Le MIPS R2000/3000 et de l'assembleur associé

Le processeur MIPS

C'est un processeur 32 bits produit par MIPS Tecnology Inc.

Il est utilisé dans la fabrication de certains stations de de travail comme le Silicon Graphics.

Le premier dans la gamme est MIPS R2000

MIPS vient d'annoncer le MIPS R10000.
(MIPS R12000 ? ?)

C'est une architecture de type chargement/rangement :

- seules les instructions chargement et rangement accèdent à la mémoire.
- Les instructions de calcul opèrent uniquement sur des valeurs contenues dans des registres

Présentation du processeur MIPS R2000

- 32 registres généraux
- PC
- HI LO
- Registres protégés :

  - SR : registre d'état
  - CR : Cause registre

Le langage d'assembleur du MIPS R2000

Le processeur possède 57 instructions qui se repartissent en 4 classes :

- 33 instructions arithmétiques/logiques entre registres
- 12 instructions de branchement
- 7 instructions de lecture/écriture mémoire
- 5 instructions systèmes

Toutes les instructions ont une longueur de 32 bits et possède une de trois formats suivants :

Format R

code d'opération	Rs	Rt	Rd	Op-Arg1	Op-Arg2
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Exemple : l'instruction ADD $10, $9, $8 sera codée :

0	8	9	A	0	0x20

Format I

code d'opération	Rs	Rt	IMD16
6 bits	5 bits	5 bits	16 bits

Format J

code d'opération	IMD26
6 bits	26 bits

Mode d'adressage

- Registre : C'est le contenu d'un registre
- IMM Immédiat
- IMM(registre) : Immédiat + registre
- Symbole : (Directe)
- Symbole ± IMM C'est la valeur d'un adresse ± Immédiat
- Symbole ± imm(registre)

La syntaxe de l'assembleur

- les commentaires commencent par # , les valeurs Hexadécimale doivent être précédées par '' 0x ''
- un identifiant ne doit pas commencer par un chiffre, les codes d'opération mnémonique sont des mots réservés
- les étiquettes se mettent en les plaçant devant une instruction et les faisant suivre par '': ''
- il existe un certain ensemble des directives de gestion de mémoire .byte , .ascII , .asciiz, .text <adr>
- les chaîne de caractères sont encadrées par '' on admet des caractères spéciaux comme /n

Le Jeu d'instructions (une Classification)

- Les instructions arithmétiques et logiques

  exemple addu $9, $9, 1

- Les instructions de chargement immédiat

  exemple li $8, 1

- Les instructions de comparaison et de branchement

  Exemple : ble $9, 100, boucle

- Les instructions de chargement et de rangement

  exemples : lb Rt , adresse
  sb Rt , adresse

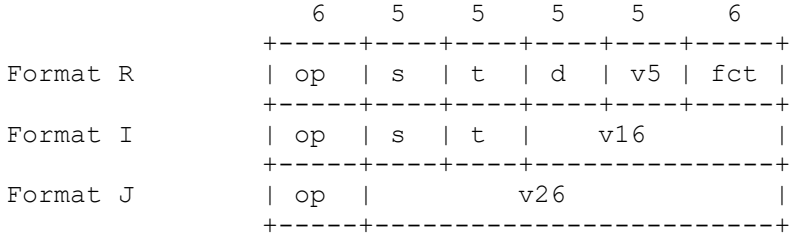
- Les instructions de transfert de données

  Exemple move $a0, $9

- Les Instructions flottantes

Instructions MIPS

FORMATS DES INSTRUCTIONS MACHINE



CONVENTIONS

Dans les codes mnémoniques
- `#` : numéro de coprocesseur = 0 (état), 1 (flottants), 2 ou 3

Dans les arguments
- `d,s,t,v5`: valeur sur 5 bits
- `v16` : valeur sur 16 bits
- `v26` : valeur sur 26 bits

