pas optimal
- Insertion de registres intermédiaires entre les étages
  - Registres pipeline

Le pipeline

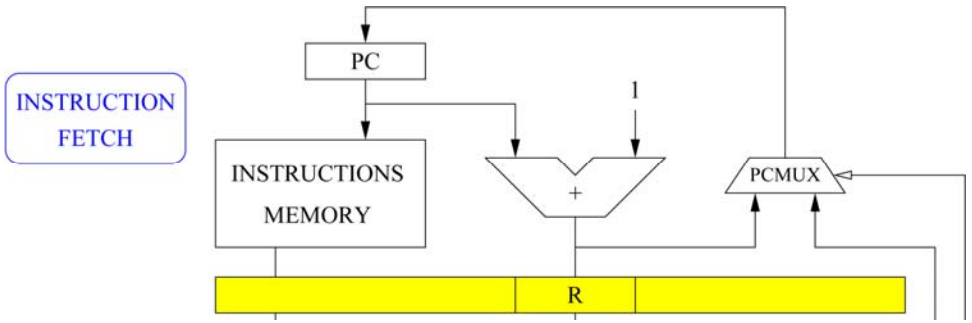
- Exemple avec un pipeline à 5 étages :
  1) Lecture de l'instruction (IF)
  2) Décodage de l'instruction (ID)
  3) Exécution de l'instruction (EX)
  4) Accès mémoire (MEM)
  5) Ecriture du résultat (WB)



Le pipeline


INSTRUCTION FETCH
DECODE
EXECUTION
MEMORY ACCESS
WRITE BACK
PC
1
+
INSTRUCTIONS MEMORY
PCMUX
R
IR
EXT
REGISTERS
B
A
C
IR
2
ALU
R
IR
MEMORY
R
IR
LOGIC

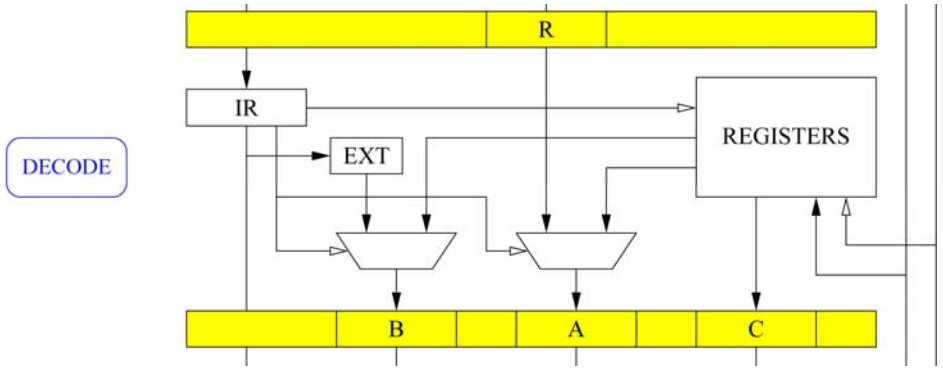
Le pipeline



1- Lecture de l'instruction (IF)

- La prochaine instruction à exécuter est chargée à partir de la case mémoire pointée par le compteur de programme (CP) dans le registre d'instruction RI.
- Le compteur de programme est incrémenté pour pointer sur l'instruction suivante.

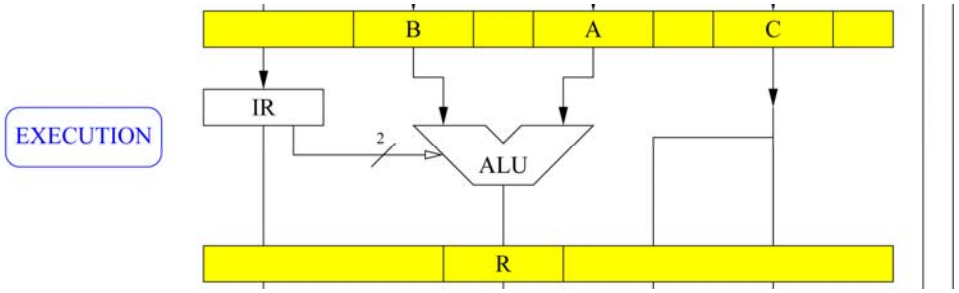
Le pipeline



2- Décodage de l'instruction (ID)

- Cette étape consiste à préparer les arguments de l'instruction pour l'étape suivante (UAL) où ils seront utilisés. Ces arguments sont placés dans deux registres A et B.
  - Si l'instruction utilise le contenu de un ou deux registres, ceux-ci sont lus et leurs contenus sont rangés dans A et B.
  - Si l'instruction contient une valeur immédiate, celle-ci est étendue (signée ou non signée) à 16 bits et placée dans le registre B.
  - Pour les instructions de branchement avec offset, le contenu de PC est rangé en A et l'offset étendu dans B.
  - Pour les instructions de branchement avec un registre, le contenu de ce registre est rangé en A et B est rempli avec 0.
  - Les instructions de rangement ST* mettent le contenu du registre qui doit être transféré en mémoire dans le registre C.

