TP Archi

.byte : Toutes les Variables qu'on peut stocker sur 8 bits. $2^8 = 256$ max = 127
c: .byte 'a'

.half : Toutes les Variables qu'on peut stocker sur 16 bits $2^{16} = 65536$ max = 32767
n1: .half 26

.word : Toutes les Variables qu'on peut stocker sur 32 bits $2^{32} = 4294967296$ max = 2147483647
n2: .word 9856

.space pour declarer un Tableau.
tab: .space 40 → nombre d'octets reservé dans la MC pour Stocker ce tableau

.asciiz pour declarer une chaine de caractéres
message1: .asciiz "Donnez la Valeur d'entrée"

.asciiz : pour declarer une chaine de caracteres qui se termine par un caractére null
m2: .asciiz "Donnez la Valeur 2"

.float : Toutes le Variables flottantes qu'on peut stocker sur 32 bits.
pi: .float 3.4159

.double : Toutes les Variables flottantes qu'on peut stocker sur 64 bits.
ti: .double 6.28944

1/ Afficher un entier:
exemple①

```
.data
  nbr: .byte 45
.text
li $v0,1
lb $a0,nbr
syscall
```

lb: load byte

exemple②:

```
.data
 nbr: .half 128
.text
li $v0,1
lh $a0,nbr
syscall
```

lh: load half

exemple③

```
.data
nbr: .word
.text
li $v0,1
lw $a0,nbr
syscall
```

lw: load word

2/ Afficher un reel:
exemple①

```
.data
 pi: .float 3.14
.text
li $v0,2
lwc1 $f12,pi
syscall
```

lwc1: load word into coprocessor 1 (FPU)

3/ Afficher un Caractere:
exemple①

```
.data
 lettre: .byte 'a'
.text
li $v0,4
la $a0,lettre
syscall
```

la: load address

4/ Afficher une chaine de caractere:
exemple①

```
.data
 message: .asciiz "hello world"
.text
li $v0,4
la $a0,message
syscall
```

5- Addition des entiers: exemple①:

```
.data
 nb1: .byte 45
 nb2: .byte 23
.text
lb $t1,nb1
lb $t2,nb2
add $t3,$t1,$t2
li $v0,1
move $a0,$t3
syscall
```

add: addition with overflow

TP Archi

6- Addition des flottants:

exemple ①:

```
.data
reel1: .float 3.198
reel2: .float 9.487
.text
lwc1 $f1, reel1
lwc1 $f2, reel2
add.s $f12, $f1, $f2
li $V0, 2
syscall
```

add.s: floating point addition single precision.

exemple ②:

```
.data
reel1: .double 3.198
reel2: .double 9.487
.text
ldc1 $f0, reel1
ldc1 $f2, reel2
add.s $f12, $f0, $f2
li V0, 3
syscall
```

Ldc1: load double word coprocessor 1 (FPU)

7/ Addition des entiers entrés au clavier:

```
.data
m1: .asciiz "donnez le nombre 1"
m2: .asciiz "donnez le nombre 2"
m3: .asciiz "la somme des deux nombres est :"
.text
# Affichage m1
li $V0,4
la $a0, m1
syscall
li $V0,5 → code pour lire un entier
syscall
move $t1,$V0
li $V0,4
la $a0, m2
syscall
li $V0,5
syscall
move $t2,$V0
li $V0,4
la $a0,m3
syscall
add $t1,$t1,$t2
li $V0,1
move $a0,$t1
syscall
```

8/ le plus grand nombre de 2 entiers entrées

```
.data
m1: .asciiz "donnez le nombre 1"
m2: .asciiz "donnez le nombre 2"
m3: .asciiz "le plus grand nombre est :"
.text
li $V0,4 → code pour afficher une chaine
la $a0, m1
syscall
li $V0,5 → code pour lire un entier
syscall
move $t1,$V0
li $V0,4
la $a0,m2
syscall
li $V0,5
syscall
move $t2,$V0
li $V0,4
la $a0,m3
syscall
bgt $t1,$t2,grand
li $V0,1
move $a0,$t2
syscall
bfin
grand:
li $V0,1
move $a0,$t1
syscall
fin:
li $V0,10 } code de fin
syscall
```

bgt: branch if Greater than

9/ Table de multiplication:

```
.data
m1: .asciiz "Donnez la Valeur de A"
m2: .asciiz "Donnez la Valeur de B"
saut: .asciiz "\n"
.text

li $V0,4
la $a0,m1
syscall

li $V0,5
syscall
move $t1,$V0

li $V0,4
la $a0,m2
syscall

li $V0,5
syscall
move $t2,$V0

mul $t3,$t2,$t1

move $t4,$t1

etiquette:

li $V0,1

move $a0,$t4
syscall
li $V0,4
la $a0,saut  syscall
beq $t4,$t3,terminer
add $t4,$t4,$t1
b etiquette

terminer:

li $V0,10
syscall
```

bne: branch if not equal

exemple:

bne $t0,$t1,label

Si la Valeur contenu dans $t0 ≠ Valeur dans $t1 alors la prochaine instruction à executer est celle placée après l'étiquette label

m2: "Donnez l'élément"

beq: Branch if equal

exemple:

beq $t0,$t1,label

Si Valeur contenue dans $t0 = contenue $t1 Dans la prochaine instruction à executer est celle placée après l'étiquette.

