Ministère de l'Enseignement Supérieur
Et de la Recherche Scientifique
Université Dr. Yahia Farés de Médéa

وزارة التعليـــــم العالـــــــي
و البحث العلمي
جامعة الدكتور يحي فارس بالمدية

FACULTE DES SCIENCES ET DE LA TECHNOLOGIE

DÉPARTEMENT DE

GÉNIE ELECTRIQUE ET DE L'INFORMATIQUE

MÉMOIRE DE FIN D'ÉTUDE

Présenté Pour l'obtention du diplôme de master

En génie électrique

Option : systèmes intelligents et communicants

Thème

Modélisation du microprocesseur MIPS R3000 en VHDL

Présenté par :

Diffallah Thouraya

Encadreur :

Mr : Tiberkak Allal

Septembre 2012

Abstract:

This project aims to implement effective modeling of a 32-bit processor. The project will detail the MIPS R3000 processor architecture and presents a model of the microprogrammed processor.

Of the many potential processors that were considered the one that was chosen was a MIPS R3000 microprocessor due to its simple instruction encodings. The R3000 microprocessor is not just a processor as it also includes cache, memory management and a coprocessor interface. (It is also capable of handling exact exceptions)

ملخص:

يهدف هذا المشروع إلى تنفيذ نماذج فعالة من معالج 32بت .وسيقوم المشروع بتفصيل المعالج MIPS R3000، ويقدم نموذجا للمعالج microprogrammed.

من بين المعالجات المحتملة الكثيرة يعتبر MIPS R3000 واحدة من التي تم اختيارها بسبب ترميزات لها تعليمات بسيطة. المعالج MIPS R3000هو ليس فقط معالج لأنه يتضمن أيضا إدارة ذاكرة التخزين المؤقت.(وهي أيضا قادرة على التعامل مع الاستثناءات).

Résumé :

Ce projet a pour objectifs de mettre en œuvre la modélisation effective d'un processeur 32 bits. Le projet décrit en détails l'architecture du processeur Mips R3000 et présente une modélisation microprogrammée de ce processeur.

Parmi les nombreux processeurs potentiels qui ont été considérés comme celle qui a été choisi était un microprocesseur MIPS R3000 en raison de ses encodages simple instructions. Le microprocesseur MIPS R3000 n'est pas seulement un processeur comme il comprend également la gestion d'antémémoire, et une interface de coprocesseur. (Il est également capable de traiter les exceptions).

Remerciement

Je tiens à remercier monsieur TIBERKAK ALLAL Pour sa contribution à ma formation et son encadrement de mon Travail, ainsi pour ces conseils qui m'ont été utiles.

J'exprime mes remerciements au corps administratif de notre faculté et surtout le chef de département : Dr BOUHEDDA MONIR

Je tiens aussi à remercier mes collègues: AMMINA, FAIZA, HASSINA, et KHADIJA.

إهداء

إلهي لايطيب الليل إلا بشكرك ولايطيب النهار إلى بطاعتك .. ولاتطيب اللحظات إلا بذكرك .. ولا تطيب الآخرة إلا بعفوك .. ولا تطيب الجنة إلا برؤيتك الله جل جلاله

إلى من بلغ الرسالة وأدى الأمانة .. ونصح الأمة .. إلى نبي الرحمة ونور العالمين..

سيدنا محمد صلى الله عليه وسلم

إلى من كلله الله بالهيبة والوقار .. إلى من علمني العطاء بدون انتظار .. إلى من أحمل إسمه بكل افتخار .. أرجو من الله أن يمد في عمرك لترى ثماراً قد حان قطافها بعد طول انتظار وستبقى كلماتك نجوم أهتدي بها اليوم وفي الغد وإلى الأبد..

والدي العزيز

إلى ملاكي في الحياة .. إلى معنى الحب وإلى معنى الحنان والتفاني .. إلى بسمة الحياة وسر الوجود

إلى من كان دعائها سر نجاحي وحنانها بلسم جراحي إلى أغلى الحبايب

أمي الحبيبة

إلى من بهم أكبر وعليم أعتمد .. إلى شموع متقدة تنير ظلمة حياتي..

إلى من بوجودهم أكتسب قوة ومحبة لا حدود لها..

إلى من عرفت معهم معنى الحياة

إخوتي وأخواتي

إلى الأخوات اللواتي لم تلدهن أمي .. إلى من تحلو بالإخاء وتميزوا بالوفاء والعطاء إلى ينابيع الصدق الصافي إلى من معهم سعدت ، وبرفقتهم في دروب الحياة الحلوة والحزينة سرت إلى من كانوا معي على طريق النجاح والخير

إلى من عرفت كيف أجدهم وعلموني أن لا أضيعهم

صديقاتي زبيدة ،أمال،مريم،فاطمة الزهراء

وإلى كل من وسعتهم ذاكرتي ولم تسعهم مذكرتي

والحمد لله الذي هدانا إلى هذا وما كنا لنهتدي لولا أن هدانا الله

ثـــــريـا

Sommaire

Introduction générale …… 1

Chapitre I : Architecture externe de MIPS R3000

I.1. Introduction ……. 2

I.2. Registre visibles du logiciel ……. 2

I.2.2. Registres protégés ……. 2

I.2.1. Registres non protégés …….. 2

I.3. Adressage mémoire …… 3

I.3.1. Adresses octet …… 3

I.3.2. Calcul d'adresse …….. 3

I.3.3. Mémoire virtuelle ……... 3

I.3.4. Segmentation …… 3

I.4. Jeu d'instructions …… 4

I.4.1.Généralités …… 4

I.4.2. Codage des instructions …… 5

I.4.3. Jeux d'instruction …… 7

I.5. Exceptions et interruptions ……. 8

I.5.1. Exceptions …… 8

I.5.2. Interruptions …… 9

I.5.3. Appels système: instructions SYSCALL et BREAK ……. 10

I.5.4. Signal RESET …….. 10

I.5.5. Retour d'interruption ……. 11

I.5.6. Gestion du registre d'état SR …… 11

I.5.7. Gestion du registre de cause CR ……... 12

I.6.Conclusion ……... 13

Chapitre II : Architecture interne de MIPS R3000

II.1.Introduction …….. 14

II.2. Interface du processeur ……. 14

II.3. Architecture interne …….. 15

II.4. Chemin de données …….. 16

II.6. Conclusion …….. 18

Chapitre III : Exemple d'une architecture simplifiée d'un processeur

III.1. Introduction …… 19

III.2. Architecture du premier ordinateur …….. 19

III.2.1. Micro-mem …….. 19

III.2.2. Micro-PC …… 19

III.2.3. Décodeur d'instructions …… 20

III.2.4. ALU …… 20

III.2.5. Mémoire centrale …… 21

III.2.6. Le registre d'adresse ……. 21

III.2.7. PC …… 21

III.3. Contenu de la micro mémoire ……. 21

III.3.1. Chargement et décodage d'une instruction ……. 22

III.3.2. Micro programme pour l'instruction NOP (No OPeration) ……. 22

III.3.3.Micro programme pour l'instruction LDIMM (load immediate) …. 23

III.3.4. Micro programme pour l'instruction LD …… 24

III.3.4. Micro programme pour l'instruction ST ……. 25

III.4. Conclusion …….. 26

Chapitre IV : Conception et simulation de mips R3000par VHDL

IV.1. Introduction …… 27

IV.2. Registre …….. 27

IV.2.1.Description …… 27

IV.2.2. Simulation ……. 29

IV.3. Compteur …… 29

IV.3.1. Description …… 29

IV.3.2.Simulation …… 30

IV.4. Compteur ordinal(PC**)** …… 31

IV.4.1. Description ……. 31

IV.4.2.Simulation ……. 32

IV.5. Décodeur d'instruction …….. 32

IV.5.1.Description …… 33

IV.5.2.Simulation …… 34

….

IV.6.Unité arithmétique logique(UAL) ……. 34

IV.6.1.Description …….. 34

VI.6.2.Simulation …… 36

IV.7. Opérateur trois états ……. 36

IV.7.1.Description …….. 36

IV.7.2.Simulation …… 37

IV.8. Micro mémoire …… 37

IV.8.1.Description …… 38

IV.8.2.Simulation …… 38

IV.9. Mémoire …….. 38

IV.9.1.Description …… 39

IV.9.2.Simulation …… 39

IV.10.Mux1 …… 41

IV.10.1.Description …… 41

IV.10.2.Simulation …… 44

IV.11.Mux2 …… 45

IV.11.1.Description …… 45

IV.11.2.Simulation …… 47

IV.12.Schéma fonctionnelle global …… 48

IV.12.1.Description …… 48

IV.12.2.Simulation …… 49

IV.13.Conclusion …… 49

Conclusion générale …… 50

Annexes …… 51

Bibliographie

Liste des tables

Tableau I.1. Les formats d'instructions …… 4
Tableau I.2. Le codage de l'instruction INS31 :26 …… 5
Tableau I.3. Le codage de l'instruction INS 5 :0 ……. 6
Tableau I.4. Le codage de l'instruction INS 16 et INS 20 …….. 6
Tableau I.5. Le codage de l'instruction INS 23 et INS 25 ……. 7
Tableau I.6. Le format de registre d'état SR …….. 11
Tableau I.7. Format de registre de cause CR …….. 13
Tableau II.1. Les requêtes possibles de processeur …… 14
Tableau III.1. Le micro programme pour chargement et décodage d'instruction .. 22
Tableau III.2. Le micro programme de l'instruction NOP …… 23
Tableau III.3. Le micro programme de l'instruction LDIMM …… 24
Tableau III.4. Le micro programme de l'instruction LD …… 25
Tableau III.5. Le micro programme de l'instruction ST …… 26
Tableau IV.1 . Codage de l'instruction ADDI. …… 33

Liste des figures

Figure I.1. Décalage de registre d'état SR …… 12

Figure I.2. Décalage inverse de registre d'état SR …… 12

Figure II.1. Architecture externe de MIPS R3000 …… 15

Figure II.2. Architecture interne de MIPS R3000 …… 15

Figure II.3. Chemin de données …… 17

Figure II.4. Exemple d'un transfert élémentaire …… 18

Figure III.1. Architecture du premier ordinateur …….. 20

Figure IV.1. Schéma de MIPS 3000 ……. 27

Figure. IV.2. Registre ……. 28

Figure IV.3. Simulation du registre …… 29

Figure IV.4. Compteur …….. 30

Figure IV.5. Simulation de compteur ……. ……. 30

Figure IV.6. Compteur ordinal …….. 31

Figure IV.7. Simulation de compteur ordinal …… 32

Figure IV.8. Décodeur d'instruction …… 33

Figure VI.9. Simulation de décodeur d'instruction …… 34

Figure IV.10. Unité arithmitique logique ……. 35

Figure IV.11. Simulation de l'unité arithmétique logique ……..…… 36

Figure IV.12.Opérateur trois états …….. 37

Figure VI.13.Simulation d'opérateur trois états …… 37

Figure IV.14. Micro mémoire …… 38

Figure IV.15. Simulation de micro mémoire …… 38

Figure IV.16. Mémoire …… 39

Figure IV.17. Simulation de la mémoire ……... 40

Figure IV.18. Mux1 (multiplexeur) …… 43

Figure IV.19. Simulation de mux1 …… 44

Figure IV.20. Mux2 (démultiplexeur) …….. 46

Figure IV.21. Simulation de mux2 …….. 47

Figure IV.22.Schéma fonctionnelle global …… 48

Figure IV.23.Simulation de chargement d'instruction …… 49

Introduction Général

Introduction général

Le processeur **MIPS R3000** est un processeur 32 bits industriel conçu dans les années 80. Son jeu d'instructions est de type RISC. Il existe plusieurs réalisations Industrielles de cette architecture (SIEMENS, NEC, LSI LOGIC, SILICON GRAPHICS, etc...). Cette architecture est suffisamment simple pour présenter les principes de base de L'architecture des processeurs, et suffisamment puissante pour supporter un système D'exploitation multitâche telle qu'UNIX.

Le but de ce travail est de modéliser l'architecture en utilisant le VHDL. Pour cette fin, ce mémoire est contient quatre chapitres :

Dans le premier chapitre nous allons voir l'architecture externe de microprocesseur MIPS R3000 (Les registres visibles du logiciel, L'adressage, Les deux modes utilisateur / superviseur comme support matériel au fonctionnement multi-tâches et multi-utilisateurs, et le traitement des interruptions et exceptions). Dans la deuxième chapitre nous allons voir l'architecture interne de microprocesseur MIPS R3000 (l'interface du processeur, Décomposition entre partie opérative et partie contrôle, Et chemin de données). Dans le chapitre numéro trois nous allons présenter une architecture simplifiée d'un simple microprocesseur. Le quatrième chapitre sera consacré a notre conception et implémenter du processeur MIPS R3000 en VHDL. Nous allons fini le travaille par une conclusion.

chapitre I:Architecture externe de MIPS R3000

I.1. Introduction

Ce chapitre présent une version légèrement simplifiée de l'architecture interne du processeur **MIPS R3000** (pour des raisons de simplicité, tous les mécanismes matériels de gestion de la mémoire virtuelle ont été délibérément supprimés).

L'architecture interne représente ce que doit connaitre un programmeur souhaitant programmer en assembleur, ou la personne souhaitant écrire un compilateur pour ce processeur:

- Les registres visibles.
- L'adressage de la mémoire.
- Le jeu d'instructions.
- Les mécanismes de traitement des interruptions et exceptions.

I.2. Registre visibles du logiciel [1]

Le processeur possède deux modes de fonctionnement :

utilisateur/superviseur. Ces deux modes de fonctionnement imposent d'avoir deux catégories de registres.

I.2.1. Registres non protégés

• **Ri :**($0 \leq i \leq 31$) 32 registres généraux

Le registre **R0** est un registre particulier:

- la lecture fournit la valeur constante "0x00000000"
- l'écriture ne modifie pas son contenu.

Le registre **R31** est utilisé par les instructions d'appel de procédures (Instructions BGEZAL, BLTZAL, JAL et JALR) pour sauvegarder l'adresse de retour.

• **PC :** Ce registre contient l'adresse de l'instruction en cours d'exécution. Sa

Valeur est modifiée par toutes les instructions.

• **HI et LO :** Ces deux registres 32 bits sont utilisés pour stocker le résultat d'une multiplication ou d'une division, qui est un mot de 64 bits.

I.2.2. Registres protégés

• **SR** Registre d'état (Status Register) : Il contient en particulier le bit qui définit le mode : superviseur ou utilisateur, ainsi que les bits de masquage des interruptions.

• **CR** Registre de cause (Cause Register) : En cas d'interruption ou d'exception, son contenu définit la cause pour laquelle on fait appel au programme de traitement des interruptions et des exceptions.

• **EPC** Registre d'exception (Exception Program Counter) : Il contient l'adresse de retour (PC + 4) en cas d'interruption. Il contient l'adresse de l'instruction fautive en cas d'exception (PC).

• **BAR** Registre d'adresse illégal (Bad Address Register): En cas d'exception de type "adresse illégale", il contient la valeur de l'adresse mal formée.

I.3. Adressage mémoire

I.3.1. Adresses octet : Toutes les adresses émises par le processeur sont des adresses octet, codées sur 32 bits. Le processeur peut lire des instructions (32 bits), des mots (32 bits), des demi-mots (16 bits) et des octets (8 bits). Il peut écrire des mots, des demi-mots et des octets. Les adresses des mots et les instructions doivent être des multiples de 4, et celles des demi-mots doivent être des multiples de 2.

I.3.2. Calcul d'adresse : Il existe un seul mode d'adressage, consistant à effectuer la somme entre le contenu d'un registre général Ri, défini dans l'instruction, et d'un déplacement qui est une valeur immédiate signée, sur 16 bits, contenue également dans l'instruction: Adresse = Ri + Déplacement.

I.3.3. Mémoire virtuelle : Pour des raisons de simplicité, cette version du processeur R3000 ne possède pas de mémoire virtuelle, c'est à dire que le processeur ne contient aucun mécanisme matériel de traduction des adresses logiques en adresses physiques. Les adresses calculées par le logiciel sont donc transmises au système mémoire sans modifications. On suppose que la mémoire répond en un cycle. Un signal permet au système mémoire de "geler" le processeur s'il n'est pas capable de répondre en un cycle.

Si une anomalie est détectée au cours du transfert entre le processeur et la mémoire, le système mémoire peut le signaler au moyen d'un signal d'erreur, qui déclenche un départ en exception.

I.3.4. Segmentation : L'espace mémoire est découpé en 2 segments identifiés par le bit de poids fort de l'adresse :

adr 31 = 0 ==> segment utilisateur

adr 31 = 1 ==> segment système

Quand le processeur est en mode superviseur, les 2 segments sont accessibles. Quand le processeur est en mode utilisateur, seul le segment utilisateur est accessible.

I.4. Jeu d'instructions

I.4.1. Généralités : Le processeur possède 57 instructions qui se répartissent en 4 classes:

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

On distingue 3 formats d'instructions :

Le format **J** n'est utilisé que pour les branchements à longue distance (Inconditionnels).

Le format **I** est utilisé par les instructions de lecture/écriture mémoire, par les Instructions utilisant un opérande immédiat, ainsi que par les branchements courte distance (conditionnels).

Le format **R** est utilisé par les instructions nécessitant 2 registres sources (désignés par RS et RT) et un registre résultat désigné par RD.

Format R

31 25	20 15	10 5	0
OPCOD	RS	RT	RD	SH	FUNC

Format I

OPCOD	RS	RT	IMD16

Format J

OPCOD	IMD26

Tableau I.1. Les formats d'instructions

Avec :

- OPCOD : code opération sur 6 bits.
- RS : n° de registre source sur 5 bits.
- RT: n° de registre de transit sur 5 bits.
- RD : n° de registre de destination sur 5 bits.
- IMD16 : valeur immédiate 16 bits.
- IMD26 : adresse cible sur 26 bits.

- SH : valeur de décalage sur 5 bits (pour les instructions de décalage, rotations).
- FUNC : champ de fonction sur 6 bits.

I.4.2. Codage des instructions : Le codage des instructions est principalement défini par les 6 bits des champs code opération de l'instruction (INS 31:26). Cependant, trois valeurs particulières de ce champ définissent en fait une famille d'instructions : il faut alors analyser d'autres bits de l'instruction pour décoder l'instruction. Ces codes particuliers sont :

SPECIAL (valeur "000000"), BCOND (valeur "000001") et COPRO (valeur"010000").

