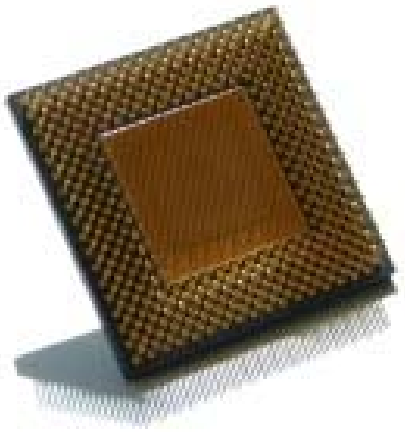
PROJET :

ETUDE ET REALISATION D'UN MICROPROCESSEURS RISC



Vicheka PHOR

Master 2 Professionnel

INFORMATIQUE

CAMSI

Conception d'Architecture de Machine et Systèmes Informatiques

2012-2013

PROJET :

ETUDE ET REALISATION D'UN MICROPROCESSEURS RISC

Vicheka PHOR

Master 2 Professionnel

INFORMATIQUE

CAMSI

Conception d'Architecture de Machine et Systèmes Informatiques

2012-2013

REMERCIEMENTS

Étudier à l'étranger nécessite de la détermination, de la patience, de travailler avec acharnement, et surtout des encouragements de la famille, des amis et des professeurs. Ce rapport a pu être complété grâce à eux.

Tout d'abord, je tiens à remercier vivement M. Abdelaziz M'ZOUGHI et M. Jacques JORDA, Responsables de la formation CAMSI (Concepteur en architecture de machines et systèmes informatiques) du Département d'Informatique, qui m'a accepté au sein de la formation de master.

Je tiens à exprimer mes profondes gratitudes à M. François THIEBOLT, Professeur de la formation CAMSI. J'apprécie vraiment son aide, ses conseils et lui suis reconnaissant pour le temps précieux qu'il m'a consacré.

Je vous aussi remercier Mme Daniela DRAGOMIRESCU et Mme Genevieve ESTADIEU de votre enseignement et de votre gentillesse.

Enfin, je voudrais dire un grand merci à ma famille qui m'a prodigué de l'amour, à mes amis et à mes professeurs qui ont toujours su me supporter et m'encourager au cours de cette année.

Je vous souhaite à tous tout le bonheur, toujours.

RÉSUMÉ

Les processeurs généraux actuels se répartissent en deux grandes catégories appelées CISC pour *Complex Instruction Set Computer* et RISC pour *Reduced Instruction Set Computer*. Grâce à sa simplicité et sa meilleure performance, le processeur RISC est de plus en plus utilisé. Ce rapport va consacrer sur le processeur RISC, à travers trois parties principales : conception de processeur RISC, réalisation et simulation, et synthèse. Les points les plus saillants et significatifs dans ce mémoire sont la conception, la mise en œuvre de processeur RISC en VHDL surtout la mise en œuvre matérielle d'une unité d'envoi pour résoudre les problèmes des aléas de données, et la synthèse. En plus, on a étudié sur deux solutions pour les problèmes aléas de données : une avec suspension du pipeline, et l'autre sans suspension du pipeline.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS

LISTE DES TABLEAUX

CHAPITRE I
INTRODUCTION

Ce chapitre va aborder sur l'introduction au sujet de projet « étude et réalisation d'un microprocesseur RISC », et l'objectif du projet.

I.1 L'INTRODUCTION AU SUJET DE PROJET

I.1.1 La description du projet

On considère un processeur RISC 32 bits possédant la structure suivante :

- un banc de 32 registres R0-R31 de 32 bits chacun avec 2 ports de lecture et 1 port d'écriture. Le registre R0 est câblé à 0, On peut tenter d'y écrire, mais sa lecture donnera toujours 0 ;
- une Unité Arithmétique et Logique (UAL) avec 2 entrées et une sortie de 32 bits. Elle permet de réaliser les opérations arithmétiques et logiques sur 32 bits du jeu d'instructions donné dans les tableaux ci-après ;
- une logique d'état composée de 4 bits de tests C (Carry), Z (Zero), V (oVerflow), N (Negative) permettant de donner une indication sur l'état des opérations réalisées et de pouvoir réaliser les branchements conditionnels ;
- un chemin de données pipeliné de 5 étages à cycle unique : EI / DI / EX / MEM / ER :
  - Le premier étage EI : réalise l'extraction de l'instruction réalise.
  - Le deuxième étage DI : réalise le décodage d'instruction et extraction des opérandes.
  - Le troisième étage EX : réalise l'exécution et calcul de l'adresse effective.
  - Le quatrième étage MEM : réalise l'accès à la mémoire.
  - Le cinquième étage ER : réalise l'écriture du résultat de l'instruction précédente.
- un registre : compteur de programme (CP) ;

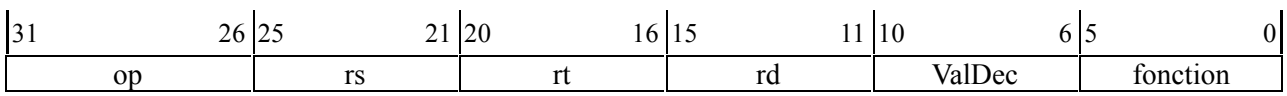
Les modes d'adressage sont de quatre types : registre à registre, immédiat, indirect et relatif. Toutes les instructions s'exécutent en un seul cycle. Les opérations d'accès mémoire peuvent être réalisées au niveau de l'octet, le demi-mot (16 bits) et le mot (32 bits) donc les adresses manipulées sont des adresses octet. Toutes les opérations, arithmétiques et logiques sont effectuées toujours sur des opérandes de 32 bits.

L'architecture interne du processeur est du type Harvard avec deux caches intégrés :

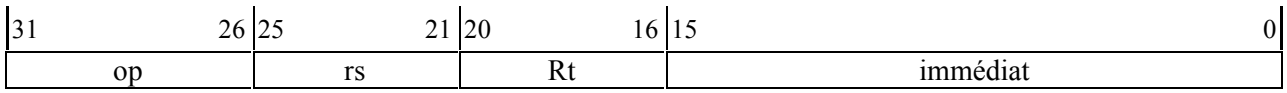
- cache d'instructions de 64 KO géré en application directe, la taille d'un bloc est de 16 mots de 32 bits (64 octets).
- cache de données de 64 KO géré en application associative par ensemble avec 4 blocs par ensemble. La taille d'un bloc est de 16 mots de 32 bits (64 octets).
  - La politique de remplacement est type LRU (Least Recently Used).
  - La politique de mise à jour de la mémoire centrale est du type WB (Write back) : mise jour différé.

Dans toute l'étude on ne considérera pas les exceptions.

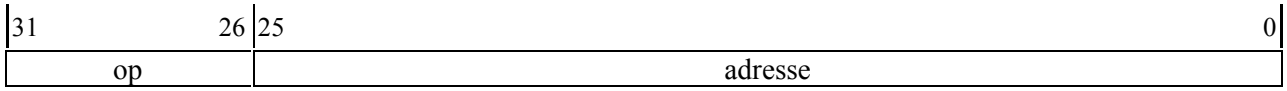
L'architecture RISC considérée ici possède 3 formats d'instructions fixes comme suit :



Format R ou registre



Format I ou immédiat



Format J ou saut

- Le champ *op* : code opération sur 6 bits, indique le format et la nature du traitement à effectuer.
- Le champ *rs* : formé de 5 bits, représente toujours un numéro du registre source
- Le champ *rt* : formé de 5 bits, représente un numéro de registre (source ou destination) ou une condition de branchement.
- Le champ *rd* : formé de 5 bits, représente un numéro de registre de destination.
- Le champ *ValDec* : formé de 5 bits, représente le nombre de décalage dans les instructions de décalage.
- Le champ fonction : formé de 5 bits, permet de différencier certaines instructions ayant le même code opération.
- Le champ *immédiat* : formé de 16 bits, représente soit un déplacement signé dans le cas des instructions de branchement, ou une constante signée immédiate dans le cas des opérations arithmétiques signées, ou une constante non signée immédiate dans le cas des opérations logiques et arithmétiques non signées .
- Le champ *adresse* : représente une adresse sur 26 bits.

Le jeu d'instructions est listé dans le Tableau I.1 suivant :

Tableau I.1 : le jeu d'instructions

Instruction	op					fonction
lsl	000000	0	rt	rd	ValDec	000000
lsr	000000	0	rt	rd	ValDec	000010
jr	000000	rs	0	0	0	001000
add	000000	rs	rt	rd	0	100000
addu	000000	rs	rt	rd	0	100001
sub	000000	rs	rt	rd	0	100010
subu	000000	rs	rt	rd	0	100011
and	000000	rs	rt	rd	0	100100
or	000000	rs	rt	rd	0	100101
xor	000000	rs	rt	rd	0	100110
nor	000000	rs	rt	rd	0	100111
slt	000000	rs	rt	rd	0	101010
sltu	000000	rs	rt	rd	0	101011
jalr	000000	rs	0	rd	0	001001
bltz	000001	rs	00000	déplacement signé		
bgez	000001	rs	00001	déplacement signé		
bltzal	000001	rs	10000	déplacement signé		
bgezal	000001	rs	10001	déplacement signé		
j	000010	adresse				
jal	000011	adresse				
beq	000100	rs	rt	déplacement signée		
bne	000101	rs	rt	déplacement signée		
blez	000110	rs	0	déplacement signé		
bgtz	000111	rs	0	déplacement signé		
addi	001000	rs	rt*	constante signée (sign_extend)		
addiu	001001	rs	rt*	constante signée (sign_extend)		
slti	001010	rs	rt*	constante signée (sign_extend)		
sltiu	001011	rs	rt*	constante signée (sign_extend)		
andi	001100	rs	rt*	constante non signée (zero_extend)		
ori	001101	rs	rt*	constante non signée (zero_extend)		
xori	001110	rs	rt*	constante non signée (zero_extend)		
lui	001111	0	rt*	constante non signée		

lb	100000	Rs**	rt*	déplacement signé
lh	100001	Rs**	rt*	déplacement signé
lw	100011	rs**	rt*	déplacement signé
lbu	100100	rs**	rt*	déplacement signé
lhu	100101	rs**	rt*	déplacement signé
sb	101000	rs**	rt	déplacement signé
sh	101001	rs**	rt	déplacement signé
sw	101011	rs**	rt	déplacement signé

* ➔ Le registre rt joue le rôle de registre de destination
** ➔ Le registre rs joue le rôle de registre de base.
sign_extend ➔ extension à 32 bits avec recopie du bit de signe dans les 16 bits de poids fort.
zero_extend ➔ extension à 32 bits avec mise à 0 des 16 bits de poids fort.

Le Tableau I.2 ci-dessous donne un exemple de chaque instruction avec son interprétation.

Tableau I.2 : un exemple de chaque instruction

Catégorie	Instruction	Exemple	Signification	Commentaires
Arithm-étique	Addition	add R1,R2,R3	R1 = R2 + R3	3 opdes; excep. possible
	Soustraction	sub R1,R2,R3	R1 = R2 - R3	3 opdes; excep. possible
	Addition immédiat	addi R1,R2,100	R1 = R2 + 100	+ cste; excep. possible
	Addition non signé	addu R1,R2,R3	R1 = R2 + R3	3 opdes; pas d'exception
	Soustr. non signé	subu R1,R2,R3	R1 = R2 - R3	3 opdes; pas d'exception
	Add. imm. non signé	addiu R1,R2,100	R1 = R2 + 100	+ cste; pas d'exception
Logique	ET	and R1,R2,R3	R1 = R2 & R3	3 registres opérandes
	OU	or R1,R2,R3	R1 = R2 I R3	3 registres opérandes
	XOR	xor R1,R2,R3	R1 = R2 xor R3	3 registres opérandes
	Non OU	nor R1,R2,R3	R1 = R2 nor R3	3 registres opérandes
	ET immédiat	andi R1,R2,100	R1 = R2 & 100	2 registres opdes + cste
	OU immédiat	ori R1,R2,100	R1 = R2 I 100	2 registres opdes + cste
	XOR immédiat	xori R1,R2,100	R1 = R2 xor 100	2 registres opdes + cste
	Déc. logique gauche	lsl R1,R2,10	R1 = R2 << 10	Décalage à gauche
	Déc. logique droite	lsr R1,R2,10	R1 = R2 >> 10	Décalage à droite
Chart. imm.	Chart. des pds forts	Lui 100(R1)	R1 = 100 << 16	chargement poids forts
Transfert de Données	Chart. octet signé	Lb R1,100(R2)	R1 = $M[R2+100]_8$	Chart. avec ext. signe
	Chart. demi mot signé	Lh R1,100(R2)	R1 = $M[R2+100]_{16}$	Chart. avec ext. signe
	Chargement mot	lw R1,100(R2)	R1 = M[R2+100]	Chart. mot 32 bits
	Chart. octet non signé	Lbu R1,100(R2)	R1 = $M[R2+100]_8$	Chart. avec ext. des zéros
	Chart. demi non signé	Lhu R1,100(R2)	R1 = $M[R2+100]_{16}$	Chart. avec ext. des zéros
	Rangement octet	sb R1,100(R2)	$M[R2+100] = (R1)_8$	Rangt. Octet pf en mém.
	Rangement demi mot	sh R1,100(R2)	$M[R2+100] = (R1)_{16}$	Rangt. Demi pf en mém.
	Rangement mot	sw R1,100(R2)	M[R2+100] = R1	Rangt. mot en mém.
Branche-ment conditionnel	Branchement si =	beq R1,R2,100	si (R1==R2) aller en CP+4+100	Test d'égalité ; branch. relatif à CP
	Branchement si ≠	bne R1,R2,100	si (R1!=R2) aller en CP+4+100	Test d'inégalité ; brancht. relatif à CP
	Branchement si ≤ 0	blez R1,100	si (R1≤0) aller en CP+4+100	Test ≤ 0 compl. à 2 ; brancht. relatif à CP
	Branchement si > 0	bgtz R1,100	si (R1>0) aller en CP+4+100	Test > 0 compl. à 2 ; brancht. relatif à CP
	Branchement si < 0	bltz R1,100	si (R1<0) aller en CP+4+100	Test < 0 compl. à 2 ; brancht. relatif à CP
	Branchement si ≥ 0	bgez R1,100	si (R1≥0) aller en CP+4+100	Test ≥ 0 compl. à 2 ; brancht. relatif à CP
	Bt. Avec lien si < 0	Bltzal R1,100	si (R1<0) aller en CP+4+ 100 avec R31 <= CP+4	Test < 0 compl. à 2 ; brancht. relatif à CP
	Bt. Avec lien si ≥ 0	Bgezal R1,100	si (R1≥0) aller en CP+4+ 100 avec R31 <= CP+4	Test ≥ 0 compl. à 2 ; brancht. relatif à CP
	Positionner si <	slt R1,R2,R3	si (R2 < R3) alors R1=1 ; sinon R1=0	Test d'infériorité ; complément à 2

	Positionner si < immédiat	slti R1,R2,100	si (R2 < 100) alors R1=1 ; sinon R1=0	Test < constante ; complément à 2
	Positionner si < non signé	sltu R1,R2,R3	si (R2 < R3) alors R1=1 ; sinon R1=0	Test d'infériorité ; nomb. entiers naturels
	Positionner si < immédiat non signé	sltiu R1,R2,100	si (R2 < 100) alors R1=1 ; sinon R1=0	Test < constante ; nomb. entiers naturels
Saut Incondition-nel	Saut	j 10000	aller en 10000	Saut vers adresse de destination
	Saut par registre	jr R1	aller à l'adresse contenu dans R1	Pour switch ; retour de sous programme
	Saut avec lien	jal 10000	R31 = CP+4 ; aller en 10000	Pour appel de sous programme
	Saut avec lien par registre	Jalr R1,R3	R3 <= CP+4 ; Aller à l'adresse dans R1	Pour appel de sous programme

I.1.2 Le travail demandé

Travail demandé : Conception et simulation en VHDL du chemin de données et de l'unité de contrôle de l'ensemble du processeur pour le jeu d'instructions décrit dans le Tableau I.1 ci-dessus, ainsi que les mémoires caches d'instructions et de données.

Le travail sera sanctionné par une présentation orale, une démonstration sur machine ainsi qu'un rapport décrivant l'essentiel de votre conception en justifiants vos choix et solutions. Les sources VHDL des différents modules seront fournies en annexe dans le rapport.

N.B. : Les rapports doivent rendus au plus tard le jour de la soutenance.

I.2 L'OBJECTIF DU PROJET

Objectif est de réaliser sous forme de projet la conception et la mise en ouvre d'une architecture de processeur types RISC en s'appuyant sur les concepts abordés dans le cours UE11P1 : Techniques Avancées de Conception des Machines Informatiques et celui de l'UE1 : Structure et Conception de Base Machines Informatiques, pour une conception avancée.

La mise en ouvre du projet se découpe en trois parties :

1. Conception de processeur RISC : cette partie présentera l'architecture de processeur RISC, les chemins de données pipeline et l'unité de contrôle pipeline de l'ensemble du processeur RISC pour le jeu d'instructions décrit dans le Tableau I.1, et les aléas dans les pipelines instruction et les solutions.
2. Réalisation et simulation : la mise en ouvre est réalisée en VHDL sous la suite du logiciel de conception Cadence (c'est outil de conception par excellence utilisé dans le monde industriel et académique) ou ModelSim PE Student Edition 10.1c et en utilisant d'un petit assembleur ou compilateur pour l'exécution de programme écrit en langage machine.
3. Synthèse : la synthèse sera faite en utilisant les outils de synthèse sous la suite Cadence de l'AIME (Atelier Interuniversitaire de Micro-Electronique) de Toulouse.

CHAPITRE II
CONCEPTION DE PROCESSEUR RISC

Ce chapitre présentera l'architecture de processeur RISC, les chemins de données pipeline et l'unité de contrôle pipeline de l'ensemble du processeur RISC pour le jeu d'instructions décrit dans le tableau I.1, et les problèmes et les solutions des aléas dans les pipelines.

II.1 L'ARCHITECTURES DE PROCESSEUR RISC

II.1.1 Qu'est-ce que un processeur RISC ? et son historique

Le microprocesseur à jeu d'instruction réduit ou *reduced instruction-set computer*: RISC en anglais, est un type d'architecture matérielle de microprocesseurs, qui se caractérise par un jeu d'instructions réduit, facile à décoder et comportant uniquement des instructions simples.

Le principe de machine à jeu d'instructions réduit, ou RISC, a été inventé par IBM. Le premier processeur de ce type, apparu en 1975 (John Cocke) sous le nom IBM801 (le numéro du bâtiment dans lequel ce projet a été poursuivi), est issu d'un contrôleur de commutateur téléphonique. L'étude de ce type d'architecture s'est ensuite poursuivie au sein des universités de Stanford et Berkeley. L'historique de RISC est montré dans la Figure II.1. Les processeurs RISC commerciaux sont listés dans le Tableau II.1.

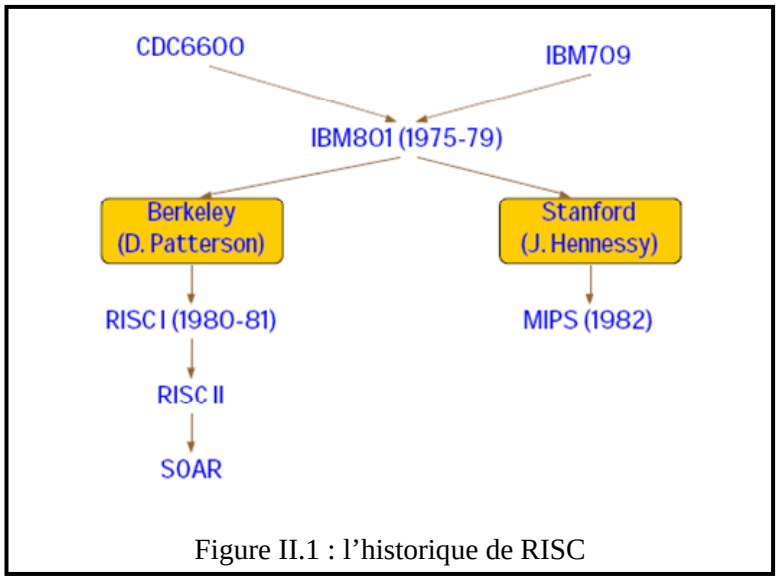


Figure II.1 : l'historique de RISC

Tableau II.1 : les processeurs RISC commerciaux

Année	Nom
1986	Silicon Graphics: MIPS (pipeline)
1986	Hewlett-Packard: HP-PA (pipeline)
1987	Sun: SPARC (superscalaire)
1990	Apple-IBM-Motorola: Power PC (superscalaire)
1992	DEC: Alpha (pipeline)

II.1.2 Les caractéristiques de processeur RISC

Les principales caractéristiques des processeurs RISC sont les suivantes :

- Jeu d'instructions simplifié : toutes les instructions sont codées avec un même nombre de bits, généralement un mot machine. L'op-code se trouve à la même position pour toutes les instructions. Ceci facilite le décodage des instructions.
- Registres indifférenciés et nombreux : tous les registres peuvent être utilisés dans tous les contextes. Il n'y a par exemple pas de registre spécifique pour la pile. Les processeurs séparent cependant les registres pour les valeurs flottantes des autres registres.
- Limitation des accès mémoire : les seules instructions ayant accès à la mémoire sont les instructions de chargement et de rangement. Toutes les autres instructions opèrent sur les registres. Il en résulte une utilisation intensive des registres.
- Nombre réduit de modes d'adressage : il n'y pas de mode d'adressage complexe. Les modes d'adressages possibles sont généralement immédiats, direct, indirect et relatifs. Exemple : les instructions arithmétiques ont généralement 3 adresses : 2 registres servant d'opérandes et un registre de sortie.
- Nombre réduit de types de données : les seuls types de données supportés sont les entiers de différentes tailles (8, 16, 32 et 64 bits) et des nombres flottants en simple et double précision.
- Chemin de données pipeline : le pipelining permet d'exécuter plus d'une instruction à la fois en décomposant le chemin d'exécution en différentes étapes, cela améliore le débit pour la charge totale de travail.

II.1.3 Quelques architectures de processeur RISC

Le Tableau II. suivant donne quelques caractéristiques de processeurs RISC actuels.

Tableau II.2 : caractéristiques de processeurs RISC actuels

	Alpha 21264B	IBM Power3-II	Sun Ultra-III	HP PA-8600	MIPS R12000
Fréquence (MHz)	833	450	900	552	400
Pipeline	7/9	7/8	14/15	7/9	6
Technologie	0,18-6	0,22-6	0,18-7	0,25-2	0,25-4
Taille	115	163	210	477	204
Transistors (M)	15.4	23	29	130	7.2
Puissance (W)	75	36	65	60	25
Cache (I/D)	64K/64K	32K/64K	32K/64K	512K/1M	32K/32K

II.2 LE CHEMIN DE DONNEES PIPELINE DE PROCESSEUR RISC

II.2.1 Le chemin de données pipeline

Le pipelining permet d'exécuter plus d'une instruction à la fois en décomposant le chemin d'exécution en différentes étapes comme indiquant dans la Figure II.2 . Cela améliore le débit pour la charge totale de travail.

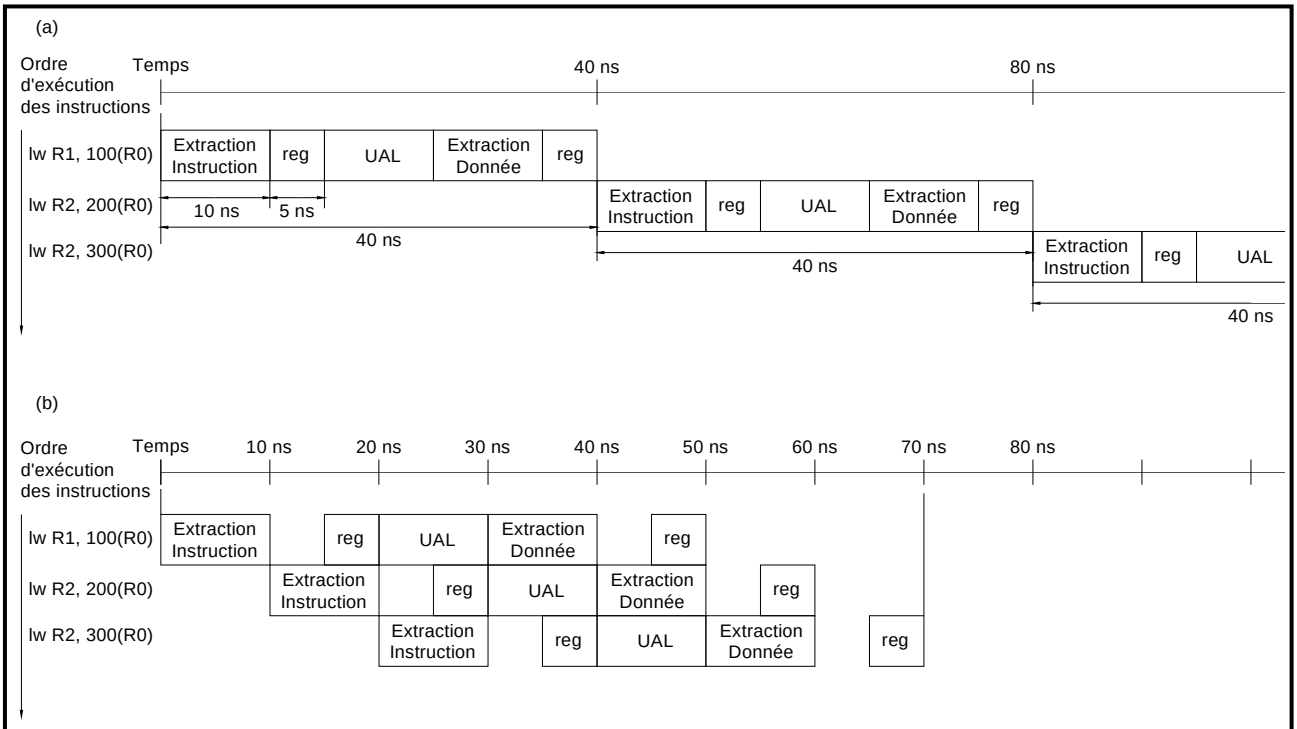


Figure II.2 : le modèle à cycle unique non-pipeline (a) vs le modèle à cycle unique pipeline (b)

Afin de mettre en œuvre un pipeline, la première tâche est de découper l'exécution des instructions en plusieurs étapes. Pour le processeur RISC, le chemin de données pipeline sera découpé en 5 étages :

1. EI : Extraction d'instruction ;
2. DI : Décodage d'instruction et extraction de registre ;
3. EX : Exécution et calcul d'adresse effective ;
4. MEM : Accès mémoire ;
5. ER : Ecriture du résultat.

