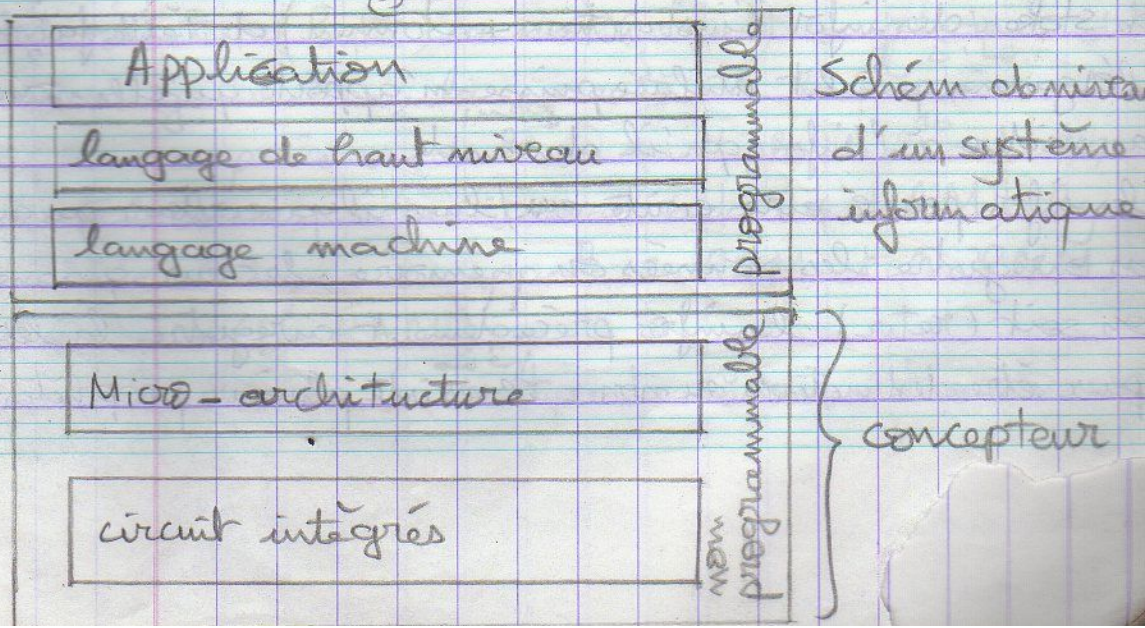
Chapitre I :

- Introduction :

- un ordinateur est une machine qui traite une information par un organe d'entrée suivant un programme qui délivre en info sur un organe de sortie.

- La partie de l'ordinateur chargée de traitement d l'info est appelé unité central (U.C) ou central processing Units (CPU) dans le sens le plus général l'architecture est la conception est l'organisation des composants matériels de l'ordinateur basé sur la technologie des circuits intégré, et plus particulièrement du processeur.

Le fonctionnement d'un ordinateur peux s'envisager suivant la hiérarchie.



Schém de mista d'un système informatique

pour programmé un ordinateur, on utilise des lan
dits evolués ou des hauts niveaux tel que: pasc
C++ ; JAVA, ada, delphi
mais l'unité central ne peux exicuter que les
ôn machine qui sont des codes binaires.
le therme langage disigné un jeu des instruct
est des règle syntasique, à l'aide d'un langa
evolué, le programmeur ecrit un code source
sera pas exicuté par la machine mais traduit
machine ou code d'objet. si le role des compil
ou assembleur ou interpreteur.
un interpreteur ne produit pas le code d'objet il
les instructions directement au fure et à mesur
l'exicution
La Memoire: on appelle mémoire tout despositif
de stoker des infor (instruction + donnée) de telle f
l'organne qui lit utilise puisse n'apporte qu'elle
accsider à l'infor qu'il demande.
les infor peuve être ecrite ou lues il y a ecriteur
on enregistre les données on memoire, lecteur q
on soit (retir) les infor précédement enregistré le
peux être destructives ou non - il existe plusieur prop

pour programmé un ordinateur, on utilise des langage dits evolués ou des hauts niveaux telque: pascal, C++, JAVA, ada, delphi.
mais l'unité central ne peux exicuter que les instru ion machine qui sont des codes binaires.
le therme langage disigne un jeu des instructions est des règle syntasique, à l'aide d'un langage evolué, le programmeur ecrit un code source qui ne sera pas executé par la machine mais traduit en cod machine ou code d'objet. si le role des compelateur ou assembleur ou interpreteur.
un interpreteur ne produit pas le code d'objet il traduit les instructions directement au fure et à mesure de l'exicution
La Memoire: on appelle mémoire tout desposetife capable de stoker des infor (instruction + donnée) de telle façon qu l'organne qui lit utilise puisse n'apporte qu'elle moment accsicder à l'infor qu'il demande.
les infor peuve être ecrite ou lues il ya ecrteur quand on enregistre les données on memoire, lecteur quand on soit (retir) les infos précédement enregistré la lectur peux être destructives ou non - il existe plusieur proprietes:

- Le temps d'acces: si le temps entre l'envoi de l'adress mémoire et l'obtenssion de la donnée
- Le temps de cycle: represente l'intervalle minimum qui doit s'paré deux operation mémoire succisive (lecture/ecriture) le temps de cycle ≥ le temps d'acces cadence de transfert ou débite de mémoire ci le nombre masim-al d'un infor lut ou ecrite par unite de temps.
- Une mémoire est fomé: d'un certain nombre de cillule, ou case contenant chaque une info chaqu cillule à un numéro qui permet de la référence ou localiser ce numéro est applé adress mémoire. avec une adress de un bit, il est possible de referencé directement ou plus 2^n cillule.