INS 31:29 \ INS 28 :26	000	001	010	011	100	101	110	111
000	SPECIAL	BCOND	J	JAL	BEQ	BNE	BLEZ	BGTZ
001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
010	COPRO							
011								
100	LB	LH		LW	LBU	LHU		
101	SB	SH		SW				
110								
111								

Tableau I.2. Le codage de l'instruction INS31 :26

Ce tableau exprime que l'instruction LHU (par exemple) possède le code opération "100101". Lorsque le code opération a la valeur SPECIAL ("000000"), il faut analyser les 6 bits de poids faible de l'instruction (**INS 5:0**):

INS 5 :3 \ INS 2 :0	000	001	010	011	100	101	110	111
000	SLL		SRL	SRA	SLLV		SRLV	SRAV
001	JR	JALR			SYSCALL	BREAK		
010	MFHI	MTHI	MFLO	MTLO				
011	MULT	MULTU	DIV	DIVU				
100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
101			SLT	SLTU				
110								
111								

Tableau I.3. Le codage de l'instruction INS 5 :0

Lorsque le code opération a la valeur BCOND, il faut analyser les bits 20 et 16 de l'instruction. Lorsque le code opération a la valeur COPRO, il faut analyser les bits 25 et 23 de l'instruction. Les trois instructions de cette famille COPRO sont des instructions privilégiées.

OPCOD=BCOND

INS 20 \ INS 16	0	1
0	BLTZ	BGEZ
1	BLTZAL	BGEZAL

Tableau I.4. Le codage de l'instruction INS 16 et INS 20

OPCOD=COPRO

INS 23

INS 25

0	MFCO	MTCO
1	RFE	

Tableau I.5. Le codage de l'instruction INS 23 et INS 25

I.4.3. Jeux d'instruction : Le jeu d'instructions est "orienté registres". Cela signifie que les instructions Arithmétiques et logiques prennent leurs opérandes dans des registres et rangent le résultat dans un registre. Les seules instructions permettant de lire ou d'écrire des données en mémoire effectuent un simple transfert entre un registre général et la mémoire, sans aucun traitement arithmétique ou logique.

La plupart des instructions arithmétiques et logiques se présentent sous les 2 formes registre-registre et registre-immédiat:

- ADD : R(rd) <--- R(rs) op R(rt) format R
- ADDI : R(rt) <--- R(rs) op IMD format I

L'opérande immédiat 16 bits est signé pour les opérations arithmétiques et non signé pour les opérations logiques.

Le déplacement est de 16 bits pour les instructions de branchement conditionnelles (Bxxx) et de 26 bits pour les instructions de saut inconditionnelles (Jxxx). De plus les instructions JAL, JALR, BGEZAL, et BLTZAL sauvegardent une adresse de retour dans le registre R31. Ces instructions sont utilisées pour les appels de sous programme.

Toutes les instructions de branchement conditionnel sont relatives au compteur ordinal pour que le code soit translatable. L'adresse de saut est le résultat d'une addition entre la valeur du compteur ordinal et un déplacement signé.

Les instructions MTC0 et MFC0 permettent de transférer le contenu des registres SR, CR, EPC et BAR vers un registre général et inversement. Ces 2 instructions ne peuvent

être exécutées qu'en mode superviseur, de même que l'instruction RFE qui permet de restaurer l'état antérieur du registre d'état avant de sortir du gestionnaire d'exception

I.5. Exceptions et Interruptions

Il existe quatre types d'évènements qui peuvent interrompre l'exécution "normale" d'un programme:

– les exceptions

– les interruptions

– les appels système (instructions SYSCALL et BREAK)

– le signal RESET

I.5.1. Exceptions : Les exceptions sont des évènements "anormaux", le plus souvent liés à une erreur de programmation, qui empêchent l'exécution correcte de l'instruction en cours. La détection d'une exception entraîne l'arrêt immédiat de l'exécution de l'instruction fautive. Ainsi, on assure que l'instruction fautive ne modifie pas la valeur d'un registre visible ou de la mémoire. Les exceptions ne sont évidemment pas masquables. Il y a 7 types d'exception dans cette version du processeur R3000 :

• ADEL (Adresse illégale en lecture): adresse non alignée ou se trouvant dans le segment système alors que le processeur est en mode utilisateur.

• ADES (Adresse illégale en écriture) : adresse non alignée ou accès à Une donnée dans le segment système alors que le processeur est en mode utilisateur.

• DBE (Data bus erreur): le système mémoire signale une erreur en activant le signal BERR à la suite d'un accès de donnée.

• IBE (Instruction bus erreur) : le système mémoire signale une erreur en activant le signal BERR à l'occasion d'une lecture instruction.

• OVF (Dépassement de capacité) : lors de l'exécution d'une instruction arithmétique (ADD, ADDI ou SUB), le résultat ne peut être représenté sur 32 bits.

• RI(Codop illégal): le codop ne correspond à aucune instruction connue (il s'agit probablement d'un branchement dans une zone mémoire ne contenant pas du code exécutable.

• CPU (Coprocesseur inaccessible): tentative d'exécution d'une instruction privilégiée (MTC0, MFC0, RFE) alors que le processeur est en mode utilisateur. Le processeur doit alors passer en mode superviseur, et se brancher au gestionnaire d'exceptions qui est une

routine logicielle implantée Conventionnellement à l'adresse "0x80000080". Toutes les exceptions étant fatales dans cette version du processeur R3000, il n'est pas nécessaire de sauvegarder une adresse de retour car il n'y a pas de reprise de l'exécution du programme contenant l'instruction fautive. Le processeur doit cependant transmettre au gestionnaire d'exceptions l'adresse de l'instruction fautive et indiquer dans le registre de cause le type d'exception détectée.

Lorsqu'une exception est détectée, le processeur :
– sauvegarde l'adresse de l'instruction fautive dans le registre EPC
–sauvegarde l'ancienne valeur du registre d'état SR
– passe en mode superviseur et masque les interruptions dans SR
– écrit le type de l'exception dans le registre CR
– branche à l'adresse "0x80000080".

I.5.2. Interruptions : Les requêtes d'interruption matérielles sont des évènements asynchrones provenant généralement de périphériques externes. Elles peuvent être masquées. Le processeur possède 6 lignes d'interruptions externes qui peuvent être masquées globalement ou individuellement. L'activation d'une de ces lignes est une requête d'interruption. Elles sont inconditionnellement écrites dans le registre CR, et elles sont prises en compte à la fin de l'exécution de l'instruction en cours si elles ne sont pas masquées. Cette requête doit être maintenue active par le périphérique tant qu'elle n'a pas été prise en compte par le processeur.

Le processeur passe alors en mode superviseur et se branche ici encore au Gestionnaire d'exceptions. Comme il faut reprendre l'exécution du programme En cours à la fin du traitement de l'interruption, il faut sauvegarder une adresse de retour.
Lorsqu'une requête d'interruption non-masquée est détectée, le processeur :

– sauvegarde l'adresse de retour (PC + 4) dans le registre EPC
– sauvegarde l'ancienne valeur du registre d'état SR
– passe en mode superviseur et masque les interruptions dans SR
– écrit qu'il s'agit d'une interruption dans le registre CR
– branche à l'adresse "0x80000080".

En plus des 6 lignes d'interruption matérielles, le processeur R3000 possède un Mécanisme d'interruption logicielle: Il existe 2 bits dans le registre de cause CR qui peuvent être écrits par le logiciel au moyen de l'instruction privilégiée MTC0. La Mise à 1

de ces bits déclenche le même traitement que les requêtes d'interruptions Externes, s'ils ne sont pas masqués.

I.5.3. Appels système: instructions SYSCALL et BREAK : L'instruction SYSCALL permet à une tâche (utilisateur ou système) de demander un service au système d'exploitation, comme par exemple effectuer une entrée sortie.
Le code définissant le type de service demandé au système, et un éventuel paramètre doivent avoir été préalablement rangés dans des registres généraux.

L'instruction BREAK est utilisée plus spécifiquement pour poser un point d'arrêt (Dans un but de déverminage du logiciel): on remplace brutalement une instruction Du programme à déverminer par l'instruction BREAK. Dans les deux cas, le Processeur passe en mode superviseur et se branche au gestionnaire D'exceptions. Ces deux instructions sont exécutables en mode utilisateur. Elles Effectuent les opérations suivantes :

– sauvegarde de l'adresse de retour (PC + 4) dans le registre EPC
– sauvegarde de l'ancienne valeur du registre d'état SR
– passage en mode superviseur et masquage des interruptions dans SR
– écriture de la cause du déroutement dans le registre CR
– branchement à l'adresse "0x80000080".

I.5.4. Signal RESET : Le processeur possède également une ligne RESET dont l'activation, pendant au moins un cycle, entraîne le branchement inconditionnel au logiciel d'initialisation. Cette requête est très semblable à une septième ligne d'interruption externe avec les Différences importantes suivantes :

- elle n'est pas masquable.
- il n'est pas nécessaire de sauvegarder une adresse de retour.
- le gestionnaire de reset est implanté à l'adresse "0xBFC00000".

Dans ce cas, le processeur :

- passe en mode superviseur et masque les interruptions dans SR.
- branche à l'adresse « 0xBFC00000 ».

I.5.5. Retour d'interruption : Avant de reprendre l'exécution d'un programme qui a effectué un appel système (Instructions SYSCALL ou BREAK) ou qui a été interrompu, il est nécessaire D'exécuter l'instruction RFE. Cette instruction effectue la restitution de l'état précédent dans le registre SR.

I.5.6. Gestion du registre d'état SR : Le registre d'état contient l'état courant du processeur, défini par le bit de mode KUC et par le bit de masque global des interruptions IEC. La valeur 00 pour les Bits KUC et IEC correspond à l'état superviseur et interruptions masquées. Le Registre SR contient aussi l'état précédent (bits KUP et IEP) et l'état antérieur (bits KUO et IEO). Il constitue donc une petite pile matérielle capable d'empiler 3 états successifs du processeur. Le registre SR contient par ailleurs 6 bits MIRQ (5:0) permettant de masquer individuellement les 6 interruptions externes et 2 bits MSWI (1:0) permettant de masquer les deux interruptions logicielles. Les 16 bit de poids fort et les deux bits 6 et 7 du registre SR ne sont pas utilisés: On récupère la valeur 0 en lecture et ils ne sont pas modifiés par les écritures.

15 14 13 12 11 10	9	8	7	6	5	4	3	2	1	0
MIRQ	MSWI	0	0	KUO	IEO	KUP	IEP	KUC	IEC	

Tableau I.6. Le format de registre d'état SR

Lors d'un appel au gestionnaire d'exception, la sauvegarde de l'ancien état et le Passage dans l'état "superviseur avec interruptions masquées" peuvent s'effectuer en Une seule opération par décalage de deux bits vers la gauche des six bits de poids Faible du registre SR (les 8 bits MIRQ et MSWI ne sont pas modifiés) (Figure I.1) :

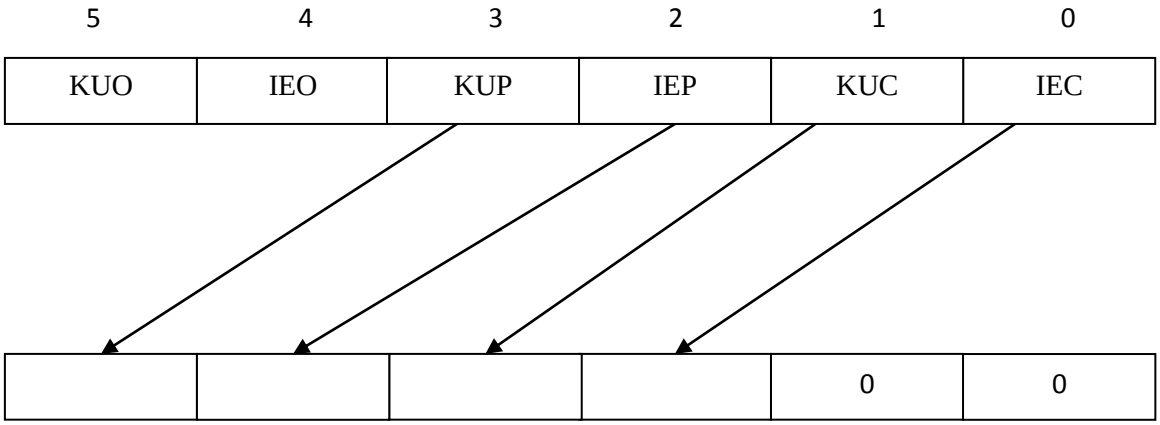


Figure I.1. Décalage de registre d'état SR

La restitution de l'ancien état, réalisée par l'instruction RFE, est obtenue par un Simple décalage de deux bits vers la droite des six bits de poids faible du registre SR (les 8 bits MIRQ et MSWI ne sont pas modifiés) (Figure I.2):

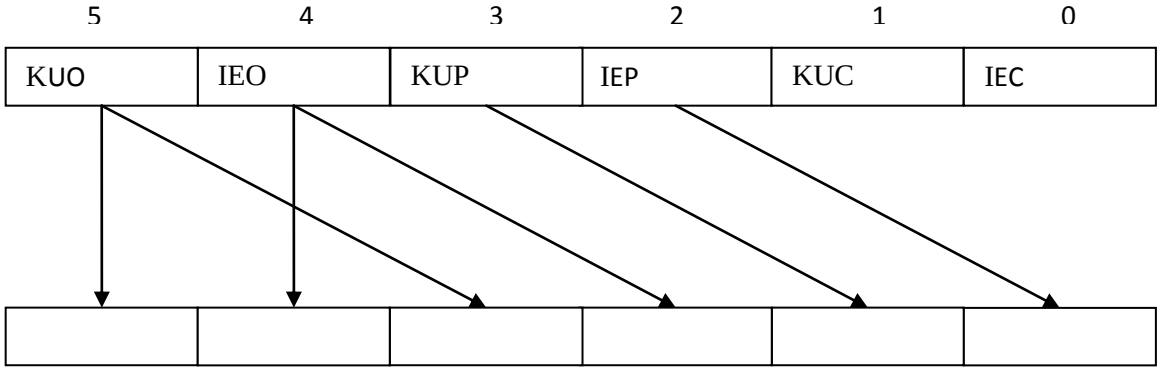


Figure I.2. Décalage inverse de registre d'état SR

I.5.7. Gestion du registre de cause CR : Le registre CR contient trois champs. Les 4 bits des champs XCODE (3:0) définissent la cause de l'appel au gestionnaire d'exceptions. Les 6 bits du Champs IRQ (5:0) représentent l'état des lignes d'interruption externes au moment De l'appel au gestionnaire d'exception. Les 2 bits SWI (1:0) représentent les Requêtes d'interruption logicielle.

15 14 13 12 11 10	9 8	7	6	5 4 3 2	1	0
IRQ	SWI	0	0	XCOD	0	0

Tableau I.7. Format de registre de cause CR

I.6. Conclusion

Le MIPS R3000 est la deuxième génération de processeur de l'architecture MIPS, et n'est plus commercialisé aujourd'hui. Son successeur est le R4000 qui lui ressemble fort. Des informations sur cette famille de processeurs, très utilisée Dans les applications embarquées et dans les machines parallèle à mémoire partagées.

L'architecture interne est le niveau d'abstraction nécessaire à l'écriture de programmes assembleur, de la partie génération de code d'un compilateur, et du programmeur de systèmes d'exploitation multi-processus (et/ou multi-tâches). Dans ce chapitre nous avons détaillés :

- les registres visibles du logiciel ;
- l'adressage de la mémoire ;
- le jeu d'instruction ;
- les mécanismes de traitement des exceptions, interruptions et appels systèmes.

chapitre II:Architecture Interne de MIPS R3000

II.1. Introduction

Ce chapitre présent l'architecture interne de Processeur MIPS R3000. Ce processeur contient deux parties, la partie opérative qui contient le chemin de données, et la partie contrôle qui contient séquenceur.

II.2. Interface du processeur [2]

L'interface entre le processeur et la mémoire est réalisé par les signaux ADR [31:0], DATA [31:0], RW [2:0], FRZ, BERR. Les requêtes possibles vers la mémoire sont les suivantes :

RW	**REQUETE**	
000	NO	**ni écriture, ni lecture**
001	WW	**écriture d'un mot**
010	WH	**écriture d'un demi-mot**
011	WB	**écriture d'un octet**
1**	RW	**lecture d'un mot**

Tableau II.1. Les requêtes possibles de processeur

Dans le cas WW, les 4 octets du bus DATA sont écrits en mémoire à une adresse alignée sur les mots (les 2 bits de poids faible de ADR ne sont pas pris en compte). Dans le cas WH, les 2 octets de poids faibles du bus DATA sont écrits à une adresse alignée sur les demi-mots (le bit de poids faible de ADR n'est pas pris en compte).

Dans le cas WB, l'octet de poids faible du bus DATA est écrit à l'adresse ADR. Dans le cas RW, les deux bits de poids faible de ADR ne sont pas pris en compte, et la mémoire doit fournir sur le bus DATA un mot aligné. Dans le cas des instructions LBU et LHU, c'est le processeur qui effectue le recadrage et l'extension de signe. Dans le cas où le système mémoire ne peut pas satisfaire en un cycle la requête d'écriture ou de lecture, le signal FRZ doit être activé : le processeur maintient sa requête tant que le signal FRZ est actif.

Dans le cas d'une erreur matérielle lors d'un accès à la mémoire, cette erreur peut être signalée au processeur par le signal BERR (Figure II.1).