Le pipeline



3- Exécution de l'instruction (EX)

Cette étape utilise l'UAL pour combiner les arguments. L'opération réalisée dépend du type de l'instruction.

- Instruction arithmétique ou logique (ADD, AND et NOT)

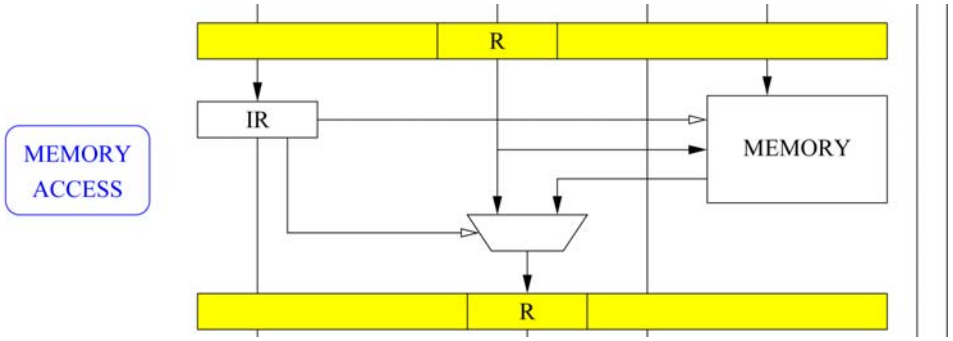
  Les deux arguments contenus dans les registres A et B sont fournis à l'unité arithmétique et logique pour calculer le résultat.

- Instruction de chargement et rangement (LD* et ST*)

  Le calcul de l'adresse est effectué à partir de l'adresse provenant du registre A et de l'offset contenu dans le registre B.

- Instruction de branchement (BR*, JMP, JSR, JSRR et TRAP)
  - Pour les instructions contenant un offset, addition avec le contenu du CP.
  - Pour les instructions utilisant un registre, le contenu du registre est transmis.

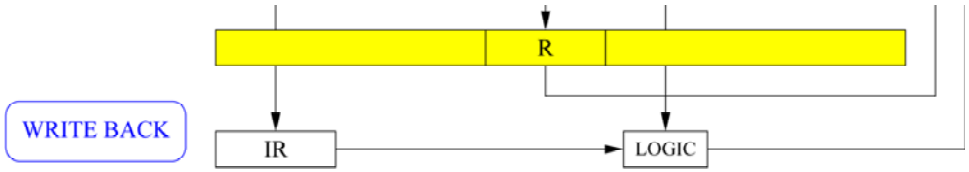
Le pipeline



4- Accès mémoire (MEM)

- Cette étape est uniquement utile pour les instruction de chargement et de rangement.
  - Pour les instructions arithmétiques et logiques ou les branchements, rien n'est effectué. L'adresse du mot mémoire est contenue dans le registre R.
- Dans le cas d'un rangement, la valeur à ranger provient du registre C.
- Dans le cas d'un chargement, la valeur lue en mémoire est mise dans le registre R pour l'étape suivante.

Le pipeline



5- Ecriture du résultat (WB)

- Le résultat des opérations arithmétiques et logiques est rangé dans le registre destination.
- La valeur lue en mémoire par les instructions de chargement est aussi rangée dans le registre destination.
- Les instructions de branchement rangent la nouvelle adresse dans CP.

Exemples de profondeur de pipeline

Intel Pentium 4 Prescott	31
Intel Pentium 4	20
AMD K10	16
IBM POWER5	16
IBM PowerPC 970	16
Intel Core 2 Duo	14
Intel Pentium II	14
Sun UltraSPARC IV	14
Sun UltraSPARC IIi	14
AMD Opteron 1xx	12
AMD Athlon	12
IBM POWER4	12
Intel Pentium III	10
Intel Itanium	10
MIPS R4400	8
Motorola PowerPC G4	7

Le pipeline

- Le gain se situe au niveau du débit
  - Le temps de traitement d'une instruction n'est pas réduit.
  - Il est même souvent augmenté
    - Gestion du passage entre les étages
    - Temps de stabilisation

- Exemple avec le pipeline précédent :
  - Débit max atteint lorsque tous les étages sont chargés :
    - 1 instruction par cycle

Le pipeline

- Dans un cas idéal
  - L'accélération serait donnée par le nombre d'étages du pipeline
  - Tps moy. instruction NP / tps moy. instruction pipelinée