La capacité d'une mémoire est le nombre total de cillule qu'elle contient. elle s'exprime en nombre de octets d'octet (Byte) ou mot (word) compte tenu de l'adressage binaire, la capacité de la mémoire s'exprime en puicense de 2 ou multiple de $2^{10} = 1024$.
Il existe 2 classes de circuit realisant une RAM, il y a les RAMs statique en appele S-RAM et les RAMs dynamique (D-RAM)

Mémoire statique: sont points mémoire est constituer

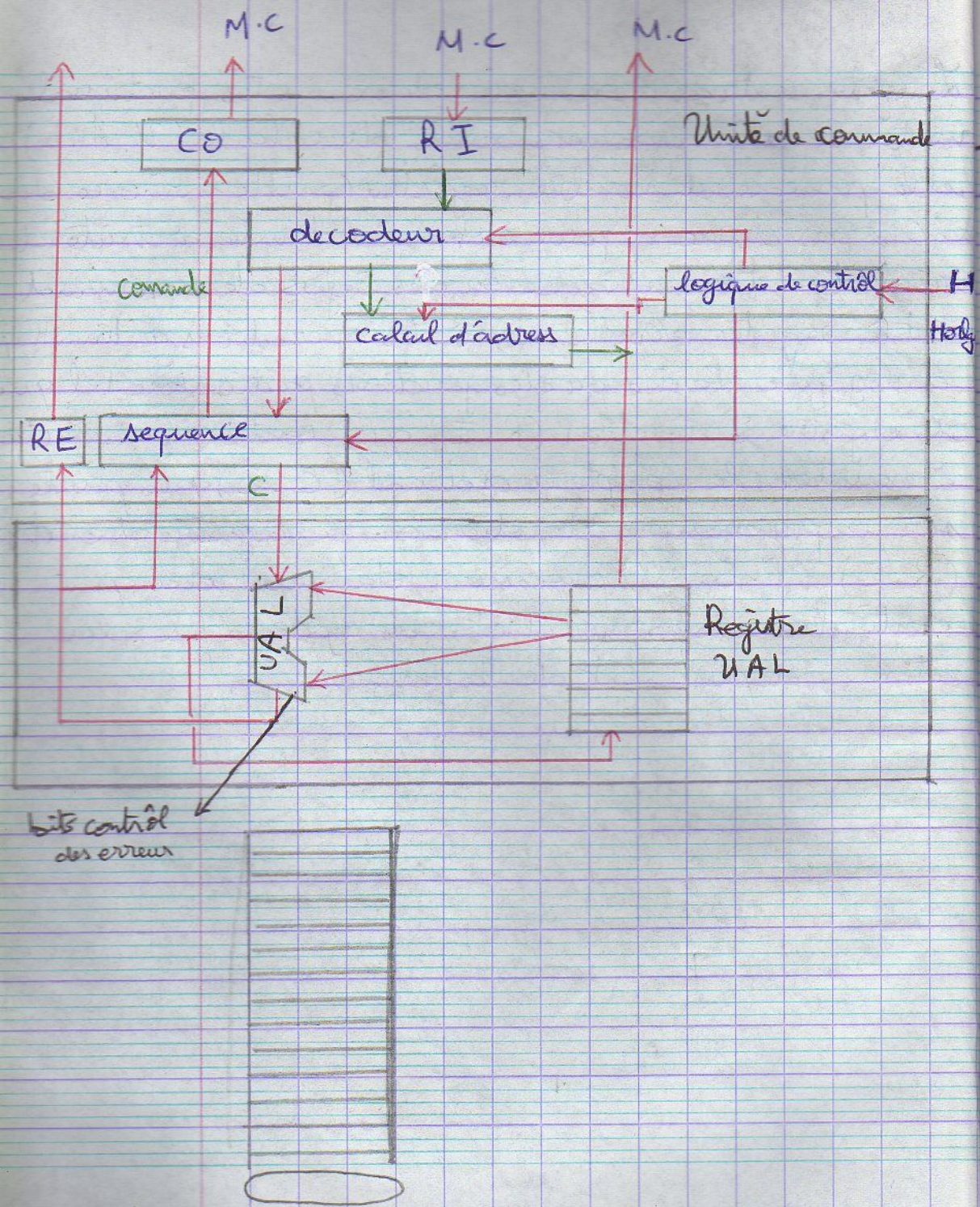
avec des portes logiques (circuit and, or ...)
La memorisation est permenante, la mémoire alimanté
(La lecture n'est pas discriptive)
- Mémoire dynamique : sont points mémoire est consti
avec une capacité (circuit) est un trasistor.
Sa capacité est 4 fois superieur à une S.RAM.

2/- Unité Central :
Toute ces unités composante fonctionne au même rythme à
une vitesse à une cadance imposé par une orloge,
généralement exterieur à l'unité central - à chaque cycle
d'horloge, chaque unité vas ouvrire ou fermer certain
porte, pour deplacer lire, ecrire ~~ses~~ comparer, additionner des bits, ceci ce fait au fonction d'ordre (commandes) donnée par l'unité de commande dans les machine dar une et une seulle instruction s'effectué entre deux Top d'horloge, La frequence d'horloge ce situer (Les années 90) entre 20 MHz et 66 MHz bien sur plus la frequence et elever plus le Micro-processeur et capable de traiter un grand nombre d'instruction

$$\text{performance} = \frac{1}{\text{tps d'exécution}}$$

3/- Unité de commande :