Dans l'effet
- `d,s,t` : contenu sur 32 bits du registre général de numéro d, s ou t
- `ra` : contenu sur 32 bits du registre général de numéro 31
- `hi` : contenu sur 32 bits du registre spécialisé "high"
- `lo` : contenu sur 32 bits du registre spécialisé "low"
- `hi_lo` : contenu sur 64 bits des registres hi et lo concaténés
- `pc` : contenu sur 32 bits du registre spécialisé "program counter"
- `v5` : valeur de v5 (0 à 31)
- `v16` : valeur de v16 étendue à 32 bits par extension du signe
- `0_v16` : valeur de v16 étendue à 32 bits par des zéros à gauche
- `0_v26` : valeur de v26 étendue à 32 bits par des zéros à gauche
- `byte(x)` : contenu sur 8 bits de l'octet d'adresse x
- `half(x)` : contenu sur 16 bits du demi-mot d'adresse x (paire)
- `word(x)` : contenu sur 32 bits du mot d'adresse x (multiple de 4)
- `cc(#)` : bit cc du coprocesseur #
- `d(#)` : contenu du registre de numéro d du coprocesseur #
- `s(#)` : contenu du registre de numéro s du coprocesseur #

MNEMO	ARGUMENTS	EFFET
lui	d,v16	d = v16 <<< v5 ** BUG utiliser srlv **
srlv	d,t,s	d = (unsigned)t s
sra	d,t,v5	d = t v5
srav	d,t,s	d = t s
slt	d,s,t	d = s 0 alors pc = pc + 4 * v16
blez	s,v16	si s <= 0 alors pc = pc + 4 * v16
bgez	s,v16	si s = 0 alors pc = pc + 4 * v16
bgezal	s,v16	si s = 0 alors ra = pc , pc = pc + 4 * v16
beq	s,t,v16	si s == t alors pc = pc + 4 * v16
bne	s,t,v16	si s != t alors pc = pc + 4 * v16
bc# f	v16	si cc(#) == 0 alors pc = pc + 4 * v16
bc# t	v16	si cc(#) == 1 alors pc = pc + 4 * v16
jr	s	pc = s
j	v26	pc = 4 * 0_v26
jalr	t,s	t = pc , pc = s
jal	v26	ra = pc , pc = 4 * 0_v26

```
lb      t,v16(s)        t = byte( s + v16 )
lh      t,v16(s)        t = half( s + v16 ) , s+v16 multiple de 2
lw      t,v16(s)        t = word( s + v16 ) , s+v16 multiple de 4
lwl     t,v16(s)        t = word( s + v16 ) , octets de gauche du mot
lwr     t,v16(s)        t = word( s + v16 ) , octets de droite du mot
lbu     t,v16(s)        t = (unsigned)byte( s + v16 )
lhu     t,v16(s)        t = (unsigned)half( s + v16 )
sb      t,v16(s)        byte( s + v16 ) = t
sh      t,v16(s)        half( s + v16 ) = t , s+v16 multiple de 2
sw      t,v16(s)        word( s + v16 ) = t , s+v16 multiple de 4
swl     t,v16(s)        word( s + v16 ) = t , octets de gauche du mot
swr     t,v16(s)        word( s + v16 ) = t , octets de droite du mot
lwc#    t,v16(s)        t(#) = word( s + v16 )
swc#    t,v16(s)        word( s + v16 ) = t(#)

cop#    ?               opération ? du coprocesseur #

syscall                 déroutement au système
break   v               provoque une exception (pc = 0x80000080)
rfe                     retour d'exception (restaure pc, status, ... )
```

Les simulateurs de MIPS (SPIM):

1/ Le simulateur de l'université de Paris 6 (version windows uniquement)

2/ Le simulateur de l'université du Wisconsin USA, fait par Jame R. Larus (cf. page A-38 de Organisation et conception des ordinateurs D. patterson, J. Hennessy, Dunod 1994).
Disponible à l'adresse : ftp.cs.wisc.edu

C'est un simulateur logiciel qui exécute des programmes écrits pour le processeur MIPS R2000/R3000.
Il y a une version tournant sous Windows 32 bits : (Windows 98, Windows nt)
Sous UNIX il existe deux versions :

- interactive : SPIM
- X-windows : XSPIM

Les appels système et les macros d'entrées/sorties

(uniquement disponibles dans la version windows)

putc imprime un caractère, exemples : putc $8, putc ''/n ''

getc lit un caractère, exemple : getc $8

puts
gets

puti
geti
...

Une macro d'entrées/sortie se traduit par un appel système qui prend comme argument (code d'appel système) le contenu du registre $v0 et comme paramètre le contenu du registre $a0.