La clé de la mise en œuvre du chemin de données pipeline est d'utiliser des registres pipelines pour séparer le chemin de données en 5 étages : EI, DI, EX, MEM, et ER. C'est-à-dire que la séparation entre 2 étages doit être réalisée par un registre pipeline qui sert d'élément d'état. Il y a quatre registres qui sont étiquetés selon les étapes qu'ils séparent par exemple EI/DI, DI/EX, EX/MEM, et MEM/ER. Si le pipeline possède n étages, il y a n instructions en cours d'exécution simultanée, chacune dans une étape différente. Le facteur d'accélération est donc le nombre n d'étages. La fréquence d'horloge est limitée par l'étape qui est la plus longue à réaliser. Le chemin de données pipeline sans unité de contrôle, unité de détection d'aléas et unité d'envoi est illustrée sur la figure A1.1 dans l'annexe A1.

Note : dans la figure A1.1 de l'annexe A1, la troisième entrée du multiplexeur MUX_ALU_SCRB dont sa valeur est égale à 16, est mise pour l'instruction *lui.* Et, la quatrième entrée du multiplexeur MUX_ALU_SCRB dont sa valeur est égale à 0, est mise pour les instructions *bltz*, *bgez*, *bltzal*, et *bgezal*. En plus, la troisième entrée du multiplexeur MUX_REG_DST dont sa valeur est égale à 31, est mise pour les instructions *jal*, *jalr*, *bltzal*, et *bgezal.*

II.2.2 Le fonctionnement du chemin de données pipeline

Pour expliquer le fonctionnement du pipeline, nous considérons une instruction de chargement parce qu'elle est active durant les cinq étages.

1. Extraction d'instruction :
   - Lecture de l'instruction à partir de la mémoire d'instruction ou la cache d'instruction en utilisant l'adresse du CP dans le registre CP.
   - Stockage de l'instruction dans le registre pipeline EI/DI.

- Incrémentation de l'adresse du CP de 4 puis rechargée dans le registre CP pour être prêt pour le prochain cycle d'horloge.

2. Décodage d'instruction et extraction de registre :
   - À partir du champ instruction Inst[31..0] en sortie du registre pipeline EI/DI :
     >> Extension du signe à partir du champ 16 bits immédiat (Inst[15..0]) étendu à 32 bits ;
     >> Lecture registre 1 à partir du champ *rs* (Inst[25..21]) ; (c'est le registre de base pour *lw*).
     >> Lecture registre 2 à partir du champ *rt* (Inst[20..16]).
   - Stockage de la valeur du registre 1 et de la valeur immédiate étendue obtenues ainsi que le CP incrémentée dans le registre pipeline DI/EX.
3. Exécution et calcul d'adresse effective :
   - Récupération à la sortie du registre pipeline DI/EX :
     >> Le contenu du registre 1 sur la 1ère entrée de l'ALU (en anglais arithmetic and logic unit (ALU), ou bien UAL, l'**unité arithmétique et logique** en français) ;
     >> La valeur immédiate étendue sur la 2ème entrée de l'ALU.
   - Addition de ces deux valeurs en vue de l'obtention de l'adresse effective.
   - Stockage de l'adresse effective dans le registre pipeline EX/MEM.
4. Accès mémoire : cette étape est uniquement utile pour les instructions de chargement et de rangement.
   - Lecture de la mémoire de données ou la cache de données en utilisant l'adresse contenue dans le registre pipeline EX/MEM.
   - Stockage de la donné dans le registre pipeline MEM/ER.
5. Ecriture du résultat :
   - Lecture de la donnée dans le registre pipeline MEM/ER.
   - Ecriture dans le banc de registre de deuxième étage

II.3 L'UNITE DE CONTROLE PIPELINE DE PROCESSEUR RISC

L'unité de contrôle permet de positionner tous les signaux de contrôle en se fondant sur le code opération (OP[31..26], le code fonction (F[5..0]) et le champ B (B[20..16) pour des instructions de branchements conditionnels *bltz, bgez, bltzal* et *bgezal*. Le chemin de données pipeline avec unité de contrôle, et sans unité d'envoi et unité de détection d'aléas est illustrée sur la figure A1.2 dans l'annexe A1.

Pour spécifier le contrôle du pipeline de processeur RISC, nous n'avons qu'à positionner les valeurs de contrôle à chacun de ses étages. Puisque chaque ligne de contrôle est associée à un composant qui n'est actif que pendant un seul étage, nous pouvons diviser les lignes de contrôle en cinq groupes, selon l'étage du pipeline :

1. Étage EI : les signaux de contrôle pour lire la mémoire d'instruction et pour écrire le PC sont toujours actifs, il n'ya donc rien de spécial à contrôler dans cet étage.

2. Étage DI : les signaux de contrôle à positionner dans cet étage sont les suivant :
   - ✦ SIGNED_EXT : indique si l'extension de la valeur immédiate est signée ou non. L'extension signée s'il est égal à 1 et l'extension non signée si non.
   - ✦ J_j : qui n'est activé (valeur 1) que lorsque l'instruction est un saut inconditionnel *j*.
   - ✦ J_jal : qui n'est activé (valeur 1) que lorsque l'instruction est un saut inconditionnel *jal*.

3. Étage EX : les signaux de contrôle à positionner dans cet étage sont les suivant :
   - ✦ ALU_OP : qui détermine l'opération à effectuer en fonction de l'instruction. Les valeurs possibles de l'ALU_OP sont les suivantes : ALU_ADD, ALU_SUB, ALU_AND, ALU_OR, ALU_NOR, ALU_XOR, ALU_SLT, ALU_LSL, ou ALU_LSR.
   - ✦ ALU_SIGNED : indiquera si l'opération en cours est signée (valeur 1) ou non (valeur 0).

- ✦ ALU_SRCA : qui détermine la sortie de multiplexeur MUX_ALU_SRCA pour l'entrée A de l'ALU. Les valeurs que peut prendre ce signal sont les suivantes :
  - REGS_QA (l'entrée A de l'ALU est donnée par le champ *rs*) ;
  - REGS_QB (l'entrée A de l'ALU est donnée par le champ *rt*) ; ou
  - IMMD (l'entrée A de l'ALU est donnée par la valeur étendue du champ immédiat).
- ✦ ALU_SRCB : qui détermine la sortie de multiplexeur MUX_ALU_SRCB pour l'entrée B de l'ALU. Les valeurs que peut prendre ce signal sont les suivantes :
  - REGS_QB (l'entrée B de l'ALU est donnée par le champ *rt*) ;
  - IMMD (l'entrée B de l'ALU est donnée par la valeur étendue du champ immédiat) ;
  - VAL_DEC (l'entrée B de l'ALU est donnée par la valeur étendue du champ ValDec) ;
  - VAL_16 (l'entrée B de l'ALU est donnée par la valeur X"00000010"); ou
  - VAL_0 (l'entrée B de l'ALU est donnée par la valeur X"00000000").
- ✦ REG_DST : qui détermine la sortie de multiplexeur MUX_REG_DST pour le numéro de registre d'écriture. Les valeurs que peut prendre ce signal sont les suivantes :
  - REG_RD (le numéro du registre d'écriture est donné par le champ *rd*) ;
  - REG_RT (le numéro du registre d'écriture est donné par le champ *rt*) ; ou
  - R31 (le numéro du registre d'écriture est le registre *R31*).
- ✦ J_jr : qui n'est activé (valeur 1) que lorsque l'instruction est un saut inconditionnel *jr*.
- ✦ J_jalr : qui n'est activé (valeur 1) que lorsque l'instruction est un saut inconditionnel *jalr*.

4. Étage MEM : les signaux de contrôle à positionner dans cet étage sont les suivant :
   - ✦ DC_AS : indiquera s'il y a d'opération sur la mémoire (valeur 1) ou non (valeur 0).
   - ✦ DC_SIGNED : indiquera si l'opération en cours est signée (valeur 1) ou non (valeur 0).
   - ✦ DC_DS : qui défini le type d'accès 8/16/32/64 bits. Les valeurs que peut prendre ce signal sont les suivantes : MEM_DS, MEM_8, MEM_16, ou MEM_32.
   - ✦ DC_RW : indiquera si l'opération est une lecture (valeur 1) ou une écriture (valeur 0).
   - ✦ BRANCH : qui n'est activé (valeur 1) que lorsque l'instruction est un branchement conditionnel *beq*, *bne*, *blez*, *bgtz*, *bltz*, *bgez*, *bltzal*, ou *bgezal*.
   - ✦ B_type : qui détermine la sortie de multiplexeur MUX_BRANCH pour le type de branchement *beq*, *bne*, *blez*, *bgtz*, *bltz*, *bgez*. Il peut prendre la valeur : B_beq, B_bne, B_blez, B_bgtz, B_bltz, B_bgez. B_bltzal, ou B_bgezal.

5. Étage ER : les signaux de contrôle à positionner dans cet étage sont les suivant :
   - ✦ REGS_W* : indiquera si l'opération d'écriture est active (valeur 0) ou non (valeur 1).
   - ✦ REGS_SRCD : qui détermine la sortie de multiplexeur MUX_ REGS_SRCD ou la donnée à écrire au registre. Il peut prendre la valeur soit le résultat l'ALU (ALU_S), soit la valeur sortie de mémoire (mem_Q), soit l'adresse de la prochaine instruction (NextPC).

Les lignes de contrôle de chaque étage pour les différentes instructions sont montrées dans le tableau A2.1, A2.2, A2.3 et A2.4, dans l'annexe A2.

II.4 LES ALEAS DANS LES PIPELINES INSTRUCTION ET LES SOLUTIONS

Le bon fonctionnement du pipeline peut être perturbé par plusieurs événements appelés *aléas* (*pipeline hazard* en anglais). Ces événements sont classés en trois catégories :

1. **Les aléas structurels** : ce type de problèmes survient lorsque deux instructions dans des étages différents du pipeline nécessitent la même ressource. Ce problème n'existe pas dans notre processeur RISC parce que notre processeur RISC est muni de deux caches intégrés : une cache d'instructions de 64 KO et une cache de données de 64 KO.

   ✦ Considérons par exemple le morceau de code suivant.

   ```
   LW R7,4(R0)
   ADDI R6,R0,1
   ADDI R2,R0,1
   ADDI R1,R0,1
   ```

   Le déroulement de l'exécution des quatre premières instructions dans le pipeline devrait être le suivant.

Instructions	Cycles d'horloge							
	1	2	3	4	5	6	7	8
LW R7,4(R0)	EI	DI	EX	MEM	ER			
ADDI R6,R0,1		EI	DI	EX	MEM	ER		
ADDI R2,R0,1			EI	DI	EX	MEM	ER	
ADDI R1,R0,1				EI	DI	EX	MEM	ER

   Problème : l'étape MEM (accès mémoire) de l'instruction LW R7,4(R0) a lieu en même temps que l'étape EI (chargement de l'instruction) de l'instruction ADDI R1,R0,1. Ces deux étapes nécessitent simultanément l'accès à la mémoire. Il s'agit d'un aléa structurel.

   Solution : ce problème est généralement résolu en séparant la mémoire où se trouvent les instructions de celle où se trouvent les données.

2. **Les aléas de données** : les aléas de données interviennent quand une instruction dépend du résultat d'une instruction précédente.

   ✦ Les aléas de données à l'étage EX

   Considérons le morceau de code suivant.

   ```
   ADDI R1,R0,1
   ADD R2,R1,R0
   ```

   Le déroulement de l'exécution de ces deux instructions dans le pipeline devrait être le suivant.

Instructions	Cycles d'horloge					
	1	2	3	4	5	6
ADDI R1,R0,1	EI	DI	EX	MEM	ER	
ADD R2,R1,R0		EI	DI	EX	MEM	ER

   Problème : le problème est que le résultat de la première instruction est écrit dans le registre R1 après la lecture de ce même registre par la seconde instruction. La valeur utilisée par la seconde instruction est alors erronée.

Solution : le résultat de la première instruction est disponible dès la fin de l'étape EX (exécution de l'instruction) de celle-ci. Il est seulement utilisé à l'étape EX de la seconde instruction. Il suffit alors de le fournir en entrée de l'ALU à la place de la valeur lue dans R1 par la seconde instruction. Ceci est réalisé en ajoutant un chemin de données ou une unité d'envoi.

Instructions	Cycles d'horloge					
	1	2	3	4	5	6
ADD R1,R0,1	EI	DI	EX	MEM	ER	
ADD R2,R1,R0		EI	DI	EX	MEM	ER

Le chemin de données pipeline avec unité de contrôle et unité d'envoi, et sans unité de détection d'aléas est illustrée sur la figure A1.3 dans l'annexe A1.

✦ Les aléas de données à l'étage MEM

Considérons le morceau de code suivant.

```
LW R1,4(R0)
ADD R2,R0,2
ADD R3,R1,R0
```

Le déroulement de l'exécution de ces deux instructions dans le pipeline devrait être le suivant.

Instructions	Cycles d'horloge						
	1	2	3	4	5	6	7
LW R1,4(R0)	EI	DI	EX	MEM	ER		
ADD R2,R0,2		EI	DI	EX	MEM	ER	
ADD R3,R1,R0			EI	DI	EX	MEM	ER

Problème : le problème est que le résultat de la première instruction est écrit dans le registre R1 après la lecture de ce même registre par la troisième instruction. La valeur utilisée par la troisième instruction est alors erronée.

Solution : le résultat de la première instruction est disponible dès la fin de l'étape MEM (exécution de l'instruction) de celle-ci. Il est seulement utilisé à l'étape EX de la troisième instruction. Il suffit alors de le fournir en entrée de l'ALU à la place de la valeur lue dans R1 par la troisième instruction. Ceci est réalisé en ajoutant un chemin de données ou une unité d'envoi.

Instructions	Cycles d'horloge						
	1	2	3	4	5	6	7
LW R1,4(R0)	EI	DI	EX	MEM	ER		
ADD R2,R0,2		EI	DI	EX	MEM	ER	
ADD R3,R1,R0			EI	DI	EX	MEM	ER

✦ Les aléas de données lorsqu'une instruction tente de lire un registre à la suite d'une instruction de chargement écrivant dans le même registre

Considérons un autre morceau de code assez semblable.

```
LW R1,4(R6)
ADD R0,R1,R2
```

Le déroulement de l'exécution de ces deux instructions dans le pipeline devrait être le suivant.

Instructions	Cycles d'horloge					
	1	2	3	4	5	6
LW R1,4(R6)	EI	DI	EX	MEM	ER	
ADD R2,R1,R0		EI	DI	EX	MEM	ER

Problème : dans cet exemple encore, le résultat de la première instruction est écrit dans le registre R1 après la lecture de ce même registre par la seconde instruction. Par contre, le résultat n'est pas disponible avant l'étape MEM (accès mémoire) de la première instruction.

Solution : comme cette étape a lieu après l'étape EX de la seconde instruction, il ne suffit pas d'ajouter un chemin de données ou une unité d'envoi pour faire disparaître l'aléa. Il faut en outre suspendre la seconde instruction. Ceci introduit une bulle dans le pipeline ou l'unité de détection d'aléas pour les chargements.

Instructions	Cycles d'horloge								
	1	2	3	4	5	6	7	8	9
LW R1,4(R6)	EI	DI	EX	MEM	ER				
ADD R2,R1,R0		EI	DI		MEM	ER	EI		
Inst. n° 3			EI		DI	EX	MEM	ER	
Inst. n° 4					EI	DI	EX	MEM	ER

Le chemin de données pipeline avec unité de contrôle, unité d'envoi, et unité de détection d'aléas est illustrée sur la figure A1.4 dans l'annexe A1.

3. **Les aléas de contrôle ou de branchement** : les aléas de contrôle ou de branchement résultent de l'exécution en pipeline des branchements et des autres instructions qui modifient le CP.

   Lors de l'exécution d'une instruction de branchement conditionnel, on dit que le branchement est *pris* si la condition est vérifiée et que le programme se poursuit effectivement à la nouvelle adresse. Un branchement sans condition est toujours pris.

   Problème : lorsqu'un branchement est pris, l'adresse de celui-ci est calculée à l'étape EX (exécution de l'instruction) et rangée dans le registre CP à l'étape ER (rangement du résultat). Toutes les instructions qui suivent l'instruction de branchement doivent être laissées de côté. Au niveau du pipeline, on obtient le diagramme d'exécution suivant.

Instructions	Cycles d'horloge											
	1	2	3	4	5	6	7	8	9	10	11	12
Branchement	EI	DI	EX	MEM	ER							
Inst. suivante n° 1		EI	DI	EX								
Inst. suivante n° 2			EI	DI								
Inst. suivante n° 3				EI								
Inst. cible n° 1						EI	DI	EX	MEM	ER		
Inst. cible n° 2							EI	DI	EX	MEM	ER	
Inst. cible n° 3								EI	DI	EX	MEM	ER

Solution : laisser de côté des instructions qui suivent l'instruction de branchement, signifie donc qu'on doit être capable de vider les instructions des étages EI, DI et EX du pipeline. Ceci est réalisé en ajoutant une ligne de contrôle, appelée Flush, à l'étage EI, DI et EX. Lorsqu'un branchement est pris, le signal Flush d'étage EI, DI et EX est actif qui indique que tous les signaux de contrôle à l'étage EI, DI et EX doivent être remis à la valeur défaut, c'est-à-dire qu'il n'y a pas d'opération d'accès mémoire ni d'opération d'écriture de données dans le registre.

Le schéma final de notre processeur RISC mettant en œuvre le contrôle des aléas est montré sur la figure A1.5 dans l'annexe A1.

CHAPITRE III
REALISATION ET SIMULATION

Les études de conception de processeur RISC menée dans les parties précédentes seront complétées dans ce chapitre par la mise en œuvre sous la suite du logiciel ModelSim PE Student Edition 10.1c et en utilisant d'un petit assembleur ou compilateur pour l'exécution de programme écrit en langage machine. Ce chapitre abordera les thèmes ci-dessous :

1. Mise en œuvre de processeur RISC en VHDL : la description des fichiers sources VHDL est donnée dans cette session. on va traiter sur les sous programme pour l'unité de contrôle de pipeline, pour les aléas dans le pipeline, surtout la solution de problèmes des aléas de données avec suspension du pipeline et sans suspension du pipeline.
2. Le compilateur
3. Résultat de simulation

III.1 LA MISE EN ŒUVRE DE PROCESSEUR RISC EN VHDL

Afin de mettre en œuvre de processeur RISC en VHDL, on a créé des fichiers ci-dessous :

- Le fichier *registres.1.vhd* et le fichier *test_registres.0.vhd* correspondent à un banc de 32 registres R0-R31 de 32 bits chacun avec 2 ports de lecture et 1 port d'écriture. Le registre R0 est câblé à 0, On peut tenter d'y écrire, mais sa lecture donnera toujours 0.
- Le fichier *memory.1.correction.vhd* et le fichier *test_memory.1.vhd* correspondent à un banc mémoire avec la taille d'un mot de 32 bits par défaut, la largeur de bus d'adresses de 32 bits par défaut, et le nombre de mots mémoire de 16 par défaut. En plus, il est capable d'opérer des accès au format 8/16/32/64 bits avec ou sans extension de signe.
- Le fichier *cpu_package.2_1.vhd* contient la fonction de log2. Pour le logiciel ModelSim PE Student Edition 10.1c, il y a des erreurs «*# **Error: Cannot call subprogram "log2" before it is elaborated*.» si l'on met cette fonction dans la même *V5cpu_package.2.vhd ou V6cpu_package.2.vhd*.
- Le fichier *V5cpu_package.2.vhd ou V6cpu_package.2.vhd* contient des constants (définition des matériels et des logiciels), des types (définition des multiplexeurs et des registre pipeline), et des fonctions.
- Le fichier *V5risc.0.vhd*, le fichier *V6risc.0.vhd*, et le fichier *test_risc.0.vhd* correspondent à un processeur RISC. Le fichier *V5risc.0.vhd* et le fichier *V6risc.0.vhd* sont les fichiers top-level pour notre processeur RISC.
- Le fichier *logique.i.0.txt* contient des instructions en binaire pour la mémoire d'instructions.
- Le ficher *logique.d.0.txt* contient des données en binaire pour la mémoire de données.

Note : On utilise soit le fichier *V5cpu_package.2.vhd* et le fichier *V5risc.0.vhd* pour la mise en œuvre d'une solution de problèmes des aléas de données avec la méthode de suspension du pipeline, soit le fichier *V6cpu_package.2.vhd* et le fichier *V6risc.0.vhd* pour la mise en œuvre d'une solution de problèmes des aléas de données avec la méthode sans suspension du pipeline.

Les sources VHDL des différents modules se trouvent dans l'annexe A3.

III.1.1 L'unité de control de pipeline

On a créé un sous-programme dans le *V5cpu_package.2.vhd ou V6cpu_package.2.vhd*, nommé procédure control pour l'unité de control de pipeline dans notre processeur RISC.

- Procédure control : permet de positionner les signaux de control pour chaque étage (DI, EX, MEM, et ER) en fonction de l'instruction identifiée soit par son code op, soit par son code fonction, soit par son code branchement. Les signaux de contrôle de chaque étage pour les différentes instructions sont indiqués dans le tableau A2.1, A2.2, A2.3 et A2.4, dans l'annexe A2.

III.1.2 Les aléas dans le pipeline

Il y a deux types aléas dans notre processeur RISC : aléas de données et aléas de branchement.

1. Les aléas de données

Il y a deux solutions pour les problèmes aléas de données : une avec suspension du pipeline, et l'autre sans suspension du pipeline.

a. La solution de problèmes des aléas de données avec suspension du pipeline

On a créé deux sous-programmes dans le *V5cpu_package.2.vhd* : procédure envoi (unité d'envoi), et procédure aleaLW (unité de détection d'aléas).

- Procédure envoi (unité d'envoi) : permet de résoudre des problèmes d'aléas de données sauf des problèmes d'aléas pour l'instruction de type chargement (Figure III.1, a, b). Les aléas de données interviennent quand une instruction dépend du résultat d'une instruction précédente.

  - Pour les instructions *addi*, *addiu*, *slti*, *sltiu*, *andi*, *ori*, *xori*, *lui*, *lb*, *lh*, *lw*, *lbu*, *lhu*, son champ *rt* joue le rôle de registre destination, donc il n'y pas d'aléa de données sur le multiplexeur MUX_ALU_XB, c'est-à-dire qu'on n'envoi pas le résultat de l'instruction précédente à l'entrée B de l'ALU.
  - Pour les instructions *sb*, *sh* et *sw*, le registre lu à partir son champ *rt* est une donné à stocké dans la mémoire de données. Pour ces instructions, on n'envoi pas le résultat de l'instruction précédente à l'entrée B de l'ALU. Mais, on l'envoi à l'entrée pour la mémoire de données *D* en traversant le multiplexeur MUX_mem_data.
  - Pour les instructions bgez, bltzal, et bgezal, la donné à l'entrée B de l'ALU est toujours égal à zéro. Donc, on n'envoi pas non plus le résultat de l'instruction précédente à l'entrée B de l'ALU.
  - Pour les instructions jr, jalr, bltz, blez, et bgtz, son champ *rt* est égal à zéro, qui est le registre R0. On peut tenter d'y écrire ou d'envoyer le résultat de l'instruction précédente à l'entrée B de l'ALU, mais sa valeur donnera toujours 0.
  - Les aléas peuvent avoir lieu à la fois aux étages EX et MEM pour une même entrée de l'ALU. Dans ce cas, la priorité est à l'aléa se trouvant le plus proche de l'étage DI dans l'ordre d'exécution du programme : ici l'aléa EX.

- Procédure aleaLW (unité de détection d'aléas) : permet de résoudre des problèmes d'aléas de donnés pour l'instruction de type chargement avec suspension du pipeline (Figure III.1, c). L'envoi ne fait pas l'affaire lorsqu'une instruction tente de lire un registre à la suite d'une instruction de chargement écrivant dans le même registre. La donnée est toujours lue dans la mémoire au cycle d'horloge 4 tandis que l'ALU effectue l'opération de l'instruction suivante. Dans le cas où il y a des aléas de donnée, on doit suspendre le pipeline pour la combinaison d'instructions composée d'un chargement suivi d'une instruction lisant le résultat de celui-ci. Pour suspendre le pipeline, on va faire donc revivre l'unité de détection d'aléas.

  - Elle opère pendant l'étage DI de l'instruction de type chargement, et continue à fonctionner en présence de l'unité d'envoi.
  - Elle recherche les instructions de chargement dans l'étage EX en testant si le signal de contrôle DI_EX_Regs_W = '0', DI_EX_DC_AS = '1', et DI_EX_DC_RW = '1'.
  - Si l'instruction est de type chargement par exemple LW, et si son registre écriture (le champ *rt* ou registre écriture) dans le registre pipeline DI/EX est la même que les registre lectures de l'instruction suivante (le champ *rs* et *rt* dans le registre pipeline EI/DI), alors suspendre le pipeline. Pourtant, on ne suspend pas le pipeline si le branchement est effectué même si la condition est vérifiée.

