

Compte Rendu

Objectifs :



Nous allons travailler sur une architecture MIPS 32 bits (Microprocessor without Interlocked Pipeline Stages). L'architecture MIPS est une architecture de processeur de type RISC (Reduced Instruction Set Computer). Son architecture est une référence. Les processeurs fabriqués selon cette architecture sont surtout utilisés dans les systèmes embarqués.



L'objectif n'est pas d'étudier cette architecture en particulier, ni d'étudier le jeu d'instruction de ce processeur. Néanmoins nous allons examiner le déroulement d'un logiciel à l'aide d'un simulateur d'architecture MIPS32. Ceci nous permettra de mieux comprendre l'organisation et l'exécution du code pour le microprocesseur.

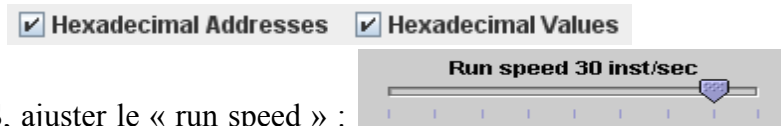
Présentation du simulateur MARS :

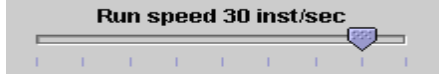
MARS est un IDE (Integrated development environment) léger pour apprendre le langage assembleur MIPS Multiplateforme: MARS est écrit en langage JAVA, donc compatible avec tous les OS avec la machine virtuelle de java grâce au format .jar

Prise en main de l'environnement :

- Au démarrage il y a seulement trois options: New (créer un document en blanc)
Open (ouvrir un fichier assembleur existant)
Help (affiche l'aide)
- En ouvrant ou créant un fichier s'activent des fonctions classique d'édition et de manipulation de fichier plus la fonction "assemble" dans le menu "run"
- Quand on active "assemble" si il y a des erreurs dans le code elles sont indiquées dans la fenêtre "Mars Messages" en donnant les lignes et les explications relatives aux erreurs. Si le code est écrit sans erreurs. L'onglet "Edit" disparaît et est remplacé par l'onglet "execute" qui montre le suivi des données en mémoire et le segment en action du code. On a alors à sa disposition d'autres fonctions: exécuter en entier, exécuter une seule instruction et redémarrer.
- L'icône  permet de lancer l'exécution du programme jusqu'à la fin. En utilisant cette fonction vous observez le surlignement jaune qui montre l'adresse du programme en cours d'exécution. Les valeurs du programme qui sont calculées apparaissent dans le segment de donnée.
- L'icône  permet de faire un reset du programme et des valeurs de l'ensemble des registres du simulateur.

- L'icône  permet d'exécuter une instruction après l'autre. Autrement dit, de faire de l'exécution pas à pas. Cette fonction est complétée par l'icône  qui permet de revenir d'un pas : "single-step backwards".
- Le processeur utilise un adressage par octet. Vous pouvez changer le format de visualisation du contenu des adresses et des adresses en hexadécimal ou en décimal :



- Dans le logiciel MARS, ajuster le « run speed » :  afin d'avoir le temps de voir l'évolution de l'algorithme en fonction du temps.
- Depuis l'aide, (**Help>Help>Basic_Instructions**) vous trouverez l'ensemble des instructions disponibles dans ce microprocesseur.

exemple1:

1. Dans le logiciel MARS créer un nouveau fichier et copier le programme assembleur ci-dessous

```
.data
vars: .word 5

.text
li $t0, 10
li $t1, 15
la $t2, vars
sw $t1, 0($t0)
lw $t2, 4($t0)
addi $t2, $t0, 4
subi $t2, $t0, 4

syscall
```