Exemple : puti sera traduit par la suite d'instructions :

```
li $v0, 1
la  $a0, 120
syscall
```

Quelques code d'appel système :

Code	fonction
1	**print_in**
2	**print_float**
5	**read_int**
8	**read_string**

INSTRUCTIONS DE BASE(SPIM)

```
Dans les arguments
        $x      : x est un numéro ou un nom de registre
        cste    : constante numérique (signée)
        symb    : étiquette symbolique
        val     = "cste" ou "$x"
        s/r     = "symb" ou "$x"
        adr     = "cste($x)" ou "symb" ou "symb($x)"
Dans l'effet
        valeurs : pour "$x" : contenu du registre x
                  pour "cste" : valeur de cette constante
                  pour "symb" : adresse symbolisée par l'étiquette
                  pour "cste($x)" : valeur de la cste + contenu du registre x
                  pour "symb($x)" : valeur du symbole + contenu du registre x
        x       : contenu sur 32 bits du registre général x
        [x+1]   : contenu sur 32 bits du registre général de numéro x +1
        ra      : contenu sur 32 bits du registre général de numéro 31
        pc      : contenu sur 32 bits du registre spécialisé "program counter"
        byte(x) : contenu sur 8 bits de l'octet d'adresse x
        half(x) : contenu sur 16 bits du demi-mot d'adresse x (paire)
        word(x) : contenu sur 32 bits du mot d'adresse x (multiple de 4)


MNEMO   ARGUMENTS       EFFET

nop

move    $r,$p           r = p
lui     $r,cste         r = cste << (unsigned)val
sra     $r,$p,val       r =           p  (unsigned)val
rol     $r,$p,val       r = décalage circulaire à gauche de p
ror     $r,$p,val       r = décalage circulaire à droite de p

slt     $r,$p,val       r = p  val
sle     $r,$p,val       r = p <= val
sge     $r,$p,val       r = p = val
seq     $r,$p,val       r = p == val
sne     $r,$p,val       r = p != val
sltu    $r,$p,val       r = (unsigned)p   (unsigned)val
sleu    $r,$p,val       r = (unsigned)p <= (unsigned)val
sgeu    $r,$p,val       r = (unsigned)p = (unsigned)val

b       symb            pc = symb
bltz    $p,symb         si  p   0    alors  pc = symb
blez    $p,symb         si  p <= 0    alors  pc = symb
bgez    $p,symb         si  p = 0    alors  pc = symb
beqz    $p,symb         si  p == 0    alors  pc = symb
bnez    $p,symb         si  p != 0    alors  pc = symb
blt     $p,val,symb     si  p   val  alors  pc = symb

ble     $p,val,symb     si  p <= val  alors  pc = symb
bge     $p,val,symb     si  p = val  alors  pc = symb
beq     $p,val,symb     si  p == val  alors  pc = symb
bne     $p,val,symb     si  p != val  alors  pc = symb
bltu    $p,val,symb     si  (unsigned)p   (unsigned)val  alors  pc = symb
bleu    $p,val,symb     si  (unsigned)p <= (unsigned)val  alors  pc = symb
bgeu    $p,val,symb     si  (unsigned)p = (unsigned)val  alors  pc = symb
```

```
j       s/r             pc = s/r
jal     s/r             ra = pc , pc = s/r

lb      $r,adr          r = byte(adr)
lh      $r,adr          r = half(adr) , adr doit être paire
lw      $r,adr          r = word(adr) , adr doit être multiple de 4
ld      $r,adr          r = word(adr) , [r+1] = word(adr+4) , adr mult. de 4
sb      $p,adr          byte(adr) = p
sh      $p,adr          half(adr) = p , adr doit être paire
sw      $p,adr          word(adr) = p , adr doit être multiple de 4
sd      $p,adr          word(adr) = p , word(adr+4) = [p+1] , adr mult. de 4
lbu     $r,adr          r = (unsigned)byte(adr)
lhu     $r,adr          r = (unsigned)half(adr) , adr doit être paire

syscall                 déroutement au système

(*) sauf si "cste" nécessite plus de 16 bits,
    dans ce cas spim traite la lui comme une li !
```