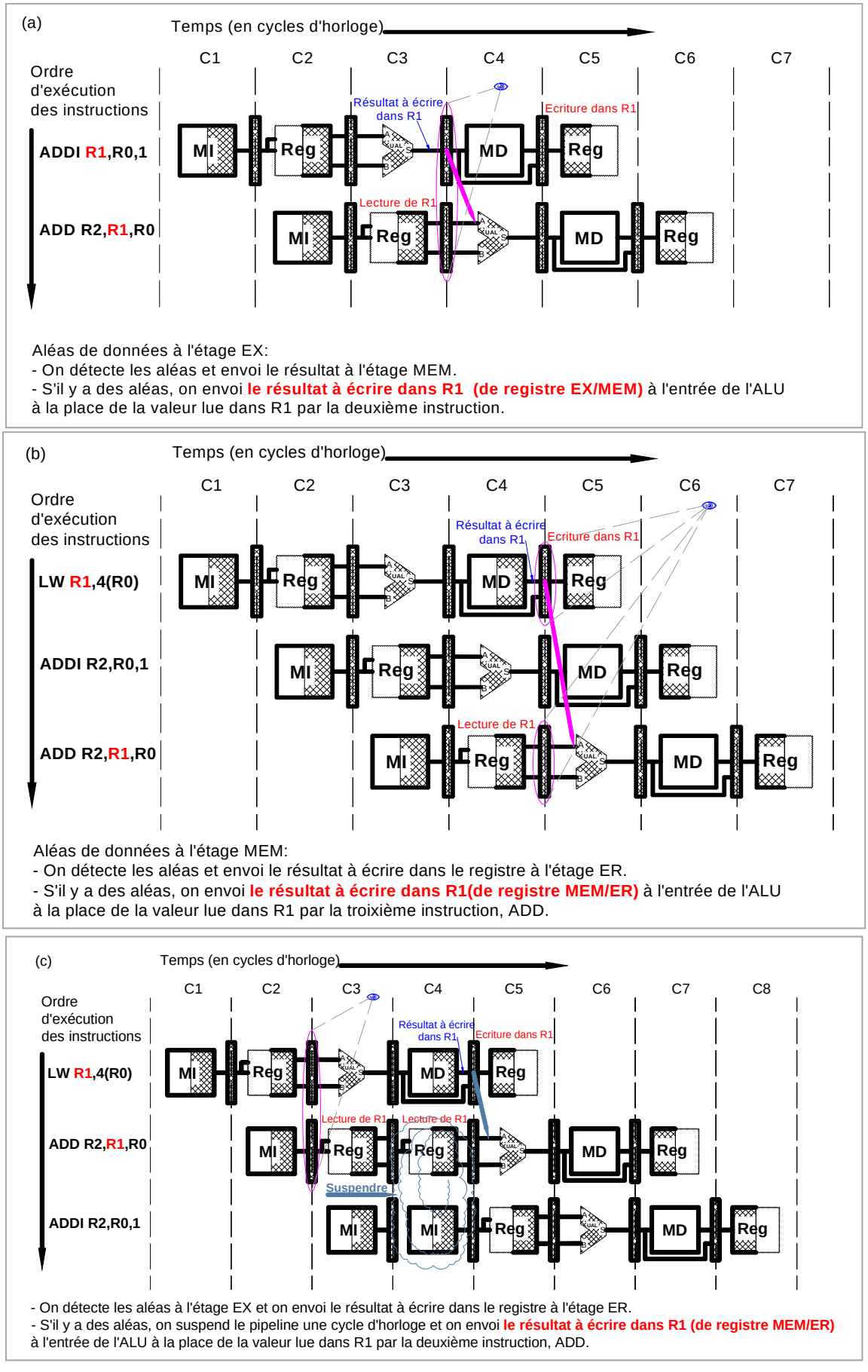


Figure III.1 : les aléas de données à l'étage EX (a), les aléas de données à l'étage MEM (b), et la détection des aléas de donnés avec suspension du pipeline (c)

b. La solution de problèmes des aléas de données sans suspension du pipeline

De même, on a créé deux sous-programmes dans le *V6cpu_package.2.vhd* : procédure envoi (unité d'envoi), et procédure aleaLW (unité de détection d'aléas). Le schéma de processeur RISC mettant en œuvre le contrôle des aléas sans suspension du pipeline est montré sur la figure A1.6 dans l'annexe A1.

- Procédure aleaLW (unité de détection d'aléas) : permet de résoudre des problèmes d'aléas de donnés pour l'instruction de type chargement sans suspension du pipeline (Figure III. 2). La donnée est toujours lue dans la mémoire au cycle d'horloge 4 à l'étage MEM) tandis que l'ALU effectue l'opération de l'instruction suivante à l'étage EX. Dans le cas où il y a des aléas de donnés, on peut envoyer les données lu de la mémoire de données à l'entré de l'ALU pour l'instruction suivante. Par exemple, pour l'instruction LW suivi par l'instruction ADD, quand l'instruction LW est à l'étage MEM, l'instruction ADD est à l'étage EX. L'opération de la mémoire et l'opération de l'ALU est exécuté simultanément et la donné lue de la mémoire (de l'instruction LW) est prêt et on peut l'envoyer à l'entré de l'ALU (de l'instruction ADD). De même, on va créer l'unité de détection d'aléas. En plus, pour résoudre ce problème, on doit ajouter un d'autre entrée pour la donné lue de la mémoire (*MEM_mem_Q*) aux multiplexeurs MUX_ALU_XA et MUX_ALU_XB.

  - Elle opère pendant l'étage EX de l'instruction de type chargement, et continue à fonctionner en présence de l'unité d'envoi.
  - Elle recherche les instructions de chargement dans l'étage MEM en testant si le signal de contrôle EX_MEM_Regs_W = '0', EX_MEM_DC_AS = '1', et EX_MEM_DC_RW = '1'.
  - Si l'instruction est de type chargement par exemple LW, et si son registre écriture (le champ *rt* ou registre écriture) dans le registre pipeline EX/MEM est la même que les registre lectures de l'instruction suivante (le champ *rs* et *rt* dans le registre pipeline DI/EX), alors suspendre le pipeline.

- Procédure envoi (unité d'envoi) : c'est presque la même que celle du fichier *V5cpu_package.2.vhd*. La différence, c'est qu'on a ajouté des conditions pour des problèmes d'aléas de donnés pour l'instruction de type chargement sans suspension du pipeline. Ces conditions sont faites pour l'objectif de renseigner aux multiplexeurs MUX_ALU_XA et MUX_ALU_XB de sélectionner la donné lue de la mémoire (*MEM_mem_Q*) pour l'entrée de l'ALU.

  - Si l'unité de détection d'aléas a détecté des problèmes d'aléas de donnés pour l'instruction de type chargement, elle active le signal *mem_halt*. Si le signal *mem_halt* est actif et le code opération (OP [31..26]) dans l'étage MEM est une instruction de type chargement, alors l'unité d'envoi va envoyer le signal de sélection aux multiplexeurs MUX_ALU_XA et MUX_ALU_XB de sélectionne la donné lue de la mémoire (*MEM_mem_Q*) pour l'entrée de l'ALU.

2. Les aléas de branchement

Les aléas de branchement interviennent quand une instruction est de type saut inconditionnel et de type branchement conditionnel qui est effectué. Les solutions des aléas de branchement sont les suivantes :

- Le saut inconditionnel J et JAL sont effectués dans l'étage DI, on va vider l'étage précédent EI.
- Le saut inconditionnel JR et JALR sont effectués dans l'étage EX, on va donc vider les étages précédents EI et DI.
- Le saut branchement conditionnel est effectué dans l'étage MEM, alors vider les étages précédents EX, DI et EI (Figure III.3).

III.2 LE COMPILATEUR

Le fichier *r3kasm2.c* et *r3kasm2.h* sert à convertir le code assembleur en code de langage machine (code binaire). Il nous facilite de créer le fichier d'instructions (binaire), le fichier *logique.i.0.txt*, pour faire la simulation. Ce code est mis dans l'annexe A4.

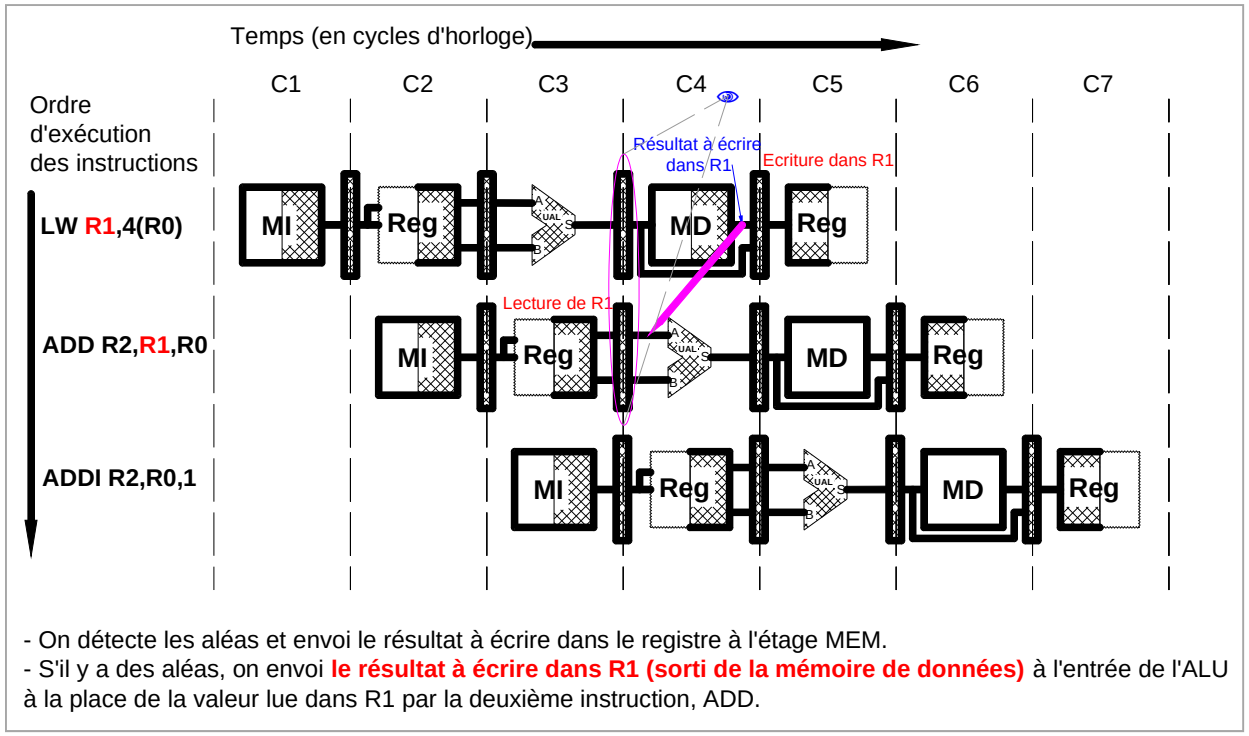


Figure III. 2 : la détection des aléas de donnés sans suspension du pipeline

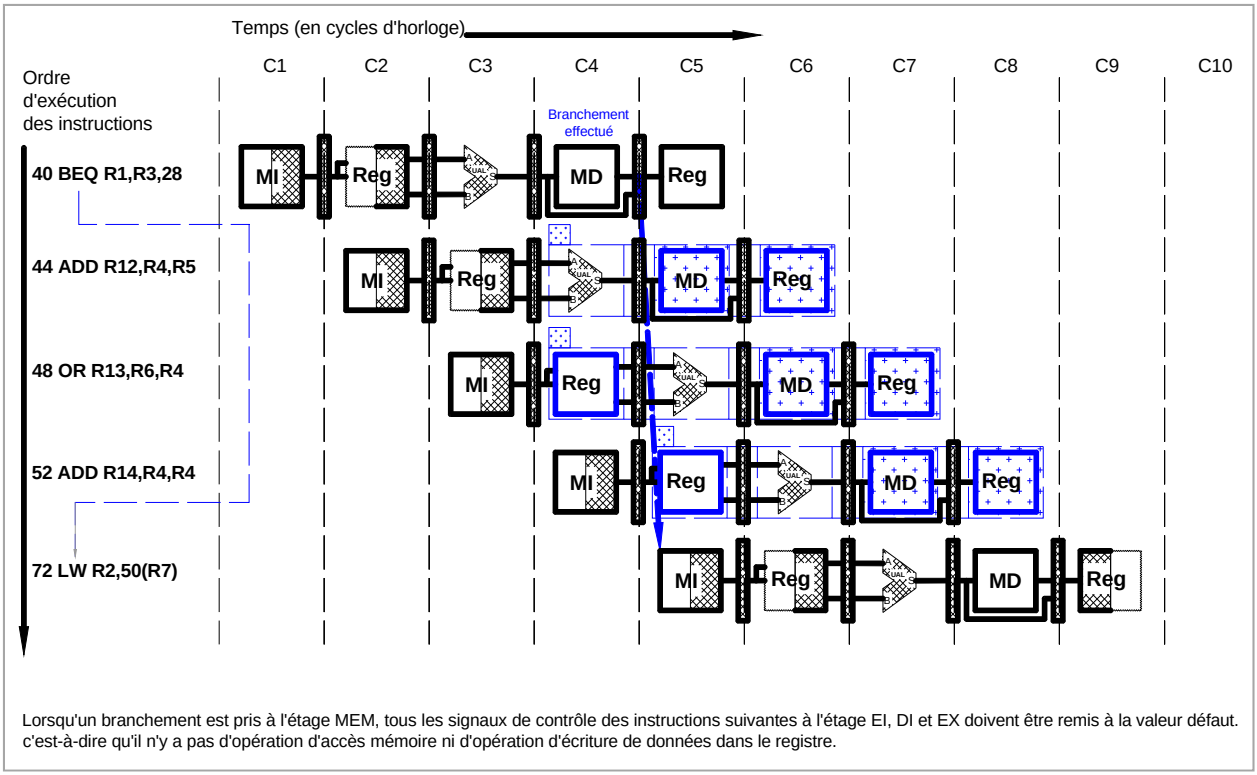


Figure III.3 : les aléas de branchement

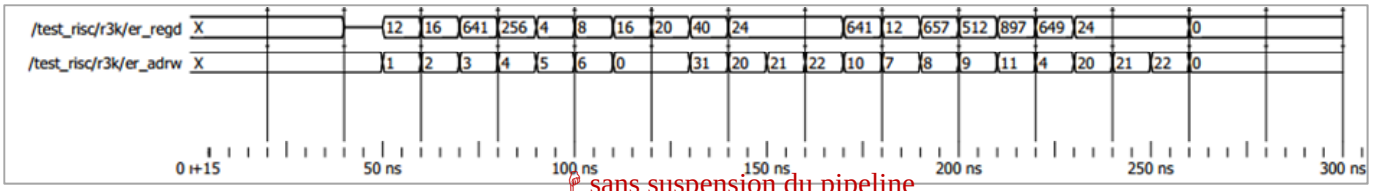
III.3 LES RESULTATS DE SIMULATION

On a choisi trois fichiers assembleurs : *testadd.asm*, *bench_1.asm* et *test_aleas_plus.asm* pour faire la simulation en utilisant d'un petit assembleur (*r3kasm2.c* et *r3kasm.h*) pour l'exécution de programme écrit en langage machine. Et on a mis la taille des caches L1 en nombre de mots égal à 32. Pour le fichier d'instructions *testadd.txt* venant de *testadd.asm*, *bench_1.txt* venant de *bench_1.asm*, et *test_aleas_plus.txt* et

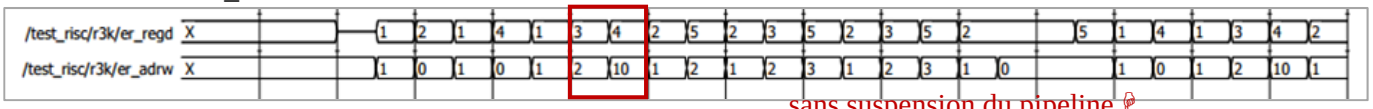
test_aleas_plus.asm, on va comparer les résultats de la simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd* avec ceux des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*. Pour faire vite fait une comparaison, on ne regarde que les données à écrire (*er_regd*) dans le banc de registres et son adresse (*er_adrw*). Les contenus des fichiers .asm et .txt sont donnés dans l'annexe A5. Tous les résultats détaillés de la simulation se trouvent dans l'annexe A6.

a. Les résultats de simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd*

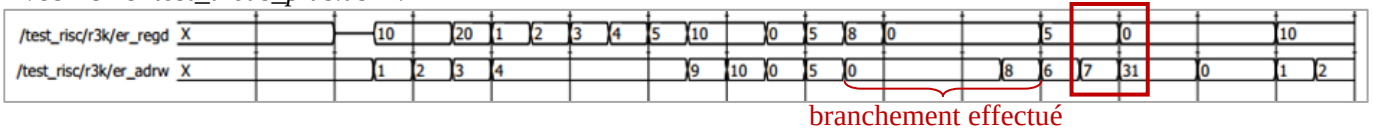
Avec fichier *testadd.txt* :



Avec fichier *bench_1.txt* :

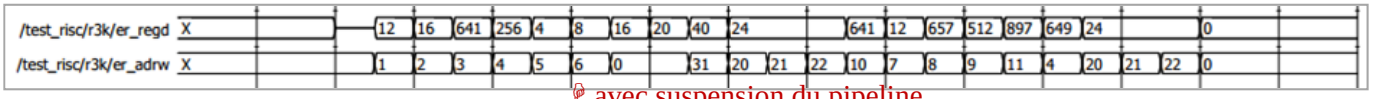


Avec fichier *test_aleas_plus.asm* :

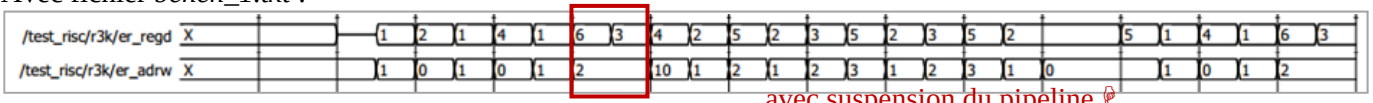


b. Les résultats de simulation des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*

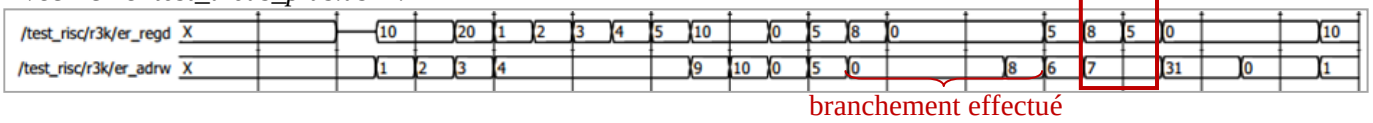
Avec fichier *testadd.txt* :



Avec fichier *bench_1.txt* :



Avec fichier *test_aleas_plus.asm* :




```
//************PROGRAMME DE TEST******************
//bench_1.asm
//;Aleas de type irresolvable
//;Aleas de rangement
addi  R1,R0,1          // R1 ← 1
sw R1,2(R0)            // M[2+R0] ← 1
//;Aleas de chargement
addi  R1,R0,1          // R1 ← 1
sw R1,4(R0)            // M[4+R0] ←1
lw R1,4(R0)            // R1 ← M[4+R0]     (R1 = 1)
addi R2,R1,2           // R2 ← R1 + 2      (R2 = 3)
add R10,R2,R1          // R10 ← R1 + R2    (R10 = 4)
```

D'après les résultats et un morceau de code dans le fichier bench_1.asm ci-dessus, on peut noter que pour les résultats de la simulation des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*, à l'instruction *addi R2,R1,2*, la donnée (*er_regd* = 3) à écrire dans le banc de registres R2 (*er_adrw* = 2) est disponible 1 cycle plus tard par rapport à celle de la simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd*.

CHAPITRE IV
SYNTHESE

Ce chapitre présentera le cycle de conception et les résultats de synthèse pour notre processeur RISC.

IV.1 LE CYCLE DE CONCEPTION

Le cycle de conception est donné dans la Figure IV.1 ci-dessous :

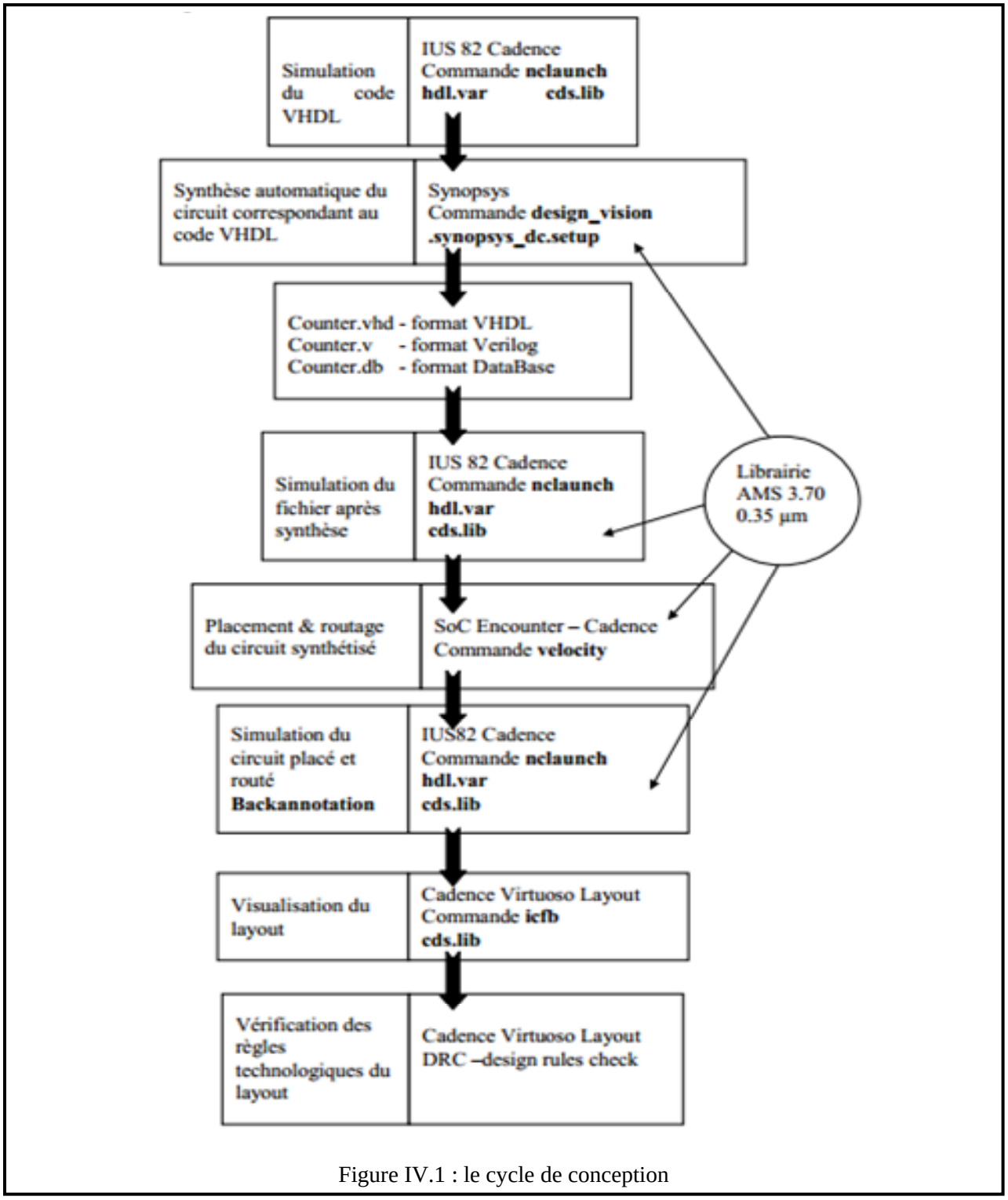


Figure IV.1 : le cycle de conception

- La simulation du système est faite pour vérifier la validité du code. Cette étape est faite à plusieurs moments entre autres : avant-synthèse, après-synthèse, et après placement-routage. Ex: VSS (Synopsys) et ModelSim (Mentor Graphics).
- La synthèse est l'étape qui transforme la description VHDL d'une puce en portes logiques. Cette description VHDL et ne doit pas comporter d'éléments comportementaux non compréhensibles par l'outil de synthèse. La synthèse est guidée par des contraintes imposées par l'utilisateur (vitesse et superficie). Ex: Design Compiler (Synopsys) et BuildGates (Cadence).
- Le placement et Routage sert à placer les cellules en accord avec le floorplan, créer des lignes d'alimentation de puissance, créer des lignes de distribution d'horloge, et connecter les cellules entre elles.

IV.2 LES RESULTATS DE SYNTHESE

En utilisant le tutorial dans l'annexe A et en utilisant les outils de synthèse sous la suite Cadence de l'AIME (Atelier Interuniversitaire de Micro-Electronique) de Toulouse, on peut obtenir les résultats de synthèse pour notre processeur RISC dans le Tableau IV.1 ci-dessous. Il est à noter que le fichier de mémoire *memory.1.correction.vhd* comporter d'éléments comportementaux non compréhensibles par l'outil de synthèse qu'il faut les corriger selon les indications des messages erreurs, et que il faut ajouter les signales sorties pour le fichier *V5risc.0.vhd* ou *V6risc.0.vhd* pour que la synthèse marche.