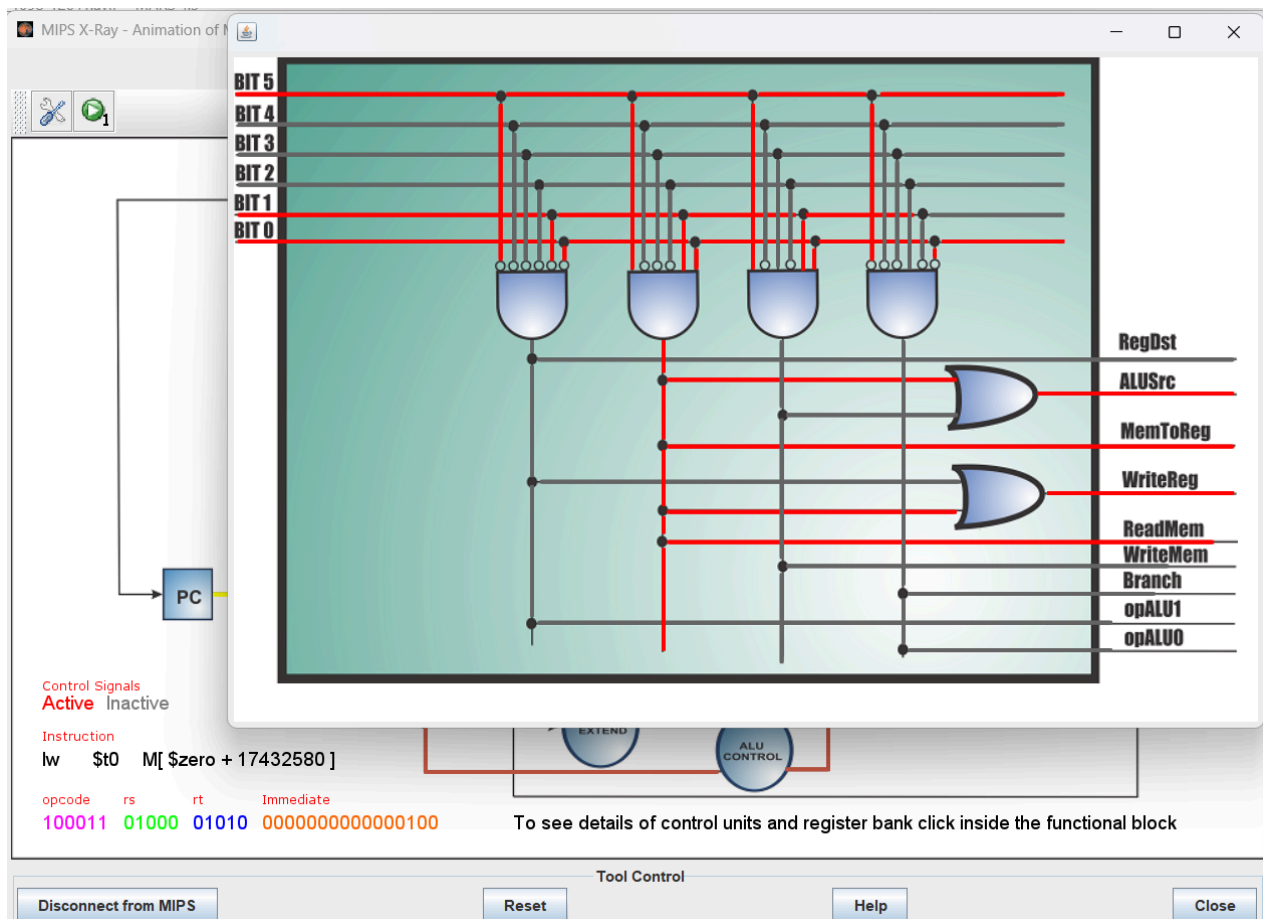
2. Déterminer ce que fait le programme en utilisant l'aide (**Help>Help**) pour expliquer le rôle de chaque ligne du code.

sol:

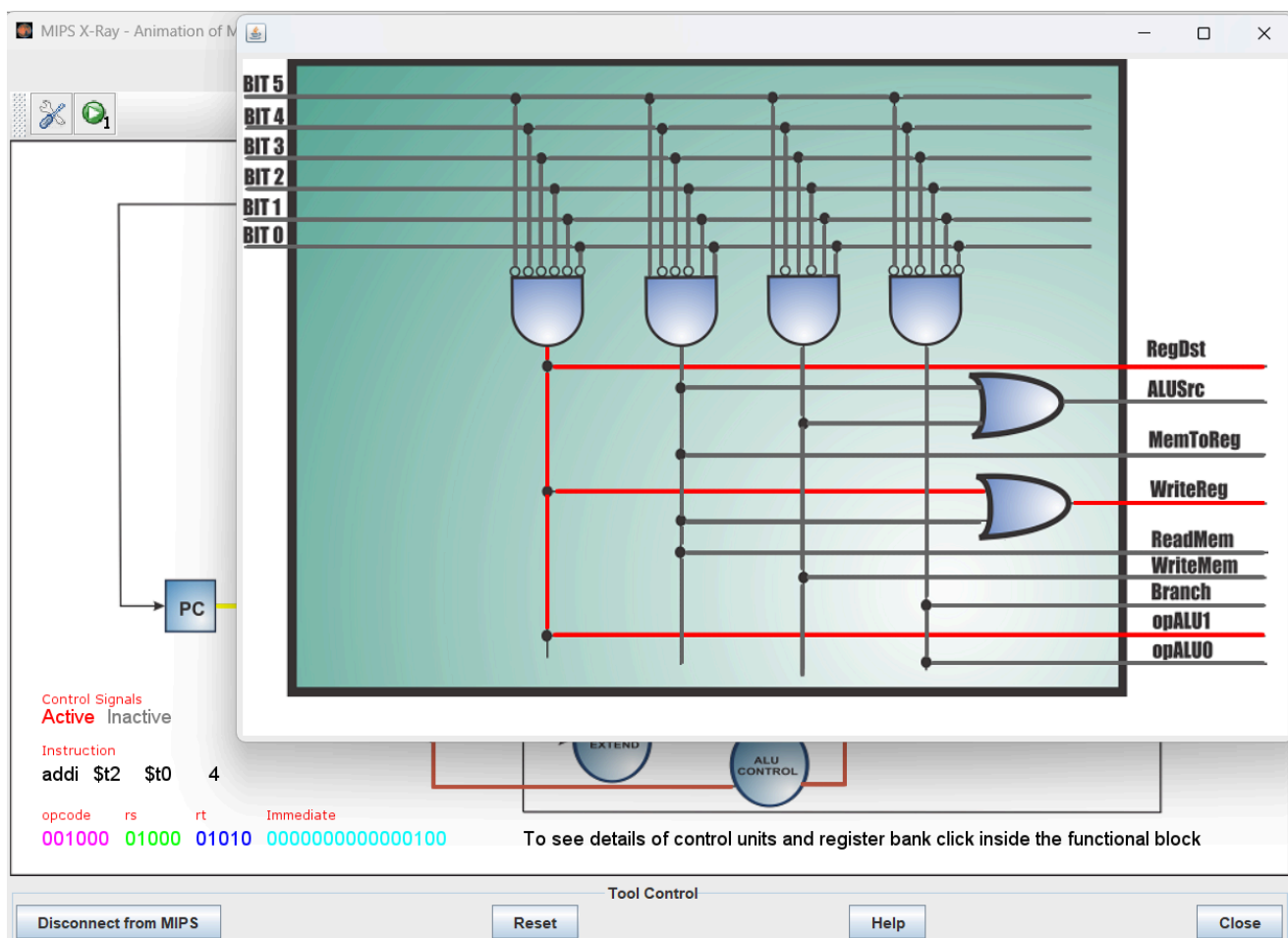
 - ligne 1:initialiser le registre t0 avec la valeur 10.
 - ligne 2:initialiser le registre t1 avec la valeur 15.
 - ligne 3:initialiser le registre t2 avec la valeur de la variable vars (valeur 5).
 - ligne 4: stocker la valeur du registre t1 (15) dans la case mémoire de l'adresse t0+0 (10).
 - ligne 5: charger la valeur de la case mémoire de l'adresse t0+4 (14) dans le registre t2.
 - ligne 6: additionner la valeur 4 avec la valeur de registre t0 et mettre le résultat dans t2.
 - ligne 7: soustraire la valeur 4 de la valeur de registre t0 et mettre le résultat dans t2.
3. Donner le code binaire de chaque instruction et le groupe de format .
4. Pour vérifier ouvrir la fenêtre MIPS X-RAY avec **Tools>mips X-RAY**.