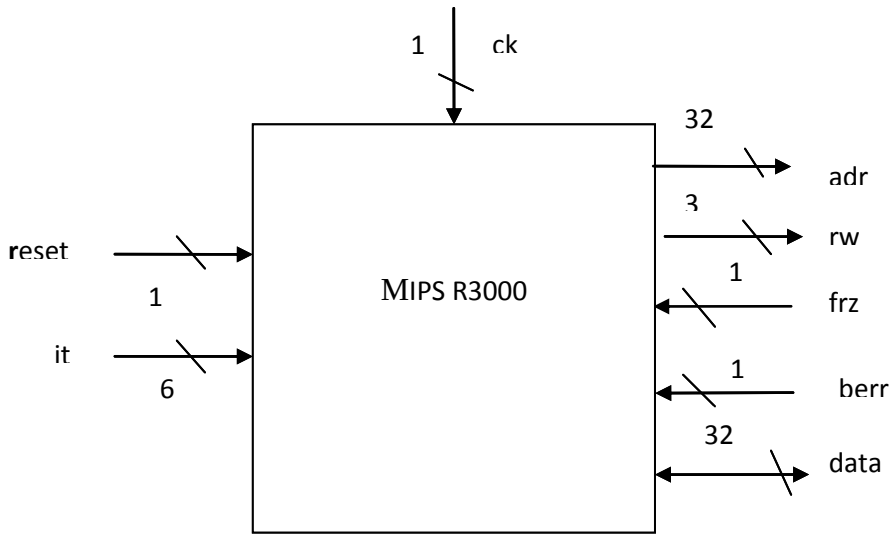


Figure II.1. Architecture externe de MIPS R3000

II.3. Architecture interne

L'architecture interne du processeur se décompose en une partie opérative et une partie contrôle (Figure II.2).

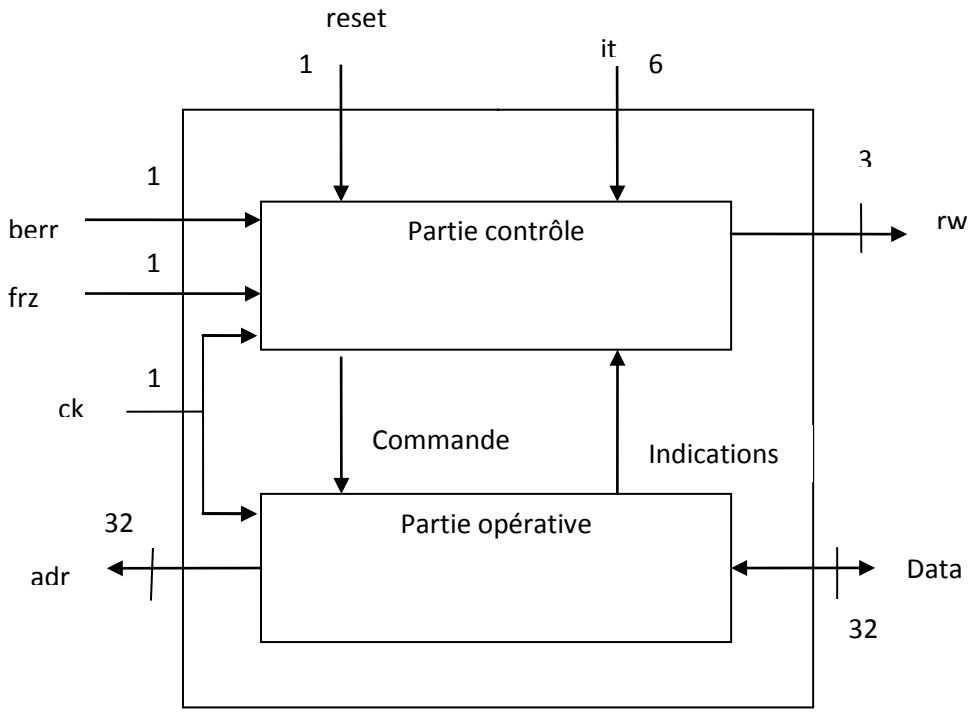


Figure II.2. Architecture interne de MIPS R3000

–La partie opérative (ou « chemin de données ») contient les registres 32 bits et les opérateurs permettant de combiner entre eux les mots de 32 bits.

Elle est capable d'exécuter des « transferts élémentaires », c'est-à-dire des transferts qui peuvent s'exécuter en un cycle.

–La partie contrôle (ou « séquenceur ») est chargée d'enchaîner les différents transferts élémentaires correspondant à l'exécution d'une instruction.

Pour prendre ses décisions, elle dispose des indicateurs fournis par la partie opérative, comme par exemple la valeur des champs OPCOD, RS, RT, et RD de l'instruction stockée dans le registre IR.

II.4. Chemin de données [3]

Les registres « visibles du logiciel » définissent l'état du processeur « du point de vue du programmeur » : On décrit l'état de la machine après l'exécution de l'instruction i, et avant l'exécution de l'instruction i+1. Ces registres ne sont pas suffisants pour permettre l'exécution des instructions. Donc On ajoute 3 registres supplémentaires dans le chemin de données :

– registre AD : permet de stocker un résultat intermédiaire, comme par exemple l'adresse de la donnée à lire ou écrire en mémoire.

– registre IR : permet de stocker les 32 bit du code de l'instruction courante pendant toute la durée d'exécution de cette instruction.

– registre DT : permet de stocker les 32 bits de la donnée qu'on vient de lire en mémoire avant de la stocker dans un accumulateur Ri.

Les registres SR et CR sont implantés dans la partie contrôle.

Le processeur possède deux bus opérandes X et Y et une seule "boîte à opérations" capable d'effectuer des opérations logiques, arithmétiques, comparaison, ou des décalages multi-bits. Toutes les opérations s'exécutent en in cycle, sauf les opérations de multiplication et de division qui s'exécutent en deux cycles. Seuls les registres PC et AD permettent d'adresser la mémoire externe.

Un mot de 32 bits provenant de la mémoire peut être écrit dans les registres DT ou IR.

Un mot de 32 bits provenant de la boîte à opérations peut être écrit dans les registres PC, EPC, SR, CR, BAR, HI, LO, AD, ou dans un registre général R(n) (Figure II.3).

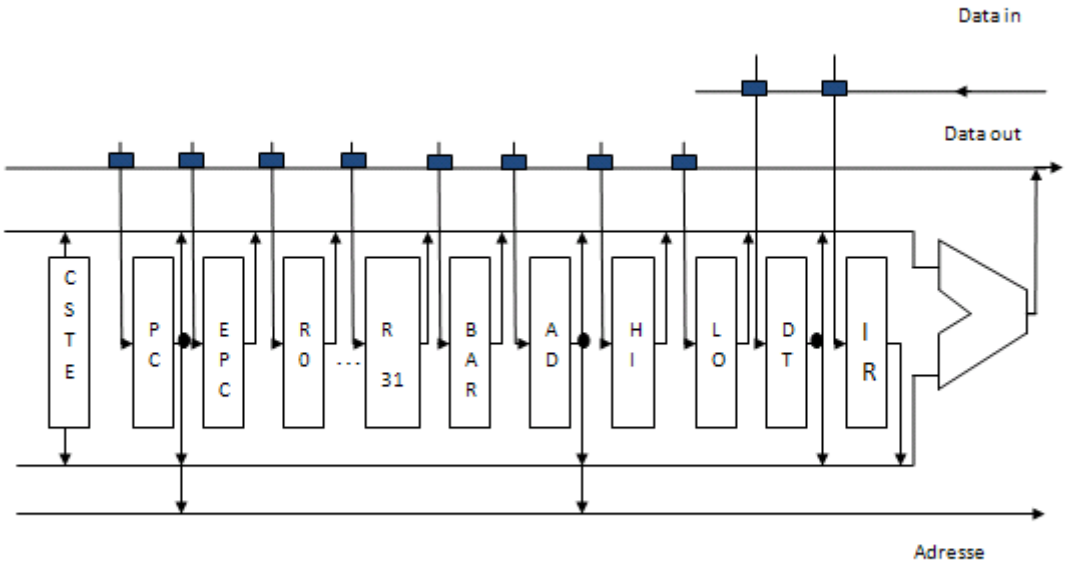


Figure II.3. Chemin de données

Ceci signifie que cette partie opérative est capable d'effectuer deux transferts élémentaires à chaque cycle :

- un transfert interne (qui utilise deux registres sources, la boîte à opérations et un registre résultat).
- et un transfert externe (lecture d'un mot en mémoire externe et écriture dans IR ou DT) (Figure II.4).

Exemple : PC <= PC + 1 / DT <= M[AD]

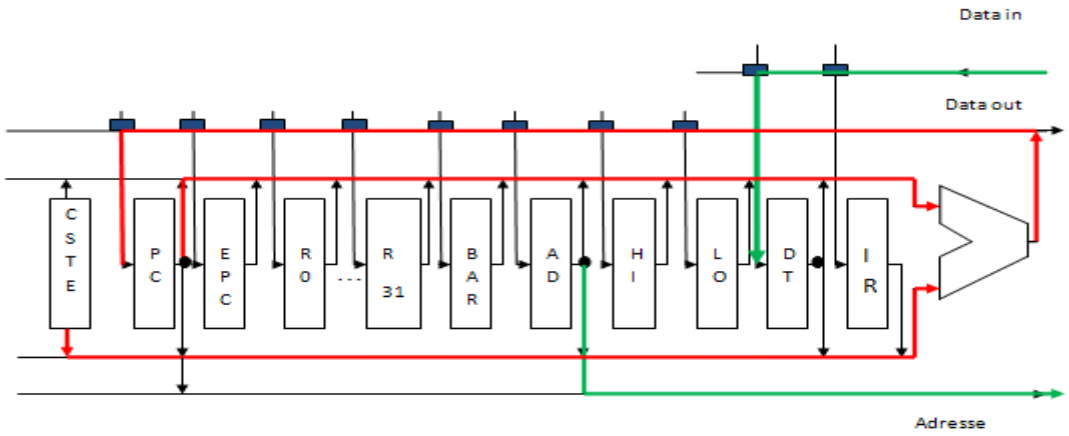


Figure II.4. Exemple d'un transfert élémentaire

Le banc de registre R(n) ne possède que 2 accès simultanés (un en lecture et un en écriture).Des pseudo-registres permettent de forcer des valeurs constantes sur les bus X et Y. Les 32 bits du registre d'instruction IR sont envoyés à la partie contrôle pour lui permettre de prendre des décisions. Elle renvoie également vers la partie contrôle les bits de signe des deux opérandes X et Y, ainsi que celui du résultat, la retenue de l'alu et les deux bits de poids faibles du résultat. Ces signaux, calculés par la micro-instruction i peuvent être utilisés par la partie contrôle pour choisir la micro-instruction i+1.

II.6. Conclusion

L'architecture externe du processeur MIPS R3000 se décompose en une partie opérative et une partie contrôle. La partie opérative (PO) contient les registres et les opérateurs. Cette architecture réalise des transferts élémentaires de données entre un ou plusieurs registres sources et un registre destination. Un transfert élémentaire est exécuté en un cycle. La partie opérative est commandée par la partie contrôle (PC). La partie contrôle est chargée de définir, pour chaque cycle d'horloge, les transferts élémentaires qui doivent être réalisés par la partie opérative.

chapitre III: Exemple d'une architecture simplifiée d'un processeur

III.1. Introduction

Dans ce chapitre nous avons étudié une architecture simple d'un processeur. Le but est le savoir comment implémenter les instructions en utilisant la micro programmation.

III.2. Architecture du premier ordinateur

III.2.1.Micro-mem : micro-mémoire adressable par 64 adresses différentes, chacune désignant mot de 15 bits qui correspond à une *micro-opération* ou *MOP*.

III.2.2.Micro-PC : *compteur ordinal* pour le microprogramme. Il s'agit d'un registre compteur avec mise à zéro. Micro-PC a 6 bits de large pour un nombre total de 64 valeurs différentes. Micro-PC est piloté par les MOP 1et 2.

- Lorsque MOP1 vaut 1, micro-PC sera mis à zéro après le prochain front d'horloge.
- lorsque MOP2 vaut 1, une nouvelle valeur de micro-PC sera chargée à partir du décodeur d'instructions.

Micro-PC peut être mis à zéro manuellement parle bouton *reset*. Puisque l'entrée *clr* de micro-PC est prioritaire par rapport à l'entrée *ld*, celui-ci sera toujours mis à zéro par le bouton *reset*, indépendamment de la valeur de MOP2.

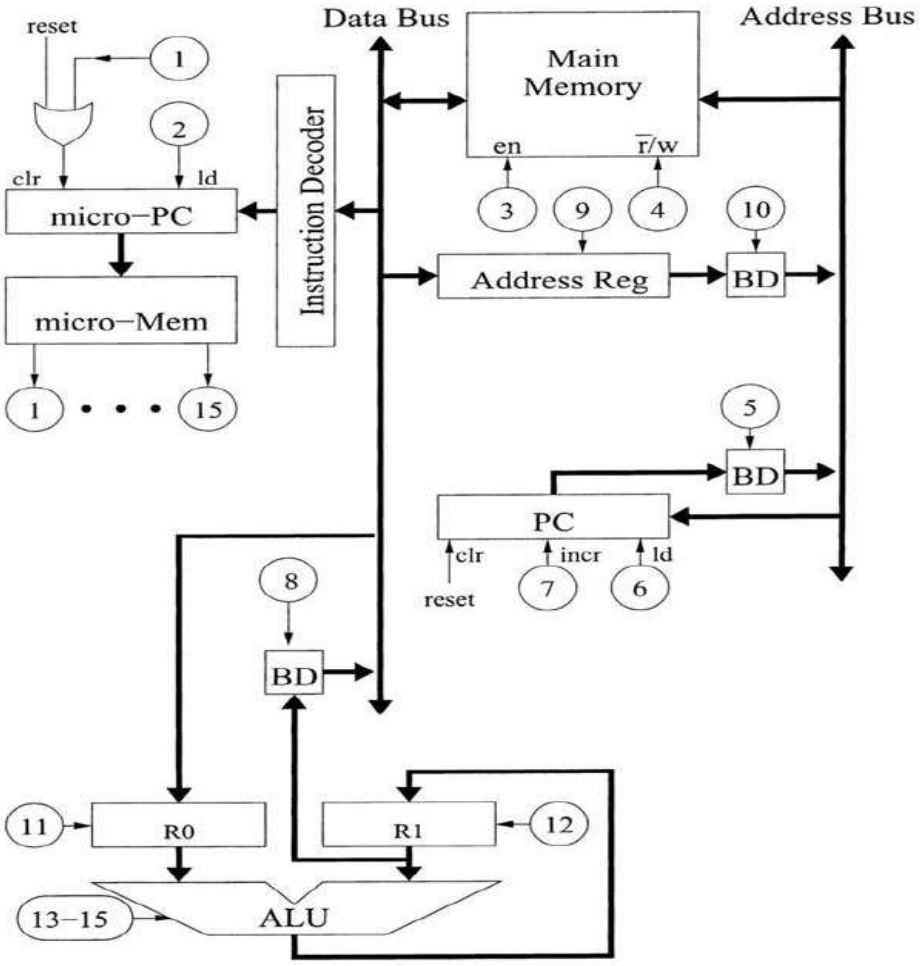


Figure III.1. Architecture du premier ordinateur

III.2.3.Décodeur d'instructions : Son rôle est de traduire le code d'une instruction en une adresse de la micro-mémoire, celle où commence le micro programme qui réalise l'instruction.

- L'entrée du décodeur d'instructions correspond aux 5 bits les moins significatifs du bus de données ce qui donne la possibilité de définir 32 instructions différentes.
- La sortie contient 6 bits ; ces derniers sont utilisés pour adresser la micro-mémoire.

***III.2.4. ALU* :** l'unité arithmétique et logique avec deux registres ordinaires R0 et R1.

- MOP11 est utilisée pour déterminer si R0 doit être chargé à partir du bus de données ou rester inchangé.

- MOP12 est utilisée pour déterminer si R1 doit être chargé à partir de la sortie de l'unité arithmétique et logique. L'unité arithmétique et logique utilise les MOP 13, 14et 15 pour sélectionner l'opération à effectuer.
- La sortie de R1 peut être connectée au bus de données grâce à MOP8 branchée sur le pilote de bus.

III.2.5. Mémoire centrale :

- L'entrée *enabel* est pilotée par MOP3.
- Et l'entrée *r/w* par MOP4. Lorsque MOP4 vaut 1, une écriture en mémoire est effectuée.

L'adresse à utiliser pour les opérations sur la mémoire (lecture ou écriture) est prise sur le bus d'adresse :

- En cas de lecture, les données lues dans la mémoire à l'adresse indiquée par le bus d'adresse seront disponibles sur le bus de données.
- Dans le cas d'une écriture, les données présentes sur le bus de données seront écrites en mémoire à l'adresse indiquée par le bus d'adresse.

III.2.6. Le registre d'adresse : Il est utilisé pour communiquer des données du bus de données au bus d'adresse. Ce registre est chargé lorsque MOP9 vaut 1. Le contenu du registre d'adresse peut être transmis sur le bus d'adresse à l'aide de MOP10.

III.2.7. PC: compteur ordinal qui est un registre avec mise à zéro et incrémentation

Explicite. Pour que le bouton *reset* fonctionne toujours, le signal *clr* est prioritaire.

- Si le signal *clr* vaut 0, mais que le signal *ld* (MOP6) vaut 1, alors PC est chargé à partir du bus d'adresse.
- Si *clr* et *ld* valent 0, mais que *incr* (MOP7) vaut 1, alors PC sera incrémenté après le front d'horloge suivant.
- Si *clr*, *ld* et *incr* valent 0, alors la valeur de PC reste intacte après le prochain front d'horloge. La valeur de PC peut être transmise au bus d'adresse par le biais de MOP5.

C'est de cette manière que les instructions à exécuter sont récupérées en mémoire centrale (Figure III.1).

III.3. Contenu de la micro mémoire

Nous allons maintenant étudier le contenu de la micro mémoire.

La largeur de cette mémoire est de 15 bits, un bit par MOP. Nous allons organiser le contenu de la micro mémoire de telle sorte que l'adresse numéro 0 contienne toujours le début d'un micro programme dont le but est de récupérer le code d'une instruction en mémoire centrale à l'adresse indiquée par PC. Il faut donc être sûr que PC contienne l'adresse de l'instruction suivante à exécuter lorsque ce micro programme démarre.