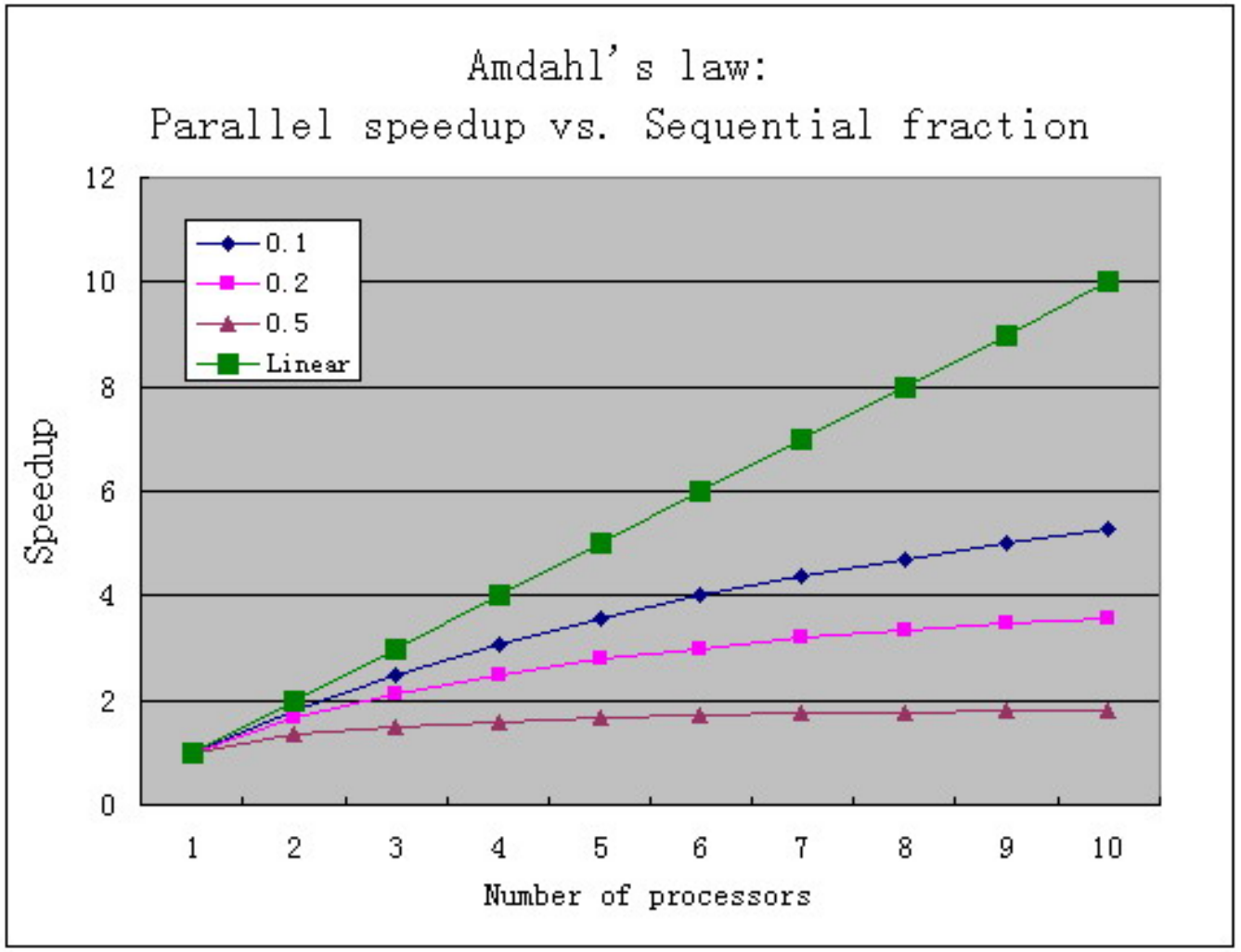
- Le pipeline peut diminuer, selon les approches adoptés
  - le nombre de cycles par instructions (CPI)
  - le temps de cycle
  - ou une combinaison des deux

La loi d'Amdahl

- La loi d'Amdahl, énoncée par Gene Amdahl, exprime le gain de performance qu'on peut attendre d'un ordinateur en améliorant une composante de sa performance. Sous sa forme générale elle indique que le gain de performance égale le temps d'exécution d'une tâche complète sans l'amélioration divisé par le temps d'exécution de la même tâche avec l'amélioration.
- Soit :
- T le temps d'exécution d'un programme sur le système d'origine.
- Ta le temps d'exécution du même programme sur le système amélioré.
- s est la fraction du temps T concernée par l'amélioration.
- Ac est l'accélération obtenue par l'amélioration.
- S est l'accélération globale

$$Ta = (1-S).T + (s.T/Ac)$$

La loi d'Amdahl



Le pipeline - Aléas

- Les aléas structurels
  - Ils interviennent lors des conflits de ressources.
- Les aléas de données
  - Ils interviennent lorsqu'une instruction dépend du résultat d'une instruction précédente.
- Les aléas de contrôle
  - Ils interviennent lors de l'exécution des branchements
  - et des instructions modifiant le CP
- Conséquences possibles
  - Blocage du pipeline
  - Augmentation du CPI