AUTRES INSTRUCTIONS

```
Conventions supplémentaires
        #       : numéro de coprocesseur = 0 (état), 1 (flottants), 2 ou 3
        hi      : contenu sur 32 bits du registre spécialisé "high"
        lo      : contenu sur 32 bits du registre spécialisé "low"
        hi_lo   : contenu sur 64 bits des registres hi et lo concaténés
        cc(#)   : bit cc du coprocesseur #
        x(#)    : contenu du registre x du coprocesseur #

MNEMO   ARGUMENTS       EFFET

mfhi    $r              r = hi
mflo    $r              r = lo
mthi    $r              hi = r
mtlo    $r              lo = r
mfc#    $r,$p           r = p(#)
mtc#    $r,$p           p(#) = r

addiu   $r,$p,cste      r = p + cste
addi    $r,$p,cste      r = p + cste , exception en cas de débordement
mult    $p,$q           hi_lo = p * q
div     $p,$q           hi = p % q , lo = p / q
multu   $p,$q           hi_lo = (unsigned)p * (unsigned)q
divu    $p,$q           hi = (uns)p % (uns)q , lo = (uns)p / (uns)q

ori     $r,$p,cste      r = p | cste
andi    $r,$p,cste      r = p & cste
xori    $r,$p,cste      r = p ^ cste

sllv    $r,$p,val       r = (unsigned)p < (unsigned)val
srav    $r,$p,val       r = p  (unsigned)val

slti    $r,$p,cste      r = p = 0  alors  ra = pc , pc = symb
bc#f    symb            si  cc(#) == 0  alors  pc = symb
bc#t    symb            si  cc(#) == 1  alors  pc = symb

jr      s/r             pc = s/r
jalr    s/r             ra = pc , pc = s/r
jalr    $r,$p           r  = pc , pc = p

lwl     $r,adr          charge dans r les octets de gauche de word((adr+4)& ~3)
lwr     $r,adr          charge dans r les octets de droite de word(adr & ~3)
```

```
swl     $p,adr          stock les octets de gauche de p dans word((adr+4)& ~3)
swr     $p,adr          stock les octets de droite de p dans word(adr & ~3)
ulh     $r,adr          r = byte(adr)_byte(adr+1)
ulhu    $r,adr          r = (unsigned)byte(adr)_byte(adr+1)
ush     $p,adr          byte(adr)_byte(adr+1) = r
ulw     $r,adr          r = byte(adr)_byte(adr+1)_byte(adr+2)_byte(adr+3)
usw     $p,adr          byte(adr)_byte(adr+1)_byte(adr+2)_byte(adr+3) = p

lwc#    $r,adr          r(#) = word(adr) , adr doit être multiple de 4
swc#    $p,adr          word(adr) = p(#) , adr doit être multiple de 4

break   cste            provoque une exception (pc = 0x80000080)
rfe                     retour d'exception (restaure pc, status, ... )
```