J: Jump unconditionally:

exemple:

J label

la prochaine instruction à executer est celle après l'étiquette label PC ← label

Jr: jump register unconditionally:

exemple

Jr registre

la prochaine instruction à executer est celle à l'adresse dans le registre: PC ← registre

Jal: Jump and Link:

exemple:

Jal label

La prochaine instruction à executer est celle placée après l'étiquette label et l'adresse de l'instruction suivant l'instruction courante (Adresse de retour) est stockée dans $ra: $ra ← PC+4

PC ← label

div

Mult $t0,$t1

mflo $t3

move $a0,$t3

.data
message: .asciiz ....
.text
main:

Jal display Message

display message:

li $V0,4

la $a0,message

syscall

Jr $ra

li $V0,6 ← Code pour lire un float

li $V0,7 ← " " " " double

li $V0,2 ← Lecture texte

* Espace d'adressage Total : Combien de Mots Mémoire ?

$2^{\text{Taille bus @}}$ mots mémoire de bits

* Capacité Totale :

$$Cap = 2^{T@} * T\text{bus de données}$$

Remarque :

Si on a pas la Taille du bus de données dans une question pour calculer Taille du bus @, on prend sa Taille à 1 Ø après faut trouver le bus d'adresse en bits.

* Nbre de Mode d'adressage que peut avoir un Processeur :

$2^{\text{Taille champ mode d'adressage}}$.

* Taille du Repertoire d'instruction :

$2^{\text{Taille champ code operation}} + 2^{\text{Taille du champ mode d'adressage}} / 2^{\text{Taille du champ code OP}} + 2^{\text{Taille du champ N registre}}$

* La Plage de Valeur de Constante :

CA2 : $[-2^{n-1}, 2^{n-1}-1]$

SVA et CA1 : $[-2^{n-1}+1, 2^{n-1}-1]$

* Le nombre de registre :

$2^{\text{Taille champ n registre}}$

Taille du bus @ = Taille du CØ

1 GHz → 10^9 Hz
1 MHz → 10^6 Hz
1 KHz → 10^3 Hz
1 µHz → 10^{-6} Hz
1 Ø → 10^{-39} GO

1 s → 10^{-9} s = 1 ns
10^{-6} s = 1 µs
10^{-3} s = 1 ms

CHAPITRE 3 : Notions sur les instructions d'un ordinateur

03/02/2022

I/ Mode d'adressage :

* Adressage implicite :
une instruction ne comportant pas d'operande.

* Adressage immediat :
L'operande est une valeur immédiate se trouvant directement dans l'instruction.

* Adressage direct :
L'adresse de la donnée à chercher se trouve directement dans l'instru

* Adressage indirect :
L'instruction contient une adresse qui contient l'adresse de la donnée à chercher, ce mode necessite 2 accès à la memoire.

* Adressage par registre :
L'information ne se trouve pas en mémoire mais dans un registre interne du processeur et ne necessite pas d'accès à la Memoire.

* Adressage basé :
L'adresse de l'operande est obtenue en additionnant le contenu du registre de base au contenu du champ adresse de l'instruction.

* Adressage relatif :
L'adresse de l'operande est obtenue en additionnant le contenu du CO au contenu de champ adresse de l'instruction, ce mode est utilisé dans le instructions de branchement.

II/ Cycle d'execution d'une instruction:

1- phase de recherche (fetch):

- Un ordre de lecture sera donné par le séquenceur à la mémoire centrale. au bout d'un certain Temps, dit Temps d'accès mémoire, le contenu de la case mémoire selectionnée sera disponible sur le bus de données.
- cette instruction sera ensuite transférée dans le Registre RI.
- mise à jour du CO pour preparer l'adresse de l'instruction pre.

2- phase decodage:

- Le RI contient l'instruction.
- un ensemble d'ordres sont préparés par le séquenceur pour l'execution de l'instruction.
- Si l'inst necessite une donnée à recup de la MC, des ordres sont generés pour chercher cette donnée et la Mettre dans un registre à usage général.

3- La phase d'execution:

- L'instruction est proprement executée.

Remarque:

Le nombre de phases d'execution d'une instruction peut etre superieur à 3 phases.

comme: mips il a 5 phases:

- Recherche d'instruction, decodage, recherche d'operandes, execution, ecriture resultat.

III/ Temps d'execution d'une instruction :

$$TE(I_i) = CPI_{(i)} \cdot T_c$$

$CPI_{(i)}$: Nombre de cycles necessaires pour l'execution de l'instruction $I_{(i)}$

Le nombre de cycle : depend de la complexité de l'inst et aussi du mode d'adressage : il est plus long d'accéder à la MP qu'à un registre du processeur.