L'unité central a ou moins deux unités fonctional unité de commande et l'unité de calcul (U.A.L) elle possède de nombre registre (petite mémoire 64 bits n'est pas déplacer) pour stoker les donnée à traité les résultats intermediaire ou des infos de commande, certain registes serve (stocker) à des operations arithmitique et logique d'autre ou des fonctions particulière tel que le registre instruction (R.I) qui contien l'instruction à exécuter, le compteur ordinal (C.O) qui pointe sur la prochène instruction où le registre d'état (R.E) (P.S.W) processeur statique word qui contient les infos sur l'état du système ex: la retenu, dépassement de la mémoire,



contaur ordinal : registre qui contien inst ← 1er @ de pr

décodeur : donnée la nature de l'instru

Il existe :
- commande de sequence vers CO ⇒ ou le transfer @ inst 2programme vers le CO pour retirer l'instruction.
ou transfer d'@ de variable vers la mémoire pour transferer le contenu de la variable vers les Registres de UAL.
- Le décodeur décode le code d'opération de chaque instruction
- on disting 6 group d'instruction :
* transfer le donnée : il peut etre de memoire à Registre, de registre à registre ou de registre à memoire
* operation arithmitique : + , − , / , =
* operation logique : et, ou, or, not, and ...
* contrôl : if, then, else ....
Les branchement conditionnel ou non, apelle de procedu
* entrie / sortie :
* manipulation diverse : décalage, convertion de format, ... et ...

| code operation | → instruction a 0 @ operands (exp : Add les operand sont sur la pile)

| code operation | adresse | → instruction à 1 @ (exp : Add @B 1ei operande de l'UAL).

code operation	adress 1	adress 2

→ instruction à 2 @

(exp: Add @A @B @C)

Chapitre 5 : Architecture Risc Versus CISC

(complex Instructions Set computer).

- il existe 2 grand constructeurs intel (processeur intel 8088, 8086, 80286, pentium, ...)

Motorola (68000, ..., 68040) des remis en cause : CISC

- complexité des jeux d'instruction (toute l'inst assembleur)
- Unité des décodage de sequencement : elle occupé de 50% de la surface de silicium.

l'inité de compatibilité ascendante

vers
faire des approche très simple et rapible : d'exution

- prévilégier les instructions à forte occurrence (temp
- regeter les insts complesces au niveau de compilateur.
- prévilégier les instruction de registre à registre. et a la fin reduit la complexité de control (simple et rapide 5%).

L'approche RISC (Reduced Inst Set Computer) en nombre réduit et s'exicuter on un 1 cycle machine.

- Les instruction d'acsser on memoire sont l'inité de 2 insts (load, store).
- le décodage des insts est caller plutôt que microprogrammer.

- Le jeu d'inst doit reposé sur un format fixe (32 bits)
- l'ensemble de l'architecture doit aux maximum

tière partie du modèle pipeline
le processeur doit posséder un l'arge banc de registre interne
Le processeur doit être adapter à la gestion de mémoire cache et de coprocesseur. Fin
quelque processeur risque Risc (Reduced Instruction Set computer) processeur Alpha : Digaital (Dec)
- Mips R x 000 = Mips Technology (silicon graphics)
- PA Risc 7100 : HP
- power PC : Appele - IBM - MOTorola
- Sparc : sun

Les étapes d'exécution d'une instruction en Pipeline :

une instruction Risc s'exécute en passe à 5 pipe.
Il existe 5 étapes (fase) d'exécution de l'instruction (LI, DI, EX, MEM, ER) ou l'exécution de chaque étape nécessite un cycle machine (cycle horloge)

1er étape : lecture d'inst (LI) = envoi de sigo CO
lecture d'inst (CO ← @ 1 inst/ @proch "registre Inst, RI ← Instruction de la M.C)
(prendre d'inst de la M.C vers le processeur)

CO ← CO + 1 ; k : Longueur nombre d'octet

toute inst exécutée → 1 cycle machine

@ programme = @ 1 inst

processeur | M. cache | M.C

M. cache : très rapide

coprocesseur → calcul réel intégré

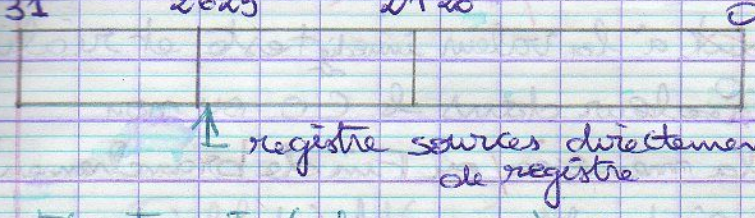
A. Risc = A. store/load

2ème étape : décodage de l'inst DI

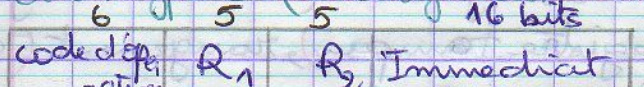
- décode de l'instruction : (décodé le code d'opération)
- accès au bon de registre pour lire les registre
- incrémentation de CO et passage à la prochaine instruction

CO ← CO + K

inst de à format fixe

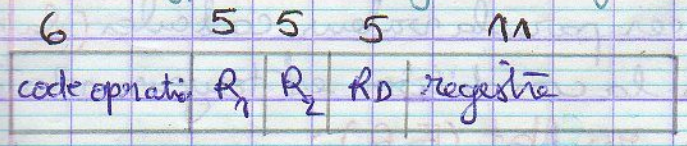


- Instruction type I : (chargement - rangement)



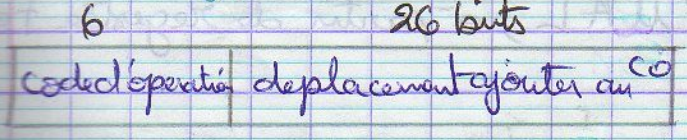
exp : lw R1 @ A
sw R1 @ A
Add R2 R1 15
Beq R1 R2 suite

- Instruction type R (Registre - Registre)



exp : Add R1 R2 R3

- Instruction de type (Saut) :



