CEP1

× Partie Conception de processeur

modèle d'un état basique :

```
when nom_état =>
    -- Opérations effectuées ds l'état
    cmd. ALU_X_Sel ⇐ Entrée X de l'ALU;
    cmd. ALU_Y_Sel ⇐ Entrée Y de l'ALU;
    cmd. ALU_OP ⇐ Opération de l'ALU;
    cmd. RF_Sel ⇐ Registre de destinat°;
    cmd. RF_we ⇐ true; (Autorisat° d'écriture de la destinat°)
    Si on touche au registre PC ou à la mémoire
    cmd. mem_we ⇐ true;
    state_d ⇐ S_fetch_wait;
    Sinon
    cmd. mem_ce ⇐ true;
    state_d ⇐ S_fetch;
```

Pour SW : 2 états : AD ⇐ RS + IMM16
mem[AD] ⇐ RT (mem_we ET mem_ce à TRUE)

Pour LW : 4 états : AD ⇐ RS + IMM16
dataIn ⇐ mem[AD]
DT ⇐ dataIn
RT ⇐ dataIn

Rq : Ne pas oublier :
→ ajouter l'état à State_type au début
→ ajouter la transit° vers l'état ds S_Decode

Gestion des conditions
→ utiliser les signaux de statuts ds l'état où on fait l'opérat° avec l'ALU

→ Syntaxe :
```
if status_x then
    state_d ⇐ ;
else
    state_d ⇐ ;
end if;
```

→ Usage
status.z ↙ 1 si le résultat de l'ALU = 0
↘ 0 sinon
status.s ↙ 1 si le résultat de l'ALU < 0
↘ 0 sinon
status.c ↙ 1 si le res à une retenue
↘ 0 sinon

↳ res < 0 pour une soustract° entre registre non signés (cf SLTI et SLTIU)

× Partie Exploitation des processeurs

qemu-system-mips -M mipscep -nographic -S -s -kernel executable
mips-elf-gdb executable

× Sections en code assembleur → directives

- .text : à mettre au début du code assemblé.
- .data : contient les variables globales modifiables initialisées (à mettre en fin de code)

ex : .data
nom_variable taille_à_réserver valeur_initiale

- .comm : contient les variables globales modifiables non initialisées

ex : .comm
nom_variable taille_à_réserver

- .globl : à mettre avant le nom de fonct° ou variable

ex .globl est_vivant
est_vivant :

Type Cpp	Taille en octets
int64_t uint64_t	8
int32_t uint32_t	4
int16_t uint16_t	2
int8_t uint8_t	1
char	1
pointeur	4

× Convention registres

Noms	Usage
$zero	toujours à 0
$v0 et $v1	valeurs de retours
$a0 - $a3	arguments
$t0 - $t9	val. temporaires
$sp	pointeur de pile
$ra	adresse de retour

× Enchaînements courants

- if (x == 0) x = 12 else x = 24 =>

```
if: bnez $t1, else
    li $t1, 12
    j endif
else: li $t1, 24
endif:
```

- while (i > 0) { ... i--; } =>

```
loop: blez $t0, endloop
      ...
      addiu $t0, $t0, -1
      j loop
endloop:
```

- for (i=0; i<n; i++) =>

```
      li $t0, 0
loop: bge $t1, $t0, endloop
      ...
      addiu $t0, $t0, 1
      j loop
endloop:
```

- Accès indicé (tableau / chaîne de caractère) : tab[j]
  ↳ utiliser lbu et sb à la place de lw et sw

si j stocké ds $t0 et tab ds $a0

(sll $t1, $t0, $\log_2$(taille_objet)) pas utile pour un char → $\log_2(1) = 0$

```
addu $t1, $t1, $a0   ← on accède à l'adresse de tab[j]
lw $t1, ($t1)        ← on met la valeur de tab[j] ds $t1
```

- Instruct° de condition :

blt	$t0, $t1, label	saute à label si $t0 < $t1
bgt	$t0, $t1, label	saute à label si $t0 > $t1
ble	$t0, $t1, label	saute à label si $t0 ≤ $t1
bge	$t0, $t1, label	saute à label si $t0 ≥ $t1
beq	$t0, $t1, label	saute à label si $t0 = $t1
bne	$t0, $t1, label	saute à label si $t0 ≠ $t1

Rq : Si comparaison avec 0, ajout d'un z à la fin
ex : bnez, bltz ...

- Initialisat° variable / Accès mémoires

lw $t0, X($t1)	charge ds $t0 ce qui est à l'adresse contenue ds $t1 + X
sw $t0, X($t1)	stocke $t0 à l'adresse contenue ds $t1 + X
li $t0, val	stocke val ds $t0 (val en décimal)
la $t0, adress	stocke l'adresse ds $t0
move $t0, $t1	copie le registre $t1 ds $t0
lbu : $t0, X($t1)	équivalent de lw mais sur 1 octet → pour les char
sb $t0, X($t1)	équivalent de sw pour 1 octet → pour les char

- Appel de fonction : jal fonction (ne pas oublier de générer ses paramètres avant)

× Contexte

Expliciter l'utilisat° de la pile, les registres utilisés pour stocker chaq variable (et son type)

Pour la pile : $N = (n_{variable} + n_{registre} + n_{parametre\_max}) \times 4$

ex :

```
int affichage(struct structure_t s){
    x = affiche(s.entier, s.pti)
    return x }
/* Contexte : fonct° non feuille : $ra et 2 param à sauver
              → Pile de +12
   x : registre $t0, type int
   s : registre $a0, type structure_t
       s.entier en 0($a0)
       s.pti en 4($a0) */
.globl affichage
affichage :
   /* Prologue */
   addiu $sp, $sp, -12   ← on fait la place ds la pile
   sw $ra, 8($sp)        ← on sauve l'adresse de retour
   /* affichage */
   jal affichage
   move $t0, $v0         ← on récupère x
   move $v0, $t0         ← on renvoie x
   /* Epilogue */
   lw $ra, 8($sp)        ← on récupère l'adresse de retour
   addiu $sp, $sp, 12    ← on libère la pile
   jr $ra                ← on sortie.
```

valeur de $ra	$SP + (N-4)
registres à sauver	
variables à sauver	$SP + n_p*4, ... $SP + n_p+n_v
paramètres appelés	$SP + 0, $SP + 4*n_p

× Interrupt°

On sauvegarde en pile tous les registres utilisés ds le traitant

agir

restaurer les registres sauvegardés

retour avec eret