Tableau IV.1 : les résultats de synthèse

Fichier de synthèse	Période d'horloge T_{CLK} [ns]	Fréquence de l'horloge [MHz]	Chemin critique [ns]	Voltage [V]	Puissance dynamique [mW]	Puissance statique [µW]	Nombre de cellules
V5risc.0.vhd	10	100	9,85	3,30	54,5345	1,4224	14185
	9	111,11	8,85	3,30	60,7387	1,4325	14322
	8	125	7,85	3,30	68,3236	1,4376	14383
	7	142,86	**7,68**	3,30	78,2718	1,4425	14415
V6risc.0.vhd	8	125	7,96	3,30	68,7665	1,4603	14583
	7	142,86	**7,79**	3,30	78,4423	1,4596	14603

Après le tableau IV.1, on peut observer que :

- Pour les résultats de synthèse du fichier *V5risc.0.vhd*, quand la période d'horloge TCLK = 7 ns, le chemin critique est égal à 7,68 qui est supérieur au période de l'horloge. Donc, la fréquence maximale de fonctionnement est égale à **125 MHz** avec la période d'horloge de 8 ns et le chemin critique de **7,85**.
- Pour les résultats de synthèse du fichier *V6risc.0.vhd*, quand la période d'horloge TCLK = 7 ns, le chemin critique est égal à 7,79 qui est supérieur au période de l'horloge. Donc, la fréquence maximale de fonctionnement est égale à **125 MHz** avec la période d'horloge de 8 ns et le chemin critique de **7,96**.
- Le chemin critique du fichier *V6risc.0.vhd* est supérieur à celui du fichier *V5risc.0.vhd*. La raison, c'est que le nombre de cellules du fichier *V6risc.0.vhd* est plus grand que celui du fichier *V5risc.0.vhd*. En fait, pour le fichier *V6risc.0.vhd*, on a ajouté une entrée pour la donné lue de la mémoire (*MEM_mem_Q*) aux multiplexeurs MUX_ALU_XA et MUX_ALU_XB, et une entrée (le signal *mem_halt*) à l'unité d'envoi.

CHAPITRE V
CONCLUSION

On constate que l'exécution d'un branchement pris dégrade notablement la performance du pipeline puisque quatre cycles sont perdus. Comme les branchements constituent en général 20% à 30% des instructions exécutées par un programme, il est primordial d'améliorer leur exécution. Une façon simple d'optimiser les branchements est de ne pas leur faire suivre toutes les étapes du pipeline afin que la nouvelle adresse soit écrite le plus tôt possible dans le registre PC.

Pour les branchements conditionnels, la condition ne dépend que des indicateurs N, Z et P et du code de l'instruction. Cette condition peut donc être calculée à l'étape DI (décodage de l'instruction). De même, l'adresse du branchement est soit le contenu d'un registre soit la somme de PC et d'un offset. Dans les deux cas, cette valeur peut être rendue disponible à la fin de l'étape DI. Le prix à payer est l'ajout d'un nouvel additionneur dédié à ce calcul. La nouvelle adresse est alors écrite dans le registre PC à la fin de l'étape DI. Le diagramme précédent devient alors le diagramme ci-dessous qui montre qu'il ne reste plus qu'un seul cycle d'horloge perdu.

Instructions	Cycles d'horloge											
	1	2	3	4	5	6	7	8	9	10	11	12
Branchement	EI	DI	EX	MEM	ER							
Inst. suivante n° 1		EI										
Inst. cible n° 1			EI	DI	EX	MEM	ER					
Inst. cible n° 2				EI	DI	EX	MEM	ER				
Inst. cible n° 3					EI	DI	EX	MEM	ER			

En plus, il est probable que le temps pour sélectionner les entrées de l'ALU fasse partie du chemin critique et donc le temps de positionnements des signaux de contrôle des multiplexeurs d'envoi par l'unité d'envoi serait limite. Le matériel serait peut-être plus rapide avec une mise en œuvre différente de l'unité d'envoi qui consisterait à déterminer pendant l'étage DI le contrôle des multiplexeurs d'envoi aux entrées de l'ALU.

D'autre part, la mémoire rapide situé dans le processeur (cache L1) est de petite taille, et la cache L1 ainsi que le processeur sont encore beaucoup plus rapides que la mémoire RAM (jusqu'à 50 fois plus rapide). Lorsque que les données ne sont pas dans la cache L1, le processeur doit aller les chercher dans la mémoire RAM, il y a alors un ralentissement notable. Le processeur doit alors attendre un long moment (par rapport à sa vitesse) pour que la mémoire RAM lui rende l'information, le processeur ne peut alors rien faire d'autre qu'attendre. Ainsi entre la cache rapide L1 (et le processeur) et la mémoire lente RAM, est insérée une deuxième cache, la cache niveau 2 ou L2. Le cache L2 est la mémoire tampon qui permet le stockage d'informations (instructions et données) redondantes (qui se répètent) et l'accès plus rapide à ces dernières. Cette cache est fabriquée à partir de mémoire rapide, mais relativement peu dispendieuse, appelée mémoire statique ou SRAM et est approximativement 10 fois plus rapide que la mémoire RAM standard.

Travaux futurs :

- Placer l'unité d'envoi dans l'étage DI afin d'anticiper le positionnement des signaux de contrôle.
- Placer les branchements conditionnels BEQ et BNE dans l'étage DI.
- Créer un cache L2 pour le cache d'instructions et le cache de données.
- La simulation après la synthèse, le placement et le routage, et la simulation après le placement et le routage.

REFERENCES BIBLIOGRAPHIQUES

[1] Computer Organization & Design. David A. Patterson and John L. Hennessy, ISBN 1-55860-428-6, p 476-501, 525-256.

[2] University of California at Davis Computer Science Museum, http://wwwcsif.cs.ucdavis.edu/~csclub/museum/cpu.html, October 1999.

[3] Advanced Microprocessors, Daniel Tabak, ISBN 0-07-062843-2, p 79-99.

[4] The Practical XILINX Designer Lab Book, Dave Van den Bout, ISBN 0-13-095502-7, p 30-31.

[5] XILINX datasheet library, http://www.xilinx.com/partinfo/4000.pdf, November 1999.

[6] Evaluation of a reconfigurable computing engine for digital communication applications, Jonas Thor, ISSN 1402-1617, p 12-17.

[7] « Architecture des ordinateurs : une approche quantitative » John L. Hennessy, David A. Patterson – Vuibert (3ème édition)

[8] « Conception des circuits en VHDL » Dominique Houzet –Cepadues

[9] http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/Cours/Architectures/risc.html

[10] http://perso.telecom-paristech.fr/~blanchet/mupts/risc.html

[11] http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/Cours/Pipeline/

ANNEXE A1

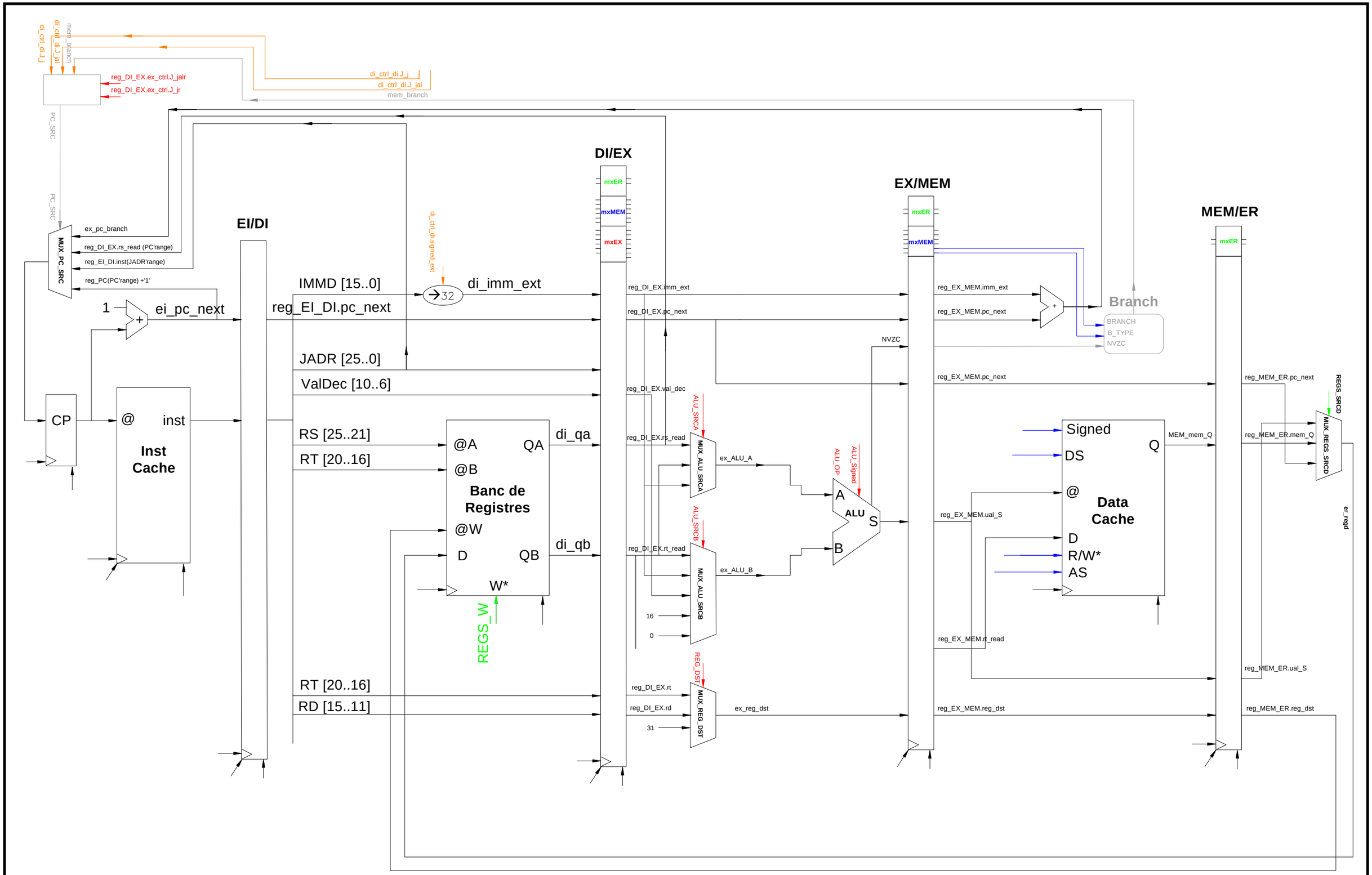


Figure A1.1 : le chemin de données pipeline sans unité de contrôle, unité de détection d'aléas et unité d'envoi

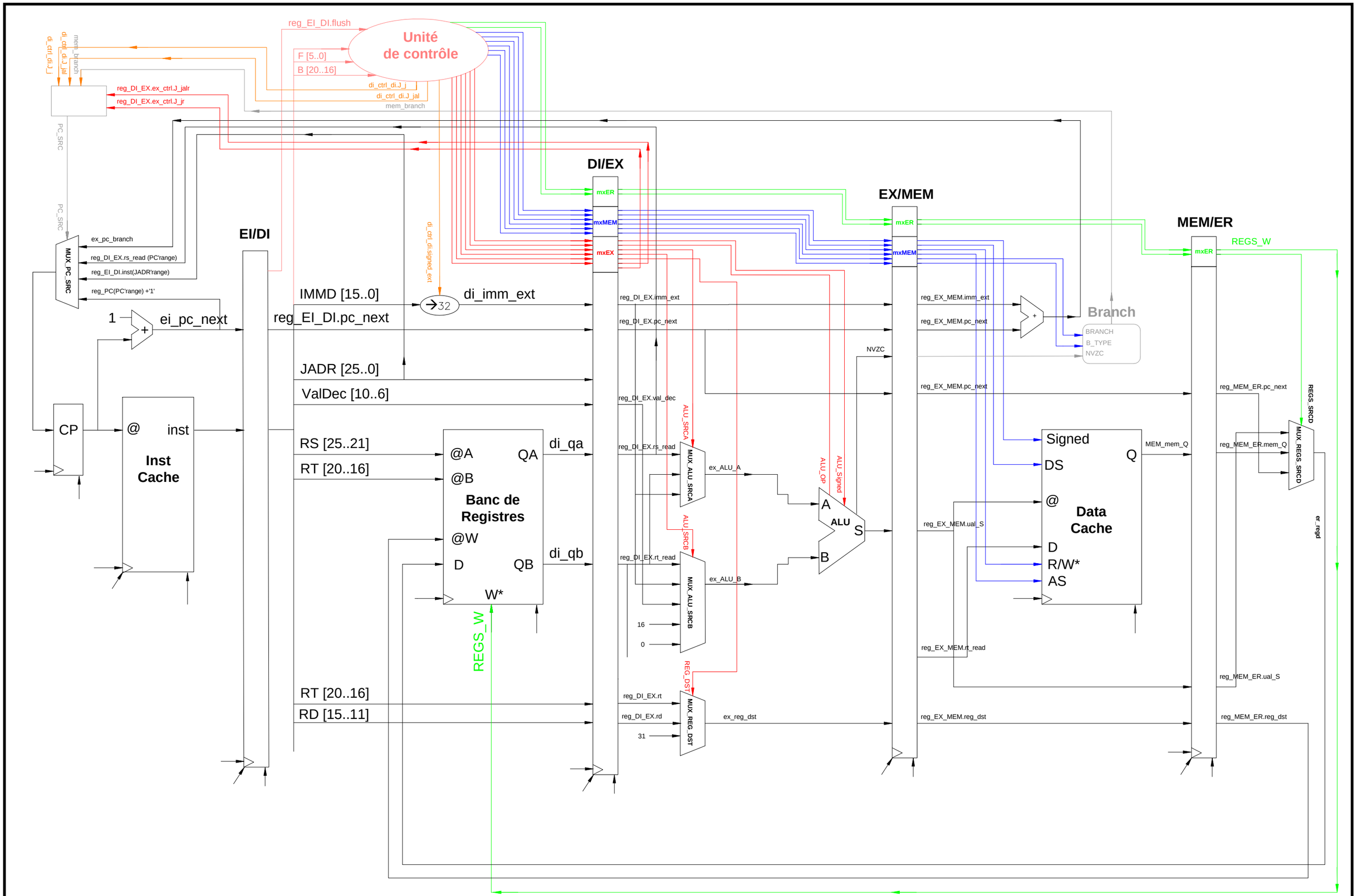


Figure A1.2: le chemin de données pipeline avec unité de contrôle, et sans unité d'envoi et unité de détection d'aléas

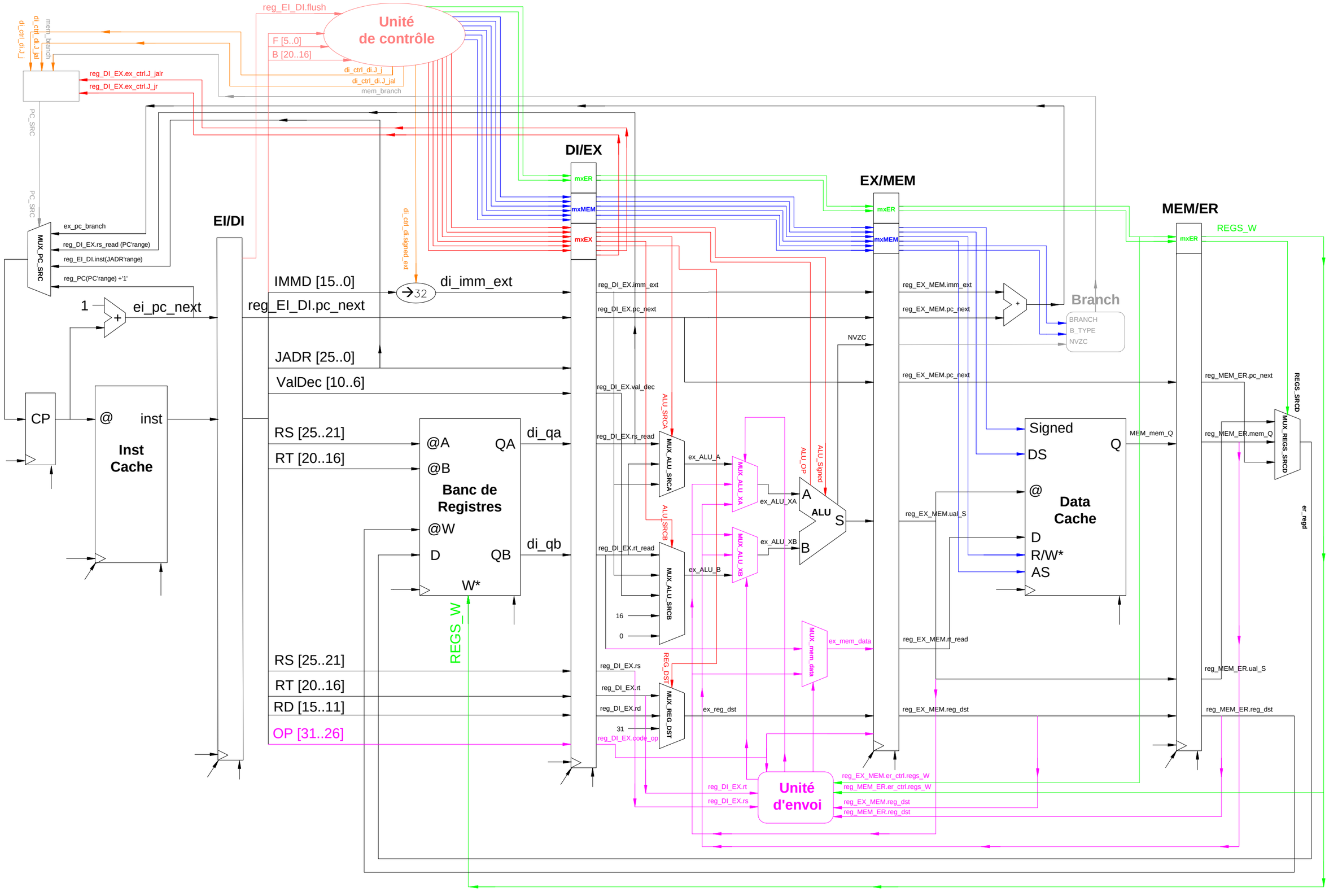


Figure A1.3: Le chemin de données pipeline avec unité de contrôle et unité d'envoi, et sans unité de détection d'aléas

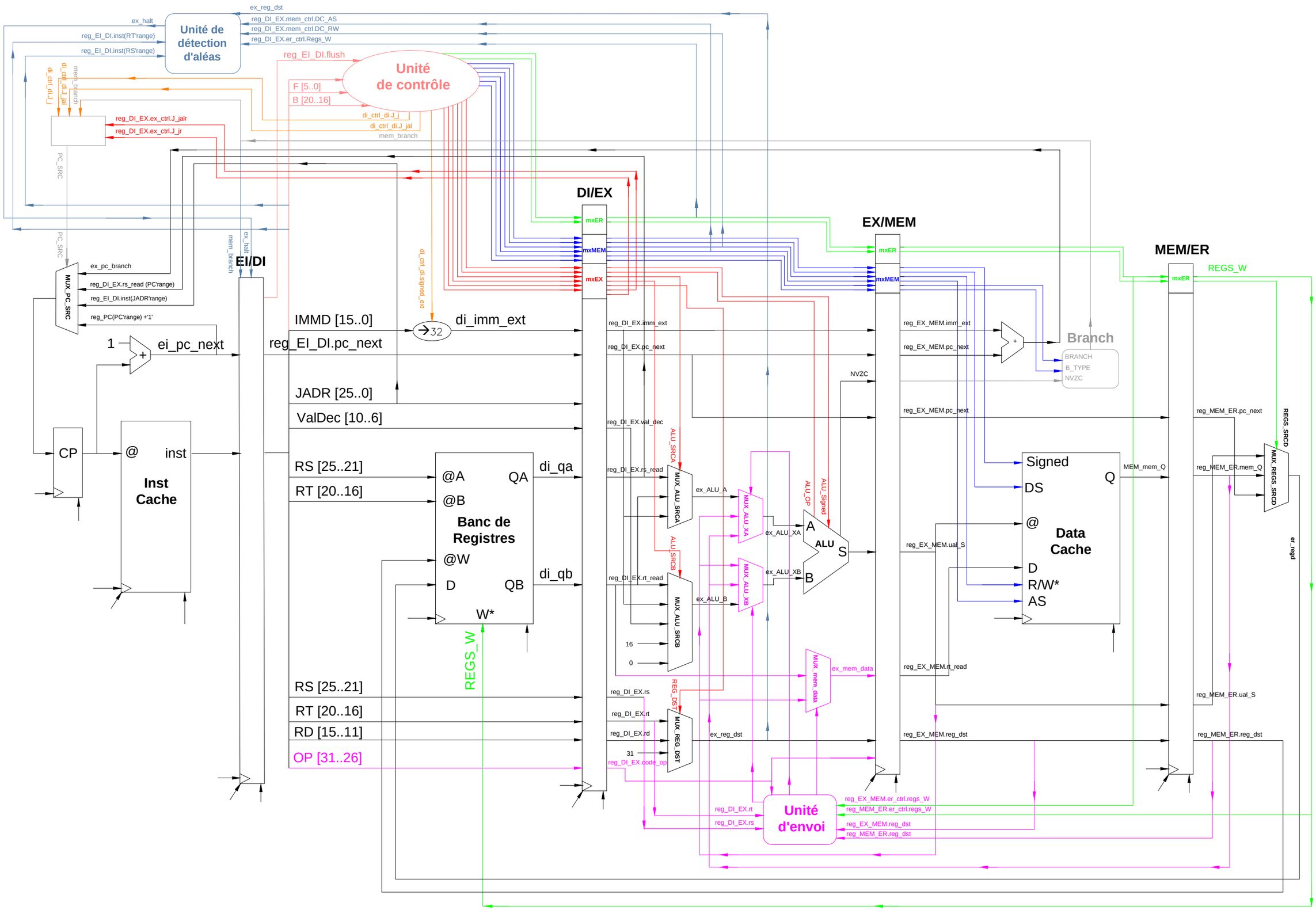


Figure A1.4 : le chemin de données pipeline avec unité de contrôle, unité d'envoi, et unité de détection d'aléas

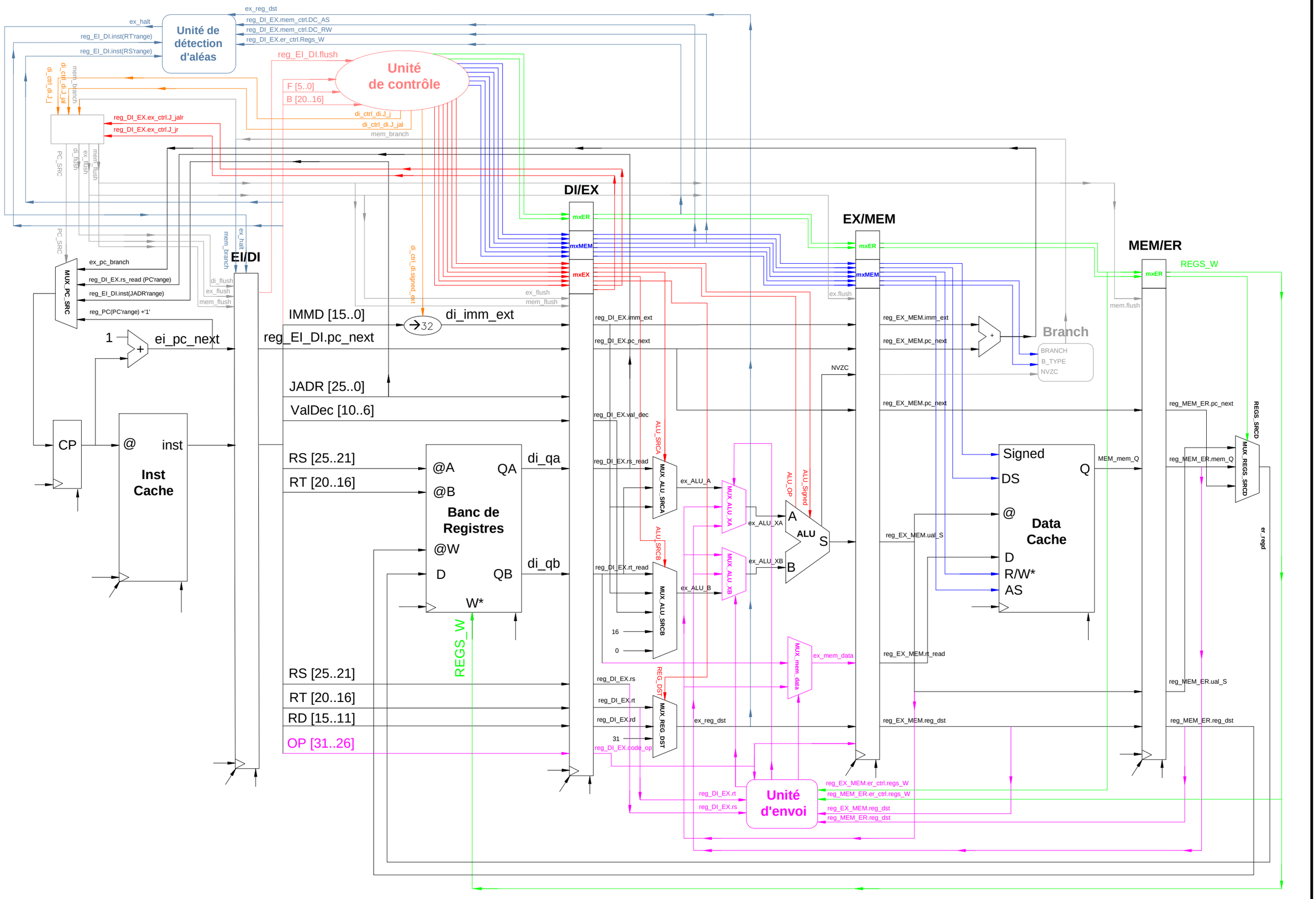


Figure A1.5 : le schéma final de notre processeur RISC mettant en œuvre le contrôle des aléas

Figure A1.6 : le schéma de processeur RISC mettant en œuvre le contrôle des aléas sans suspension du pipeline

ANNEXE A2

Tableau A2.1 :

OPCODE		FCODE	rt / B	Étage DI Lignes de contrôle SIGNED_EXT	J_j	J_jal
TYPE_R "000000"		ADD		0	-	-
		ADDU		0	-	-
		SUB		0	-	-
		SUBU		0	-	-
		iAND		0	-	-
		iOR		0	-	-
		iNOR		0	-	-
		iXOR		0	-	-
		SLT		0	-	-
		SLTU		0	-	-
		LSL		0	-	-
		LSR		0	-	-
		JR		0	-	-
		JALR		0	-	-
TYPE_B "000001"			BLTZ	1	-	-
			BGEZ	1	-	-
			BLTZAL	1	-	-
			BGEZAL	1	-	-
TYPE_J	J			0	1	-
	JAL			0	-	1
TYPE_I	ADDI			1	-	-
	ADDIU			1	-	-
	SLTI			1	-	-
	SLTIU			1	-	-
	ANDI			0	-	-
	ORI			0	-	-
	XORI			0	-	-
	LUI			0	-	-
	LB			1	-	-
	LH			1	-	-
	LW			1	-	-
	LBU			1	-	-
	LHU			1	-	-
	SB			1	-	-
	SH			1	-	-
	SW			1	-	-
	BEQ			1	-	-
	BNE			1	-	-
	BLEZ			1	-	-
	BGTZ			1	-	-