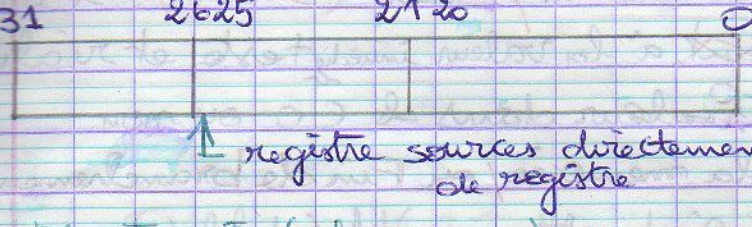
exp : J label

2ème étape : décodage de l'inst DI

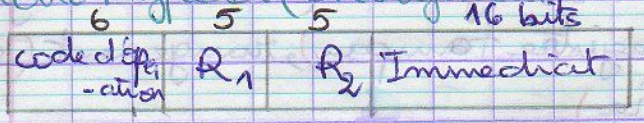
- décode de l'instruction : (décodé le code d'opération)
- accept au xxx bon de registre pour lire les registre
- incrémentation de CO et passage et la prochaine instruction

CO ← CO + K

inst de à format fixe

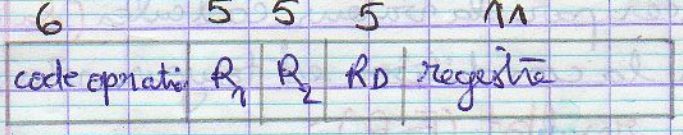


- Instruction type I : (chargement - rangement)



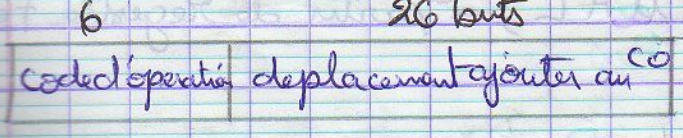
exp lw R_1 @ A
sw R_1 @ A
Add R_2 R_2 15
Beq R_1 R_2 suite

- Instruction type R (Registre - Registre)



exp : Add R_1 R_2 R_3

- Instruction de type (Saut) :



exp : J label

3ème étape :
exécution / calcule d'adresse
- 3èmes acssé mémoire : UAL Additionne les operandes pour former l'adresse éfective, ou registre de donnée chargé pour un stockage.
- instruction UAL : l'UAL efectue une operation arithmitique ou logique.
- Branch Branchement / Saut (Sauter) : l'UAL additionne le Saut à la valeur immédiat testé et réaliser pour placer la valeur dans le CO ou non
- accès à la mémoire / ou Fin de branchement (etp même)
- accès mémoire : chargement (donnée envoyer depuis la mémoire vers registre taunpan), rangement (donnée écrite dans la mémoire)
- branchement ou saut : si branchent à lieu ⇒ co est remplacer par la valeur calculer (la nouvelle @)
pour les sauts la condition est toujour vraie
5. Ecriteur du resultat (E R) :
c'est l'écriteur de resultat dans le bande registre : à partir de l'UAL, à partir de registre taunpen mémoire

R. taunpan : intermidiaire des calcul.
Co ← etiquette (instruction de branchement).
Co ← Co + instruction. 32/8/4

nbr de cycle	1	2	3	4	5	6	
inst 1	LI	DI	EXE	MEM	ER	-	-
inst 2		LI	DI	EXE	MEM	ER	-
inst 3			LI	DI	EXE	MEM	ER
inst 4				LI	DI	EXE	MEM
inst 5					LI	DI	EXE
inst 6						LI	DI
inst 7							LI
inst 8							

Tableau d'exécution de pipeline

Chapitre II Programmation Assembleur

Introduction : MIPS R3000.

MIPS R3000 dispose elle p c'est un processeur possède 32 Registres raumeroté de 0 -- 31 chaque nom de registre est précéder d'un dolar $ le registre $0 est le registre Zéro (0) est sepiial il contien toujour 0.

Le Fabriquant MIPS propose la taille de registre suivant :

nom	Numéro	Usage
Zero	0	Zero (tgs).
at	1	Reserve par l'assembleur
V_0 --- V_1	2 -- 3	Retour de Valeurs
a_0 --- a_3	4 -- 7	passage d'arguments
t_0 --- t_7	8 -- 15	Temperaire non sauvegardés
S_0 --- S_7	16 -- 23	// sauvegardés
t_8 --- t_9	24 -- 25	// non sauvgardés
k_0 --- k_1	26 -- 27	Reservé par le systeme.
gp	28	pointeur global
Sp	29	// de pile
fp	30	// de fraume.
Va	31	

Les registre $at, $k0, $k1 sont reserver par l'assembleur et le systeme d'explotation.

Chapitre II Programmation Assembleur

Introduction : MIPS R3000.

MIPS R3000 dispose elle p c'est un processeur possède 32 Registres numerotés de 0 -- 31 chaque nom de registre est précéder d'un dolar $ le registre $0 est le regist Zéro (0) est sepcial il contien toujour 0.

Le Fabriquant MIPS propose la talle de registre suivant :

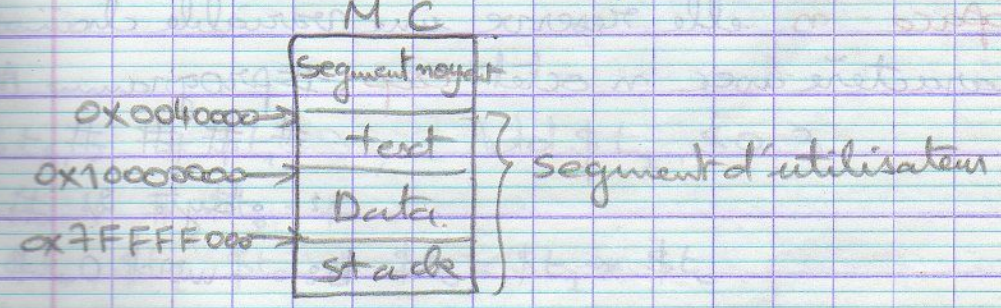
nom	Numéro	Usage
Zero	0	Zero (tgs).
at	1	Reserve par l'assembleur
V_0 --- V_1	2 -- 3	Retour de Valeurs
a_0 --- a_3	4 -- 7	passage d'arguments
t_0 --- t_7	8 -- 15	Temperaire non sauvegardés
S_0 --- S_7	16 -- 23	// sauvegardés
t_8 --- t_9	24 --- 25	// non sauvgardés
k_0 --- k_1	26 -- 27	Reservé par le systeme.
gp	28	pointeur global
Sp	29	// de pile
fp	30	// de fraume.
Va	31	

Les registre $ at, $ k_0, $ k_1 sont reserver par l'assembleur est le systeme d'esplotation.