III.3.1. Chargement et décodage d'une instruction : Une instruction en mémoire centrale commence toujours par un *code* d'instruction sur un octet suivi éventuellement d'arguments. Le chargement et le décodage du code d'une instruction est une opération relativement simple qui ne nécessite qu'un seul cycle d'horloge. Il faut mettre le contenu de PC sur le bus d'adresse en utilisant MOP5. La mémoire centrale doit être lue à l'aide de MOP3. Le contenu de la mémoire centrale à l'adresse indiquée par PC sera automatiquement traduit par le décodeur d'instructions en une adresse qui indique le début du micro programme réalisant l'instruction. Cette adresse est chargée dans micro-PC à l'aide de MOP2. Finalement, par convention, nous allons incrémenter PC pour que l'instruction puisse charger ses *arguments* (qui suivent le code de l'instruction en mémoire centrale). Si l'instruction n'a pas d'argument, alors PC contiendra simplement l'adresse du code de l'instruction suivante. Nous allons veiller à ce que cette convention soit toujours respectée.

Nous avons maintenant le micro programme entier pour le chargement et le décodage d'une instruction :

adresse	contenu	opération
000000	011010100000000	fetch

Tableau III.1. Le micro programme pour chargement et décodage d'instruction

Comme on peut le constater, les MOP 2, 3, 5 et 7 valent 1 et toutes les autres valent 0.

III.3.2.Micro programme pour l'instruction NOP (No OPeration): l'instruction NOP qui ne fait absolument rien. Nous allons utiliser le code 0 pour cette instruction. Le micro programme de l'instruction est écrit appartir de la première adresse libre en micro-mémoire, à savoir 1 (car l'adresse 0 contient le micro programme pour le chargement d'une instruction. Le contenu du décodeur d'instructions à l'adresse 0 sera donc le suivant :

Adresse	Contenu	opération
00000	000001	NOP

Le micro programme de l'instruction NOP ne fait rien. Par conséquent, il faut Just faire en sort que l'adresse du micro-PC après son exécution vaille 0 pour que l'instruction suivante soit chargée. En fait le micro programme de chaque instruction sera terminé par une valeur de 1pour MOP1afin de garantir le bon chargement de l'instruction suivante. Avec le micro programme de l'instruction NOP, voici le contenu actuel de la micro mémoire :

adresse	contenu	opération
000000	011010100000000	Fetch
000001	100000000000000	NOP

Tableau III.2.le micro programme de l'instruction NOP

III.3.3.Micro programme pour l'instruction LDIMM (load immediate): C'est une instruction à un seul argument, d'une longueur totale de 2 octets en comptant celui du code de l'instruction. Le premier octet contient comme d'habitude le code de l'instruction et le second contient la valeur à charger dans R0. Une telle instruction est utile pour le chargement de *constantes* (à savoir des valeurs connues à la compilation) dans des registres.

Le code de l'instruction sera 1, premier code libre, et l'adresse en micro mémoire du début du micro programme de sa réalisation sera 2, la première adresse libre. Le décodeur d'instructions contient maintenant ceci :

adresse	contenu	opération
00000	000001	NOP
00001	000010	LDIMM

Pour implémenter l'instruction LDIMM, il faut adresser la mémoire en utilisant le contenu de PC qui contient déjà l'adresse de l'octet en mémoire centrale stockant l'argument, car le micro programme de chargement d'instruction a déjà incrémenté PC. Pour adresser la mémoire à partir de PC, nous utilisons MOP5. De plus, il nous fautMOP3 pour la lecture de la mémoire centrale, ainsi que MOP11 pour stocker la valeur lue dans R0. Par convention, nous allons aussi incrémenter PC pour qu'il contienne le code de l'instruction suivante. Tout ceci peut être fait en un seul cycle d'horloge. Afin de pouvoir charger l'instruction suivante, nous mettons MOP1 à 1 comme dans le cas de l'instruction NOP. Voici donc le contenu de la micro-mémoire :

adresse	contenu	opération
000000	011010100000000	Fetch
000001	100000000000000	NOP
000010	101010100010000	LDIMM

Tableau III.3. Le micro programme de l'instruction LDIMM

III.3.4.Micro programme pour l'instruction LD: Le chargement de constantes connues à la compilation est utile, mais il est aussi nécessaire de pouvoir charger le contenu de variables dont les valeurs ne sont pas connues à la compilation. De telles variables correspondent à des cellules en mémoire dont les adresses sont, quant à elles, connues à la compilation. Nous avons donc besoin d'une instruction de chargement dont l'argument indique une *adresse* en mémoire centrale et qui charge le contenu de cette adresse dans R0. Le premier code d'instruction disponible est 2 et la première adresse en micro-mémoire disponible est 3, ce qui donne le contenu du décodeur d'instructions suivant :

adresse	contenu	opération
00000	000001	NOP
00001	000010	LDIMM
00010	000011	LD

Le plan de l'implémentation de LD est le suivant : d'abord utiliser l'adresse contenue dans PC pour adresser la mémoire (MOP 5 et 3) et charger la valeur se trouvant à cette adresse de la mémoire dans le registre d'adresse à l'aide de MOP9. Simultanément, incrémenter PC(MOP7). Puis, pendant le cycle suivant, utiliser le contenu du registre d'adresse pour adresser la mémoire centrale (MOP 10 et 3) et charger le contenu dans R0 (MOP11). Comme on peut le constater, cette instruction nécessite deux cycles d'horloge, simplement parce que deux adresses différentes en mémoire centrale doivent être utilisées, et parce que nous n'avons qu'un seul bus de données et un seul bus d'adresse. Par convention, nous incrémentons PC le plus vite possible, à savoir après le premier cycle d'horloge. Voici le contenu actuel de la micro-mémoire :

adresse	contenu	opération
000000	011010100000000	Fetch
000001	100000000000000	NOP
000010	101010100010000	LDIMM
000011	001010101000000	LD
000100	101000000110000	LD

Tableau III.4. Le micro programme de l'instruction LD

III.3.4.Micro programme pour l'instruction ST: De la même façon qu'il est nécessaire de pouvoir charger la valeur d'une variable, il faut aussi pouvoir stocker une nouvelle valeur dans une variable. Pour cela, il nous faut une instruction capable de prendre le contenu d'un registre (ici R1) et de le stocker dans la mémoire centrale à l'adresse indiquée par l'argument de l'instruction. Nous appelons cette instruction ST (pour store).

Comme toujours, nous prenons le premier code d'instruction disponible et la première adresse libre de la micro mémoire, ce qui nous donne le contenu du décodeur d'instructions:

adresse	contenu	opération
00000	000001	NOP
00001	000010	LDIMM
00010	000011	LD
00011	000101	ST

Pour implémenter l'instruction ST, il faut prendre des précautions qui jusque là n'étaient pas nécessaires. En effet, la mémoire n'est pas synchrone. Il faut donc s'assurer que les valeurs sur le bus d'adresse et le bus de données sont stables avant de commencer l'écriture en mémoire. La seule manière d'en être sûr est de séparer la production de l'adresse et des données d'une part et l'écriture d'autre part en deux cycles d'horloge différents, et de maintenir l'adresse et la donnée un cycle de plus après avoir activé l'écriture.

Voici le plan de réalisation de l'instruction ST : pendant le premier cycle d'horloge, charger l'argument de l'instruction (indiquant l'adresse en mémoire à utiliser pour le stockage) dans le registre d'adresse, comme pour l'instruction LD. Pendant le deuxième cycle, activer les MOP 8 et 10 pour que les bus soient stables. Pendant le cycle suivant, ajouter le signal d'écriture (MOP4) ainsi que le signal enable (MOP3). L'écriture en mémoire sera alors effectuée. Dans le cycle final, supprimer les MOP 3 et 4 tout en maintenant les MOP 8 et 10 .Voici le contenu actuel de la micro-mémoire :

adresse	contenu	opération
000000	011010100000000	Fetch
000001	100000000000000	NOP
000010	101010100010000	LDIMM
000011	001010101000000	LD
000100	101000000110000	LD
000101	001010101000000	ST
000110	000000010100000	ST
000111	001100010100000	ST
001000	100000010100000	ST

Tableau III.5. Le micro programme de l'instruction ST

III.4. Conclusion :

Nous avons vu une architecture simple de processeur suivi par le micro programme de ce processeur. Nous allons essayer de réaliser notre processeur MIPS par la même méthode de ce processeur.

chapitre IV:Conception et simulation de MIPS R3000 par VHDL

IV.1. Introduction

Dans ce chapitre nous allons aborder en détails les blocs qui composent le processeur MIPS R3000, Chaque bloc va être présenté et simulé séparément et enfin le fonctionnement des différents blocs sera réalisé à l'aide d'un séquenceur (micro mémoire) (Figure IV.1).

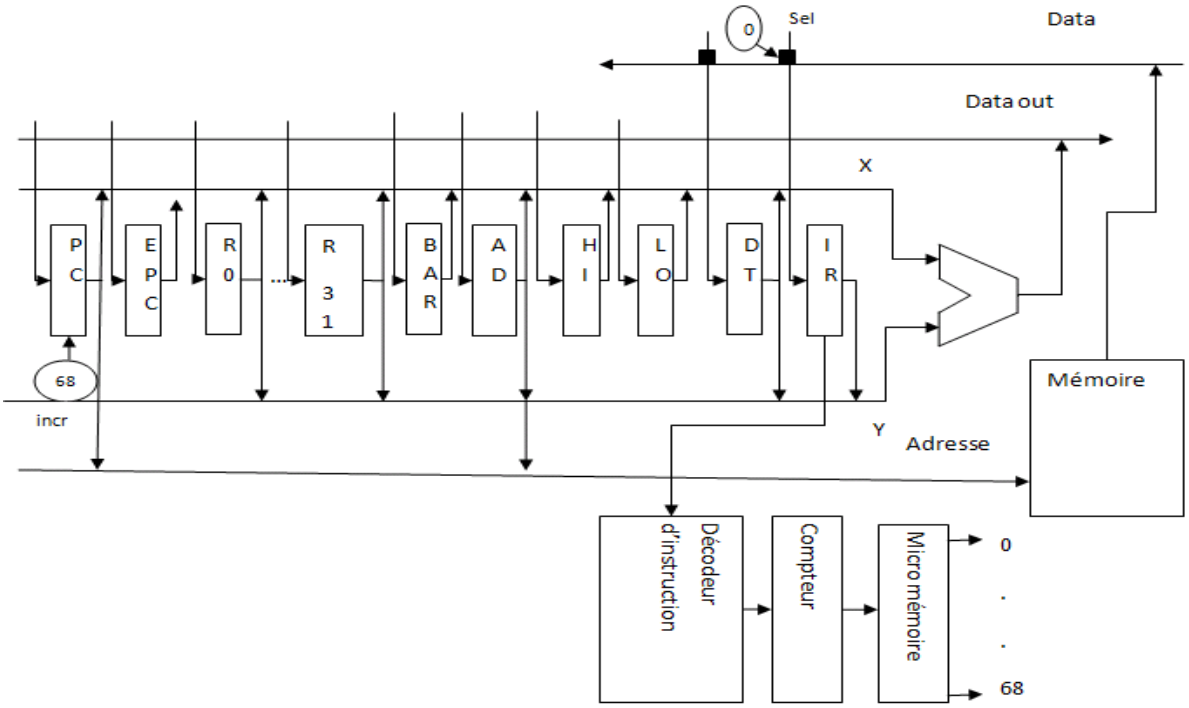


Figure IV.1. Schéma de MIPS R3000

IV.2. Registre :

Ce sont utilisé pour le stockage et le transfert des donner. Ils présentent les registres protégés et non protégés du processeur MIPS R3000.

IV.2.1. Description : Le rôle de registre c'est le sauvegarde des donne ces composants contiennent trois signaux. Ce sont clk, ld, e et s.

- **clk:** représente l'horloge, il permet d'activer les fonctionnements du registre.
- **e:** C'est un signale d'entrée de 32 bits, il permet d'alimenter le registre par de nouvelles données.

- **s:** C'est un signal de sortie de 32 bits, il permet de sortir les données stockée dans le registre.
- **ld:** représente le signale de chargement, l'activation de ce signal force le registre de stocker la données qui se trouvent dans les signale d'entre (Figure. IV.2).

Lorsque ld vaut 0, les sorties ne sont pas affectées par un front d'horloge. Par contre, lorsque ld vaut 1, les entrées (e) sont copiées dans le registre après le prochaine front d'horloge (Figure IV.3).

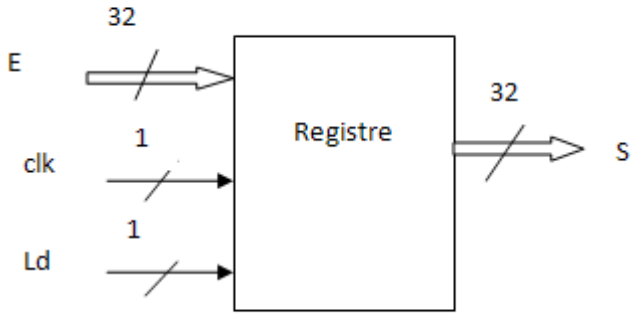


Figure. IV.2. Registre

IV.2.2. Simulation

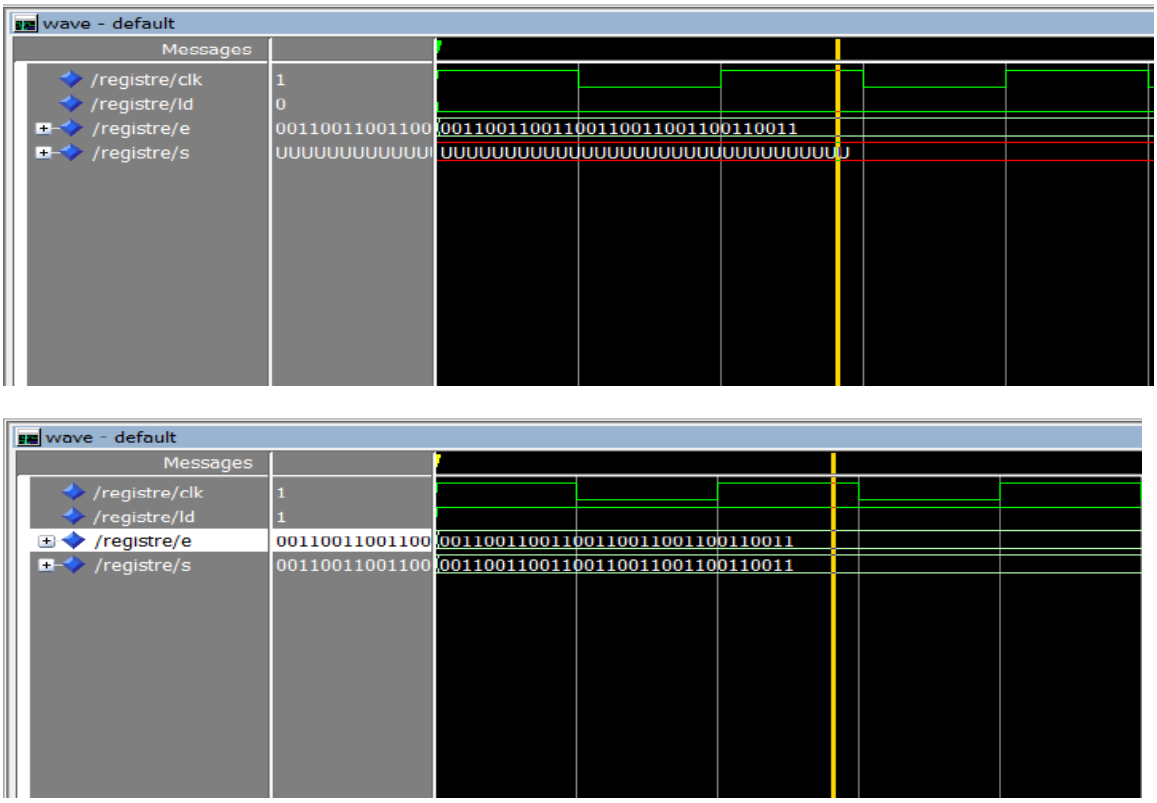


Figure IV.3. Simulation du registre

IV.3. Compteur

IV.3.1. Description : Le compteur contiens quatre signaux d'entrées *rst* , *clk* ,load,*e* et un signal de sortie *code*.

- **rst :** Ce signale permet le remise de la valeur contenu dan le compteur à zéro.
- **clk:** représente l'horloge, il permet d'activer les fonctionnements du compteur.
- **load:** représente le signale de chargement, l'activation de ce signal force le compteur de stocker la données qui se trouvent dans les signale d'entre.
- **e :** C'est un signale d'entrée, il permet d'alimenter le compteur par de nouvelles données.
- **code:** C'est un signal de sortie, il permet de sortir les données stockées dans le compteur (Figure IV.4).

Si rst prend la valeur 1 le signal code est remis à 0, lorsque on donne la valeur 1 au signal load le signal code prend la valeur de signal e, lorsque le signal rst est à 0 et load est à 0 le signal de sortie code va avoir code+1 après chaque front montant du signal clk (Figure IV.5).