EXPANSION DES INSTRUCTIONS

La valeur d'une constante ou d'un symbole (valeur d'adresse dans ce cas) conditionne les expansions faites par le compilateur suivant que cette valeur se représente sur 16 bits (notée alors v16) ou pas (valeur notée alors v32).

```
MNEMO   ARGUMENTS       val ou adr: EXPANSION

nop                     : or

li      $r,val          v16: ori        v32: lui,ori
lui     $r,val          v16: lui        v32: lui,ori (traité comme une li !)
la      $r,adr          symb: lui,ori   v16($p): addi   v32($p): lui,ori,add
mfhi    $r              : mfhi
mflo    $r              : mflo
mthi    $p              : mthi
mtlo    $p              : mtlo
mfc#    $r,$p           : mfc#
mtc#    $r,$p           : mtc#
cfc#    $r,$p           : cfc#
ctc#    $r,$p           : ctc#
move    $r,$p           : addu

add[u]  $r,$p,val       $p: add[u]      v16: addi[u]    v32: lui,ori,add[u]
sub[u]  $r,$p,val       $p: sub[u]      v16: addi[u]    v32: lui,ori,add[u]
neg[u]  $r,$p           : sub[u]
abs[u]  $r,$p           : addu,bgez,sub
mult[u] $p,$p           : mult[u]
mul[o]  $r,$p,val       $p: mult[,...],mflo        v16/32: (lui,)ori,...
mul[o]u $r,$p,val       $p: mult[,...],mflo        v16/32: (lui,)ori,...
div[u]  $p,$p           : div[u]
div[u]  $r,$p,val       $p: bne,break,div[u],mflo  v16/32: (lui,)ori,div[u],mflo
rem[u]  $r,$p,val       $p: bne,break,div[u],mfhi  v16/32: (lui,)ori,div[u],mfhi

or      $r,$p,val       $p: or          v16: ori        v32: lui,ori,or
ori     $r,$p,val                       v16: ori        v32: lui,ori,or
and     $r,$p,val       $p: and         v16: andi       v32: lui,ori,and
andi    $r,$p,val                       v16: andi       v32: lui,ori,and
xor     $r,$p,val       $p: xor         v16: xori       v32: lui,ori,xor
xori    $r,$p,val                       v16: xori       v32: lui,ori,xor
nor     $r,$p,val       $p: nor         v16: ori,nor    v32: lui,ori,nor
not     $r,$p           : nor

sll[v]  $r,$p,val       $p: sllv                v16/32: sll
srl[v]  $r,$p,val       $p: srlv                v16/32: srl
sra[v]  $r,$p,val       $p: srav                v16/32: sra
rol     $r,$p,val       $p: subu,srlv,sllv,or   v16/32: srl,sll,or
```

```
ror     $r,$p,val       $p: subu,sllv,srlv,or   v16/32: sll,srl,or

slt[u]  $r,$p,val       $p: slt[u]      v16: slti[u]    v32: lui,ori,slt[u]
slti[u] $r,$p,val                       v16: slti[u]    v32: lui,ori,slt[u]
sgt[u]  $r,$p,val       $p: slt[u]      v16: ori,...    v32: lui,ori,...
sle[u]  $r,$p,val       $p: bne,ori,beq,slt[u]  v16/32: (lui,)ori,...
sge[u]  $r,$p,val       $p: bne,ori,beq,slt[u]  v16/32: (lui,)ori,...
seq     $r,$p,val       $p: beq,ori,beq,ori     v16/32: (lui,)ori,...
sne     $r,$p,val       $p: beq,ori,beq,ori     v16/32: (lui,)ori,...

blt[u]  $p,val,symb     $p: slt[u],bne  v16/32: (lui,ori,)slt[u]i,bne
ble[u]  $p,val,symb     $p: slt[u],beq  v16/32: (lui,)[ori,beq,]slt[u]i,bne
bgt[u]  $p,val,symb     $p: slt[u],bne  v16/32: (lui,ori,)slt[u]i,beq
bge[u]  $p,val,symb     $p: slt[u],beq  v16/32: (lui,ori,)slt[u]i,beq
beq     $p,val,symb     $p: beq         v16/32: (lui,)ori,beq
bne     $p,val,symb     $p: bne         v16/32: (lui,)ori,beq
bltz    $p,symb         : bltz
bgtz    $p,symb         : bgtz
blez    $p,symb         : blez
bgez    $p,symb         : bgez
beqz    $p,symb         : beq
bnez    $p,symb         : bne
bgezal  $p,symb         : bgezal
bltzal  $p,symb         : bltzal
b       symb            : bgez
bc#f    symb            : bc#f
bc#t    symb            : bc#t

j[r]    s/r             $p: jr          symb: j
jal[r]  s/r             $p: jalr        symb: jal
jalr    $r,$p           : jalr

lb[u]   $r,adr          symb: lui,lb[u] v16/32($p): (lui,addu,)lb[u]
lh[u]   $r,adr          symb: lui,lh[u] v16/32($p): (lui,addu,)lh[u]
lw      $r,adr          symb: lui,lw            v16/32($p): (lui,addu,)lw
lwl     $r,adr          symb: lui,lwl           v16/32($p): (lui,addu,)lwl
lwr     $r,adr          symb: lui,lwr           v16/32($p): (lui,addu,)lwr
sb      $r,adr          symb: lui,sb            v16/32($p): (lui,addu,)sb
sh      $r,adr          symb: lui,sh            v16/32($p): (lui,addu,)sh
sw      $r,adr          symb: lui,sw            v16/32($p): (lui,addu,)sw
swl     $r,adr          symb: lui,swl           v16/32($p): (lui,addu,)swl
swr     $r,adr          symb: lui,swr           v16/32($p): (lui,addu,)swr
ulh[u]  $r,adr          : (lui,(addu,))lb[u],(lui,(addu,))lbu,sll,or
ush     $r,adr          : (lui,(addu,))sb,srl,(lui,(addu,))sb
ulw     $r,adr          : (lui,(addu,))lwl,(lui,(addu,))lwr
usw     $r,adr          : (lui,(addu,))swl,(lui,(addu,))swr
ld      $r,adr          : (lui,(addu,))lw,(lui,(addu,))lw
sd      $r,adr          : (lui,(addu,))sw,(lui,(addu,))sw
lwc#    $r,adr          symb: lui,lwc#          v16/32($p): (lui,addu,)lwc#
swc#    $r,adr          symb: lui,swc#          v16/32($p): (lui,addu,)swc#

syscall                 : syscall
break   v               : break
rfe                     : rfe
```

```
#  auteur nakechbandi

.data
m_donnee:   .asciiz "\n donner n : "
m_resultat: .asciiz "\n le resultat est : "
N     :     .word 0
FACN  :     .word 0
      .text
main :
        li $7, 1
#       puts  m_donnee
        li $v0, 4
        la  $a0,m_donnee
        syscall

#       geti $8
        li $v0, 5
        syscall

        move $8, $v0
        move $9, $8
        sw $8, N
        li $10, 1

boucle: add $0, $0, $0
        mul  $10, $10, $9
        sub $9, $9, $7
        bgtz  $9, boucle

#       puts m_resultat
        li $v0, 4
        la  $a0, m_resultat
        syscall

#       puti $10
        li $v0, 1
        move  $a0, $10
        syscall
        sw $10, FAC
        done
```