- Les registre $\$a_0$ --- $\$a_3$ sont utilisés pour passer des arguments à des sous programme
- Les registre $\$V_0$ --- $\$V_1$ sont utilisés pour retourné des valeurs depuis des fonctions de syntaxe de l'assembleur dans l'architecture MIPS R3000, l'espace adressable (M.C) est diviser en deux segment.

le segment noyeau et le segment utilisateur. le syntaxe de l'assembleur un pgm utilisateur utilisant le segment utilisateur sera diviser en 3 sous segment appelé Section.

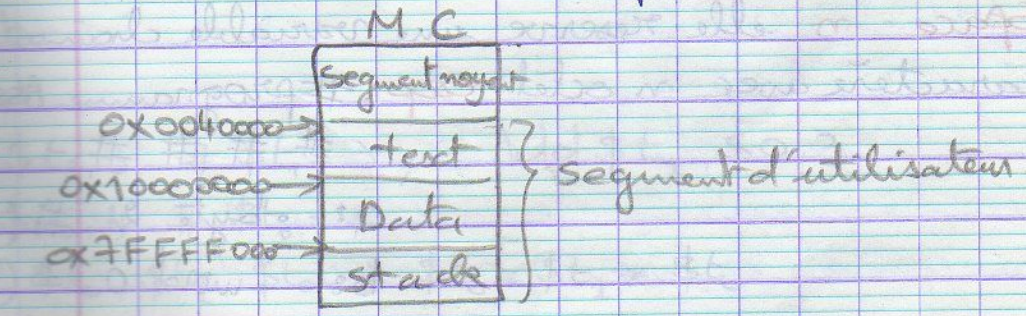
- La section text: qui contient le code exécutable en mode d'utilisateur elle est amplantée par l'adresse 0x00400000 sa taille est fixe.
- La section DATA: Data = donnée contient les données globales manipuler par pgm elle est amplantée à l'adresse suivant 0x10000000.
- La section Stacke: contient la pile d'exuction de pgm d'un utilisateur elle est amplanté 0x7FFFF000



- Les registre $\$a_0$ --- $\$a_3$ sont utilisés pour passer des arguments à des sous programme
- Les registre $\$V_0$ --- $\$V_1$ sont utilisés pour retourné des valeurs depuis des fonctions de syntaxe de l'assembleur dans l'architecture MIPS R3000, l'espace adressable (M.C) est diviser en deux segment.

le segment noyeau et le segment utilisateur. le segment de syntaxe d'un pgm utilisateur utilisant le segment utilisateur sera diviser en 3 sous segment appelé Section.

- La section text: qui contient le code exécutable en mode d'utilisateur elle est amplantée par l'adresse 0x00400000 sa taille est fixe.
- La section DATA: Data = donnée contient les données globales manipuler par pgm elle est amplantée à l'adresse suivant 0x10000000.
- La section Stack: contient la pile d'exuction de pgm d'un utilisateur elle est amplanté 0x7FFFF000



Nom de fichier :
un pgm assembleur est sauvgaradée par "nom.fichier"
- les commentaire : tout les commentaire commence par un "#"
- Identificateur : les identificateur ou variable sont former par des caratere & numérique caracter souligner mais ne débute jamais par un chiffre.
Les variable sont suivre l'autre de la declaration par plusieur directive indiquant leur Type
• Ascii chaine
• Asciz chaine reserve une chaine de caractere qui termine par un zéro Z
• byte n,[m] déclare une variable de type byte avec comme valeur petite n exp : • byte 5
reserve un octet • byte 5,3,2,1
• word n,[m] reserve 16 bits (2 octet) pour la variable.
• Space n elle reserve une variable chaine de caractere avec n octet exp : #programme Assembleur

```
# # # #
a:  .byte 2  #(a=2)
b:  .word 0  #(b=0)
ch:  .Ascii "boyoure"
                ch="boyour"
     .Texte
```

Instruction Arithmétiques et logique

Add R_d, R_s, R_t ; $Rd \leftarrow R_s + R_t$
Sub R_d, R_s, R_t ; $Rd \leftarrow R_s - R_t$
Addu R_d, R_s, R_t ; $Rd \leftarrow R_s + R_t$
Addi R_d, R_s, I ; $Rd \leftarrow R_s + I$
Addiu R_d, R_s, I ; $Rd \leftarrow R_s + I$
Mul R_d, R_s, R_t ; $Rd \leftarrow R_s + R_t$
Li R_d, I ; $Rd \leftarrow I$
or R_d, R_s, R_t ; $Rd \leftarrow R_s$ ou R_t
And R_d, R_s, R_t ; $Rd \leftarrow R_s$ and R_t
Xor R_d, R_s, R_t ; $Rd \leftarrow R_s$ xor R_t
Nor R_d, R_s, R_t ; $Rd \leftarrow R_s$ Nor R_t

Exercice : Ecrire le pgm assembleur qui calcul l'expretion suivante $(3*4)+2$ et met le resultat dans t_0.