Le pipeline - Aléas

- Les aléas structurels
  - S'expliquent par le fait qu'une unité fonctionnelle n'est pas complètement pipelinée
  - Ou qu'une ressource n'a pas été dupliquée suffisamment
    - Exemple d'un bus d'écriture unique dans les registres et un besoin d'écriture simultané provoqué par 2 instructions durant le même cycle d'horloge
  - Ce problème est généralement résolu en séparant la mémoire où se trouvent les instructions de celle où se trouvent les données.
    - Ceci est réalisé au niveau des caches de niveau 1. Le micro-processeur doit alors avoir deux bus distincts pour accéder simultanément aux deux caches.

Le pipeline - Aléas

- Ces aléas peuvent amener à suspendre des opérations
  - Etage vacant: bulle (*bubble* = NOP - *No OPeration*)
  - Dans l'attente
    - De la libération de la ressource (structurel)
    - De la disponibilité de la valeur (donnée)

Programme	Cycles d'horloge											
	1	2	3	4	5	6	7	8	9	10	11	12
Inst. n° 1	IF	ID	IE	MA	WB							
Inst. n° 2		IF	ID	IE	MA	WB						
Inst. n° 3			IF	ID	Bulle		IE	MA	WB			
Inst. n° 4				IF	Bulle		ID	IE	MA	WB		
Inst. n° 5							IF	ID	IE	MA	WB	
Inst. n° 6								IF	ID	IE	MA	WB

Le pipeline - Aléas

- Exemple (aléa structurel à cause d'un seul bus mémoire données et instructions)
  - LDR R7,R6,0
  - ADD R6,R6,1
  - ADD R0,R0,1
  - ADD R1,R1,1

Programme	Cycles d'horloge							
	1	2	3	4	5	6	7	8
LDR R7,R6,0	IF	ID	IE	MA	WB			
ADD R6,R6,1		IF	ID	IE	MA	WB		
ADD R0,R0,1			IF	ID	IE	MA	WB	
ADD R1,R1,1				IF	ID	IE	MA	WB

Programme	Cycles d'horloge										
	1	2	3	4	5	6	7	8	9	10	11
LDR R7,R6,0	IF	ID	IE	MA	WB						
ADD R6,R6,1		IF	ID	IE	MA	WB					
ADD R0,R0,1			IF	ID	IE	MA	WB				
ADD R1,R1,1					IF	ID	IE	MA	WB		
Inst. n° 5						IF	ID	IE	MA	WB	
Inst. n° 6							IF	ID	IE	MA	WB

Le pipeline - Aléas

- Les aléas de données
  - S'expliquent par le fait que la temporalité des instructions est modifiée par le pipeline
    - Le recouvrement des opérations peut provoquer l'apparition d'états faux (lecture de valeurs erronées par exemple)

    - ADD R1,R2,R3
    - SUB R4,R1,R5
    - AND R6,R1,R7
    - OR R8,R1,R9

Le pipeline - Aléas

- Exemple (aléa de données)
  - LDR R1,R6,0
  - ADD R0,R1,R2

Programme	Cycles d'horloge					
	1	2	3	4	5	6
`LDR R1,R6,0`	IF	ID	IE	MA	WB	
`ADD R2,R1,R0`		IF	ID	IE	MA	WB

Le pipeline - Aléas

- Le conflit d'accès à la mémoire se produit à chaque fois qu'une instruction de chargement ou de rangement est exécutée (MA)
- Celle-ci rentre systématiquement en conflit avec le chargement d'une instruction qui a lieu à chaque cycle d'horloge (IF)
- Ce problème est résolu en séparant la mémoire où se trouvent les instructions de celle où se trouvent les données.
  - Ceci est réalisé au niveau des caches de niveau 1.
  - Le micro-processeur doit alors avoir deux bus distincts pour accéder simultanément aux deux caches.

Le pipeline - Aléas

- Exemples
  - A
    - ADD R1, R2, R3 // R1 = R2 + R3
    - STORE R1, 1000 // C(1000) = R1
  - B
    - MOV R1, #1000 // R1 = 1000
    - JUMP R1 // Saut