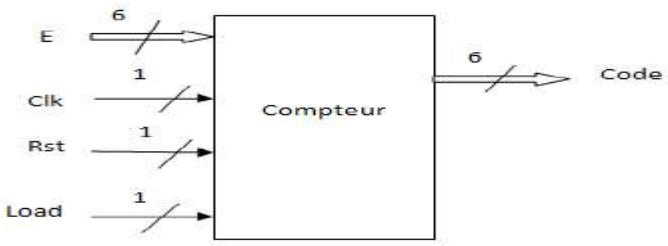


Figure IV.4. Compteur

IV.3.2.Simulation

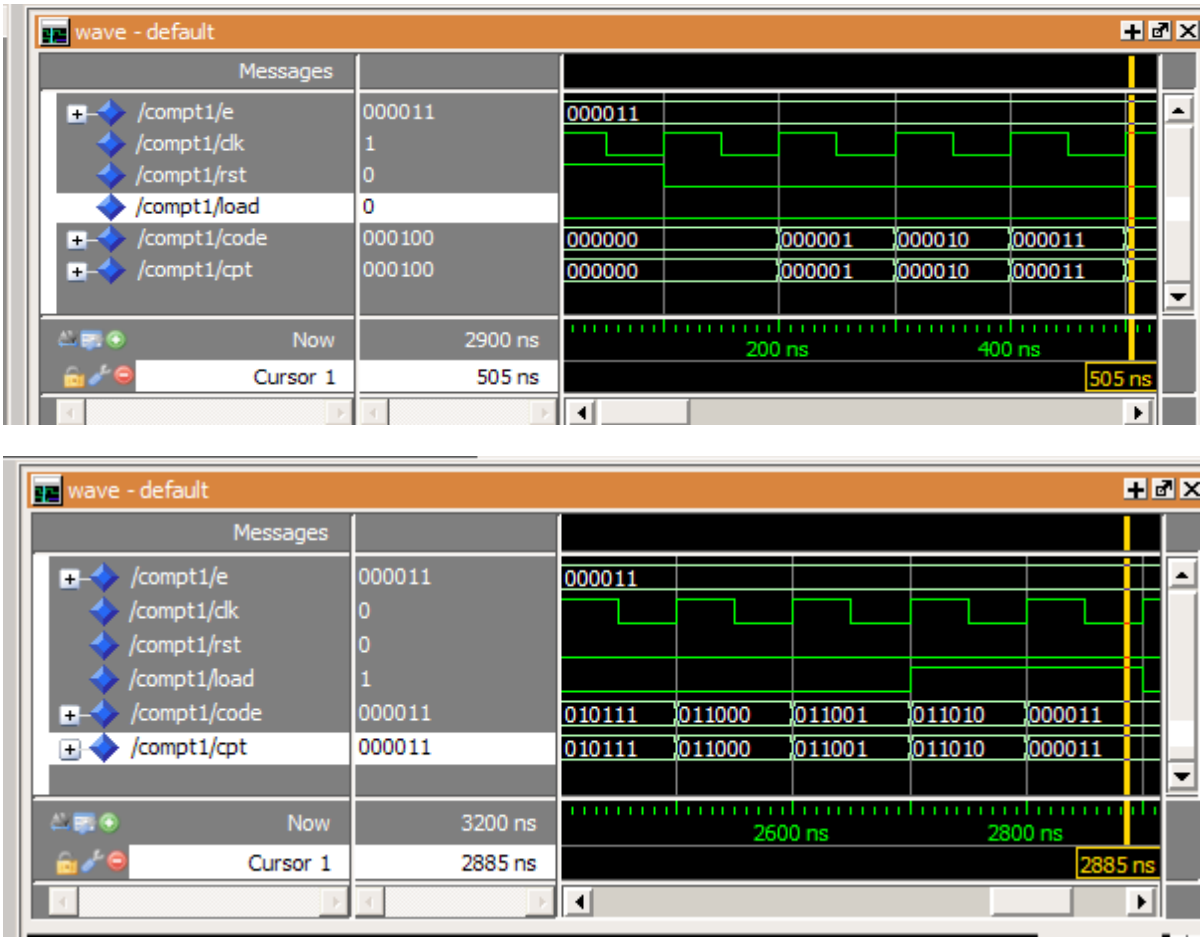


Figure IV.5. Simulation de compteur

IV.4. Compteur ordinal(PC)

IV.4.1. Description: Le compteur ordinal (PC) contiens cinqe signaux d'entrées (*rst* , *clk* ,load,incr,*e*) et un signal de sortie (so):

- **rst:** Ce signale permet le remise de la valeur contenu dans le compteur ordinal à zéro.
- **clk:** représente l'horloge, il permet d'activer les fonctionnements du compteur ordinal.
- **load:** représente le signale de chargement, l'activation de ce signal force le compteur ordinal de stocker la données qui se trouvent dans le signale d'entrée.
- **incr:** ce signal permet au compteur ordinal à s'incrémenter par 4.
- **so:** C'est un signal de sortie, il permet de sortir les données stockées dans le compteur ordinal (Figure IV.6).

Si rst prend la valeur 1 le signal so est remis à 0, lorsque on donne la valeur 1 au signal load le signal so prend la valeur de signal e, lorsque Le signal rst=0 et load = 0 et le signal incr=1 le signal de sortie so prend la valeur so+4 et chaque front montant incrément le signal so (Figure IV.7).

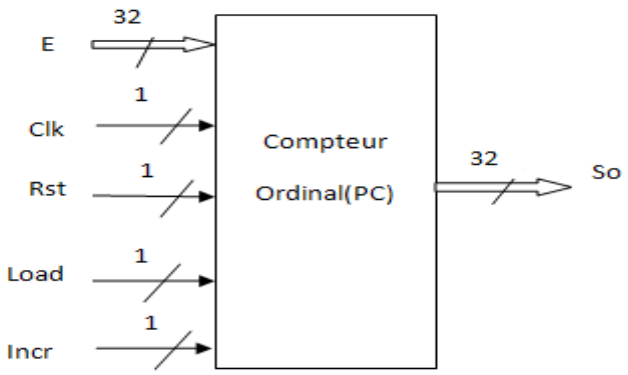


Figure IV.6. Compteur ordinal

IV.4.2. Simulation

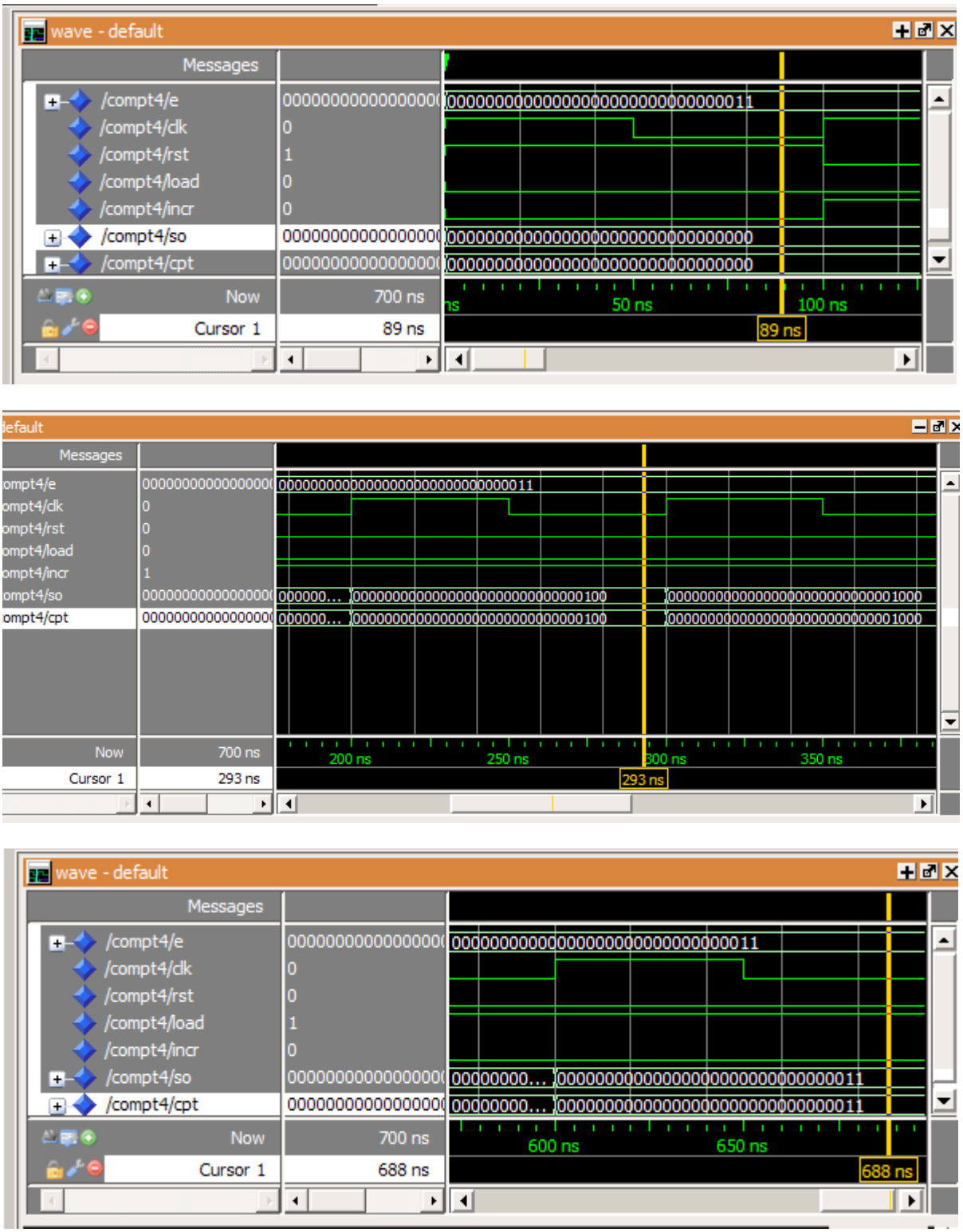


Figure IV.7. Simulation de compteur ordinal

IV.5. Décodeur d'instruction

Son but est de traduire le code d'une instruction en l'adresse de la micro-mémoire.

IV.5.1.Description : Le décodeur d'instruction prend en entré la sortie du registre RI registre d'instruction RI qui présenté par un signal d'entrée. Les sorties de ce composant sont:

- RS :c'est un signal de sortie de 5 bits, il est utilisé pour récupérer la partie représentant le numéro de registre RS dans RI, ces valeur ce trouvant dans les cases d'indices de 20 à 24 dans RI.
- RT : c'est un signal de sortie de 5 bits, il est utilisé pour récupérer la partie représentant le numéro de registre RT dans RI, ces valeur ce trouvant dans les cases d'indices de 15 à 19 dans RI.
- RD : c'est un signal de sortie de 5 bits il est utilisé pour récupérer la partie représentant le numéro de registre RD dans RI, ces valeur ce trouvant dans les cases d'indices de 10 à 14 dans RI.
- S(OPCOD) : il représente la partie code opération dans RI, ces valeur ce trouvant dans les cases d'indices de 26 à 31 dans RI. Le tableau 1 illustre le code de l'instruction ADD. Pour le reste des instructions voire l'annexe A.
- SH : c'est un signal de sortie de 5 bits, il représente la valeur SH, ces valeur se trouvant dans les cases d'indices de 6 à 10 dans RI (Figure IV.9).

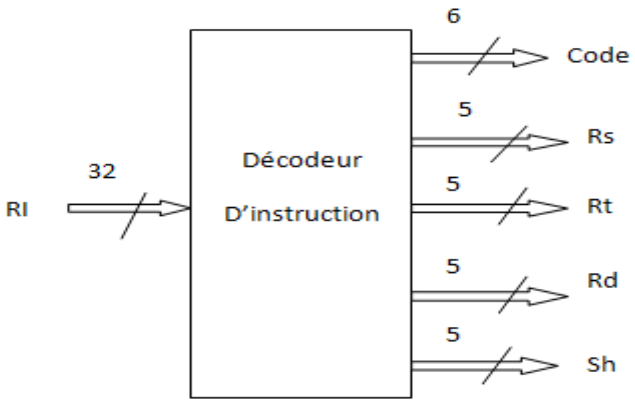


Figure IV.8. Décodeur d'instruction

Instructions arithmétique/ logique			
assembleur	Opération	Format	code
Addi RT, RS, I	Addition immediate avec detection over flow Rt<- Rs +I	I	101011

Tableau .IV.1: Codage de l'instruction ADDI.

Pour savoir les déférentes valeurs possibles du signal code voir l'annexe A.

IV.5.2.Simulation

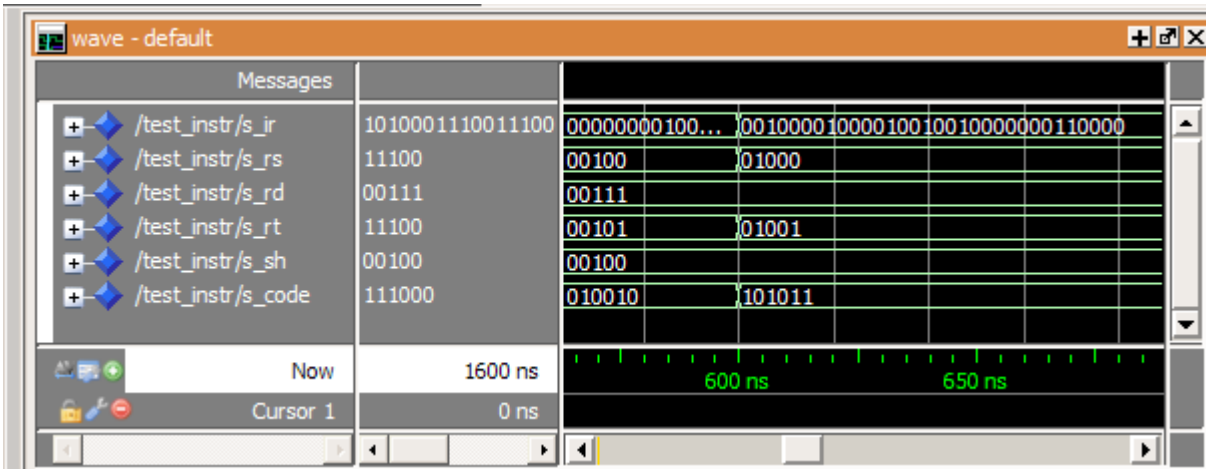


Figure VI.9. Simulation de décodeur d'instruction

IV.6.Unité arithmétique logique(UAL)

Une *unité arithmétique et logique* (UAL) d'un processeur est un circuit capable d'effectuer les opérations de base comme l'addition, la soustraction, le *ou* et le *et* bit-à-bit,… etc.

IV.6.1.Description : Une boite à opération pour les opérations arithmétique, logique, comparaison, et décalage multi bits. Avec deux signaux d'entrée a et b de taille 32 bits, un signal d'entrée de sélection « selll » de taille 6 bits, un signal de sortie « acc » de 32 bits, une bit de sortie « z » et une bit de sortie « c »(Figure IV.10).

- A et B: pour enter les opérandes sur les quelle on va exécuter l'opération.

On appliquer les opérations sur les signaux a et b et stocker le résultat dans le signale ACC. En cas de détection d'une over flow le bit c prendre la valeur 1. Cependant, en cas d'opération de division et l'opérande diviseur = 0 le bit z prendre la valeur 1 (Figure IV.11).

L'UAL contient un registre interne Rn de 32 bits. Ce registre est utilisé dans les opérations de la devisions et la multiplication. Dans le cas de la multiplication le signal ACC prend les 32 bit de poids fort du résultat et le Rn prend les 32 bites de poids faible du résultat. Dans le cas de la division, le signal ACC contiendra le résultat de la division, alors que Rn contiendra

le reste de la division. Pour mettre la valeur de registre Rn dans le signal ACC on doit mettre le code '111011' dans le signal selll.

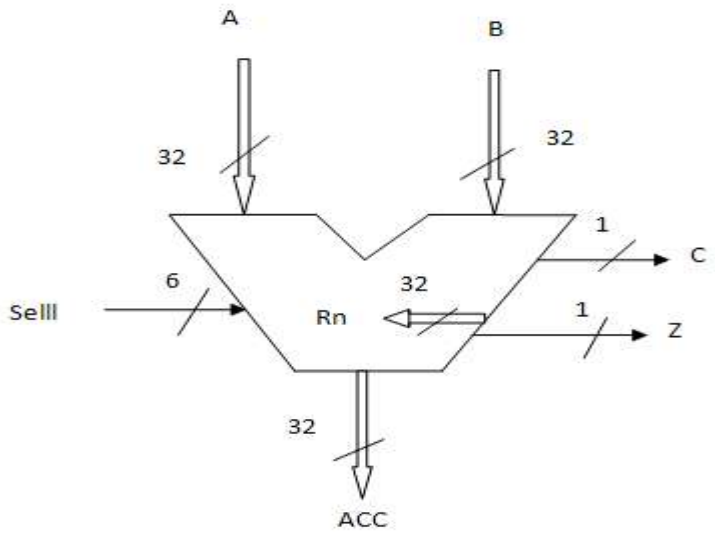


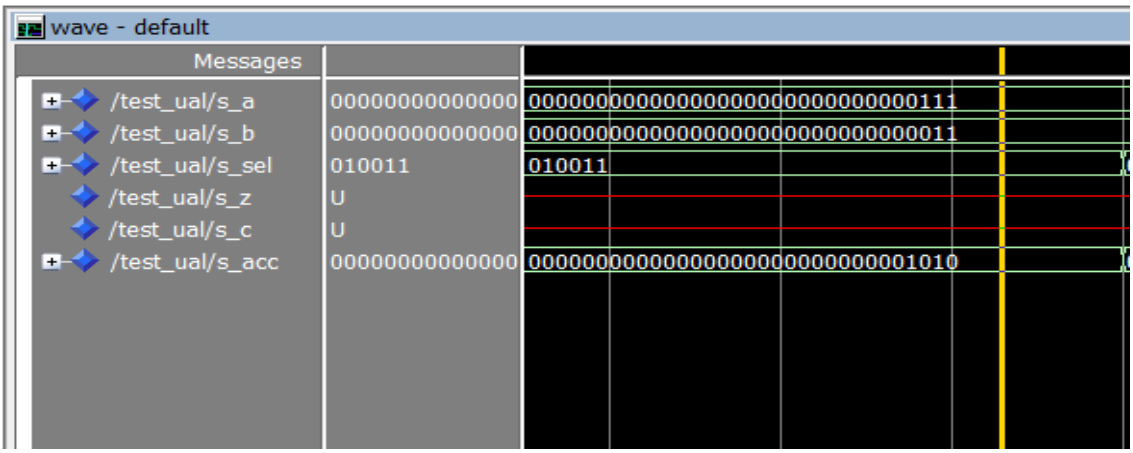
Figure IV.10. Unité arithmitique logique

Pour savoir les déférentes valeurs possibles du signal selll voir l'annexe B.

Ci-dessous la simulation du fonctionnement de l'UAL pour quelques opérations.