Tableau A2.2 :

OPCODE		FCODE	rt / B	Étage EX						
				Lignes de contrôle						
				ALU_OP	ALU_SIGNED	MUX ALU_SRCA	MUX ALU_SRCB	MUX REG_DST	J_jr	J_jalr
TYPE_R "000000"		ADD		ALU_ADD	1	REGS_QA	REGS_QB	REG_RD	-	-
		ADDU		ALU_ADD	0	REGS_QA	REGS_QB	REG_RD	-	-
		SUB		ALU_SUB	1	REGS_QA	REGS_QB	REG_RD	-	-
		SUBU		ALU_SUB	0	REGS_QA	REGS_QB	REG_RD	-	-
		iAND		ALU_AND	0	REGS_QA	REGS_QB	REG_RD	-	-
		iOR		ALU_OR	0	REGS_QA	REGS_QB	REG_RD	-	-
		iNOR		ALU_NOR	0	REGS_QA	REGS_QB	REG_RD	-	-
		iXOR		ALU_XOR	0	REGS_QA	REGS_QB	REG_RD	-	-
		SLT		ALU_SLT	1	REGS_QA	REGS_QB	REG_RD	-	-
		SLTU		ALU_SLT	0	REGS_QA	REGS_QB	REG_RD	-	-
		LSL		ALU_LSL	0	REGS_QB	VAL_DEC	REG_RD	-	-
		LSR		ALU_LSR	0	REGS_QB	VAL_DEC	REG_RD	-	-
		JR		-	-	REGS_QA	REGS_QB	REG_RD	1	-
		JALR		-	-	REGS_QA	REGS_QB	REG_RD	-	1
TYPE_B "000001"			BLTZ	ALU_SUB	1	REGS_QA	VAL_0	-	-	-
			BGEZ	ALU_SUB	1	REGS_QA	VAL_0	-	-	-
			BLTZAL	ALU_SUB	1	REGS_QA	VAL_0	R31	-	-
			BGEZAL	ALU_SUB	1	REGS_QA	VAL_0	R31	-	-
TYPE_J	J			-	-	-	-	-	-	-
	JAL			-	-	-	-	R31	-	-
TYPE_I	ADDI			ALU_ADD	1	REGS_QA	IMMD	REG_RT	-	-
	ADDIU			ALU_ADD	0	REGS_QA	IMMD	REG_RT	-	-
	SLTI			ALU_SLT	1	REGS_QA	IMMD	REG_RT	-	-
	SLTIU			ALU_SLT	0	REGS_QA	IMMD	REG_RT	-	-
	ANDI			ALU_AND	0	REGS_QA	IMMD	REG_RT	-	-
	ORI			ALU_OR	0	REGS_QA	IMMD	REG_RT	-	-
	XORI			ALU_XOR	0	REGS_QA	IMMD	REG_RT	-	-
	LUI			ALU_LSL	1	IMMD	VAL_16	REG_RT	-	-
	LB			ALU_ADD	1	REGS_QA	IMMD	REG_RT	-	-
	LH			ALU_ADD	1	REGS_QA	IMMD	REG_RT	-	-
	LW			ALU_ADD	1	REGS_QA	IMMD	REG_RT	-	-
	LBU			ALU_ADD	1	REGS_QA	IMMD	REG_RT	-	-
	LHU			ALU_ADD	1	REGS_QA	IMMD	REG_RT	-	-
	SB			ALU_ADD	1	REGS_QA	IMMD	-	-	-
	SH			ALU_ADD	1	REGS_QA	IMMD	-	-	-
	SW			ALU_ADD	1	REGS_QA	IMMD	-	-	-
	BEQ			ALU_SUB	1	REGS_QA	REGS_QB	-	-	-
	BNE			ALU_SUB	1	REGS_QA	REGS_QB	-	-	-
	BLEZ			ALU_SUB	1	REGS_QA	REGS_QB	-	-	-
	BGTZ			ALU_SUB	1	REGS_QA	REGS_QB	-	-	-

Tableau A2.3 :

OPCODE		FCODE	rt / B	Étage MEM					
				Lignes de contrôle					
				DC_AS	DC_SIGNED	DC_DS	DC_RW	BRANCH	MUX B_type
TYPE_R "000000"		ADD		0	-	-	-	0	-
		ADDU		0	-	-	-	0	-
		SUB		0	-	-	-	0	-
		SUBU		0	-	-	-	0	-
		iAND		0	-	-	-	0	-
		iOR		0	-	-	-	0	-
		iNOR		0	-	-	-	0	-
		iXOR		0	-	-	-	0	-
		SLT		0	-	-	-	0	-
		SLTU		0	-	-	-	0	-
		LSL		0	-	-	-	0	-
		LSR		0	-	-	-	0	-
		JR		0	-	-	-	0	-
		JALR		0	-	-	-	0	-
TYPE_B "000001"			BLTZ	0	-	-	-	1	B_bltz
			BGEZ	0	-	-	-	1	B_bgez
			BLTZAL	0	-	-	-	1	B_bltz
			BGEZAL	0	-	-	-	1	B_bgez
TYPE_J	J			0	-	-	-	0	-
	JAL			0	-	-	-	0	-
TYPE_I	ADDI			0	-	-	-	0	-
	ADDIU			0	-	-	-	0	-
	SLTI			0	-	-	-	0	-
	SLTIU			0	-	-	-	0	-
	ANDI			0	-	-	-	0	-
	ORI			0	-	-	-	0	-
	XORI			0	-	-	-	0	-
	LUI			0	-	-	-	0	-
	LB			1	1	MEM_8	1	0	-
	LH			1	1	MEM_16	1	0	-
	LW			1	1	MEM_32	1	0	-
	LBU			1	0	MEM_8	1	0	-
	LHU			1	0	MEM_16	1	0	-
	SB			1	1	MEM_8	0	0	-
	SH			1	1	MEM_16	0	0	-
	SW			1	1	MEM_32	0	0	-
	BEQ			0	-	-	-	1	B_beq
	BNE			0	-	-	-	1	B_bne
	BLEZ			0	-	-	-	1	B_blez
	BGTZ			0	-	-	-	1	B_bgtz

Tableau A2.4 :

OPCODE		FCODE	rt / B	Étage ER			
				Lignes de contrôle			
				REGS_W*	MUX REGS_SRCD		
TYPE_R "000000"		ADD		0	ALU_S		
		ADDU		0	ALU_S		
		SUB		0	ALU_S		
		SUBU		0	ALU_S		
		iAND		0	ALU_S		
		iOR		0	ALU_S		
		iNOR		0	ALU_S		
		iXOR		0	ALU_S		
		SLT		0	ALU_S		
		SLTU		0	ALU_S		
		LSL		0	ALU_S		
		LSR		0	ALU_S		
		JR		1	-		
		JALR		0	NextPC		
TYPE_B "000001"			BLTZ	1	-		
			BGEZ	1	-		
			BLTZAL	0	NextPC		
			BGEZAL	0	NextPC		
TYPE_J	J			1	-		
	JAL			0	NextPC		
TYPE_I	ADDI			0	ALU_S		
	ADDIU			0	ALU_S		
	SLTI			0	ALU_S		
	SLTIU			0	ALU_S		
	ANDI			0	ALU_S		
	ORI			0	ALU_S		
	XORI			0	ALU_S		
	LUI			0	ALU_S		
	LB			0	MEM_Q		
	LH			0	MEM_Q		
	LW			0	MEM_Q		
	LBU			0	MEM_Q		
	LHU			0	MEM_Q		
	SB			1	-		
	SH			1	-		
	SW			1	-		
	BEQ			1	-		
	BNE			1	-		
	BLEZ			1	-		
	BGTZ			1	-		