Solution :

- data
- .text

```
li $t1,3  # $t1 = 3 ; = Add $t1, $0, 3
li $t0,4  # $t0 = 4
Mul $t0, $t1, $t0  # $t0 = $t1 * $t0
Add $t0, $t0, 2
```

u : Unsigned = non signé = mbr normal (entier).
i : immédiat
I = 5

Exercice : Ecrire pgm assembleur qui va lire 2 entier à partir du clavier faire la somme et afficher la chaine, le resultat et avec la somme

Solution :

```
.data
ch1: .Asciiz "Entrez le 1 nombre"
ch2 : .Asciiz "Entrez le 2 nombre"
ch3 : .Asciiz " le résultat est-"
.Text
li $V0, 4 # Afficher ch1
la $a0, ch1
Syscall
lui $V0, 5 # lire le 1er nombre
syscall # de 1er nombre et dans $V0
move $t0, $V0 # $t0 = $V0
li $V0 4
la $a0, ch2
Syscall
li $V0 5 # lire 2em nombre
syscall # 2em nombre ds $V0.
Add $t0, $t0, $V0.
li $a0, ch3
syscall
```

li V_0, 4
la a_0, ch3
syscall
li V_0, 1 # ecrire un entier sur ecran
nbre a_0, t_0
syscall # resulta affiché
li V_0, 10 # end
Syscall .

Les instruction du branchement :

Beq R_s, R_t label : branchement à label ssi $R_s = R_t$
Bne R_s, R_t label : branchement a label ssi $R_s \neq R_t$
Bgez R_s, label : branchement " " " $R_s \geq 0$.
Bgtz R_s, label " " " " $R_s > 0$
Blez R_s, label " " " " $R_s \leq 0$
Bltz R_s, label " " " " $R_s < 0$
J label : Jump ou branchement inconditionnel
label est sur 26 bits.
Jal label : branchement avec sauvegarde de l'adresse de retour
Jr, R_s : Branchement au registre R_s (l'adress sur 32 bits)
Exercice : ecrire le pgm qui va lire 10 nbrs à partir du clavier, la somme de 10 nbrs.
faire

Solution :

```
        .data
mess    .Asciiz "la somme est "

        .text
        li $t0, 0
        li $t1, 1
etiquette : beq $t1, $1 fin
            li $V0, 5
            syscall # $V0 contient le nombre
            Add $t0, $t0, $V0 # $t0 contient la somme
            Add $t1, $t1, 1
            j etiquette
fin : li $V0, 4 # affichage de mess
      la $a0, mess
      syscall
      li $V0, 1 # affichage de $t1
      move $a0, $t0
      Syscall
      li $V0, 10
      syscall.
```

Exercice n°2 : ecrire un pgm qui calcul (2n)! sachant que (2n)! n superieur ou egal à zero

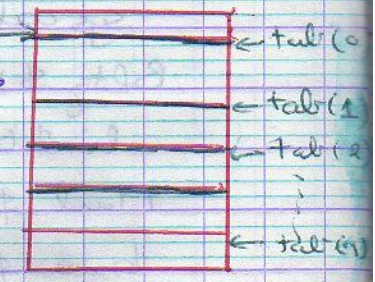
$n \geq 0$

```
        .data
mess:   .Asciiz "le factoriel est: "
        .text
etiquette = li $V0,5
            syscall
            Blez $V0, etiquette
            li $t0,2
            Mul $V0, $t0, $V0
            bne $V0, 0 for
            li $t1, 1
            J = fin
for : move $t1, $V0 # $t1 contient le produit fact
etq   beq $V0,1 fin
            sub $V0, $V0, 1
            Mul $t1, $t1, $V0
            J : etq
fin : li $V0, 4
            la $a0, mess
            syscall
            li $V0, 1 # afficher le factoriel
            move $a0, $t1
            Syscall
            li $V0, 10
            syscall
```

Les tableaux :

la multiplica

Exercice : ecrire le pgm assembleur qui fait ~~les~~ d'un tableau on suppose que on a un tableau de 10 elements deja rempli




```
    .data
tab: .word 1, -2, 3, 0, 4, 0, -10, 11, -1, 100
    .text
    li $t0, 0 # indice de la boucle
    li $t1, 10
    li $t2, 0 # indice du tableau
etiq: beq $t0, 10 : fin
    lw $t3, tab($t2) # charger l'elt dans M.C  word → R
    Mul $t3, $t3, $t1 # tab(i) ← tab(i) * 10.
    sw $t3, tab($t2)
    li $v0, 1
    move $a0, $t3 # affichage l'elt du tableau
    syscall
    Add $t0, $t0, 1
    Add $t2, $t2, 4
    J etiq
fin: li $v0, 10
    syscall
```

1 byte	2 byte	3 byte	4 byte

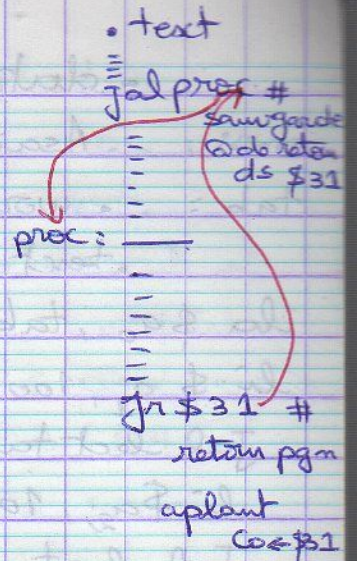
Les Sous programme :

- un sous programme est défini par : sont nom (qui est une etiquette) un ensemble d'instruction, qui se termine par l'instruction Jr Registre $31.

chaque sous pgm est appelé par :

Jal nom de ce programme .

ce pendant avant de passer à l'appel de sous pgm Jal doit sauvgarder l'adresse de retour (qui est l'inst après le Jal), dans le Registre $31