IV.6.2. Simulation

- A. Opérations addu



- B. Operations Add(detection over flow)

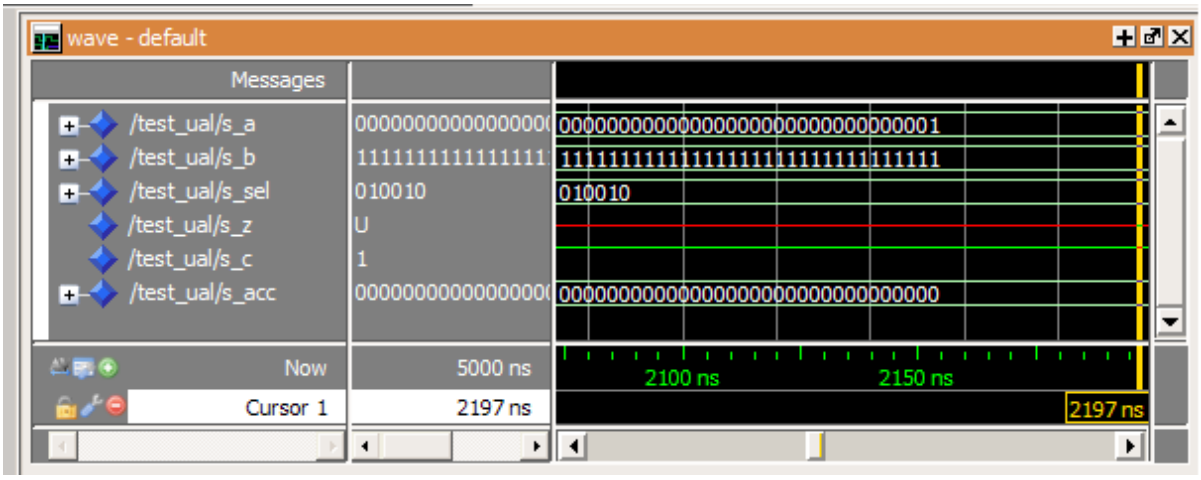


Figure IV.11. Simulation de l'unité arithmétique logique

IV.7. Opérateur trois états

IV.7.1. Description : Opérateur 3états avec un signal d'entrée « e » de 32 bits et un signal de sortie « s » de 32 bits, et une entrée de sélection « sel » (Figure IV.12). Lorsque sel =1 alors s prend la valeur de e (Figure VI.13).

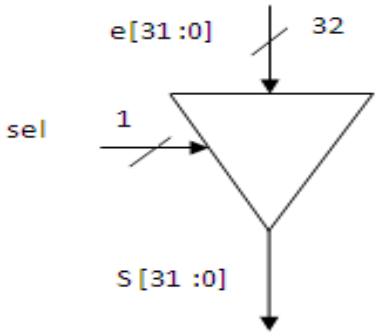


Figure IV.12. Opérateur trois états

IV.7.2.Simulation

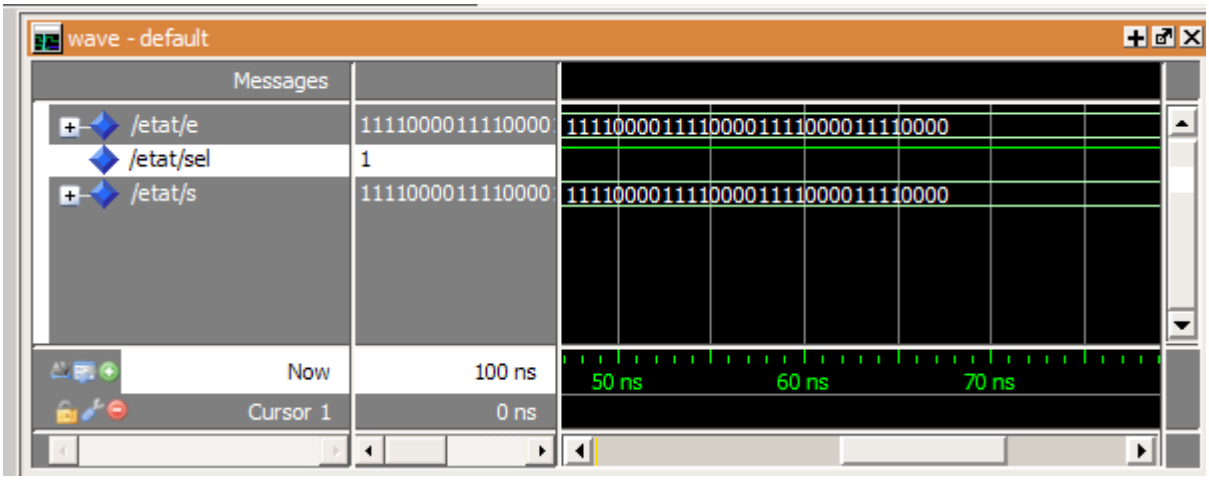


Figure VI.13. Simulation d'opérateur trois états

IV.8. Micro mémoire

La micro-mémoire sert à stocker les micros instruction pour l'exécution de chaque Instruction. La micro-mémoire d'un processeur est un circuit numérique réalisé parfois sous forme de mémoire à lecture, parfois sous forme de circuit combinatoire. Il fut un temps où il était relativement fréquent de vouloir modifier le contenu de la micro-mémoire afin de créer des instructions spécialisées pour des applications particulières. La micro-mémoire était alors similaire à une mémoire à lecture et écriture ordinaire.

IV.8.1.Description : Le micro mémoire c'est de contiens deux signaux d'entrées (*init* , *adr*) et un signal de sortie (micro_instr) (Figure IV.14), si init prend la valeur 0 le signal micro_instr est remis à 0, lorsque on donne la valeur 1 au signal init le signal micro_instr prend la valeur du mot mémoire adressé le signal adr (Figure IV.15).

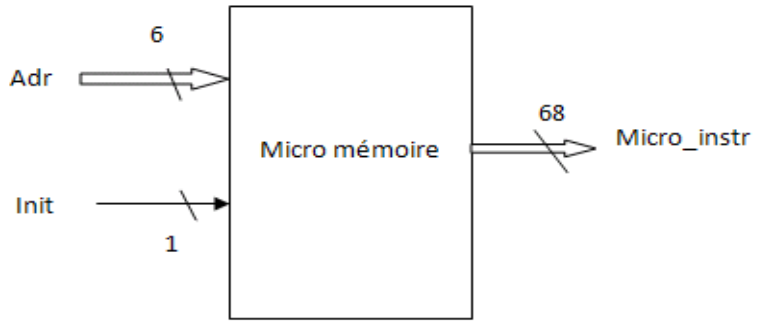


Figure IV.14. Micro mémoire

IV.8.2. Simulation

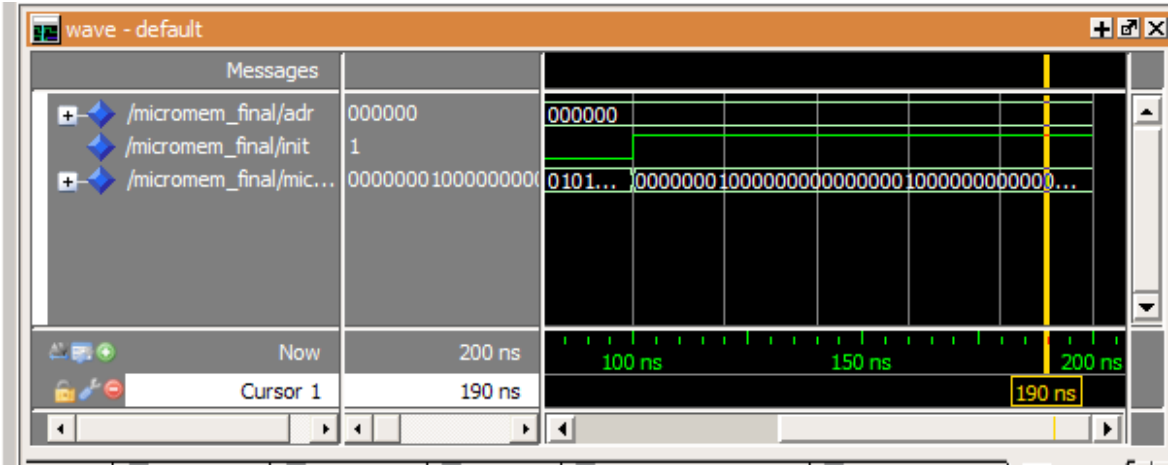


Figure IV.15. Simulation de micro mémoire

IV.9. Mémoire

Une mémoire a entrée appelée *entrée d'adresse* qui est utilisée pour sélectionner exactement 1 parmi plusieurs *mots*. De plus, elle a un fils *bidirectionnel* appelé *ligne de donnée*. Cette ligne de données servent à la fois d'entrée pour stocker des informations dans un mot sélectionné par les entrées d'adresse, et de sortie pour récupérer un mot préalablement

stocké. Finalement, la mémoire a une entrée appelée *r/w* qui détermine la direction des lignes de données.

IV.9.1.Description : La mémoire contiens deux signaux d'entrées (*adresse*, *rw*) et un signal d'entrée sortie (data) (Figure IV.16). La taille du signal adresse est 7 ce qui signifier que la taille de cette mémoire est 128 octets. Dans le cas (rw=001), les 4 octet de bus data sont écrit en mémoire a une adresse alignée sur les mots. Dans le cas (rw=010), les 2 octet de poids faibles du bus data sont écrit à une adresse alignée sur les demi mots. Dans le cas (rw=011), l'octet de poids faible du bus data est écrit à l'adresse (adresse) (Figure IV.17).

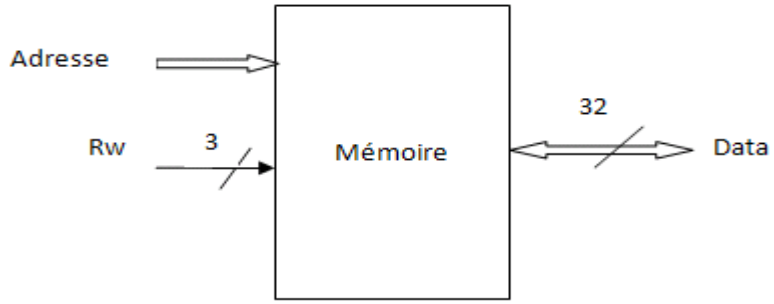
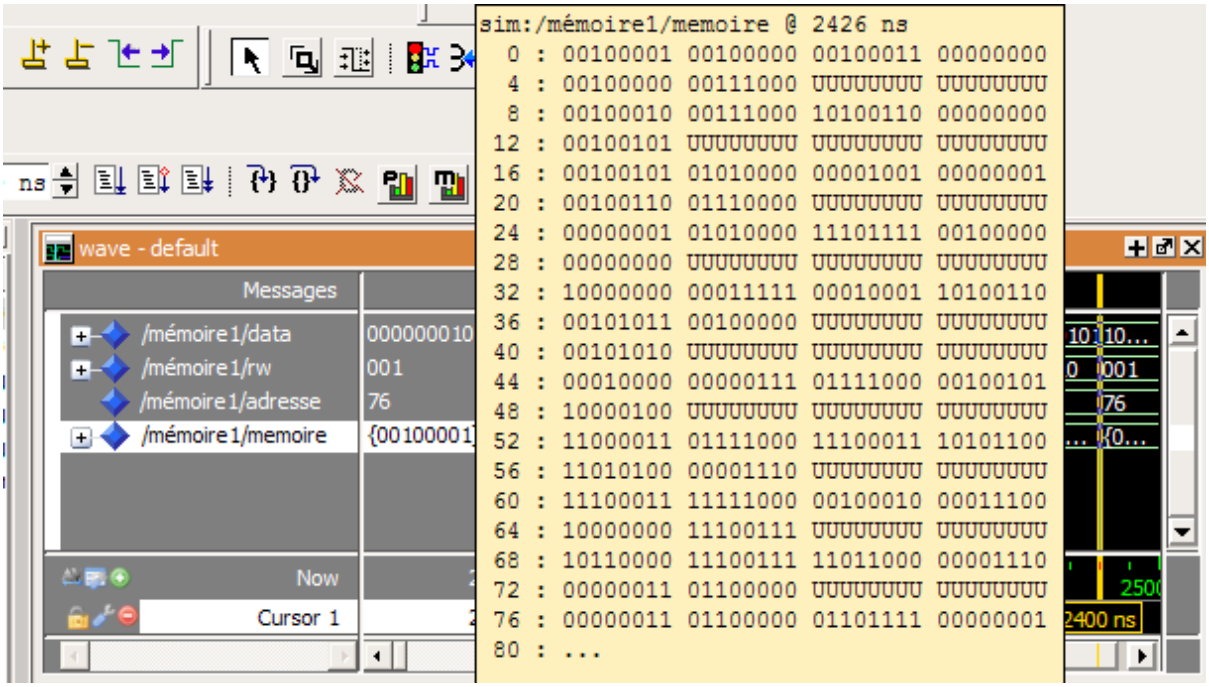


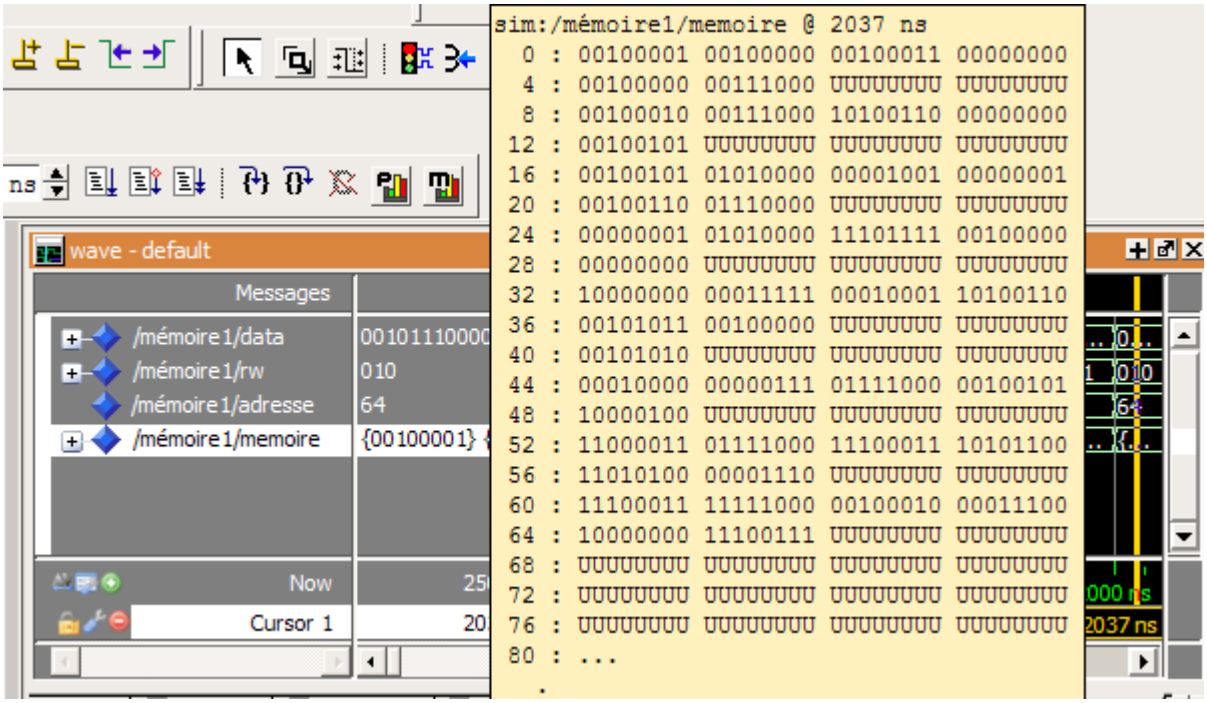
Figure IV.16. Mémoire

IV.9.2.Simulation

A. Ecriture d'un mot à l'adresse 76



B. Ecriture demi mot à l'adresse 64



C. Ecriture d'un octet à l'adresse 48

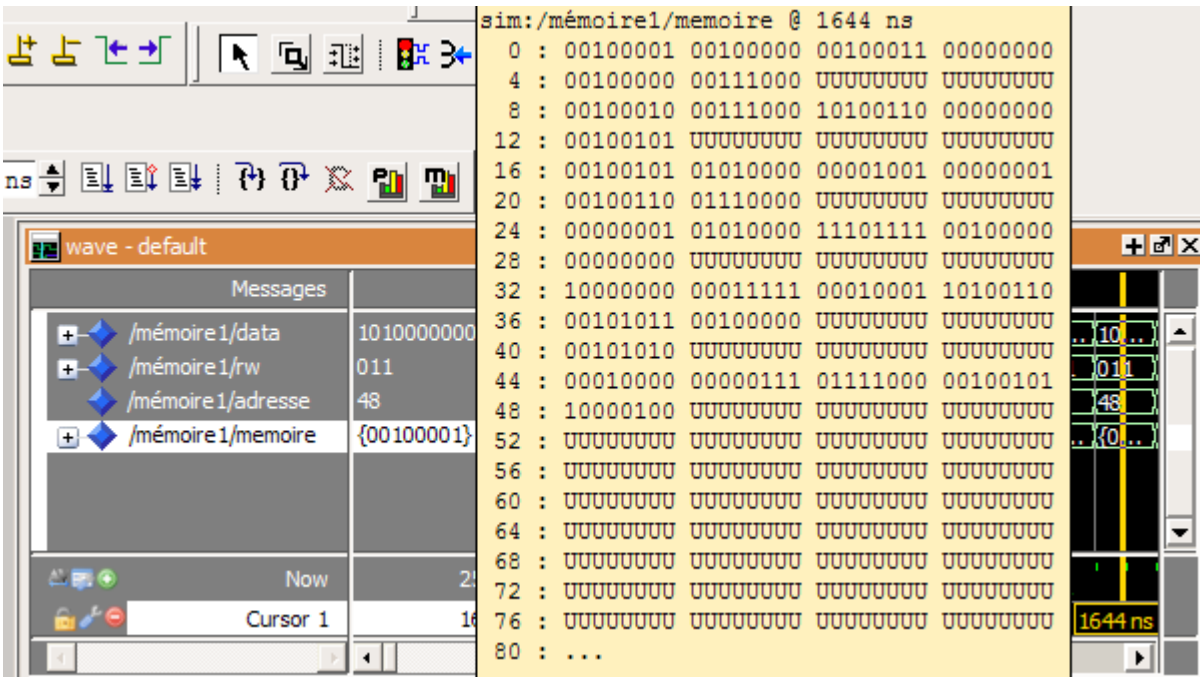


Figure IV.17. Simulation de la mémoire

IV.10. Mux1 :

IV.10.1.Description : C'est un circuit qui contient de :

- 40 signaux d'entrées chacun de taille 32 bits (r0, r1,…r31 de part ; et ir, dt, lo, hi, ad, bar, epc, pc d'autre part).
- **Sel1, sel2:** Deux signaux d'entrées de taille 1bits.
- **Sel_rs,sel_rt:** Deux signaux d'entrées de taille 5bits.
- **Sel_regx, sel_regy:** Deux signaux d'entrées de taille 3 bits.
- **Sel_adr :** Un signal d'entrée de taille 2 bits.
- **x,y, adr:** Trois signaux de sortie de taille 32 bits (Figure IV.18).