ANNEXE A3

```
--------------------------------------------------------------------------------
-- Banc de registres
-- THIEBOLT Francois le 05/04/04
-- PHOR Vicheka
--------------------------------------------------------------------------------

----------------------------------------------------------------
-- Par defaut 32 registres de 32 bits avec lecture double port
----------------------------------------------------------------

-- Definition des librairies
library IEEE;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-- Definition de l'entite
entity registres is

        -- definition des parametres generiques
        generic (
                -- largeur du bus de donnees par defaut
                DBUS_WIDTH      : integer := 32; -- registre de 32 bits par
defaut


                -- largeur du bus adr pour acces registre soit 32 (2**5) par
defaut
                ABUS_WIDTH      : integer := 5;

                -- definition du front actif d'ecriture par defaut
                ACTIVE_FRONT: std_logic := '1' );

        -- definition des entrees/sorties
        port    (
                -- signaux de controle du Banc de registres
                CLK                     : in std_logic;
                W,RST                   : in std_logic; -- actifs a l'etat
bas


                -- bus d'adresse et donnees
                ADR_A,ADR_B     : in std_logic_vector(ABUS_WIDTH-1 downto 0);
                ADR_W                   : in std_logic_vector(ABUS_WIDTH-1
downto 0);
                D                               : in
std_logic_vector(DBUS_WIDTH-1 downto 0);


                -- Ports de sortie
                QA,QB                   : out std_logic_vector(DBUS_WIDTH-1
downto 0) );


end registres;


--------------------------------------------------------------------------------
-- REGISTRES architecture
```

```
--------------------------------------------------------------------------------

-- Definition de l'architecture du banc de registres
architecture behavior of registres is

        -- definitions de types (index type default is integer)
        type FILE_REGS is array (0 to (2**ABUS_WIDTH)-1) of std_logic_vector
(DBUS_WIDTH-1 downto 0);


        -- definition des ressources internes
        signal REGS : FILE_REGS; -- le banc de registres
        signal D2QA     : boolean; -- si acces simultane R & W sur le meme
registre ==> QA <= D;
        signal D2QB     : boolean; -- si acces simultane R & W sur le meme
registre ==> QB <= D;


begin
-------------------------------------------
-- Affectations dans le domaine combinatoire
--

---------------------------------
-- test si acces simultanes
D2QA <= TRUE When (ADR_A = ADR_W and W = '0') else
        FALSE;
D2QB <= TRUE When (ADR_B = ADR_W and W = '0') else
        FALSE;

QA <= (others => 'X') when is_x(ADR_A) else
        (others => '0')
                when ADR_A = conv_std_logic_vector(0,ABUS_WIDTH) else
        REGS(conv_integer(ADR_A))
                when not(D2QA) else
        D;

--when (D2QA = TRUE and ADR_A /= conv_std_logic_vector(0,ABUS_WIDTH) and
not(is_x(ADR_A))) else
--      REGS(conv_integer(ADR_A));

QB <= D when (D2QB and ADR_B /= conv_std_logic_vector(0,ABUS_WIDTH) and
not(is_x(ADR_B))) else
        REGS(conv_integer(ADR_B))
                when (not(D2QB) and ADR_B /=
conv_std_logic_vector(0,ABUS_WIDTH) and not(is_x(ADR_B))) else
        (others=>'0')
                when not(is_x(ADR_B)) else
        (others => 'X');

-------------------
-- Process P_ReadQA
--P_ReadQA: process(ADR_A,D2QA,D)
--begin
        -- test validite adresse
--      if not( is_x(ADR_A) ) then
                -- test si acces R0
--              if (ADR_A /= conv_std_logic_vector(0,ABUS_WIDTH)) then
                        -- test si acces R & W simultane
--                      if (D2QA = FALSE) then
--                              QA <= REGS(conv_integer(ADR_A));
```

```
--                      else -- on envoie la donnee a ecrire
--                              QA <= D;
--                      end if;
--              else -- acces R0
--                      QA <= (others => '0');
--              end if;
--      else -- X
--              QA <= (others => 'X');
--      end if;
--end process P_ReadQA;

-------------------
-- Process P_ReadQB
--P_ReadQB: process(ADR_B,D2QB,D)
--begin
        -- test validite adresse
--      if not( is_x(ADR_B) ) then
--              -- test si acces R0
--              if (ADR_B /= conv_std_logic_vector(0,ABUS_WIDTH)) then
--                      -- test si acces R & W simultane
--                      if (D2QB = FALSE) then
--                              QB <= REGS(conv_integer(ADR_B));
--                      else -- on envoie la donnee a ecrire
--                              QB <= D;
--                      end if;
--              else -- acces R0
--                      QB <= (others => '0');
--              end if;
--      else -- X
--              QB <= (others => 'X');
--      end if;
--end process P_ReadQB;

------------------
-- Process P_WRITE
P_WRITE: process(CLK)
begin
        -- test du front actif d'horloge
        if (CLK'event and CLK='1') then
                -- test du reset
                if RST='0' then
                        REGS <= (others =>
conv_std_logic_vector(0,DBUS_WIDTH));
                else
                        -- test si ecriture dans le registre
                        if ((W='0') and ADR_W /= conv_std_logic_vector(0,
ABUS_WIDTH)) then
                                REGS(conv_integer(ADR_W)) <= D;
                        end if;
                end if;
        end if;
end process P_WRITE;

end behavior;
```

```
--------------------------------------
-- Fichier de test de Banc de registres
-- THIEBOLT Francois le 20/11/02
-- PHOR Vicheka
--------------------------------------

-- Definition des librairies
library IEEE;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-- Definition de l'entite
entity test_registres is
end test_registres;

-- Definition de l'architecture
architecture behavior of test_registres is

-- definition des constantes de test
        constant S_DATA : positive:=32; -- taille du bus de donnes
        constant S_ADR          : positive:=3; -- taille du bus d'adresse
        constant WFRONT : std_logic := '1'; -- front actif pour ecriture
        constant TIMEOUT        : time := 150 ns; -- timeout de la simulation

-- definition de constantes
constant clkpulse : Time := 5 ns; -- 1/2 periode horloge

-- definition de types

-- definition de ressources internes

-- definition de ressources externes
signal E_CLK                                                    : std_logic;
signal E_RST,E_W                                                : std_logic;
-- actifs a l'etat bas
signal E_ADR_A,E_ADR_B,E_ADR_W  : std_logic_vector(S_ADR-1 downto 0);
signal E_QA,E_QB,E_D                                    :
std_logic_vector(S_DATA-1 downto 0);


begin

-------------------------
-- definition de l'horloge
P_E_CLK: process
begin
        E_CLK <= '1';
        wait for clkpulse;
        E_CLK <= '0';
        wait for clkpulse;
end process P_E_CLK;

----------------------------------------
-- definition du timeout de la simulation
P_TIMEOUT: process
begin
        wait for TIMEOUT;
        assert FALSE report "SIMULATION TIMEOUT!!!" severity FAILURE;
```

```
end process P_TIMEOUT;

-------------------------------------------------
-- instantiation et mapping du composant registres
regf0 : entity work.registres(behavior)
                                        generic map (S_DATA,S_ADR,WFRONT)
                                        port map (CLK => E_CLK,
                                                                W => E_W,
                                                                RST =>
E_RST,
                                                                D => E_D,
                                                                ADR_A =>
E_ADR_A,
                                                                ADR_B =>
E_ADR_B,
                                                                ADR_W =>
E_ADR_W,
                                                                QA => E_QA,
                                                                QB => E_QB);

-----------------------------
-- debut sequence de test
P_TEST: process
begin

        -- initialisations
        E_RST <= '0';
        E_ADR_A <= (others=>'X');
        E_ADR_B <= (others=>'X');
        E_ADR_W <= (others=>'X');
        E_D <= (others=>'X');
        E_W <= '1';

        -- sequence RESET
        E_RST <= '0';
        wait for clkpulse*3;
        E_RST <= '1';
        wait for clkpulse;

        -- ecriture dans registre2
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR_W <= conv_std_logic_vector(2,S_ADR);
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"2222FFFF"));
        E_W <= '0';

        -- ecriture dans registre3
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR_W <= conv_std_logic_vector(3,S_ADR);
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"33FF33FF"));
        E_W <= '0';

        -- ecriture dans registre0
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR_W <= conv_std_logic_vector(0,S_ADR);
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"FFFF0000"));
        E_W <= '0';

        -- ecriture dans registre4 et
        -- lectures registres 0 et 3 sur respectivement QA et QB
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR_W <= conv_std_logic_vector(4,S_ADR);
```

```
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"4F4F4F4F"));
        E_W <= '0';
        E_ADR_A <= conv_std_logic_vector(0,S_ADR);
        E_ADR_B <= conv_std_logic_vector(3,S_ADR);

        -- tests
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_W <= '1';
        E_ADR_A <= (others => 'X');
        E_ADR_B <= (others => 'X');
        E_ADR_W <= (others => 'X');
        E_D <= (others => 'X');
        assert E_QA = conv_std_logic_vector(0,S_DATA)
                report "Register 0 BAD VALUE"
                severity ERROR;
        assert E_QB = to_stdlogicvector(BIT_VECTOR'(X"33FF33FF"))
                report "Register 3 BAD VALUE"
                severity ERROR;

        -- ecriture dans registre5 et lecture registre 5
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR_W <= conv_std_logic_vector(5,S_ADR);
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"F5F5F5F5"));
        E_W <= '0';
        E_ADR_A <= conv_std_logic_vector(5,S_ADR);

        -- tests de la lecture asynchrone sur l'autre front
        wait until (E_CLK=not(WFRONT));
        assert E_QA = to_stdlogicvector(BIT_VECTOR'(X"F5F5F5F5"))
                report "Register 5 BAD VALUE"
                severity WARNING;

        -- NOP
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_W <= '1';
        E_ADR_A <= (others => 'X');
        E_ADR_B <= (others => 'X');
        E_ADR_W <= (others => 'X');
        E_D <= (others => 'X');
        -- ADD NEW SEQUENCE HERE

        -- LATEST COMMAND (NE PAS ENLEVER !!!)
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        assert FALSE report "FIN DE SIMULATION" severity FAILURE;

end process P_TEST;

end behavior;
```

```
-----------------------------------
-- Banc Memoire pour processeur RISC
-- PHOR Vicheka
-----------------------------------

-------------------------------------------------------
-- Lors de la phase RESET, permet la lecture d'un fichier
-- passe en parametre generique.
-------------------------------------------------------

-----------------------------------------------------------------
-- Ne s'agissant pas encore d'un cache, le signal Ready est cable
-- a 1 puisque toute operation s'execute en un seul cycle.
--      Ceci est la version avec lecture ASYNCHRONE pour une
--      integration plus simple dans le pipeline.
-- Si la lecture du fichier d'initialisation ne couvre pas tous
--      les mots memoire, ceux-ci seront initialises a 0
-----------------------------------------------------------------

-- Definition des librairies
library IEEE;
library STD;
library WORK;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use WORK.cpu_package2_1.all;
use WORK.cpu_package.all;

-- Definition de l'entite
entity memory is

        -- definition des parametres generiques
        generic (
                -- largeur du bus de donnees par defaut
                DBUS_WIDTH : natural := 32;

                -- largeur du bus adr par defaut
                ABUS_WIDTH : natural := 32;

                -- nombre d'elements dans le cache exprime en nombre de mots
                MEM_SIZE : natural := 32;  -- 16

                -- front actif par defaut
                ACTIVE_FRONT : std_logic := '1';

                -- fichier d'initialisation
                FILENAME : string := "" );

        -- definition des entrees/sorties
        port    (
                -- signaux de controle du cache
                RST                     : in std_logic; -- actifs a l'etat
bas
                CLK,RW          : in std_logic; -- R/W*
                DS                              : in MEM_DS;            --
acces octet, demi-mot, mot...
```

```
                Signed          : in std_logic;  -- extension de signe
                AS                              : in std_logic;  -- Address
Strobe (sorte de CS*)
                Ready                   : out std_logic;-- indicateur
HIT/MISS
                Berr                    : out std_logic;-- bus error (acces
non aligne par exemple), active low


                -- bus d'adresse du cache
                ADR                     : in std_logic_vector(ABUS_WIDTH-1
downto 0);


                -- Ports entree/sortie du cache
                D                               : in
std_logic_vector(DBUS_WIDTH-1 downto 0);
                Q                               : out
std_logic_vector(DBUS_WIDTH-1 downto 0) );


end memory;

-- Definition de l'architecture du banc de registres
architecture behavior of memory is

        -- definition de constantes
        constant BITS_FOR_BYTES : natural := log2(DBUS_WIDTH/8); -- nb bits
adr pour acceder aux octets d'un mot
        constant BITS_FOR_WORDS : natural := log2(MEM_SIZE); -- nb bits adr
pour acceder aux mots du cache
        constant BYTES_PER_WORD : natural := (DBUS_WIDTH/8); -- nombre
d'octets par mot


        -- definitions de types (index type default is integer)
        subtype BYTE is std_logic_vector(7 downto 0); -- definition d'un octet
        type WORD is array (0 to BYTES_PER_WORD-1) of BYTE; -- definition d'un
mot composé d'octets


        type FILE_REGS is array (0 to MEM_SIZE-1) of WORD;
        subtype I_ADR is std_logic_vector(BITS_FOR_WORDS-1+BITS_FOR_BYTES
downto BITS_FOR_BYTES); -- internal ADR au format mot du cache


        subtype B_ADR is std_logic_vector(BITS_FOR_BYTES-1 downto 0); -- byte
ADR pour manipuler les octets dans le mot


    -- range ought to be O to 2**BITS_FOR_BYTES-1 mais lorsque l'on affecte
nb_bytes=4 on déborde !
    subtype byte_adr is natural range 0 to (2**BITS_FOR_BYTES); --
manipulation d'octets dans les mots


        -- definition de la fonction de chargement d'un fichier
        --              on peut egalement mettre cette boucle dans le process
qui fait les ecritures
        function LOAD_FILE (F : in string) return FILE_REGS is
                variable temp_REGS : FILE_REGS;
```

```
                file mon_fichier : TEXT open READ_MODE is STRING'(F); --
VHDL93 compliant
                --      file mon_fichier : TEXT is in STRING'(F); -- older
implementation
                variable line_read : line := null;
                variable line_value : std_logic_vector (DBUS_WIDTH-1 downto
0);
                variable index,i : natural := 0;
        begin
                -- lecture du fichier
                index:=0;
                while (not ENDFILE(mon_fichier) and (index < MEM_SIZE))
                loop
                        readline(mon_fichier,line_read);
                        read(line_read,line_value);
                        for i in 0 to BYTES_PER_WORD-1 loop
                                temp_REGS(index)(i):=line_value(((i+1)*8)-1
downto i*8);
                        end loop;
--                      temp_REGS(index):=line_value;
                        index:=index+1;
                end loop;
                -- test si index a bien parcouru toute la memoire
                if (index < MEM_SIZE) then
                        temp_REGS(index to MEM_SIZE-1):=(others => (others =>
(others => '0')));
                end if;
                -- renvoi du resultat
                return temp_REGS;
        end LOAD_FILE;

        -- definition des ressources internes
        signal REGS : FILE_REGS; -- le banc memoire

        -- l'adressage de la memoire se faisant par element de taile
DBUS_WIDTH, par rapport
        -- au bus d'adresse au format octet il faut enlever les bits d'adresse
de poids faible
        -- (octets dans le mot), puis prendre les bits utiles servant a
l'acces des mots du cache.
        -- ex.: mots de 32 bits => 2 bits de poids faible pour les octets dans
le mot
        --              16 mots memoire => 4 bits necessaire
        -- D'ou I_ADR = ADR (4+2-1 downto 2)

begin
-------------------------------------------------------------------
-- Affectations dans le domaine combinatoire
--

-- Indicateur acces MISS/HIT
Ready <= '1'; -- car pas encore un cache

-------------------------------------------------------------------
-- Process P_CACHE
--      La lecture etant asynchrone c.a.d qu'elle ne depend que des
--              signaux d'entree, nous sommes obliges de les mettre dans la
--              liste de sensitivite du process
P_CACHE: process(CLK,RST,ADR,AS,RW,DS,Signed)
        variable byte_low : B_ADR; -- octet de départ
        variable nb_bytes : byte_adr; -- nombre d'octets concernés par R/W*
```

```
        variable internal_adr : I_ADR;
        variable sign_bit : std_logic;
        variable tmp_berr : std_logic;
begin
        -- test du reset
        if ( is_X(RST) or (RST='0') ) then
                -- affectation des REGS
                if (STRING'(FILENAME) /= "") then
                        REGS <= LOAD_FILE(FILENAME);
                else REGS <= (others => (others => (others => '0')));
                end if;
                -- affectation de la sortie
                Q <= (others => 'Z');
        elsif (AS='1') then
                -- evaluation de l'adresse interne
                internal_adr := ADR(I_ADR'range);
                -- evaluation de l'octet de départ
                byte_low := ADR(B_ADR'range);

                -- switch sur la taille de la transaction (nombre d'octets) et
test du BERR
                tmp_berr:='1'; -- tous octets accessibles
                case DS is
                        when MEM_8 =>
                                nb_bytes:= 1;
                        when MEM_16 =>
                                nb_bytes:= 2;
                                if ( ADR(0) /= '0' ) then
                                        tmp_berr:='0';
                                end if;
                        when MEM_32 =>
                                nb_bytes:= 4;
                                if ( ADR(1 downto
0)/=conv_std_logic_vector(0,2) ) then
                                        tmp_berr:='0';
                                end if;
                        when MEM_64 =>
                                nb_bytes:= 8;
                                if ( ADR(2 downto
0)/=conv_std_logic_vector(0,3) ) then
                                        tmp_berr:='0';
                                end if;
                        when others =>
                                assert FALSE report "MEM_DS operation not yet
implemented !" severity FAILURE;
                end case;

                -- affectation du signal BERR
                berr<=tmp_berr;
                -- affectation du bus de sortie
                Q <= ( others => 'Z' );

                -- test front actif d'horloge et ecriture
                if (CLK'event and CLK=ACTIVE_FRONT and RW='0' and
tmp_berr='1') then
                        for i in 0 to nb_bytes-1 loop
                                REGS(conv_integer(internal_adr))( i +
conv_integer(byte_low)) <= D(((i+1)*8)-1 downto i*8);
                        end loop;
                end if;
```

```
                -- test si lecture
                if (RW='1' and tmp_berr='1') then
                        -- affectation de la sortie
                        Q <= ( others => '0' );
                        -- affectation de la donnee
                        for i in 0 to nb_bytes-1 loop
                                Q(((i+1)*8)-1 downto i*8) <=
REGS(conv_integer(internal_adr))(i + conv_integer(byte_low));
                        end loop;
                        -- operation signee ?
                        if (Signed = '1') then

sign_bit:=REGS(conv_integer(internal_adr))(nb_bytes-1 +
conv_integer(byte_low))(7);
                                Q(DBUS_WIDTH-1 downto nb_bytes*8) <= (others
=> sign_bit);
                        end if;
                end if;
        else -- AS inactif
                Q <= (others => 'Z');
        end if;
end process P_CACHE;

end behavior;
```

```
-----------------------------------
-- Fichier de test pour Banc Memoire
-- THIEBOLT Francois le 08/12/04
-----------------------------------

-- Definition des librairies
library IEEE;
library WORK;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.cpu_package2_1.all;
use WORK.cpu_package.all;

-- Definition de l'entite
entity test_memory is
end test_memory;

-- Definition de l'architecture
architecture behavior of test_memory is

-- definition de constantes de test
        constant S_DATA : positive := CPU_DATA_WIDTH; -- taille du bus de
donnees
        constant S_ADR          : positive := CPU_ADR_WIDTH; -- taille du bus
d'adresse
        constant S_L1           : positive := L1_SIZE; -- taille du cache L1
en nombre de mots
--      constant S_L1           : positive := 32; -- taille du cache L1 en
nombre de mots
        constant WFRONT : std_logic := L1_FRONT; -- front actif pour ecriture
--      constant FILENAME : string := ""; -- init a 0 par defaut
        constant FILENAME : string := "rom_file.0.txt"; -- init par fichier
        constant TIMEOUT        : time := 200 ns; -- timeout de la simulation

-- definition de constantes
constant clkpulse : Time := 5 ns; -- 1/2 periode horloge

-- definition de types

-- definition de ressources internes

-- definition de ressources externes
signal E_RST                            : std_logic; -- actif a l'etat bas
signal E_CLK,E_RW                       : std_logic;
signal E_DS                                     : MEM_DS;
signal E_Signed,E_AS            : std_logic;
signal E_Ready,E_Berr   : std_logic;
signal E_ADR                            : std_logic_vector(S_ADR-1 downto 0);
-- bus adresse au format octet !
signal E_D,E_Q                          : std_logic_vector(S_DATA-1 downto
0);


begin

--------------------------
-- definition de l'horloge
P_E_CLK: process
```

```
begin
        E_CLK <= '1';
        wait for clkpulse;
        E_CLK <= '0';
        wait for clkpulse;
end process P_E_CLK;

----------------------------------------
-- definition du timeout de la simulation
P_TIMEOUT: process
begin
        wait for TIMEOUT;
        assert FALSE report "SIMULATION TIMEOUT!!!" severity FAILURE;
end process P_TIMEOUT;

----------------------------------------
-- instanciation et mapping de composants
L1 : entity work.memory(behavior)
                        generic map (S_DATA,S_ADR,S_L1,WFRONT,FILENAME)
                        port map (RST => E_RST, CLK => E_CLK, RW => E_RW,
                                                                DS => E_DS, Signed =>
E_Signed, AS => E_AS,
                                                                Ready => E_Ready,
Berr => E_Berr,
                                                                ADR => E_ADR, D =>
E_D, Q => E_Q );


----------------------------
-- debut sequence de test
P_TEST: process
begin

        -- initialisations
        E_RST <= '0';
        E_RW <= '1';
        E_DS <= MEM_DS'low;
        E_Signed <= '0';
        E_AS <= '0';
        E_ADR <= (others => 'X');
        E_D <= (others => 'X');

        -- sequence RESET
        E_RST <= '0';
        wait for clkpulse*3;
        E_RST <= '1';
        wait for clkpulse;

        -- ecriture octet a l'octet 1
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR <= conv_std_logic_vector(1,S_ADR);
        E_DS <= MEM_8;
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"FFFFFFAA"));
        E_RW <= '0';
        E_AS <= '1';
        E_Signed <= '0';

        -- ecriture demi-mot a l'octet 2
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR <= conv_std_logic_vector(2,S_ADR);
        E_DS <= MEM_16;
```

```
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"FFFFBBBB"));
        E_RW <= '0';
        E_AS <= '1';
        E_Signed <= '0';

        -- ecriture mot a l'octet 4
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR <= conv_std_logic_vector(4,S_ADR);
        E_DS <= MEM_32;
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"CCCCCCCC"));
        E_RW <= '0';
        E_AS <= '1';
        E_Signed <= '0';

        -- ecriture octet a l'octet 6
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_ADR <= conv_std_logic_vector(6,S_ADR);
        E_DS <= MEM_8;
        E_D <= to_stdlogicvector(BIT_VECTOR'(X"FFFFFFDD"));
        E_RW <= '0';
        E_AS <= '1';
        E_Signed <= '0';

        -- lecture octet a l'octet 2
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_D <= (others => 'X'); -- disable Datas on DataBus
        E_RW <= '1';
        E_ADR <= conv_std_logic_vector(2,S_ADR);
        E_DS <= MEM_8;
        E_AS <= '1';
        E_Signed <= '0';

        -- tests & lecture demi-mot a l'octet 6
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        assert E_Q = to_stdlogicvector(BIT_VECTOR'(X"000000BB"))
                report "Memory 2 BAD VALUE"
                severity ERROR;
        E_RW <= '1';
        E_AS <= '1';
        E_Signed <= '0';
        E_ADR <= conv_std_logic_vector(6,S_ADR);
        E_DS <= MEM_16;

        -- tests & lecture mot a l'octet 0
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        assert E_Q = to_stdlogicvector(BIT_VECTOR'(X"0000CCDD"))
                report "Memory 6 BAD VALUE"
                severity ERROR;
        E_RW <= '1';
        E_AS <= '1';
        E_Signed <= '0';
        E_ADR <= conv_std_logic_vector(0,S_ADR);
        E_DS <= MEM_32;

        -- tests & lecture octet signe a l'adresse 6
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        assert E_Q = to_stdlogicvector(BIT_VECTOR'(X"BBBBAA00"))
                report "Memory 0 BAD VALUE"
                severity ERROR;
        E_RW <= '1';
        E_AS <= '1';
```

```
        E_Signed <= '1';
        E_ADR <= conv_std_logic_vector(6,S_ADR);
        E_DS <= MEM_8;

        -- tests
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_RW <= '1';
        E_AS <= '0';
        E_Signed <= '0';
        assert E_Q = to_stdlogicvector(BIT_VECTOR'(X"FFFFFFDD"))
                report "Memory 6 BAD VALUE"
                severity ERROR;

        -- erreur de lecture demi-mot a l'adresse 3
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_RW <= '1';
        E_AS <= '1';
        E_Signed <= '0';
        E_ADR <= conv_std_logic_vector(3,S_ADR);
        E_DS <= MEM_16;

        -- tests
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        E_RW <= '1';
        E_AS <= '0';
        E_Signed <= '0';
        assert E_Berr = '0'
                report "Berr not reporting error !"
                severity ERROR;

        -- ADD NEW SEQUENCE HERE

        -- LATEST COMMAND (NE PAS ENLEVER !!!)
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        assert FALSE report "FIN DE SIMULATION" severity FAILURE;
        -- assert (NOW < TIMEOUT) report "FIN DE SIMULATION" severity FAILURE;

end process P_TEST;

end behavior;
```

```
--------------------------------------------------------------------------------
-- RISC processor general definitions
-- THIEBOLT Francois le 08/03/04
-- PHOR Vicheka
--------------------------------------------------------------------------------

-- library definitions
library IEEE;

-- library uses
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

--
--------------------------------------------------------------------------------
-- the package contains types, constants, and function prototypes
--
--------------------------------------------------------------------------------
package cpu_package2_1 is


-- ================================================================
-- DEFINITION DE FONCTIONS/PROCEDURES de base
-- ================================================================

        -- Fonction log2
        --              calcule le logarithme base2 d'un entier naturel, ou
plus exactement
        --              renvoie le nombre de bits necessaire pour coder un
entier naturel I
        function log2 (I: in natural) return natural;

-- ================================================================
-- TYPES/CONSTANT DEFINITIONS
-- ================================================================


end cpu_package2_1;

--
--------------------------------------------------------------------------------
-- the package contains types, constants, and function prototypes
--
--------------------------------------------------------------------------------
package body cpu_package2_1 is

-- ================================================================
-- DEFINITION DE FONCTIONS/PROCEDURES
-- ================================================================

-- fonction log2
function log2 (I: in natural) return natural is
        variable ip : natural := 1; -- valeur temporaire
        variable iv : natural := 0; -- nb de bits
begin
        while ip < i loop
                ip := ip + ip; -- ou ip := ip * 2
                iv := iv + 1;
        end loop;
        -- renvoie le nombre de bits
```

```
        return iv;
end log2;


end cpu_package2_1;
```

```
--------------------------------------------------------------------------------
-- RISC processor general definitions
-- PHOR Vicheka
-- THIEBOLT Francois
--------------------------------------------------------------------------------

-- library definitions
library IEEE;

-- library uses
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.cpu_package2_1.all;
--
--------------------------------------------------------------------------------
-- the package contains types, constants, and function prototypes
--
--------------------------------------------------------------------------------
package cpu_package is


-- ================================================================
-- DEFINITION DE FONCTIONS/PROCEDURES de base
-- ================================================================

  -- Fonction log2
        --      calcule le logarithme base2 d'un entier naturel, ou plus
exactement
        --      renvoie le nombre de bits necessaire pour coder un entier
naturel I
        --function log2 (I: in natural) return natural;

-- ================================================================
-- TYPES/CONSTANT DEFINITIONS
-- ================================================================

-------------------------------------------------------------------
-- HARDWARE definitions
-------------------------------------------------------------------
        -- define CPU core physical sizes
        constant CPU_DATA_WIDTH    : positive := 32;                    --
data bus width
        constant CPU_INST_WIDTH    : positive := CPU_DATA_WIDTH; --
instruction bus width
        constant CPU_ADR_WIDTH     : positive := 32;                    --
address bus width, byte format


        -- define MISC CPU CORE specs
        constant CPU_WR_FRONT        : std_logic:= '1';                 --
pipes write active front
        constant PC_WIDTH                : positive     := 26;
     -- bits pour le PC format mot memoire
        constant PCLOW                     : positive   :=
log2(CPU_INST_WIDTH/8);


        -- define REGISTERS physical sizes