Nbre ~~de cycle~~ moyen de cycles par instruction (CPI)

La durée d'un cycle (T_c) : depend de la frequence (f) d'horloge de l'ordinateur. Plus l'Horloge bat rapidement (grande fréquence), plus un cycle est court et plus on execute un grand nombre d'inst par S.

IV/ Temps d'execution d'un programme :

Nbre moyen de cycle par inst.

$$CPI = \frac{\text{Nbre de cycles d'un Prgm}}{\text{Nbre d'instructions de ce prgm}}$$

$$T_{exec}(P) = N_{inst}(P) \times CPI \times T_c$$

$T_{exec}(P)$: Temps d'execution du Prgm.

$N_{inst}(P)$: Nbre d'instruction.

CPI : nbre de cycles par inst (nbre moyen de cycles par inst)

T_c : Temps de cycle

* Nbre d'instructions qu'un processeur peut executer par seconde

$$MIPS = \frac{f}{CPI \cdot 10^6}$$

pour mesurer les performances d'un processeur.

* Comment Augmenter la performance des processeurs,

- Augmenter la fréquence d'horloge, mais cette methode presente des limites techniques et elle est très couteuse.
- Ameliorer les Accès memoire : differents de cache.
- Recourir à un nombre élevé de coeurs de calcul.
- Permettre l'execution simultanée de plusieurs instructions: comme pipelines d'instructions, architecture super scalaire.

II/ Le Pipeline :

* 1er Methode : Maniere Séquentiel

$$T_T = K * T_V \Longrightarrow T_T = K * N * T_p$$

* 2ème Methode : Le Travail à la chaine :

$$T_T = T_1 + (K-1) * T_p \Rightarrow = N * T_p + (K-1) * T_p$$

$$\Rightarrow = (N+K-1) * T_p$$

T_p : Temps d'une phase
N : nombre de phases
T_T : Temps Total.

Conclusion:

Travailler avec la 2ème Methode permet:

- Un gain de Temps
- Une meilleure gestion (exploitation) des robots à cet assemblage.
- Un meilleur débit.

1/ Decoupage d'une instruction:

* Pipeline à 3 niveaux (ARM7):
  Fetch, Decode, Execute.
* Pipeline à 5 niveaux (processeur ARM9, MIPS):
  IF, ID, EX, MEM, WB

IF: Instruction Fetch

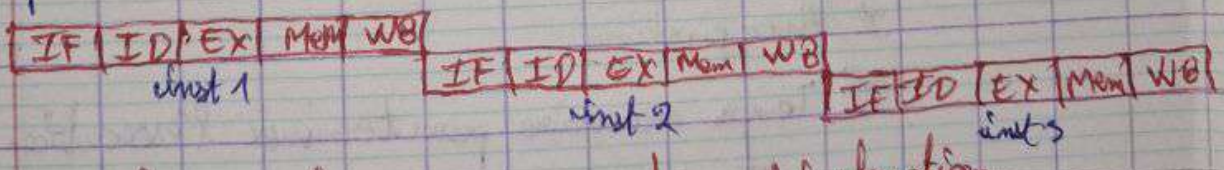
ID: Instruction decode et lecture des registres operandes.

EX: Execution / calcul de l'adresse Mémoire

MEM: Memory Access / Cache Acces.

WB: Write-Back; stocke le resultat dans un registre.

* ~~Les~~ microprocesseur sans pipeline: il faut attendre que les 5 etapes aient été réalisées sur l'instruction en cours pour passer à l'instruction suivante.



5 cycles d'Horloge pour executer 1 instruction.

* microprocesseur avec pipeline : une fois que la 1ère inst a terminé l'étape IF et passe dans l'étape ID, la 2ème inst peut commencer l'étape IF :

5 cycles d'horloges pour executer 1 inst. et
7 cycles d'horloge " " " 3 inst.

Remarque :
en mode pipeline le gain se situe au niveau du débit, le Temps de traitement d'une instruction reste le meme.

I/ Gain de performance :

I / Sans pipeline :

$T_{inst} = N * T_c$. ce Temps (durée d'execution d'une inst) est dit latence. Donc pour executer K instructions de maniere sequ il faut un Temps Total :

$$T_T = k * T_{inst} \Longrightarrow T_T = K * N * T_c$$

II / Avec pipeline :

Le Temps necessaire pour terminer l'execution de la 1ère inst :

$$T_{inst1} = N * T_c$$

Donc le Temps Totale pour terminer l'execution de la K^ème inst de maniere pipeline est :

$$T_T = T_{inst} + (K-1) * T_c$$
$$= N * T_c + (K-1) * T_c$$
$$= (N+K-1) * T_c$$

Speed up (acceleration) $= \frac{K * N * T_c}{(N + K - 1) * T_c} = \frac{K * N}{(N + K - 1)}$