-Lorsque le signal sel1 prend la valeur 0 nous avons testé le signal sel_rs :

- Lorsque le signal sel_rs = 00000 alors le signal de sortie x prend la valeur de signal « r0 ».
- Lorsque le signal sel_rs = 00001 alors le signal de sortie x prend la valeur de signal « r1 ».

Et ainsi de suite jusqu'a :

- Lorsque le signal sel_rs = 11111 alors le signal de sortie x prend la valeur de signal « r31 ».

-Lorsque le signal sel1 prend la valeur 1 nous avons testé le signal sel_regx :

- Lorsque le signal sel_regx = 001 alors le signal de sortie x prend la valeur de signal « dt ».
- Lorsque le signal sel_regx = 010 alors le signal de sortie x prend la valeur de signal «Lo ».
- Lorsque le signal sel_regx = 011 alors le signal de sortie x prend la valeur de signal« hi ».
- Lorsque le signal sel_regx prend la valeur « 100 »alors le signal de sortie x prend la valeur de signal «ad ».
- Lorsque le signal sel_regx prend la valeur « 101 »alors le signal de sortie x prend la valeur de signal «bar ».
- Lorsque le signal sel_regx =110 alors le signal de sortie x prend la valeur de signal «epc ».

- Lorsque le signal sel_regx = 111 alors le signal de sortie x pend la valeur de signal «pc ».

-Lorsque le signal sel2 prend la valeur 0 nous avons testé le signal sel_rt :

- Lorsque le signal sel_rt = 00000 alors le signal de sortie y prend la valeur de signal « r0 ».
- Lorsque le signal sel_rt = 00001 alors le signal de sortie y prend la valeur de signal « r1 ».

Et ainsi de suite jusqu'a :

- Lorsque le signal sel_rt = 11111 alors le signal de sortie y prend la valeur de signal « r31 ».

-Lorsque le signal sel2 prend la valeur 1 nous avons testé le signal sel_regy :

- Lorsque le signal sel_regy = 001 alors le signal de sortie y prend la valeur de signal « ir ».
- Lorsque le signal sel_regy = 010 alors le signal de sortie y prend la valeur de signal «dt ».
- Lorsque le signal sel_regy =011alors le signal de sortie y prend la valeur de signal « ad ».

-lorsqe le signal sel_adr prendre la valeur « 01 »alors le signal de sortie adr prend la valeur de signal « ad ».

- lorsqe le signal sel_adr prendre la valeur « 10 »alors le signal de sortie adr prend la valeur de signal « pc » (Figure IV.19).

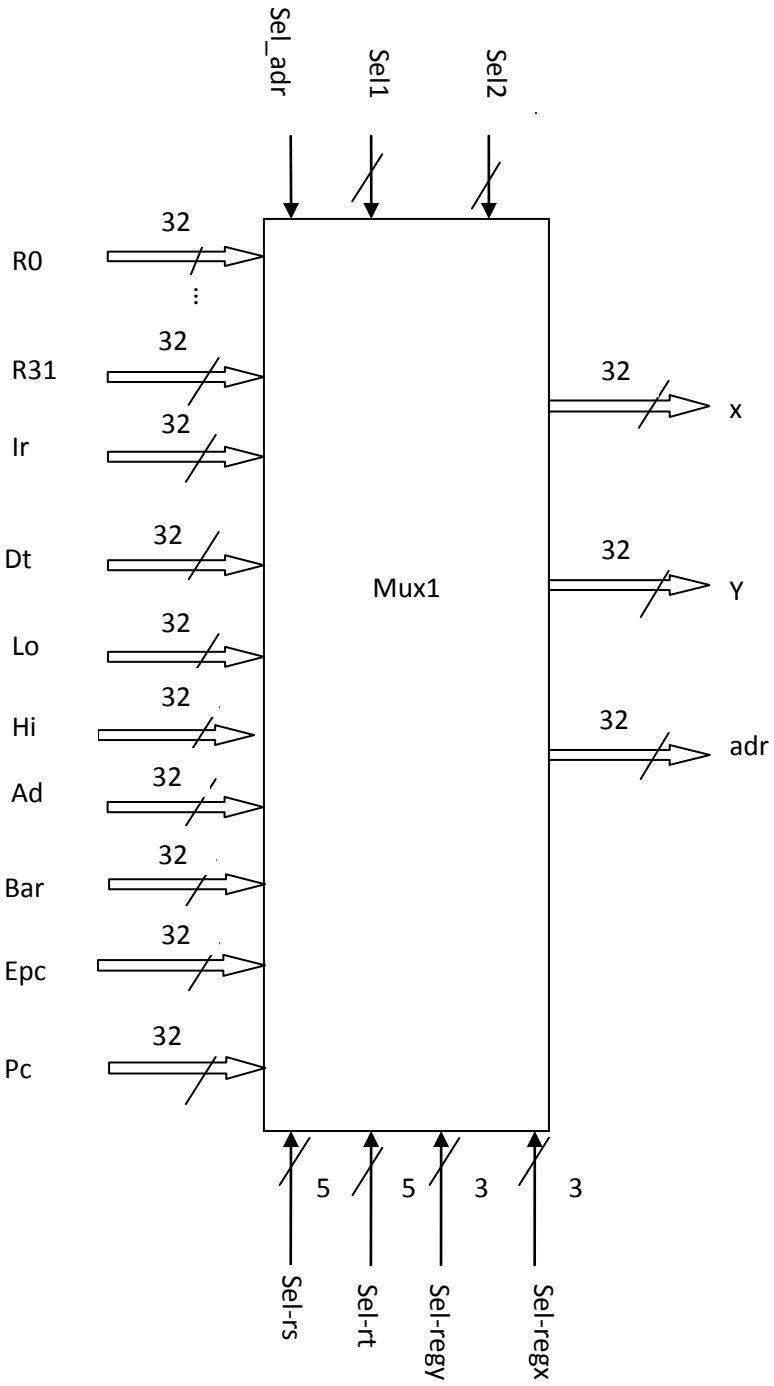


Figure IV.18. Mux1 (multiplexeur)

IV.10.2. Simulation :

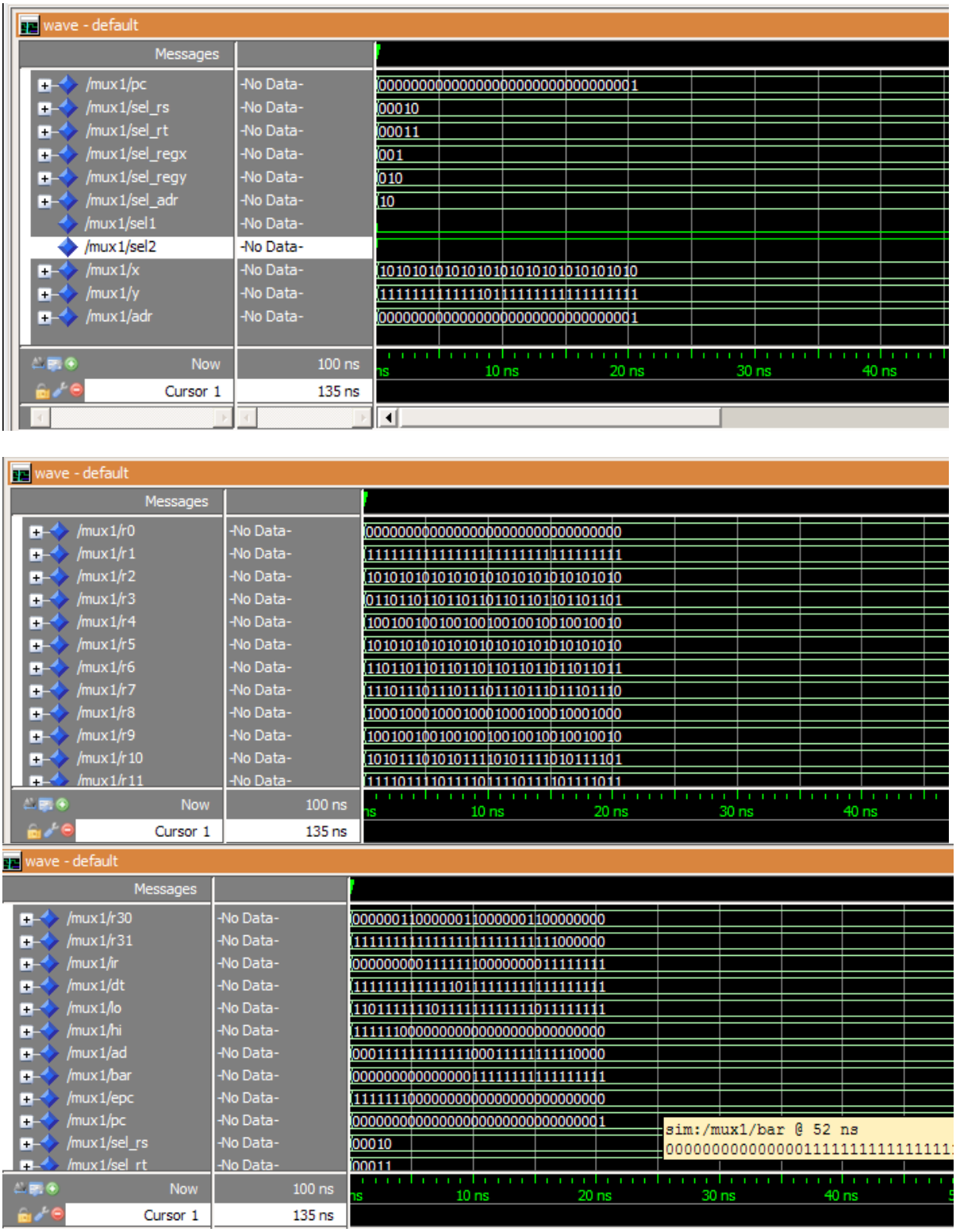


Figure IV.19. Simulation de mux1

IV.11. Mux2 :

IV.11.1. Description : C'est un circuit qui contient de :

- **Résultat :** Un signal d'entrée de taille 32 bits.
- **Sel :** un signal d'entrée de taille 1bit.
- **Sel_rd :** un signal d'entrée de taille 5bits.
- **Sel_reg :** un signal d'entrée de taille 3bits.
- 38 signal de sortie la taille de chacun est 32 bits (r0, r1,…, r31 de part ; et lo, hi, ad, bar, epc, pc d'autre part) (Figure IV.20).

-lorsque le signal sel prendre la valeur 0 alors nous avons testé la valeur de signal sel_rd :

- Lorsque le signal sel _rd = 00000 alors le signal de sortie « r0 » prend la valeur de signal résultat.
- Lorsque le signal sel _rd = 00001 alors le signal de sortie « r1 » prend la valeur de signal résultat.

Et ainsi de suite jusqu'a :

- Lorsque le signal sel _rd = 11111 alors le signal de sortie « r31 » prend la valeur de signal résultat.

-lorsque le signal sel prendre la valeur 1 alors nous avons testé la valeur de signal sel_reg:

- Lorsque le signal sel _reg= 001 alors le signal de sortie «lo » prend la valeur de signal résultat.
- Lorsque le signal sel _reg= 010 alors le signal de sortie «hi » prend la valeur de signal résultat.
- Lorsque le signal sel _reg= 011 alors le signal de sortie «ad» prend la valeur de signal résultat.
- Lorsque le signal sel _reg=100 alors le signal de sortie «bar» prend la valeur de signal résultat.
- Lorsque le signal sel _reg= 101 alors le signal de sortie «epc» prend la valeur de signal résultat.
- Lorsque le signal sel _reg=110 alors le signal de sortie «pc» prend la valeur de signal résultat (Figure IV.21).

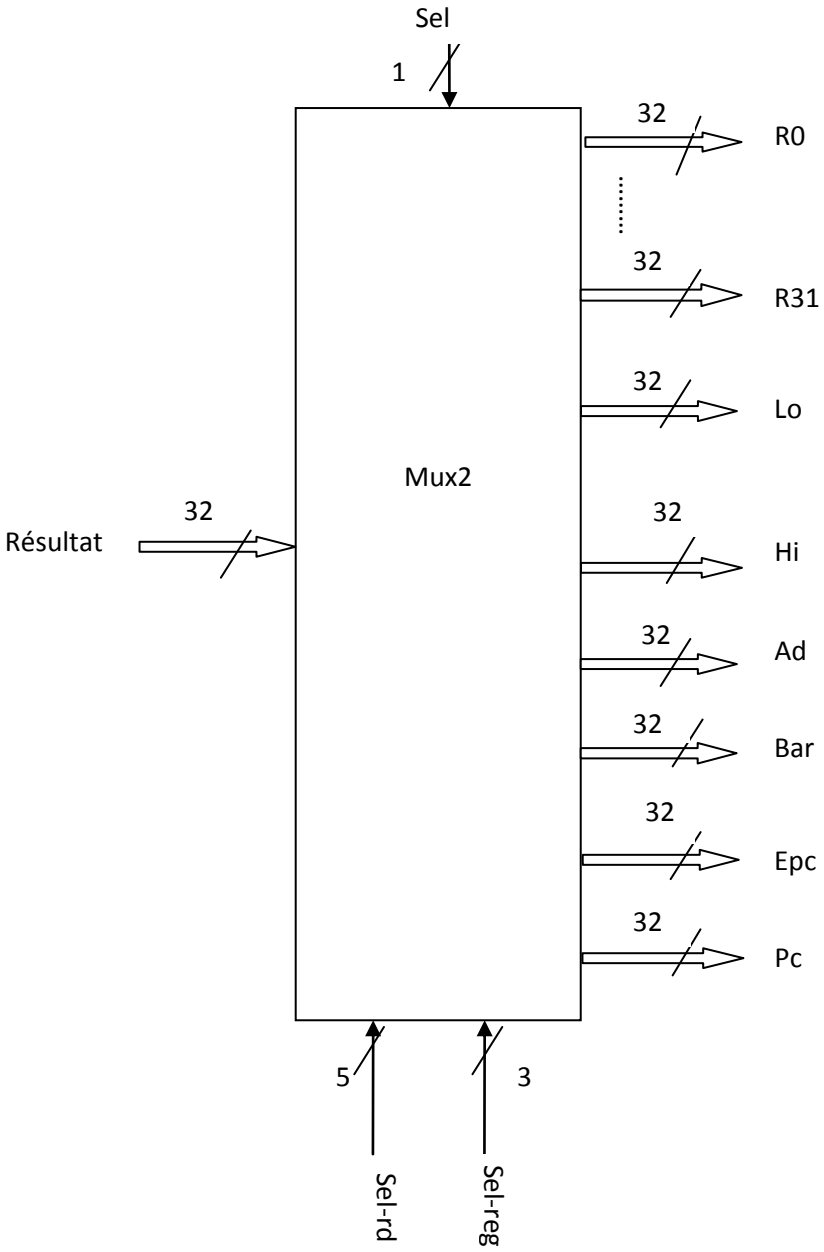


Figure IV.20. Mux2 (démultiplexeur)

IV.11.2. Simulation :

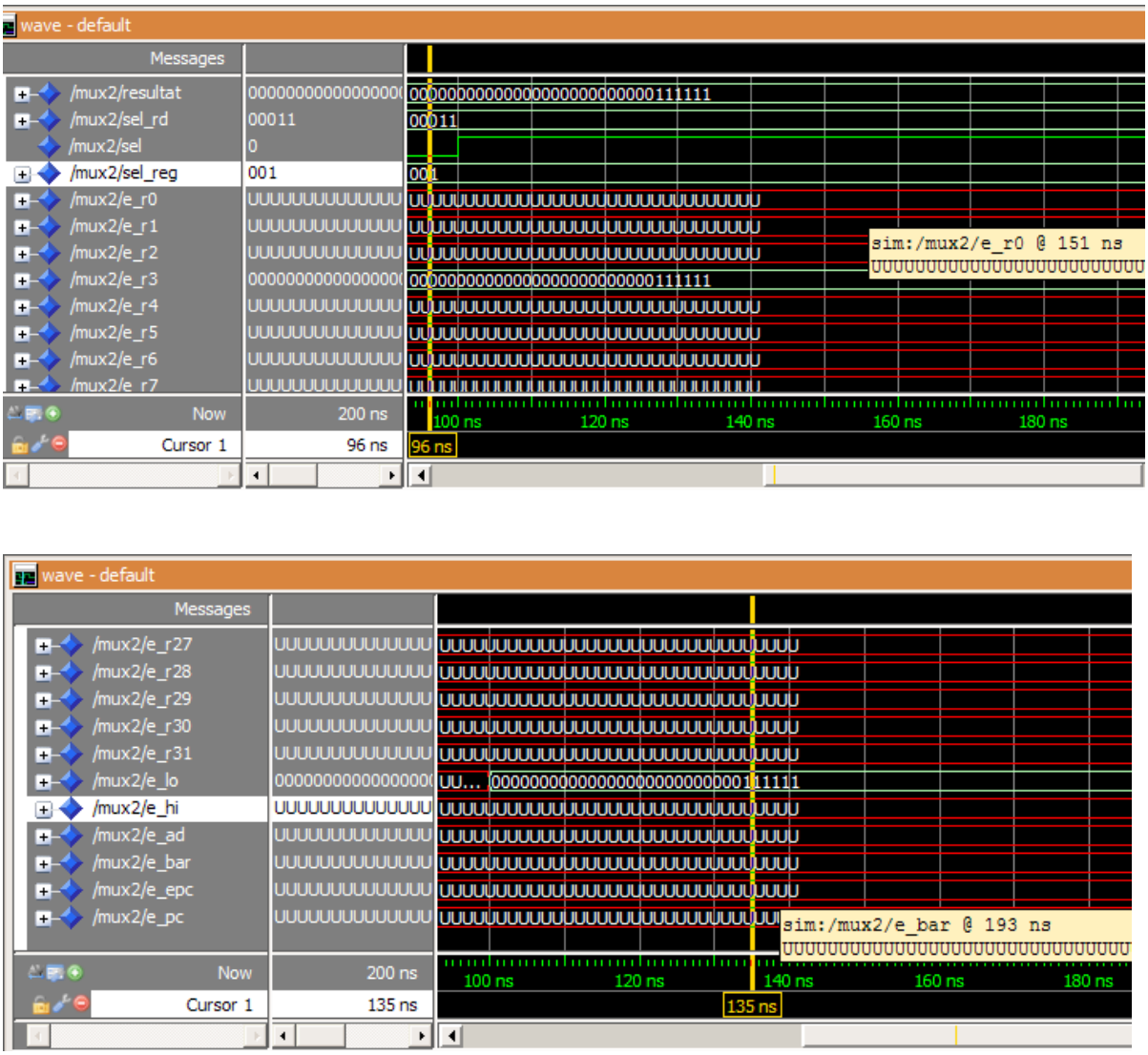


Figure. IV.21. Simulation de mux2

IV.12. Schéma fonctionnelle global

IV.12.1. Description : C'est le schéma qui rassemble tout les composants déjà vus. Au début de notre système. Pour démarrer le système, il suffit de donner les valeurs suivantes ck=1, reset=0, alors le compteur ordinal et le compteur sont mis à zéro, après le démarrage on donne au signal reset la valeur 1, le contenu de compteur ordinal passe à la mémoire pour récupère l'instruction de cette adresse dans le bus data, en suit, il la mit dans le registre d'instruction IR (Figure IV.23).