        constant REG_WIDTH         : positive   := 5;
-- registers address bus with
```

```
        constant REG_FRONT          : std_logic   := CPU_WR_FRONT;

        -- define instruction & data CACHE physical sizes
        constant L1_SIZE            : positive    := 32;    --16         --
taille des caches L1 en nombre de mots
        constant L1_ISIZE           : positive    := L1_SIZE;        -- taille
du cache instruction L1 en nombre de mots
        constant L1_DSIZE           : positive    := L1_SIZE;        -- taille
du cache donnees L1 en nombre de mots
        constant L1_FRONT           : std_logic   := CPU_WR_FRONT;

        -- define types/subtypes according to hardware specs
        subtype PC     is std_logic_vector(PC_WIDTH-1+PCLOW downto PCLOW);
        subtype INST   is std_logic_vector(CPU_INST_WIDTH-1 downto 0);
        subtype ADDR   is std_logic_vector(CPU_ADR_WIDTH-1 downto 0);
        subtype DATA   is std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
        subtype REGS   is std_logic_vector(REG_WIDTH-1 downto 0);

        -- define default values
        constant PC_DEFL : ADDR := conv_std_logic_vector(0,ADDR'length);

------------------------------------------------------------------
-- SOFTWARE definitions
------------------------------------------------------------------
        -- define the basic ALU operations
        --      Le fait qu'une operation soit signee ou non sera indique a
l'ALU par un signal
        --      supplementaire, ceci dit cela n'affecte que les bits d'etat.
        type ALU_OPS is
(ALU_ADD,ALU_SUB,ALU_AND,ALU_OR,ALU_NOR,ALU_XOR,ALU_SLT,ALU_LSL,ALU_LSR);


        -- define the size of datas during memory access
        type MEM_DS is (MEM_8,MEM_16,MEM_32,MEM_64);

        -- definition des champs d'instructions
        subtype OPCODE          is std_logic_vector(31 downto 26);
        subtype RS       is std_logic_vector(25 downto 21);
        subtype RT                  is std_logic_vector(20 downto 16);
        subtype RD       is std_logic_vector(15 downto 11);
        subtype VALDEC   is std_logic_vector(10 downto 6);
        subtype FCODE             is std_logic_vector(5 downto 0);
        subtype BCODE             is std_logic_vector(20 downto 16);
        subtype IMM                 is std_logic_vector(15 downto 0);
        subtype JADR               is std_logic_vector(25 downto 0);

        ------------------------------------------------------------------
        -- definition des constantes de type des instructions

        ------------------------------------------------------------------
        -- Instructions de type R : Registre Inst[OPCODE'range]
        constant TYPE_R : std_logic_vector := "000000" ;

        -- Fonctions associes au TYPE_R Inst[FCODE'range]
        constant ADD            : std_logic_vector := "100000" ;
        constant ADDU   : std_logic_vector := "100001" ;
        constant SUB            : std_logic_vector := "100010" ;
        constant SUBU   : std_logic_vector := "100011" ;
        constant iAND   : std_logic_vector := "100100" ; -- i pour
differencier l'operateur du meme nom
        constant iOR            : std_logic_vector := "100101" ; -- i pour
differencier l'operateur du meme nom
```

```
        constant iNOR   : std_logic_vector := "100111" ; -- i pour
differencier l'operateur du meme nom
        constant iXOR   : std_logic_vector := "100110" ; -- i pour
differencier l'operateur du meme nom
        constant SLT            : std_logic_vector := "101010" ;
        constant SLTU   : std_logic_vector := "101011" ;
        constant LSL            : std_logic_vector := "000000" ;
        constant LSR            : std_logic_vector := "000010" ;
        constant JR              : std_logic_vector := "001000" ;
        constant JALR   : std_logic_vector := "001001" ;

        ---------------------------------------------------------------
        -- Instruction de Type B : Branchement
        constant TYPE_B : std_logic_vector := "000001" ;
        -- Branch associes au TYPE_B Inst[BCODE'range]
        constant BLTZ            : std_logic_vector := "00000" ;
        constant BGEZ            : std_logic_vector := "00001" ;
        constant BLTZAL         : std_logic_vector := "10000" ;
        constant BGEZAL : std_logic_vector := "10001" ;

        ---------------------------------------------------------------
        -- Instructions de type J : Saut
        constant J                      : std_logic_vector := "000010" ;
        constant JAL     : std_logic_vector := "000011" ;

        ---------------------------------------------------------------
        -- Instruction de type I : Immediat
        constant ADDI           : std_logic_vector := "001000" ;
        constant ADDIU  : std_logic_vector := "001001" ;
        constant SLTI           : std_logic_vector := "001010" ;
        constant SLTIU  : std_logic_vector := "001011" ;
        constant ANDI           : std_logic_vector := "001100" ;
        constant ORI             : std_logic_vector := "001101" ;
        constant XORI           : std_logic_vector := "001110" ;
        constant LUI             : std_logic_vector := "001111" ;
        constant LB               : std_logic_vector := "100000" ;
        constant LH               : std_logic_vector := "100001" ;
        constant LW               : std_logic_vector := "100011" ;
        constant LBU             : std_logic_vector := "100100" ;
        constant LHU             : std_logic_vector := "100101" ;
        constant SB               : std_logic_vector := "101000" ;
        constant SH               : std_logic_vector := "101001" ;
        constant SW               : std_logic_vector := "101011" ;
        constant BEQ             : std_logic_vector := "000100" ;
        constant BNE             : std_logic_vector := "000101" ;
        constant BLEZ           : std_logic_vector := "000110" ;
        constant BGTZ           : std_logic_vector := "000111" ;

-------------------------------------------------------------------
-------------------------------------------------------------------
-- HARDWARE Multiplexer and Pipelines registers definitions
-------------------------------------------------------------------
-------------------------------------------------------------------

        --================================================
        -- Definition des multiplexeurs dans les etages
        --================================================

  -------------------------Mux_aleas--------------------------------
        type MUX_ALU_XA   is (ALU_A,EX_MEM_A,MEM_ER_A);--, MEM_Q);
        type MUX_ALU_XB   is (ALU_B,EX_MEM_B,MEM_ER_B);--, MEM_Q);
```

```
        type MUX_mem_data   is (Data_ual_s, Data_rt_read);

  ------------------------------------------------------------------
        type MUX_ALU_A    is (REGS_QA,REGS_QB,IMMD);
        type MUX_ALU_B    is (REGS_QB,IMMD,VAL_DEC, VAL_16, VAL_0);
        type MUX_REG_DST  is (REG_RD,REG_RT,R31);
        type MUX_REGS_D   is (ALU_S,MEM_Q,NextPC);
        type MUX_BRANCH   is (B_beq,B_bne,B_blez,B_bgtz,B_bltz,B_bgez);

--      constant VAL16    : std_logic_vector := X"00000010" ;
--      constant VAL0     : std_logic_vector := X"00000000" ;

  --================================================
  -- Definitions des structures de controles des etages
  --================================================

        ---------------1>>> Structure des signaux de control de l'etage EI
<<<---------------
        -- No signal de control in this stage! Halt and Flus is use by branch
harzard control and LW harzard control.


  ---------------2>>> Structure des signaux de control de l'etage DI
<<<---------------
        type mxDI is record
                SIGNED_EXT              : std_logic;               --
extension signee ou non donnee immediate
                J_j          : std_logic ;                        -- saut
immediate SAUT_IMMD= J_j
                J_jal     : std_logic;  -- SAUT   : J_jal
        end record;
        -- default DI control
        constant DI_DEFL : mxDI := ( SIGNED_EXT=>'0',others => '0');

-------------------------------------------------------------------------------------------

  ---------------3>>> Structure des signaux de control de l'etage EX
<<<---------------
        type mxEX is record
                ALU_OP                      : ALU_OPS;                --
operation sur l'ALU
                ALU_SIGNED              : std_logic;               --
operation ALU signee ou non
                ALU_SRCA                  : MUX_ALU_A;             -- mux
pour entree A de l'ALU
                ALU_SRCB                  : MUX_ALU_B;             -- mux
pour entree B de l'ALU
                REG_DST            : MUX_REG_DST;                -- mux pour
registre destinataire
                J_jr         : std_logic;     -- SAUT : J_jr
                J_jalr    : std_logic;  -- SAUT   : J_jalr
        end record;
        -- default EX control
  constant EX_DEFL : mxEX := ( ALU_OP=>ALU_OPS'low, ALU_SRCA=>MUX_ALU_A'low,
ALU_SRCB=>MUX_ALU_B'low,

                  REG_DST=>MUX_REG_DST'low, others=>'0' );

-------------------------------------------------------------------------------------------

  ---------------4>>> Structure des signaux de control de l'etage MEM
<<<---------------
```

```
        type mxMEM is record
                DC_DS             : MEM_DS;                -- DataCache
taille d'acces 8/16/32/64/...
                DC_RW             : std_logic;          -- DataCache signal
R/W*
                DC_AS             : std_logic;          -- DataCache signal
Address Strobe
                DC_SIGNED       : std_logic;            -- DataCache
operation signee ou non (lecture)
        BRANCH     : std_logic;         -- BRANCH active or not
          B_TYPE    : MUX_BRANCH;       --control de type de branchement
        end record;
        -- default MEM control
        constant MEM_DEFL : mxMEM := ( DC_DS=>MEM_DS'low, DC_RW=>'1', B_TYPE
=>MUX_BRANCH'low, others=>'0' );

-------------------------------------------------------------------------------------------

  ---------------5>>> Structure des signaux de control de l'etage ER
<<<---------------
        type mxER is record
                REGS_W           : std_logic;    -- signal d'ecriture W* du
banc de registres
                REGS_SRCD       : MUX_REGS_D;   -- mux vers bus de donnee D
du banc de registres
        end record;
        -- default ER control
        constant ER_DEFL : mxER := ( REGS_W=>'1', REGS_SRCD=>MUX_REGS_D'low);

-------------------------------------------------------------------------------------------

  --=================================================
  -- Definition des strucures des registres pipeline
  --=================================================

        --1>>> Structure du registre EI/DI
        type EI_DI is record
                -- === Data ===
                pc_next : std_logic_vector (PC'range);          -- cp
incremente
                inst              : std_logic_vector (INST'range);      --
instruction extraite
                flush    : std_logic ; --PV-05-02-2013
                code_op         : std_logic_vector (OPCODE'length-1 downto
0);
                code_func : std_logic_vector(FCODE'length-1 downto 0);
                rs                    : std_logic_vector (REGS'range);     --
champ rs
                rt                    : std_logic_vector (REGS'range);     --
champ rt
                rd                    : std_logic_vector (REGS'range);     --
champ rd

                -- === Control ===
                --di_ctrl               : mxDI;         -- signaux de control
de l'etage EX
        end record;

        --2>>> Structure du registre DI/EX
        type DI_EX is record
                -- === Data ===
```

```
                pc_next         : std_logic_vector (PC'range);              --
cp incremente propage
                rs                  : std_logic_vector (REGS'range);     --
champ rs
                rt                  : std_logic_vector (REGS'range);     --
champ rt
                rd                  : std_logic_vector (REGS'range);     --
champ rd
                val_dec         : std_logic_vector (VALDEC'range);      --
valeur de decalage
                imm_ext         : std_logic_vector (DATA'range);  -- valeur
immediate etendue
                jump_adr: std_logic_vector (JADR'RANGE);   -- champ adresse de
sauts
                rs_read         : std_logic_vector (DATA'range);  -- donnee
du registre lu rs
                rt_read         : std_logic_vector (DATA'range);  -- donnee
du registre lu rt
                code_op         : std_logic_vector (OPCODE'length-1 downto
0);
                -- === Control ===
                ex_ctrl         : mxEX;         -- signaux de control de
l'etage EX
                mem_ctrl : mxMEM;       -- signaux de control de l'etage MEM
                er_ctrl : mxER;         -- signaux de control de l'etage ER
        end record;

        --3>>> Structure du registre EX/MEM
        type EX_MEM is record
                -- === Data ===
                pc_next          : std_logic_vector (PC'range);           --
cp incremente propage
                ual_S              : std_logic_vector (DATA'range);
                ual_N              : std_logic ;
                ual_V              : std_logic ;
                ual_C              : std_logic ;
                ual_Z              : std_logic ;

                rs                   : std_logic_vector (REGS'range);     --
champ rs
                rt                   : std_logic_vector (REGS'range);     --
champ rt
                imm_ext         : std_logic_vector (DATA'range);  -- valeur
immediate etendue
                -- Remove pc_branch
          --pc_branch   : std_logic_vector (PC'range);            -- adresse
de branchement
                reg_dst         : std_logic_vector (REGS'range);  -- registre
destination (MUX_REG_DST)
                rt_read         : std_logic_vector (DATA'range);
                code_op         : std_logic_vector (OPCODE'length-1 downto
0);
                -- === Control ===
                mem_ctrl : mxMEM;                                   -- signaux de
control de l'etage MEM
                er_ctrl : mxER;                                     -- signaux de
control de l'etage ER
        end record;

        --4>>> Structure du registre MEM/ER
        type MEM_ER is record
```

```
                -- Signaux
                pc_next         : std_logic_vector (PC'range);           --
cp incremente propage
                mem_Q           : std_logic_vector (DATA'range);  -- sortie
memoire
                ual_S           : std_logic_vector (DATA'range);   --
resultat ual propage
                reg_dst         : std_logic_vector (REGS'range); -- registre
destination propage
          -- code_op            : std_logic_vector (OPCODE'length-1 downto
0);

                -- Signaux de control
                er_ctrl         : mxER;                                  --
signaux de control de l'etage ER propage
        end record;

-- ============================================================
-- DEFINITION DE FONCTIONS/PROCEDURES
-- ============================================================

-- Si on ne specifie rien devant les parametres...il considere que c'est une
variable
-- exemple : procedure adder_cla (A,B: in std_logic_vector;...)
-- ici A et B sont consideres comme etant des variables...
-- Sinon il faut : procedure adder_cla (signal A,B: in std_logic_vector;...)

        -- Fonction "+" --> procedure adder_cla
        -- function "+" (A,B: in std_logic_vector) return std_logic_vector;

        -- Procedure adder_cla
        procedure adder_cla (   A,B: in std_logic_vector; C_IN : in
std_logic;
                                                S : out std_logic_vector;
C_OUT : out std_logic; V : out std_logic);


        -- Procedure alu
        --      on notera l'utilisation d'un signal comme parametres formels
de type OUT
        procedure alu ( A,B: in std_logic_vector; signal S: out
std_logic_vector;
                                signal N,V,Z,C: out std_logic; SIGNED_OP:
in std_logic; CTRL_ALU: in ALU_OPS);


        -- Procedure control
        --              permet de positionner les signaux de control pour
chaque etage (EX MEM ER)
        --              en fonction de l'instruction identifiee soit par son
code op, soit par
        --              son code fonction, soit par son code branchement.
        procedure control (     flush : in std_logic ;
                                OP    : in std_logic_vector(OPCODE'length-1
downto 0);
                                                F     : in
std_logic_vector(FCODE'length-1 downto 0);
                                                B     : in
std_logic_vector(BCODE'length-1 downto 0);
                                               signal DI_ctrl  : out mxDI;
        -- signaux de controle de l'etage DI
```

```
                                                signal EX_ctrl  : out mxEX;
        -- signaux de controle de l'etage EX
                                                signal MEM_ctrl : out mxMEM;
-- signaux de controle de l'etage MEM
                                                signal ER_ctrl  : out mxER );
-- signaux de controle de l'etage ER

        -- Procedure envoi
        procedure envoi (   DI_EX_code_op  : in std_logic_vector
(OPCODE'length-1 downto 0);

                            --EX_MEM_code_op  : in std_logic_vector
(OPCODE'length-1 downto 0);

                            DI_EX_Register_Rs   : in std_logic_vector
(REGS'range);
                      DI_EX_Register_Rt   : in std_logic_vector (REGS'range);
                      EX_MEM_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                      MEM_ER_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                      EX_MEM_Regs_W     : in std_logic;
                      MEM_ER_Regs_W     : in std_logic;

                      --signal mem_halt : in std_logic;
                      signal mem_data   : out MUX_mem_data;
                      signal ALU_XA    : out MUX_ALU_XA;
                      signal ALU_XB    : out MUX_ALU_XB );

  -- Procedure aleaLW
  procedure aleaLW (  DI_EX_Register_Rt : in std_logic_vector (REGS'range);
                      EI_DI_Register_Rs : in std_logic_vector (REGS'range);
                      EI_DI_Register_Rt : in std_logic_vector (REGS'range);
                      DI_EX_Regs_W      : in std_logic;
                      DI_EX_DC_RW      : in std_logic;
                      Di_EX_DC_AS      : in std_logic;
                      signal halt       : out std_logic );


end cpu_package;

--
-------------------------------------------------------------------------------
-- the package contains types, constants, and function prototypes
--
-------------------------------------------------------------------------------
package body cpu_package is

  -- ============================================================
  -- DEFINITION DE FONCTIONS/PROCEDURES
  -- ============================================================

  -->>> fonction log2
--  function log2 (I: in natural) return natural is
--    variable ip : natural := 1; -- valeur temporaire
--      variable iv : natural := 0; -- nb de bits
--  begin
--      while ip < i loop
--              ip := ip + ip; -- ou ip := ip * 2
--              iv := iv + 1;
--      end loop;
```

```
--       -- renvoie le nombre de bits
--       return iv;
--   end log2;

  -->>> fonction "+" --> procedure adder_cla
--   function "+" (A,B: in std_logic_vector) return std_logic_vector is
--       variable tmp_S : std_logic_vector(A'range);
--       variable tmp_COUT,tmp_V : std_logic;
--   begin
--       adder_cla(A,B,'0',tmp_S,tmp_COUT,tmp_V);
--       return tmp_S;
--   end "+";

  -- Le drapeau overflow V ne sert que lors d'operations signees !!!
  -- Overflow V=1 si operation signee et :
  --             addition de deux grands nombres positifs dont le resultat < 0
  --             addition de deux grands nombres negatifs dont le resultat >= 0
  --             soustraction d'un grand nombre positif et d'un grand nombre
negatif dont le resultat < 0
  --             soustraction d'un grand nombre negatif et d'un grand nombre
positif dont le resultat >= 0
  --    Reviens a faire V = C_OUT xor <carry entrante du dernier bit>
  -->>> procedure adder_cla
  procedure adder_cla ( A,B: in std_logic_vector;C_IN : in std_logic;
                                                        S : out
std_logic_vector;C_OUT : out std_logic;
                                                        V : out std_logic)
is
                            variable G_CLA,P_CLA : std_logic_vector(A'length-1
downto 0);
                            variable C_CLA                      :
std_logic_vector(A'length downto 0);
  begin
        -- calcul de P et G
        G_CLA:= A and B;
        P_CLA:= A or B;
        C_CLA(0):=C_IN;
        for I in 0 to (A'length-1) loop
                C_CLA(I+1):= G_CLA(I) or (P_CLA(I) and C_CLA(I));
        end loop;
        -- mise a jour des sorties
        S:=(A Xor B) xor C_CLA(A'length-1 downto 0);
        C_OUT:=C_CLA(A'length);
    V:= C_CLA(A'length) xor C_CLA(A'length - 1);
  end adder_cla;

  -- procedure alu
  procedure alu (  A,B: in std_logic_vector;signal S: out std_logic_vector;
                                                  signal N,V,Z,C: out
std_logic;SIGNED_OP: in std_logic;
                                                  CTRL_ALU: in ALU_OPS) is
                        variable DATA_WIDTH : positive := A'length;
                         variable b_in       : std_logic_vector(DATA_WIDTH-1
downto 0);
                        variable c_in       : std_logic;
                        variable tmp_S           :
std_logic_vector(DATA_WIDTH-1 downto 0);
                        variable tmp_V           : std_logic;
                        variable tmp_N           : std_logic;
                        variable tmp_C           : std_logic;
                         variable tmp_CLA_C      : std_logic;
```

```
                        variable tmp_CLA_V       : std_logic;
  begin
        -- raz signaux
        tmp_V := '0';
        tmp_N := '0';
        tmp_C := '0';
        -- case sur le type d'operation
        case CTRL_ALU is
                when ALU_ADD | ALU_SUB | ALU_SLT =>
                        b_in := B;
                        c_in := '0';
                        if (CTRL_ALU /= ALU_ADD) then
                                b_in := not(B);
                                c_in := '1';
                        end if;
                        adder_cla(A,b_in,c_in,tmp_S,tmp_C,tmp_V);
                        if (CTRL_ALU = ALU_SLT) then
                                tmp_S := conv_std_logic_vector( (SIGNED_OP
and (tmp_V xor tmp_S(DATA_WIDTH-1))) or (not(SIGNED_OP) and not(tmp_C)) ,
S'length );
                                -- remize à 0 des flags selon definition
                                tmp_C := '0';
                                tmp_V := '0';
                        else
                                tmp_C := not(SIGNED_OP) and tmp_C;
                                tmp_N := SIGNED_OP and tmp_S(DATA_WIDTH-1);
                                tmp_V := SIGNED_OP and tmp_V;
                        end if;
                when ALU_AND =>
                        tmp_S := A and B;
                when ALU_OR =>
                        tmp_S := A or B;
                when ALU_NOR =>
                        tmp_S := A nor B;
                when ALU_XOR =>
                        tmp_S := A xor B;
                when ALU_LSL =>
                        tmp_S := shl(A,B);
                when ALU_LSR =>
                        tmp_S := shr(A,B);
                when others =>
        end case;
        -- affectation de la sortie
        S <= tmp_S;
        -- affectation du drapeau Z (valable dans tous les cas)
        if (tmp_S=conv_std_logic_vector(0,DATA_WIDTH)) then Z <= '1';
        else Z <= '0';
        end if;
        -- affectation des autres drapeaux N,V,C
        C <= tmp_C;
        N <= tmp_N;
        V <= tmp_V;
  end alu;

  -->>> === Procedure control ========================================
  --            Permet de positionner les signaux de control pour chaque etage
(EX MEM ER)
  --            en fonction de l'instruction identifiee soit par son code op,
soit par
  --            son code fonction, soit par son code branchement.
  --  If the signal control has also value X (don't care), write the code in
(if .... end if), not (if ... elsif... else ... end if)
```

```
  --  because we the signal will take the last time value for the value X.
  procedure control ( flush : in std_logic ;
                      OP    : in std_logic_vector(OPCODE'length-1 downto 0);
                                                                  F     : in
std_logic_vector(FCODE'length-1 downto 0);
                                                                  B     :in
std_logic_vector(BCODE'length-1 downto 0);
                                                                  signal
DI_ctrl : out mxDI;             -- signaux de controle de l'etage DI
                                                                  signal
EX_ctrl : out mxEX;             -- signaux de controle de l'etage EX
                                                                  signal
MEM_ctrl: out mxMEM;    -- signaux de controle de l'etage MEM
                                                                  signal
ER_ctrl : out mxER ) is -- signaux de controle de l'etage ER
  begin
        -- Initialisation

        DI_ctrl <= DI_DEFL;
        EX_ctrl <= EX_DEFL;
        MEM_ctrl <= MEM_DEFL;
        ER_ctrl <= ER_DEFL;

    if (flush = '0') then

    --==============================
    ---- ControlDI :
    -->>> SIGNED_EXT: signe-t-on ou non l'extension de la valeur immediate
    -->>> SAUT @IMMD: J_j
    --==============================

        -->>> SIGNED_EXT
        if (  (OP=TYPE_B) or (OP=ADDI) or (OP=ADDIU) or (OP=SLTI) or
(OP=SLTIU) or
              (OP=LB) or (OP=LH) or (OP=LW) or (OP=LBU) or (OP=LHU) or (OP=SB)
or
              (OP=SH) or (OP=SW) or (OP=BEQ) or (OP=BNE) or (OP=BLEZ) or
(OP=BGTZ)  )then
            DI_ctrl.SIGNED_EXT <= '1';
        else
            DI_ctrl.SIGNED_EXT <= '0';
        end if;

             -->>> SAUT @IMMD: J_j
             if (  (OP=J)) then
              DI_ctrl.J_j <= '1';
             end if ;

        -->>> J_jal
        if ( (OP= JAL)) then
                  DI_ctrl.J_jal <= '1';
        end if;

    --==============================
    ---- Control EX
    -->>> SAUT @Reg: J_jr
    -->>> REG_DST
    -->>> ALU_SRCB
    -->>> ALU_SRCA
    -->>> ALU_SIGNED
    -->>> ALU_OP
```

```
    -->>> J_jal
    -->>> J_jalr
    --==============================

        -->>> SAUT @Reg: J_jr
             if ((OP=TYPE_R) and (F=JR)) then
              EX_ctrl.J_jr <= '1';
             end if ;

        -->>> J_jalr
        if ( (OP= TYPE_R) and (F=JALR) ) then
                  EX_ctrl.J_jalr <= '1';
        end if;

        -->>> REG_DST
        if (  (OP=ADDI)or (OP=ADDIU) or (OP=SLTI)or (OP=SLTIU)or (OP=ANDI)or
(OP=ORI)or (OP=XORI)
              or (OP=LUI)or(OP= LB)or (OP= LH)or (OP= LW)or (OP= LBU)or (OP=
LHU) ) then
              EX_ctrl.REG_DST <= REG_RT;
        end if;
        if (((OP= TYPE_B)and ((B=BLTZAL)or(B=BGEZAL)))or(OP=JAL))then
                  EX_ctrl.REG_DST <= R31;
        end if;
        if ((OP= TYPE_R)) then
                  EX_ctrl.REG_DST <= REG_RD;
        end if;

        -->>> ALU_SRCB
        if ((OP=TYPE_B)) then
              EX_ctrl.ALU_SRCB <= VAL_0;
        end if;
        if ((OP=LUI)) then
              EX_ctrl.ALU_SRCB <= VAL_16;
        end if;
        if ( (OP=ADDI) or (OP=ADDIU)or (OP=SLTI)or (OP=SLTIU)or (OP=ANDI) or
(OP=ORI)or (OP=XORI) or
             (OP= LB)or (OP= LH) or (OP= LW) or (OP= LBU)or (OP= LHU)or (OP=
SB)or (OP= SH)or(OP= SW) )then
            -- EX_ctrl.ALU_SRCA <= REGS_QA;
              EX_ctrl.ALU_SRCB <= IMMD;
        end if;
        if ( (OP= TYPE_R) and ((F=LSL) or (F=LSR)) )then
              EX_ctrl.ALU_SRCB <= VAL_DEC;
        end if;
        if ( ( (OP= TYPE_R) and ((F=ADD) or (F=ADDU) or (F=SUB) or (F=SUBU) or
(F=iAND) or (F=iOR) or (F=iNOR) or
                                  (F=iXOR) or (F=SLT) or (F=SLTU) or (F=JR) or
(F=JALR)) )
                or (OP= BEQ) or (OP= BNE) or (OP= BLEZ) or (OP= BGTZ)
              ) then
              EX_ctrl.ALU_SRCB <= REGS_QB;
        end if;

        -->>> ALU_SRCA
        if ( (OP=LUI)) then
              EX_ctrl.ALU_SRCA <= IMMD;
        end if;
        if ( (OP= TYPE_R) and ((F=LSL) or (F=LSR)) )then
              EX_ctrl.ALU_SRCA <= REGS_QB;
        end if;
```

```
        if ( (OP= TYPE_R) or (OP= TYPE_B) or (OP=ADDI) or (OP=ADDIU) or
(OP=SLTI) or (OP=SLTIU) or (OP=ANDI) or (OP=ORI)or (OP=XORI) or
             (OP= LB) or (OP= LH) or (OP= LW) or (OP= LBU) or (OP= LHU) or
(OP= SB) or (OP= SH) or(OP= SW) or
             (OP= BEQ) or(OP= BNE) or(OP= BLEZ) or(OP= BGTZ) ) then
              EX_ctrl.ALU_SRCA <= REGS_QA;
        end if;

        -->>> ALU_SIGNED
        if ( ( (OP=TYPE_R) and ( (F=ADDU) or (F=SUBU) or (F=iAND) or (F=iOR)
or (F=iNOR) or (F=iXOR) or (F=SLTU) or (F=LSL) or (F=LSR) ) ) or
              (OP=ADDIU) or (OP=SLTIU) or (OP=ANDI) or (OP=ORI) or (OP=XORI) )
then
              EX_ctrl.ALU_SIGNED <= '0';
        end if;
        if ( ( (OP=TYPE_R) and ( (F=ADD) or (F=SUB) or (F=SLT) ) ) or (OP=
TYPE_B) or (OP=ADDI) or (OP=SLTI) or (OP=LUI) or
                (OP= LB) or (OP= LH) or (OP= LW) or (OP= LBU) or (OP= LHU) or
(OP= SB) or (OP= SH) or(OP= SW) or (OP=BEQ) or
                (OP=BNE) or (OP=BLEZ) or (OP=BGTZ) ) then
              EX_ctrl.ALU_SIGNED <= '1';
        end if;

        -->>> ALU_OP
        if ( ((OP=TYPE_R)and (F=iAND)) or (OP=ANDI) ) then
              EX_ctrl.ALU_OP <= ALU_AND;
         end if;

             if (((OP= TYPE_R) and (F=iOR))or (OP=ORI)) then
                  EX_ctrl.ALU_OP <= ALU_OR;
             end if;

             if (((OP= TYPE_R) and (F=iXOR))or (OP=XORI)) then
                 EX_ctrl.ALU_OP <= ALU_XOR;
             end if;

             if ((OP= TYPE_R) and (F=iNOR)) then
                 EX_ctrl.ALU_OP <= ALU_NOR;
             end if;

             if ( ((OP= TYPE_R) and ((F=SUB)or (F=SUBU)))or ((OP= TYPE_B)and
((B=BLTZ)or(B=BGEZ)or(B=BLTZAL)or(B=BGEZAL))) or
              (OP=BEQ)or(OP=BNE)or(OP=BLEZ)or(OP=BGTZ)) then
             EX_ctrl.ALU_OP      <= ALU_SUB ;
             end if ;

        if ( ((OP= TYPE_R) and ((F=ADD)or (F=ADDU)))or   (OP= ADDI) or (OP=
ADDIU) or (OP= LB) or (OP= LH) or (OP= LW) or
              (OP= LBU) or (OP= LHU) or (OP= SB) or (OP= SH) or (OP= SW) )
then
             EX_ctrl.ALU_OP      <= ALU_ADD ;
        end if;

        if ( ((OP= TYPE_R) and (F=LSL)) or (OP= LUI) ) then
                 EX_ctrl.ALU_OP <= ALU_LSL;
             end if;

             if ( ((OP= TYPE_R) and (F=LSR)) ) then
                 EX_ctrl.ALU_OP <= ALU_LSR;
             end if;
```

```
             if ( ((OP= TYPE_R) and ((F=SLT) or (F=SLTU))) or (OP= SLTI) or
(OP= SLTIU) ) then
                 EX_ctrl.ALU_OP <= ALU_SLT;
             end if;

    --==============================
    ---- Control MEM
    -->>> BRANCH
    -->>> B_TYPE
    -->>> DC_RW
    -->>> DC_DS
    -->>> DC_SIGNED
    -->>> DC_AS
    --==============================


        -->>> BRANCH
        if ( ((OP= TYPE_B)and ((B=BLTZ)or(B=BGEZ))) or
(OP=BEQ)or(OP=BNE)or(OP=BLEZ)or(OP=BGTZ)) then
             MEM_ctrl.BRANCH <= '1';
             end if ;

        -->>> B_TYPE
        if ( (OP= TYPE_B)and ((B=BLTZ) or (B=BLTZAL) )) then
                 MEM_ctrl.B_TYPE <= B_bltz;
        end if;

        if ( (OP= TYPE_B)and ((B=BGEZ) or (B=BGEZAL))) then
            MEM_ctrl.B_TYPE <= B_bgez;
        end if;

        if (OP=BEQ) then
                 MEM_ctrl.B_TYPE <= B_beq;
        end if;

        if (OP=BNE) then
            MEM_ctrl.B_TYPE <= B_bne;
        end if;

        if (OP=BLEZ) then
            MEM_ctrl.B_TYPE <= B_blez;
        end if;

        if (OP=BGTZ) then
            MEM_ctrl.B_TYPE <= B_bgtz;
        end if;