Remarque : si il existe des arguments quand vous transferez on pgm applant vers le pgm appelée dans ou a_0, a_1, a_2, a_3. si on a un resultat d'un sous pgm à transferer vers le pgm applant, ce resultat doit être (~~mett~~) met ou V_0, V_1

Exercice :

ecrire un pgm avec 2 les fonctions : (sous pgm)

1. qui lit un tableau de 100 elements "procedure lecture"
2. puis qui fait la somme de 10 elements "procedure somme"

le resultat est affiché dans le pgm applant

Solution :

```
        .data
mess:   .Asciiz "la somme est :"
tab:    .word 0
        .text
la $a0, tab
li $a1, 100
Jal lect tab
li $a2, 10
Jal lect_nbre
Move $t1, $V0
li $V0, 4
la $a0, mess
Syscall
li $V0, 1 # affichage de la somme
Move $a0, $t1
syscall
li $V0, 10
syscall
lect tab:
        li $t0, 0
        Move $t2, $a0.
lect : beq $t0, $a1, fin
```

li V_0, 5 # l'élet est dans V_0.
syscall
Sw V_0, (t_1)
Add t_0, t_0, 1
Add t_1, t_1, 4
J lect
fin : Jr $31.
lect_nbre :
li t_1, 0
li t_0, 0
lect1 : beq t_0, a_2 fin1
li V_0, 5
syscall
Add t_1, t_1, V_0 # S ← S + nbr - lect
Add t_0, t_0 1
J lect1
move V_0, t_1
fin1 : Jr $31

Exercice n°:
écrire le pgm assembleur qui calcule $\sum_{i=1}^{30} (2+i)!$

```
.data
mess : .asciiz "la somme est :"
.text
li $t0, 1 # $t0 remplacé le i (i=1)
li $t4, 0 # $t4 contient la somme
etiq : beq $t0, 31, fin
Add $t1, $t0, 2 # $t1 = (2+i)
move $t3, $t1, # $t3 contient le factoriel
Sub $t2, $t1, 1 # $t2 = (2+i) - 1
etiq1 : beq $t2, 0, fin1
Mul $t3, $t3, $t2
Sub $t2, $t2, 1
J etiq1
fin1 : Add $t4, $t4, $t3
Add $t0, $t0, 1
J etiq
fin : li $V0, 4 # afficher Message
la $a0, mess
syscall
li $V0, 1 # affichage de la somme
move $a0, $t4
syscall
```

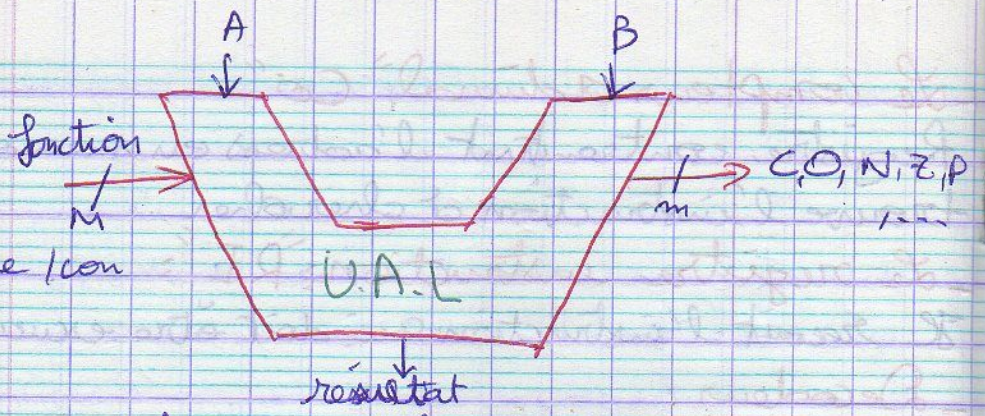
li $V0, 10
syscall.

Div $Rs, $Rt # division Rs / Rt
Lo ← quotient
Hi ← reste
Mfhi Rd # Rd ← Hi
Mflo Rd # Rd ← Lo.
Move flottante → Lo.

li V_0, 10
syscall.

Div R_s, R_t # division R_s / R_t
Lo ← quotient
Hi ← reste
Mfhi Rd # Rd ← Hi
Mflo Rd # Rd ← Lo.
Move flotant → Lo.

- Le compteur ordinal "CO":
Registre contenant l'adresse en mémoire ou se trouve l'instruction à chercher
- Le registre instruction: "RI":
Il reçoit l'instruction qui doit être exécuter.
- Décodeur:
qui détermine l'opération à effectuer et les opérandes.
- Séquenceur: Il génère le signaux de commande aux différents composant
- L'horloge: (interne ou externe):
- qui émet des impulsions permettant la synchronisation de toutes les éléments de l'unité centrale.
- Une horloge est un système logique qui émet des signaux périodiques constituant le cycle d'horloge ou cycle instruction.
- Un cycle d'instruction peut se composer en un cycle de recherche (instruction plus opérande) et un cycle d'exécution
On remarque qu'on peut indiquer par cycle CPU le temps nécessaire pour l'exécution d'une instruction (le temps le plus court)



trig: Representation symbolique d'1 U.A.L.

Sequenceur :

Le sequenceur est un automate distribuant, selon un chronogramme précis. Les signaux de commande aux diverses unités participant à l'éxécution d'une instruction. Il peut être cablé ou microprogramme. Un sequenceur cablé est un circuit séquentiel (combinatoire) complexe comprenant un sous-circuit pour chaqué une des instruction à commander. Ce sous-circuit est activé par un décodeur. De même pour reproduire une sequence d'operation élémentaire, il suffit d'un mot par "tranche de temps" cette serie de mots constitué un microprogramme. Le code operation de l'instruction à executer peut être utiliser pour définir le pointeur sur la 1ère microinstruction du microprogramme

En fonction du code opération le contenu d'un compteur est ~~opérationnelle~~ initialisé, puis celui-ci s'incrément ensuite à chaque cycle d'horloge.