sol de 3 et 4:

lw \$t2, 4(\$t0): format I



addi \$t2, \$t0, 4 :format R



- Donner l'état des signaux de l'unité de contrôle et du alu-contrôle pour les instructions addi et subi (pas demander).

exemple 2:

- Dans le logiciel MARS créer un nouveau fichier et copier le programme assembleur ci-dessous

```
.data
vars: .word 5

.text
la $t0, vars          PC REGISTER VALUE
lw $t1, 0($t0)        0X00400008
lw $t2, 4($t0)        0X0040000C
saut: bge $t1, $t2, exit 0X00400010
move $a0, $t1         0X00400018
li $v0, 1             0X0040001C

Syscall              0X00400020
addi $t1, $t1, 1     0X00400024
j saut               0X00400028
exit: li $v0, 10      0X0040002C
Syscall              0X00400030
```

- Déterminer la valeur du registre pc pour chaque instruction

exemple 3

- On donne le programme en c suivant:

```
main() {
    if (a == 0)
        b = update(g,h);
    else
        c = update(k,m);
}
```

Main Registers:
R20=a, R21=b, R22=c
R16=g, R17=h, R18=k, R19=m

```
update(a1,a2) {
    return (a1+a2)-(a2<<4);
}
```

Update Registers:
Arguments: R4=a1, R5=a2
R20=temp0, R21=temp1, R2=result

- Compléter l'écriture du code suivant dans l'éditeur MARS.

```

main:
    bne R20, R0, DoElse
    add R4, R16, R0 ; move g→R4
    add R5, R17, R0 ; move h→R5
    jal update
    addi R21, R2      ; move result→b
    j SkipElse
DoElse:
    add R4, R18, R0 ; move k→R4
    add R5, R19, R0 ; move m→R5
    jal update
    addi R22, R2      ; move result→c
SkipElse:

update:
    add R20, R4, R5
    sll R21, R5, 4
    sub R2, R20, R21
    jr $ra

```

3. Déterminer la valeur du registre pc pour les instructions du branchement.

PC 0x00400000 ben
 PC 0x0040000c jal
 PC 0x00400014 j
 PC 0x00400010 jal
 PC 0x00400034 jr

4. quel est le problème dans ce code?

Le problème, c'est que l'étiquette update est placée dans la suite du programme principal, juste après SkipElse, donc le programme continue dans la fonction update après avoir fini le main ce qui va exécuter update une deuxième fois même s'il n'est pas appelé.

5. proposer deux solution pour le résoudre

sol1: la solution est de faire un jump a une étiquette fin après l'étiquette SkipElse, et cette étiquette doit être déclarer a la fin du code

sol2: déplacer toute la fonction update après la fin du code principal