Université du Havre, Faculté des Sciences, Licence d'informatique, UV Architecture des Ordinateurs, Examen TP, janvier 2002, Enseignant : Nakechbandi M***Durée Heures, Notes de cours autorisées.***

Remarques :

1. Joignez à votre copie un exemplaire de chaque programme réalisé (listing imprimé et bien commenté).
2. Envoyez-moi également les listings en fichiers attaché par Email à l'adresse : nakech@hera.iut.univ-lehavre.fr
   subject : linf-vos-initials

Présentation : Le programme MIPS suivant calcule n ! = 1 * 2 * 3 * .. * n ; n > 1

```
#  auteur nakechbandi
.data
debut:    .asciiz "abcd------------------ "
m_debut:    .asciiz "debut  \n"
m_donnee:   .asciiz "\n donner n : "
m_resultat: .asciiz "\n le resultat est : "
N          :     .word 0, 0
FACN  :     .word 0, 0
      .text
main :
        li $7, 1
#       puts  m_donnee
        li $v0, 4
        la  $a0,m_donnee
        syscall
#       geti $8
        li $v0, 5
        syscall

        move $8, $v0
        move $9, $8
        sw $8, N
        li $10, 1
boucle: add $0, $0, $0
        mul  $10, $10, $9
        sub $9, $9, $7
        bgtz  $9, boucle
#       puts m_resultat
        li $v0, 4
        la  $a0, m_resultat
        syscall
#       puti $10
        li $v0, 1
        move  $a0, $10
        syscall
        sw $10, FAC
        done
```

Travail à faire :

1. Compiler le programme précédant puis donner l'adresse mémoire correspondant à l'adresse mnémonique debut et boucle.

2. Donner le contenu du mot mémoire pointé par l'adresse debut

3. Pour numéroter les octets d'un mot, quel est le type d'ordonnancement utilisé par mips? (petit indien ou grand indien).
4. Donner le code objet de l'instruction : `bgtz $9, boucle` puis expliquer le rôle des différents champs de ce code objet.
5. Soient a1,a2, a3, a4, a5, …an des entiers stockés à partir de l'adresse mnémonique A. n est un entier à saisir au clavier. Faire un programme en langage mips permettant de calculer X

$$X = (n * a_n) + ((n - 1) * a_{n-1}) + ((n-2) * a_{n-2}) + \ldots + a_1$$