- Les ordinateurs sont capable : d'effectuer un certain nombre d'opérations élémentaires une instruction machine comporte un à plusieur champs un champ obligatoire représente le code operation les autres champs comporte des données tels que variable (adress), registre (adress) ou mode d'adressage. Une instruction à une longeur de n' adresse, n = 0, 1, 2

inst 1 | code operation | n = 0

inst monodyadique | code opé | adresse | n = 1 (nbr operande = 1)

inst dyadique | code opé | @1 | @2 | n = 2

On distigue 6 groupes d'instructions :

1_ transfert de données : de memoire a registre, registre → registre, registre → memoire.

Mips R3000 :

Mips Technology est une filiale de la si

soccieté Silicon graphics depuis 1992 Les processeurs mips sont utilisés par plusieurs vendeurs de station de travail et de PC.
Annoncé en 1988 par la société Mips computer systems Le Mips R3000 (Microprocessor without Interlock pipeline stage) fut le 1er microprocesseur à vocation Industrielle. Il succède le R2000 qui est dérivé des travaux de l'université Stanford en 85, L'une des principles caractéristique de ce processeur est
- Sa simplicité
- La force des Microprocesseurs Mips réside ds la relation etroite entre l'architecture de processeur et le compilateur acossié
- Le Mips R4000 est La 3ème génération de processeur Mips (en 1992) il est le 1er à mettre on oeuvre une architecture à 64 bits est qui reste cependant compatible avec les generation Mips précidentes (32 bits)

Aller à TD n° 3 pipeline :

Technique de pipeline :

1/ Definition : Le principe de pipeline est celui de la chaine de montage
- Fractionner la tâche en plusieurs sous tâches

de ~~xxxxxx~~ durée egale appelés étages.
- Executer simultanément les différentes sous-tâches de plusieur tâche.

2- caractéristiques:

Un système pipeline est caractérisé par
T: Durée de l'étage
N: Nb d'étages.

3- Processeurs pipelinés: On considère indépendamment les différentes phases d'execution d'une instruction.
LI, DI, EX, MEM, ER.

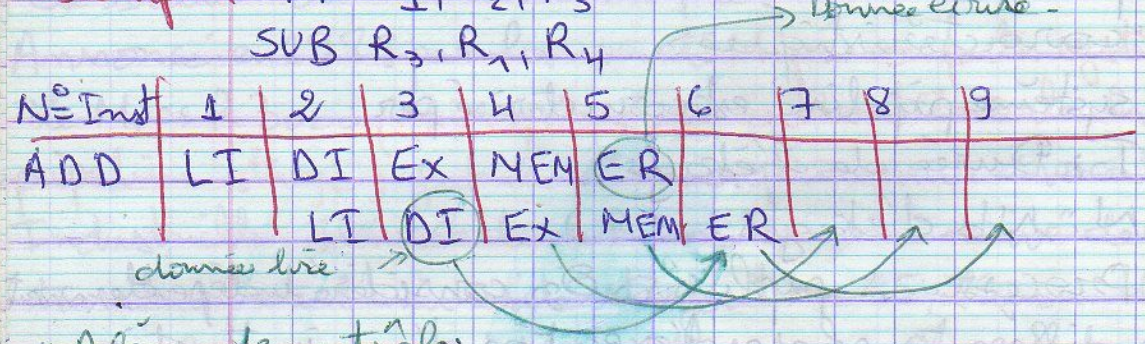
Avantages: - Augmente le nombre d'instructions executées par une unité de temps.
- Ne réduit pas le temps d'execution d'une instruction

Difficultés (Aléas): Situation empêchant l'inst suivante de s'executer au cycle d'horloge suivant
Retard d'execution.

Types d'aléas: * Aléas structurels: Material ne peut pas générer toutes les combinaisons de chevauchement possible exemple: Micro processeur ne comportant qu'un seul port mémoire (une seule operation mémoire au ^n temps)
* Aléas de données: l'ordre d'accès aux opérandes

est modifié par l'ordre d'exécution séquentielle des instructions.

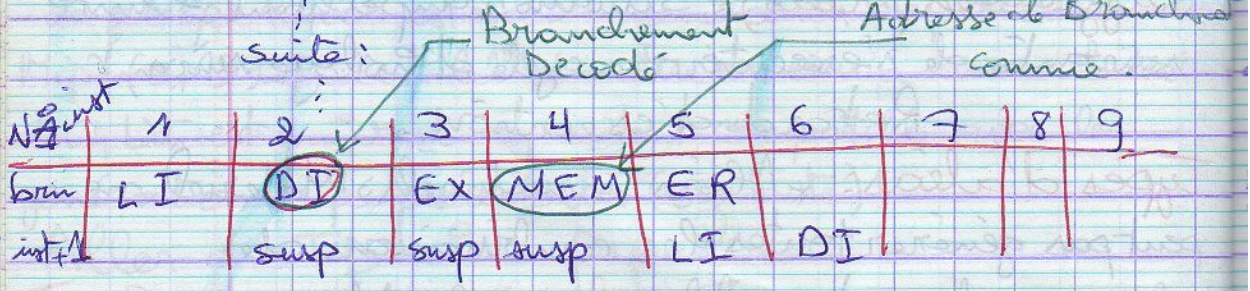
exemple: ADD R_1, R_2, R_3
SUB R_3, R_1, R_4



* Aléas de contrôle:

Fonctionnement pipeline des branchements et des instructions qui modifient le CO.

exemple: bne R_0, 12, suite
ADD R_0, R_1, R_2
:
suite:



Fact...