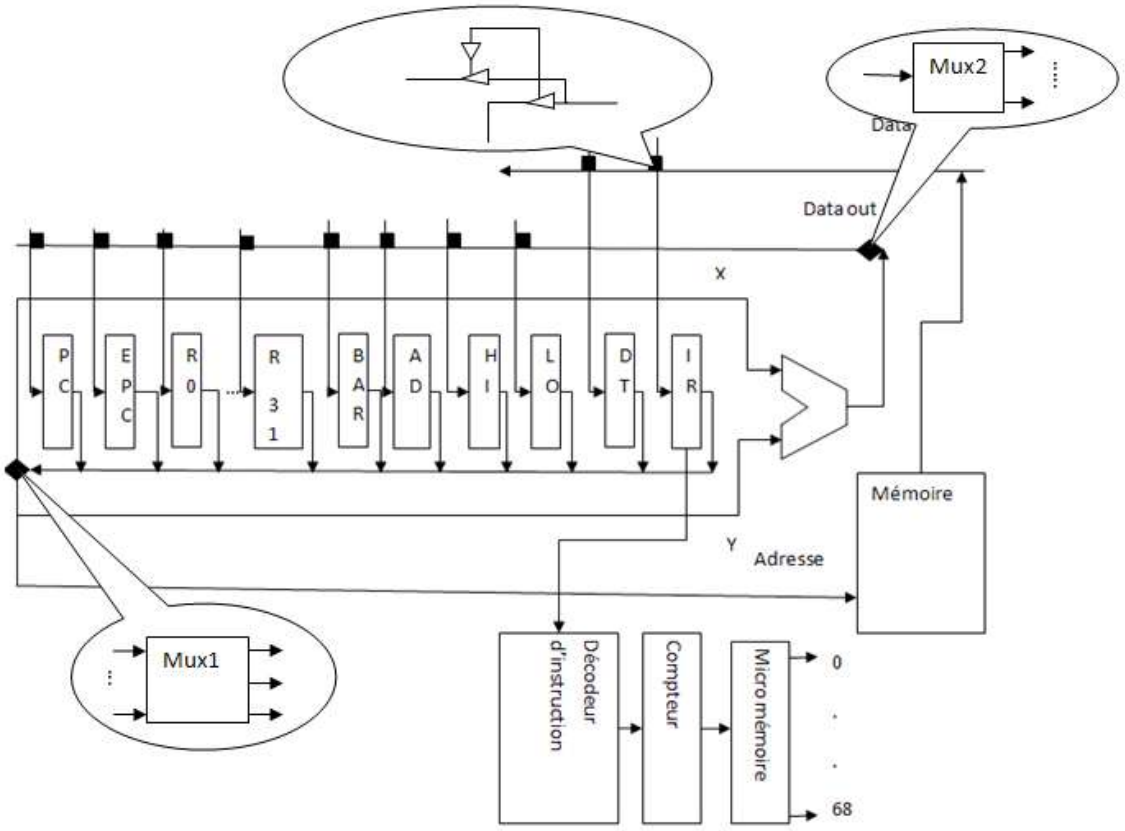


Figure. IV.22. Schéma fonctionnelle global

Nous avons maintenant la simulation de chargement d'instruction.

IV.12.2.Simulation :

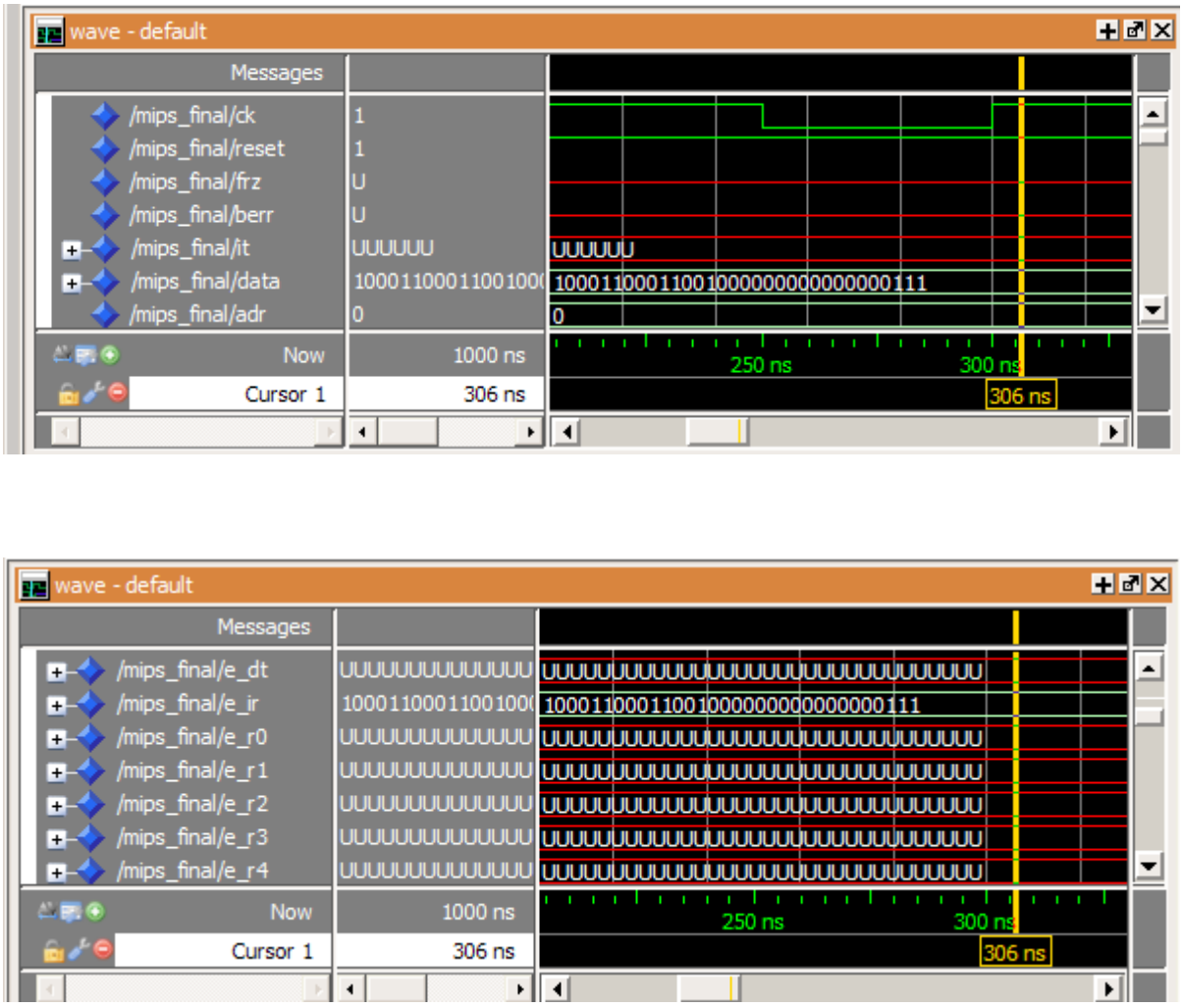


Figure IV.23. Simulation de chargement d'instruction

IV.13.Conclusion

Dans ce chapitre nous avons abordé en détaille les composants de processeur MIPS R3000, nous avons ajusté chaque composant à l'aide de schéma. La modélisation fait par le modelsim que présente la simulation de la marche de chaque composant.

Conclusion Général

Conclusion général

Dans ce travail nous avons fait une implémentation du processeur MIPS R3000. MIPS R3000 est un processeur de 32 bits, il respect la structure RISC, le jeu d'instruction se compose de 57 instructions.

Au plus des registres généraux définir par la spécification MIPS R3000, nous avons proposé la partie contrôle qui est composée de composants pour acheminer les données (mux1, mux2, opérateur trois états), de registres pour communiquer avec la mémoire, l'unité arithmétique et logique, une micro mémoire pour contenir les micros instructions et compteur pour contrôler l'accès à la micro mémoire. Les micros instructions sont stockées dans la micro mémoire de tel sort que l'exécution de l'instruction se fait en mode séquentiel.

Ce travail nous à bien permis de mieux comprendre comment fonctionne le processeur, exécute les instructions et bien sur nous permettra d'approfondir nos connaissance à propos de l'architecture des ordinateurs et la micro programmation.

Cependant, il reste plusieurs choses à faire pour améliorer l'efficacité du fonctionnement de cette architecture. Parmi les améliorations possibles nous définissons les points suivants comme perspectives:

- Rendre l'exécution des instructions en mode pipeline.
- Améliorer l'architecture interne afin de minimiser la consommation de l'énergie.
- Faire les modifications nécessaires pour passer en mode 64 bits.

ANNEXE

Annexe A : les déférents codes de décodeur d'instruction

Instructions arithmétique/ logique			
assembleur	Opération	Format	code
Add RD, RS, RT	Addition signée avec détection over flow Rd <- Rs +Rt	R	010010
SUB RD, RS, RT	Soustraction signée avec détection over flow Rd <- Rs -Rt	R	010100
Addu RD, RS, RT	Addition non- signée Rd <- Rs +Rt	R	010011
Subu RD, RS, RT	Soustraction non-signée Rd <- Rs –Rt	I	010101
Addi RT, RS, I	Addition immediate avec detection over flow Rt<- Rs +I	I	101011
Addiu RT, RS, I	Addition immediate Rt<- Rs + I	I	101100
Or Rd, Rs, Rt	Ou logique Rd <-RS or RT	R	010111
AND Rd, Rs, Rt	Et logique Rd <-RS and RT	R	010110
XOR Rd, Rs, Rt	Ou exclusive Rd <-RS xor RT	R	011000

Nor Rd, Rs, Rt	Non ou logique Rd <-RS nor RT	R	011001
Ori RT, Rs, I	Ou immédiate signée RT <-RS or I	I	110000
Andi RT, Rs, I	Et immediate signee RT <-RS and I	I	101111
Xori RT, Rs, I	Ou immédiate exclusive signée RT <-RS xor I	I	110001
Sllv RD, RT, RS	Décalage à gauche par numéro de RS RD <-RT << RS	R	000011
Srlv RD, RT, RS	Décalage à droite par numéro de RS RD <-RT >> RS	R	000100

Srav RD, RT, RS	Décalage à droite arithmétique registre RD <-RT >> RS	R	000101
Sll RD, RT, SH	Décalage à gauche logique RD <-RT << SH	R	000000
srl RD, RT, SH	Décalage à droite logique RD <-RT >> SH	R	000001

Sra RD, RT, SH	Décalage à droite arithmétique RD <-RT >> SH	R	000010
Lui RT, I	Lecture d'une constante dans les poids forts RT<-I II « 0000 »	I	110010
Slt RD, RS, RT	Comparaison signée registre registre Rd <- 1 if Rs<Rt	R	011010
Sltu RD, RS, RT	Comparison non signee Rd <- 1 if Rs<Rt else 0	R	011011
Slti RT, RS, I	Comparaison avec la valeur immédiate Rt<- 1 if Rs < I else 0	I	101101
Sltiu RT, RS	Comparaison non signée avec la valeur immédiate Rt<- 1 if Rs < I else 0	I	101110
Mult RS, RT	multiplication RS*RT LO <-32 les bits de poids faible HI <- 32 les bits de poids fort	R	001110
Multu RS, RT	Multiplication signée RS*RT LO <-32 de poids faible HI <- 32 de poids fort	R	001111
Div RS, RT	Division signée RS/RT LO <- quotient HI <- reste	R	010000
Divu RS, RT	Division non signée RS/RT LO <- quotient HI <- reste	R	010001
Mfhi RD	Copie le registre HI dans un registre général Rd <- HI	R	001010
Mflo RD	Copie le registre LO dans un registre général Rd <- LO	R	001100
Mthi RS	Copie d'un registre général dans le registre HI HI<-Rs	R	001011
Mtlo RS	Copie d'un registre général dans le registre LO LO<-Rs	R	001101

Instruction de lecture/écriture mémoire			
assembleur	opération	format	code
Lw RT, I(RS)	Lecture d'un mot de la mémoire RT<-M(RS+I)	I	110101
Sw RT, I(RS)	Écriture d'un mot en mémoire M(RS+I) <-RT	I	111010
Lh RT, I(RS)	Lecture demi mot RT<-M(RS+I)	I	110100
Lhu RT, I(RS)	Lecture demi mot RT<-M(RS+I)	I	110111
Sh RT, I(RS)	Écriture demi mot M(RS+I) <-RT	I	111001
Lb RT, I(RS)	Chargement de l'octet RT<-M(RS+I)	I	110011
Lbu RT, I(RS)	Chargement de l'octet RT<-M(RS+I)	I	110110
Sb RT, I(RS)	Écriture d'un octet M(RS+I) <-RT	I	111000

Instruction systèmes			
Assembleur	opération	format	code
Rfe	Restauration des bits d'état en fin d'exception SR <- SR 31:4 ll SR 5:2	R	100100
Break n	Arrêt et saut à la routine d'exception SR <- SR 31:6 ll SR 3:0 ll "00" PC <- "8000 0080" CR <- cause	R	001001
syscall	Appel à une fonction du système SR <- SR 31:6 ll SR 3:0 ll "00" PC <- "8000 0080" CR <- cause	R	001000
Mfc0 RT, RD	Copie d'un registre spécialisé dans d'un registre général Rt<- Rd	R	100010
Mtc0 RT, RD	Copie d'un registre général dans un registre spécialisé Rt<- Rd	R	100011

Annexe B : les déférentes valeurs possibles du signal selll

opération	selll
add	010010
addu	010011
sub	010100
subu	010101
and	010110
or	010111
xor	011000
nor	011001
slt	011010
sltu	011011
beq	100111
addiu	101100
sltu	101110
Andi	101111
ori	110000
xori	110001
lui	110010
lw	110101
mfhi	001010
mthi	001011
mflo	001100
mtlo	001101
multu	001111
div	010000
divu	010001
Divu1	111011

Annexe C : les déférentes valeurs de bits de micro mémoire

Bits de micro mémoire	composant	signal	Signaux d'entrées	Signaux de sortie
0	Opérateur trois états	Sel	Data_in	E_ir
1	Opérateur trois états	Sel	Data_in	E_dt
2	registre	ld	E_ir	S_ir
3	registre	ld	E_dt	S_dt
4	registre	ld	E_lo	S_lo
5	registre	ld	E_hi	S_hi
6	registre	ld	E_ad	S_ad
7	registre	ld	E_bar	S_bar
8	registre	ld	E_r0	S_r0
9	registre	ld l	E_r1	S_r1
10	registre	ld	E_r2	S_r2
11	registre	ld	E_r3	S_r3
12	registre	ld	E_r4	S_r4

13	registre	ld	E_r5	S_r5
14	registre	ld	E_r6	S_r6

Bits de micro mémoire	composant	signal	Signaux d'entrées	Signaux de sortie
15	registre	ld	E_r7	S_r7
16	registre	ld	E_r8	S_r8
17	registre	ld	E_r9	S_r9
18	registre	ld	E_r10	S_r10
19	registre	ld	E_r11	S_r11
20	registre	ld	E_r12	S_r12
21	registre	ld	E_r13	S_r13
22	registre	ld	E_r14	S_r14
23	registre	ld	E_r15	S_r15
24	registre	ld	E_r16	S_r16
25	registre	ld	E_r17	S_r17
26	registre	ld	E_r18	S_r18
27	registre	ld	E_r19	S_r19
28	registre	ld	E_r20	S_r20
29	registre	ld	E_r21	S_r21
30	registre	ld	E_r22	S_r22
31	registre	ld	E_r23	S_r23
32	registre	ld	E_r24	S_r24
33	registre	ld	E_r25	S_r25
34	registre	ld	E_r26	S_r26
35	registre	ld	E_r27	S_r27
36	registre	ld	E_r28	S_r28
37	registre	ld	E_r29	S_r29
38	registre	ld	E_r30	S_r30
39	registre	ld	E_r31	S_r31
40	registre	ld	E_epc	S_epc
41	Mux1	Sel1		
42	Mux1	Sel2		
44,45	Mux1	Sel_adr		
46,47,48	Mux1	Sel_regx		
49,50,51	Mux1	Sel_regy		
43	Mux2	sel		
52,53,54	Mux2	Sel_reg		
55,56,57,58,59,60	ual	selll		
61,62,63	mémoire	rw		
64	compteur	load		
65	compteur	rst		
66	Compteur ordinal	load		
67	Compteur ordinal	rst		
68	Compteur ordinal	incr		

Référence :

Site web :

[1] : Pirouz Bazargan, François Dromard, Alain Greiner,« PROCESSEUR MIPS R3000, architecture externe », version 2.2, septembre 1999.
Site: ftp://asim.lip6.fr/pub/mips/mips.externe.pdf

[2]: Pirouz Bazargan, François Dromard, Alain Greiner,« PROCESSEUR MIPS R3000, architecture externe », version 2.2, septembre 1999.

Site: ftp://asim.lip6.fr/pub/mips/mips.interne.pdf

[3]:Alain Greiner, « Architecture interne des micro processeurs », version 1.0. Site http://www.ensignement.polytechnique.fr

Bibliographies:

[4] : Robert Strandh, Irène Durand. « Architecture de l'ordinateur ». dunod paris 2005, ISBN : « 2100492144 ».