            -->>> DC_RW
        if ( (OP = LB) or (OP = LH) or (OP = LW) or (OP = LBU) or (OP = LHU) )
then
                 MEM_ctrl.DC_RW <= '1' ;
         end if ;
         if ( (OP = SB) or (OP = SH) or (OP = SW) ) then
                MEM_ctrl.DC_RW<= '0' ;
         end if ;

        -->>> DC_DS
        if ( (OP = LB) or (OP = LBU) or (OP = SB) ) then
                 MEM_ctrl.DC_DS <= MEM_8 ;
           end if ;
        if ( (OP = LH) or (OP = LHU) or (OP = SH) ) then
```

```
                MEM_ctrl.DC_DS <= MEM_16 ;
           end if ;
        if ( (OP = LW) or (OP = SW) ) then
                MEM_ctrl.DC_DS <= MEM_32 ;
           end if ;

        -->>> DC_SIGNED
        if ( (OP = LH) or (OP = LHU) or (OP = SH) or(OP = LB) or (OP = LBU) or
(OP = SB) ) then
                MEM_ctrl.DC_SIGNED <= '1' ;
           end if ;
        if ( (OP = LW) or (OP = SW) ) then
                MEM_ctrl.DC_SIGNED <= '0' ;
           end if ;

        -->>> DC_AS
        if ( (OP = LH) or (OP = LHU) or (OP = SH) or (OP = LB) or (OP = LBU)
or (OP = SB) or
             (OP = LW) or (OP = SW) ) then
                MEM_ctrl.DC_AS<=  '1' ;
              else
                   MEM_ctrl.DC_AS<=  '0' ;
           end if ;

    --==============================
    -- Control ER
    -->>> REGS_SRCD
    -->>> REGS_W : active à l'état 0

    --==============================

         -->>> REGS_SRCD
        if (  (OP= LB) or (OP= LH) or (OP= LW) or (OP= LBU) or (OP= LHU)) then
                  ER_ctrl.REGS_SRCD <= MEM_Q ;
              end if;
             if ( ((OP= TYPE_B) and ((B=BGEZAL)or(B=BLTZAL))) or ((OP=TYPE_R)
and (F=JALR)) or (OP=JAL) ) then
                  ER_ctrl.REGS_SRCD <= NextPC;
              end if;
             if ( ((OP= TYPE_R) and ((F=ADD) or (F=ADDU) or (F=SUB) or
(F=SUBU) or (F=iAND) or (F=iOR) or (F=iNOR) or
                  (F=iXOR) or (F=SLT) or (F=SLTU) or (F=LSL) or (F=LSR))) or
(OP=ADDI) or (OP=ADDIU) or (OP=SLTI) or
                  (OP=SLTIU) or (OP=ANDI) or (OP=ORI)or (OP=XORI) or (OP=LUI)
) then
                  ER_ctrl.REGS_SRCD <= ALU_S;
             end if;

             -->>> REGS_W : active à l'état 0
             if ( ((OP= TYPE_B) and ((B=BLTZ)or(B=BGEZ))) or (OP=J) or (OP =
SB)
                   or (OP = SH) or (OP = SW) or (OP=BEQ) or (OP=BNE) or
(OP=BLEZ) or (OP=BGTZ) )  then
                    ER_ctrl.REGS_W <= '1'; -- inactif
        else
              ER_ctrl.REGS_W <= '0'; -- actif
        end if;

    end if; -- if flush
```

```
  end procedure control;

  -- === Procedure envoi ========================================
  -->>> MUX: ALU_XA
  -->>> MUX: ALU_B

  procedure envoi (   DI_EX_code_op  : in std_logic_vector (OPCODE'length-1
downto 0);

                      --EX_MEM_code_op  : in std_logic_vector (OPCODE'length-1
downto 0); -- Aléas LW suivi par add sans suspendre étage MEM

                      DI_EX_Register_Rs   : in std_logic_vector (REGS'range);
                      DI_EX_Register_Rt   : in std_logic_vector (REGS'range);

                      EX_MEM_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                      MEM_ER_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                      EX_MEM_Regs_W     : in std_logic; -- active à l'état '0'
                      MEM_ER_Regs_W     : in std_logic; -- active à l'état '0'

                      --signal mem_halt : in std_logic;
                      signal mem_data   : out MUX_mem_data;
                      signal ALU_XA    : out MUX_ALU_XA;
                      signal ALU_XB    : out MUX_ALU_XB
                       ) is
   begin

        -- Initialisation

         mem_data <= Data_rt_read;
        ALU_XA <= ALU_A;
        ALU_XB <= ALU_B;


        --=================
        -->>> MUX: ALU_XA
        --=================

        -- Aléas étage MEM
        IF ( (MEM_ER_Regs_W = '0') AND
             (MEM_ER_Register_Reg_dst /= X"00000000") AND -- Register_Reg_dst
-> R0 Non envoi
             (EX_MEM_Register_Reg_dst /= DI_EX_Register_Rs) AND
             (MEM_ER_Register_Reg_dst = DI_EX_Register_Rs)) THEN

             ALU_XA <= MEM_ER_A; --MEM HAZARD

        -- Aléas étage EX
        ELSIF ( (EX_MEM_Regs_W = '0') AND
                (EX_MEM_Register_Reg_dst /= X"00000000") AND --
Register_Reg_dst -> R0 Non envoi
                (EX_MEM_Register_Reg_dst = DI_EX_Register_Rs)) THEN

            -- Aléas LW suivi par add sans suspendre étage MEM
            -- lw suivi par add. À l'instruction add d'étage DI_EX, S'il y a
de halt, entré de ALU_XA prend la valeur de MEM_Q pour addition.
--            if ((mem_halt='1') AND ( EX_MEM_code_op=LB or EX_MEM_code_op=LH
or EX_MEM_code_op=LW or EX_MEM_code_op=LBU or
--                EX_MEM_code_op=LHU ) ) then
```

```
--                ALU_XA <= MEM_Q;
--            else
--                ALU_XA <= EX_MEM_A; --EX HAZARD
--            end if;
             ALU_XA <= EX_MEM_A; --EX HAZARD

        -- Pas d'Aléas
        ELSE
            ALU_XA <= ALU_A; --NO HAZARD
        END IF;

        --================
        -->>> MUX: ALU_B
        --================

        -- Aléas étage MEM
        IF (  (MEM_ER_Regs_W = '0') AND
               (MEM_ER_Register_Reg_dst /= X"00000000") AND --
Register_Reg_dst -> R0 Non envoi
               (EX_MEM_Register_Reg_dst /= DI_EX_Register_Rt) AND
               (MEM_ER_Register_Reg_dst = DI_EX_Register_Rt) ) THEN
          IF (
               DI_EX_code_op/= SW AND DI_EX_code_op/= SB AND DI_EX_code_op/=
SH AND
               DI_EX_code_op/= LB AND DI_EX_code_op/= LH AND DI_EX_code_op/=
LW AND DI_EX_code_op/= LBU AND
               DI_EX_code_op/= LHU AND

               DI_EX_code_op/= BGEZ AND DI_EX_code_op/= BLTZAL AND
DI_EX_code_op/= BGEZAL AND

               DI_EX_code_op/= ADDI AND DI_EX_code_op/=ADDIU AND
DI_EX_code_op/=SLTI AND DI_EX_code_op/=SLTIU AND
               DI_EX_code_op/=ANDI AND DI_EX_code_op/=ORI AND
DI_EX_code_op/=XORI AND
               DI_EX_code_op/=LUI
             ) THEN
            ALU_XB <= MEM_ER_B; --MEM
          END IF;

        -- Aléas étage EX
        ELSIF (  (EX_MEM_Regs_W = '0') AND
                  (EX_MEM_Register_Reg_dst /= X"00000000") AND --
Register_Reg_dst -> R0 Non envoi
                  (EX_MEM_Register_Reg_dst = DI_EX_Register_Rt) ) THEN
           IF (
               DI_EX_code_op/= SW AND DI_EX_code_op/= SB AND DI_EX_code_op/=
SH AND
               DI_EX_code_op/= LB AND DI_EX_code_op/= LH AND DI_EX_code_op/=
LW AND DI_EX_code_op/= LBU AND
               DI_EX_code_op/= LHU AND

               DI_EX_code_op/= BGEZ AND DI_EX_code_op/= BLTZAL AND
DI_EX_code_op/= BGEZAL AND

               DI_EX_code_op/= ADDI AND DI_EX_code_op/=ADDIU AND
DI_EX_code_op/=SLTI AND DI_EX_code_op/=SLTIU AND
               DI_EX_code_op/=ANDI AND DI_EX_code_op/=ORI AND
DI_EX_code_op/=XORI AND
               DI_EX_code_op/=LUI
             ) THEN
```

```
                -- Aléas LW suivi par add sans suspendre étage MEM
                -- lw suivi par add. À l'instruction add d'étage DI_EX, S'il y
a de halt, entré de ALU_XA prend la valeur de MEM_Q pour addition.
--                if ((mem_halt='1') AND ( EX_MEM_code_op=LB or
EX_MEM_code_op=LH or EX_MEM_code_op=LW or EX_MEM_code_op=LBU or
--                     EX_MEM_code_op=LHU ) ) then
--                    ALU_XB <= MEM_Q;
--                else
--                    ALU_XB <= EX_MEM_B; --EX HAZARD
--                end if;
               ALU_XB <= EX_MEM_B; --EX HAZARD
          END IF;

          IF ((DI_EX_code_op = SW) or (DI_EX_code_op = SB) or (DI_EX_code_op =
SH)) THEN
              mem_data <= Data_ual_s;
          END IF;

       -- Pas d'Aléas
       ELSE
            ALU_XB <= ALU_B; --NO HAZARD
       END IF;

  end procedure envoi;

  -- === Procedure Alea LW ========================================
  ------ >> Alea LW suivi par une instruction arithmetique ---------
  procedure aleaLW (  DI_EX_Register_Rt : in std_logic_vector (REGS'range); --
DI_EX_Register_Rt === DI_EX_EcrireReg
                      EI_DI_Register_Rs : in std_logic_vector (REGS'range);
                      EI_DI_Register_Rt : in std_logic_vector (REGS'range);
                      DI_EX_Regs_W      : in std_logic;
                      DI_EX_DC_RW      : in std_logic; -- PV 05-02-2013
                      DI_EX_DC_AS      : in std_logic; -- PV 05-02-2013
                      signal halt       : out std_logic ) is

  begin
        -- Initialisation
        halt <= '0';
     --DETECT A LOAD/USE HAZARD e.g. LW $2, 20($1) followed by ADD $4, $2, $1
      IF ( (DI_EX_Regs_W = '0') AND DI_EX_DC_AS = '1' AND DI_EX_DC_RW = '1'
AND
           (   (DI_EX_Register_Rt = EI_DI_Register_Rs and EI_DI_Register_Rs /=
conv_std_logic_vector(0,REGS'length))
            OR (DI_EX_Register_Rt = EI_DI_Register_Rt and EI_DI_Register_Rt /=
conv_std_logic_vector(0,REGS'length)) ) ) THEN
          halt <= '1'; --LOAD/USE HAZARD
      ELSE
          halt <= '0'; --NO HAZARD
      END IF;

  end procedure aleaLW;

--  -- === Procedure Alea LW ========================================
--  ------ >> Alea LW suivi par une instruction arithmetique ---------
--  procedure aleaLW (  DI_EX_Register_Rt : in std_logic_vector (REGS'range);
--                      EI_DI_Register_Rs : in std_logic_vector (REGS'range);
--                      EI_DI_Register_Rt : in std_logic_vector (REGS'range);
--                      EX_MEM_Regs_W      : in std_logic;
--                      EX_MEM_DC_RW      : in std_logic; -- PV 05-02-2013
```

```
--                       EX_MEM_DC_AS      : in std_logic; -- PV 05-02-2013
--                       signal halt       : out std_logic ) is
--
--  begin
--       -- Initialisation
--       halt <= '0';
--      --DETECT A LOAD/USE HAZARD e.g. LW $2, 20($1) followed by ADD $4, $2,
$1
--       IF ( (EX_MEM_Regs_W = '0') AND EX_MEM_DC_AS = '1' AND EX_MEM_DC_RW =
'1' AND
--            (   (DI_EX_Register_Rt = EI_DI_Register_Rs and EI_DI_Register_Rs
/= conv_std_logic_vector(0,REGS'length))
--             OR (DI_EX_Register_Rt = EI_DI_Register_Rt and EI_DI_Register_Rt
/= conv_std_logic_vector(0,REGS'length)) ) ) THEN
--           halt <= '1'; --LOAD/USE HAZARD
--       ELSE
--           halt <= '0'; --NO HAZARD
--       END IF;
--
--  end procedure aleaLW;

end cpu_package;
```

```
-----------------------------------
-- Processeur RISC
-- THIEBOLT Francois le 09/12/04
-- PHOR Vicheka
-----------------------------------

------------------------------------------------------
-- Lors de la phase RESET, permet la lecture d'un fichier
-- instruction et un fichier donnees passe en parametre
--      generique.
------------------------------------------------------

-- Definition des librairies
library IEEE;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.cpu_package.all;
--use ieee.numeric_std.all;
--use WORK.all;

-- Definition de l'entite
entity risc is

        -- definition des parametres generiques
        generic (
                -- fichier d'initialisation cache instruction
                IFILE : string := "";

                -- fichier d'initialisation cache donnees
                DFILE : string := "" );

        -- definition des entrees/sorties
        port    (
                -- signaux de controle du processeur
                RST                     : in std_logic; -- actifs a l'etat
bas
                CLK                     : in std_logic );

end risc;

-- Definition de l'architecture du banc de registres
architecture behavior of risc is

        -- definition de constantes

        -- definitions de types/soustypes

        -- definition des ressources internes

        -- Registres du pipeline
        signal reg_EI_DI : EI_DI;               -- registre pipeline EI/DI
        signal reg_DI_EX : DI_EX;               -- registre pipeline DI/EX
        signal reg_EX_MEM       : EX_MEM;       -- registre pipeline EX/MEM
        signal reg_MEM_ER       : MEM_ER;       -- registre pipeline MEM/ER

        --1>>> Ressources de l'etage EI
        signal reg_PC                 : ADDR;                       --
compteur programme format octet
```

```
        signal ei_pc_next          : PC;                                --
pointeur sur prochaine instruction
        signal ei_inst              : INST;                             --
instruction en sortie du cache instruction
  -- signal ei_halt                 : std_logic;         -- suspension etage
pipeline
  --    signal ei_flush             : std_logic;  -- vidange de l'etage --PV:
Sans utiliser

        --2>>> Ressources de l'etage DI
        signal di_qa                          : DATA;
-- sortie QA du banc de registres
        signal di_qb                          : DATA;                    --
sortie QB du banc de registres
        signal di_imm_ext                 : DATA;
-- valeur immediate etendue
        signal di_ctrl_di                 : mxDI;
-- signaux de controle de l'etage DI
        signal di_ctrl_ex                 : mxEX;
-- signaux de controle de l'etage EX
        signal di_ctrl_mem        : mxMEM;                          -- signaux
de controle de l'etage MEM
        signal di_ctrl_er                 : mxER;
-- signaux de controle de l'etage ER
        signal di_flush             : std_logic; -- vidange de l'etage
  --  signal di_halt                 : std_logic;         -- suspension etage
pipeline

        --3>>> Ressources de l'etage EX
        signal ex_ALU_Z       : std_logic;
        signal ex_ALU_N       : std_logic;
        signal ex_ALU_V       : std_logic;
        signal ex_ALU_C       : std_logic;
  signal ex_ALU_S       : DATA;     -- sortie S de l'alu
        signal ex_ALU_A       : DATA;     -- sortie ALU_A de MUX_AUL_A
        signal ex_ALU_B       : DATA;     -- sortie ALU_B de MUX_AUL_B
        signal ex_ALU_XA      : DATA;     -- PV: ex_envoie_A === ex_ALU_XA
        signal ex_ALU_XB      : DATA;     -- PV: ex_envoie_B === ex_ALU_XB
        signal ex_mem_data    : DATA;

        signal ex_reg_dst                  : REGS;     -- PV: registres
destination

        signal ex_MUX_mem_data : MUX_mem_data;
  signal ex_MUX_ALU_XA  : MUX_ALU_XA;
        signal ex_MUX_ALU_XB  : MUX_ALU_XB;

  signal ex_pc_branch       : PC;                 -- adresse de branchement

        signal ex_flush              : std_logic;         -- vidange de
l'etage
  signal ex_halt             : std_logic;         -- suspension etage
pipeline

        --4>>> Ressources de l'etage MEM
        signal MEM_mem_Q    : DATA;   -- PV: 08-02-2013
  signal mem_branch   : std_logic ;   -- PV: mem_branch_bol === mem_branch
  signal mem_b_type   : std_logic;      -- PV: mem_branch_type_out ===
mem_b_type

  signal mem_flush                : std_logic;
```

```
  -- signal mem_halt                 : std_logic;         -- suspension etage
pipeline


        --5>>> Ressources de l'etage ER
        signal er_regd              : DATA;                     -- donnees a
ecrire dans le banc de registre
        signal er_adrw              : REGS;                     -- adresse du
registre a ecrire dans le banc


begin


-- ============================================================
-- === Etage EI ===============================================
-- ============================================================

------------------------------------------------------------------
-- instanciation et mapping du composant cache instruction
icache : entity work.memory(behavior)
                        generic map ( DBUS_WIDTH=>CPU_DATA_WIDTH,
ABUS_WIDTH=>CPU_ADR_WIDTH, MEM_SIZE=>L1_ISIZE,

ACTIVE_FRONT=>L1_FRONT, FILENAME=>IFILE )
                        port map ( RST=>RST, CLK=>CLK, RW=>'1', DS=>MEM_32,
Signed=>'0', AS=>'1', Ready=>open,
                                                          Berr=>open,
ADR=>reg_PC, D=>(others => '0'), Q=>ei_inst );


------------------------------------------------------------------
-- Affectations dans le domaine combinatoire de l'etage EI
-->>> Incrementation du PC (format mot)
ei_pc_next <=  ex_pc_branch when mem_branch = '1' else         -- mem_branch:
Branch & Branch type & (Z, N,C,V)
               reg_DI_EX.rs_read (PC'length-1 downto 0)  when
reg_DI_EX.ex_ctrl.J_jr = '1' or reg_DI_EX.ex_ctrl.J_jalr = '1' else     --
J_jr comme from reg_DI_EX.ex_ctrl.J_jr
               reg_EI_DI.inst(JADR'range)    when di_ctrl_di.J_j = '1' or
di_ctrl_di.J_jal = '1'  else  -- reg_EI_DI.di_ctrl.J_j <-> di_ctrl_di.J_j
                                reg_PC(PC'range) +'1';

--ei_halt <= '0' ;


-->>> Appel de la procedure aleaLW
UA: aleaLW (  ex_reg_dst, --reg_EX_MEM.reg_dst, --reg_DI_EX.rd,
              reg_EI_DI.inst(RS'range),      --  reg_DI_EX.rs,
                    reg_EI_DI.inst(RT'range),      --  reg_DI_EX.rt,
                reg_DI_EX.er_ctrl.Regs_W, -- er_ctrl.regs_W: active '0'
                reg_DI_EX.mem_ctrl.DC_RW,
                reg_DI_Ex.mem_ctrl.DC_AS,
                ex_halt );

------------------------------------------------------------------

-- Process Etage Extraction de l'instruction et mise a jour de
--      l'etage EI/DI et du PC
EI: process(CLK)
```

```
begin

  -- test du front actif d'horloge
  if (rising_edge (CLK)) then
     -- test du reset
           if (RST='0') then
                    -- reset du PC
                    reg_PC <= PC_DEFL;
           else
               if (ex_halt = '0' or (ex_halt = '1' and mem_branch = '1')) then

           --   if (di_ctrl_di.J_j  = '1' or di_ctrl_di.J_jal  = '1' or
reg_EX_MEM.mem_ctrl.BRANCH ='1') then
                 if ( di_flush='1' or ex_flush ='1' or mem_flush ='1' ) then
                                          reg_EI_DI.flush   <= '1';
                 else
                                          reg_EI_DI.flush   <= '0';
                 end if ;
                           reg_PC(PC'range)<= ei_pc_next;

                        -- Mise a jour du registre inter-etage EI/DI
             reg_EI_DI.code_op  <= ei_inst(OPCODE'range) ;
             reg_EI_DI.code_func  <= ei_inst(FCODE'range) ;

                          reg_EI_DI.rs                        <=
ei_inst(RS'range);
                  reg_EI_DI.rt                      <= ei_inst(RT'range);
                  reg_EI_DI.rd                      <= ei_inst(RD'range);


                          reg_EI_DI.pc_next <= ei_pc_next;
                          reg_EI_DI.inst    <= ei_inst;

        end if; --ei_halt
    end if;
  end if;
end process EI;

-- ============================================================
-- === Etage DI ===============================================
-- ============================================================

------------------------------------------------------------------
-- instantiation et mapping du composant registres
regf : entity work.registres(behavior)
                        generic map ( DBUS_WIDTH=>CPU_DATA_WIDTH,
ABUS_WIDTH=>REG_WIDTH, ACTIVE_FRONT=>REG_FRONT )
                        port map ( CLK=>CLK, W=>reg_MEM_ER.er_ctrl.regs_W,
RST=>RST, D=>er_regd,

ADR_A=>reg_EI_DI.inst(RS'range), ADR_B=>reg_EI_DI.inst(RT'range),
                                                        ADR_W=>er_adrw,
QA=>di_qa, QB=>di_qb );


------------------------------------------------------------------
-- Affectations dans le domaine combinatoire de l'etage DI

-- Calcul de l'extension de la valeur immediate
di_imm_ext(IMM'range) <= reg_EI_DI.inst(IMM'range);
```

```
di_imm_ext(DATA'high downto IMM'high+1) <= (others => '0') when
di_ctrl_di.signed_ext='0' else
                                                        (others =>
reg_EI_DI.inst(IMM'high));


---- Appel de la procedure contol
UC: control(    reg_EI_DI.flush,
                reg_EI_DI.inst(OPCODE'range),
                                    reg_EI_DI.inst(FCODE'range),
                                    reg_EI_DI.inst(BCODE'range),
                                    di_ctrl_di,
                                    di_ctrl_ex,
                                    di_ctrl_mem,
                                    di_ctrl_er );

di_flush <= '1' when di_ctrl_di.J_j  = '1' or di_ctrl_di.J_jal  = '1' else
            '0';
--------------------------------------------------------------------

-- Process Etage Extraction de l'instruction et mise a jour de
--      l'etage DI/EX
DI: process(CLK)
begin

-- test du front actif d'horloge
if (rising_edge (CLK)) then
         -- test du reset et signal flush
         if (RST='0' or ex_flush ='1' or mem_flush ='1') then
                  -- reset des controle du pipeline
                reg_DI_EX.ex_ctrl       <= EX_DEFL;
                reg_DI_EX.mem_ctrl <= MEM_DEFL;
                reg_DI_EX.er_ctrl       <= ER_DEFL;

         else
                  -- Mise a jour du registre inter-etage DI/EX
                reg_DI_EX.pc_next               <= reg_EI_DI.pc_next;
                reg_DI_EX.rs                        <=
reg_EI_DI.inst(RS'range);
                reg_DI_EX.rt                        <=
reg_EI_DI.inst(RT'range);
                reg_DI_EX.rd                        <=
reg_EI_DI.inst(RD'range);
                reg_DI_EX.val_dec               <=
reg_EI_DI.inst(VALDEC'range);
                reg_DI_EX.imm_ext               <= di_imm_ext;
                reg_DI_EX.jump_adr      <= reg_EI_DI.inst(JADR'range);
                reg_DI_EX.rs_read               <= di_qa;
                reg_DI_EX.rt_read               <= di_qb;
                  reg_DI_EX.code_op  <= reg_EI_DI.inst(OPCODE'range) ;

                -- Mise a jour des signaux de controle
                reg_DI_EX.ex_ctrl               <= di_ctrl_ex;
                reg_DI_EX.mem_ctrl      <= di_ctrl_mem;
                reg_DI_EX.er_ctrl               <= di_ctrl_er;

         end if;
end if ;

end process DI;
```

```
-- ============================================================
-- === Etage EX ===============================================
-- ============================================================
-- Process
--l'etage  EX/MEM

-- affectation combinatoire dans l'etage exécution

ex_reg_dst <= reg_DI_EX.rd when reg_DI_EX.ex_ctrl.reg_dst=REG_RD else
                                  reg_DI_EX.rt when
reg_DI_EX.ex_ctrl.reg_dst=REG_RT else
                                  (others => '1'); -- R31

-- Appel de l'unité d'envoie
UE: envoi (  reg_DI_EX.code_op,

             -- reg_EX_MEM.code_op,

             reg_DI_EX.rs,      -- Registe Lecture 1 RS
                   reg_DI_EX.rt,      -- Registre Lecture 2 RT
                   reg_EX_MEM.reg_dst,
                   reg_MEM_ER.reg_dst,
                   reg_EX_MEM.er_ctrl.regs_W,
                   reg_MEM_ER.er_ctrl.regs_W,

                   -- mem_halt,
                   ex_MUX_mem_data,
                   ex_MUX_ALU_XA,
                   ex_MUX_ALU_XB);


ex_ALU_A  <= reg_DI_EX.rs_read when reg_DI_EX.ex_ctrl.ALU_SRCA = REGS_QA  else
             reg_DI_EX.rt_read when reg_DI_EX.ex_ctrl.ALU_SRCA = REGS_QB  else
                               reg_DI_EX.imm_ext ;

ex_ALU_XA <= ex_ALU_A          when ex_MUX_ALU_XA = ALU_A                 else
             reg_EX_MEM.ual_s  when ex_MUX_ALU_XA = EX_MEM_A              else
             --MEM_mem_Q when ex_MUX_ALU_XA = MEM_Q              else
             er_regd ;

ex_ALU_B  <= reg_DI_EX.rt_read                        when
reg_DI_EX.ex_ctrl.ALU_SRCB = REGS_QB  else
             reg_DI_EX.imm_ext                               when
reg_DI_EX.ex_ctrl.ALU_SRCB = IMMD     else
             X"000000" & "000" & reg_DI_EX.val_dec  when
reg_DI_EX.ex_ctrl.ALU_SRCB = VAL_DEC  else
             X"00000010"                              when
reg_DI_EX.ex_ctrl.ALU_SRCB = VAL_16   else
                                    (others => '0');

ex_ALU_XB <= ex_ALU_B          when ex_MUX_ALU_XB =  ALU_B                 else
             reg_EX_MEM.ual_s  when ex_MUX_ALU_XB =  EX_MEM_B             else
             --MEM_mem_Q when ex_MUX_ALU_XB = MEM_Q              else
             er_regd ;

ex_mem_data  <= reg_EX_MEM.ual_s  when ex_MUX_mem_data =  Data_ual_s     else
              reg_DI_EX.rt_read; -- Need for store instruction

--P_ALU:alu
P_ALU:  alu(  ex_ALU_XA, ex_ALU_XB, ex_ALU_S, ex_ALU_N, ex_ALU_V, ex_ALU_Z,
ex_ALU_C, reg_DI_EX.ex_ctrl.ALU_SIGNED, reg_DI_EX.ex_ctrl.ALU_OP);
```

```
-- calcul d'adresse de branch
   ex_pc_branch <= reg_EX_MEM.imm_ext (PC_WIDTH -1 downto 0) +
reg_EX_MEM.pc_next ;


-- Flush
ex_flush <= '1' when reg_DI_EX.ex_ctrl.J_jr = '1' or reg_DI_EX.ex_ctrl.J_jalr
= '1' else
            '0';


EX: process(CLK)
begin

        -- test du front actif d'horloge
        if (rising_edge (CLK)) then
                -- test du reset
    if (RST='0' or mem_flush ='1') then
                    -- reset des controle du pipeline
                    reg_EX_MEM.mem_ctrl <= MEM_DEFL;
                    reg_EX_MEM.er_ctrl  <= ER_DEFL ;
          else
                -- Mise a jour du registre inter-etage DI/EX
                reg_EX_MEM.pc_next                  <= reg_DI_EX.pc_next;
                reg_EX_MEM.ual_S        <= ex_ALU_S ;
                reg_EX_MEM.ual_N        <= ex_ALU_N ;
                reg_EX_MEM.ual_V        <= ex_ALU_V ;
                reg_EX_MEM.ual_Z        <= ex_ALU_Z ;
                reg_EX_MEM.ual_C        <= ex_ALU_C ;
                reg_EX_MEM.reg_dst      <= ex_reg_dst;
                reg_EX_MEM.rt                       <= reg_DI_EX.rt;
                reg_EX_MEM.rs                       <= reg_DI_EX.rs;

    -- reg_EX_MEM.rt_read      <= reg_DI_EX.rt_read; -- Need for store
instruction
    reg_EX_MEM.rt_read      <= ex_mem_data;
    reg_EX_MEM.imm_ext      <= reg_DI_EX.imm_ext;
                reg_EX_MEM.code_op  <= reg_DI_EX.code_op ;

                -- Mise a jour des signaux de controle
                reg_EX_MEM.mem_ctrl         <= reg_DI_EX.mem_ctrl ;
                reg_EX_MEM.er_ctrl      <= reg_DI_EX.er_ctrl ;
          end if;
  end if;

end process EX ;

-- ============================================================
-- === Etage MEM ==============================================
-- ============================================================

--instanciation et mapping du composant cache donnée
dcache : entity work.memory(behavior)

                        generic map ( DBUS_WIDTH=>CPU_DATA_WIDTH,
ABUS_WIDTH=>CPU_ADR_WIDTH, MEM_SIZE=>L1_ISIZE,

ACTIVE_FRONT=>L1_FRONT, FILENAME=>DFILE )
                        port map    ( RST=>RST, CLK=>CLK,
RW=>reg_EX_MEM.mem_ctrl.DC_RW , DS=>reg_EX_MEM.mem_ctrl.DC_DS,
```

```
                                        Signed=>reg_EX_MEM.mem_ctrl.DC_SIGNED,
AS=> reg_EX_MEM.mem_ctrl.DC_AS, Ready=>open,

Berr=>open, ADR=>reg_EX_MEM.ual_s, D=>reg_EX_MEM.rt_read , Q=> MEM_mem_Q );

-- Branchement
mem_b_type  <=  reg_EX_MEM.ual_Z
                                                          when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BEQ   else
               not(reg_EX_MEM.ual_Z)

                                                          when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BNE   else
                    (reg_EX_MEM.ual_Z) or ((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V))
                                                              when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BLEZ  else
                    (not(reg_EX_MEM.ual_Z)) and not((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V))
                                                              when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BGTZ  else
                    (not(reg_EX_MEM.ual_Z)) and ((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V))
                                                              when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BLTZ  else
                    ((reg_EX_MEM.ual_Z)) or not((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V)) ;
                                                       --when
reg_EX_MEM.mem_ctrl.BRAN_TYPE = B_BGEZ

mem_branch    <= (reg_EX_MEM.mem_ctrl.BRANCH and mem_b_type) ;

-- Flush
mem_flush <= '1' when mem_branch = '1' else
             '0';

MEM: process(CLK)
begin

    if (rising_edge (CLK)) then
        if (RST='0') then
                     reg_MEM_ER.er_ctrl         <= ER_DEFL ;
           else
                    -- Mise a jour du registre inter-etage DI/EX
                    reg_MEM_ER.mem_Q <= MEM_mem_Q;
                    reg_MEM_ER.pc_next           <= reg_EX_MEM.pc_next;
                    reg_MEM_ER.reg_dst   <= reg_EX_MEM.reg_dst;
                    reg_MEM_ER.ual_S     <= reg_EX_MEM.ual_S;

                    -- reg_MEM_ER.code_op  <= reg_EX_MEM.code_op ;

                    -- Mise a jour des signaux de controle
                    reg_MEM_ER.er_ctrl   <= reg_EX_MEM.er_ctrl;
           end if;
    end if ;
end process MEM ;

-- ============================================================
-- === Etage ER ===============================================
-- ============================================================
```

```
er_adrw <= reg_MEM_ER.reg_dst    ;

er_regd <= reg_MEM_ER.ual_S  when reg_MEM_ER.er_ctrl.REGS_SRCD = ALU_S  else
           reg_MEM_ER.mem_Q  when reg_MEM_ER.er_ctrl.REGS_SRCD = MEM_Q   else
                       reg_MEM_ER.pc_next;

end behavior;
```

```
--------------------------------------------------------------------------------
-- RISC processor general definitions
-- PHOR Vicheka
-- THIEBOLT Francois
--------------------------------------------------------------------------------

-- library definitions
library IEEE;

-- library uses
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.cpu_package2_1.all;
--
-------------------------------------------------------------------------------
-- the package contains types, constants, and function prototypes
--
-------------------------------------------------------------------------------
package cpu_package is


-- ================================================================
-- DEFINITION DE FONCTIONS/PROCEDURES de base
-- ================================================================

  -- Fonction log2
        --      calcule le logarithme base2 d'un entier naturel, ou plus
exactement
        --      renvoie le nombre de bits necessaire pour coder un entier
naturel I
        --function log2 (I: in natural) return natural;

-- ================================================================
-- TYPES/CONSTANT DEFINITIONS
-- ================================================================

-----------------------------------------------------------------
-- HARDWARE definitions
-----------------------------------------------------------------
        -- define CPU core physical sizes
        constant CPU_DATA_WIDTH    : positive := 32;                    --
data bus width
        constant CPU_INST_WIDTH    : positive := CPU_DATA_WIDTH; --
instruction bus width
        constant CPU_ADR_WIDTH     : positive := 32;                    --
address bus width, byte format


        -- define MISC CPU CORE specs
        constant CPU_WR_FRONT        : std_logic:= '1';                 --
pipes write active front
        constant PC_WIDTH                : positive     := 26;
     -- bits pour le PC format mot memoire
        constant PCLOW                     : positive   :=
log2(CPU_INST_WIDTH/8);


        -- define REGISTERS physical sizes
        constant REG_WIDTH         : positive   := 5;
-- registers address bus with
```

```
        constant REG_FRONT         : std_logic   := CPU_WR_FRONT;

        -- define instruction & data CACHE physical sizes
        constant L1_SIZE           : positive    := 32;    --16         --
taille des caches L1 en nombre de mots
        constant L1_ISIZE          : positive    := L1_SIZE;        -- taille
du cache instruction L1 en nombre de mots
        constant L1_DSIZE          : positive    := L1_SIZE;        -- taille
du cache donnees L1 en nombre de mots
        constant L1_FRONT          : std_logic   := CPU_WR_FRONT;

        -- define types/subtypes according to hardware specs
        subtype PC     is std_logic_vector(PC_WIDTH-1+PCLOW downto PCLOW);
        subtype INST   is std_logic_vector(CPU_INST_WIDTH-1 downto 0);
        subtype ADDR   is std_logic_vector(CPU_ADR_WIDTH-1 downto 0);
        subtype DATA   is std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
        subtype REGS   is std_logic_vector(REG_WIDTH-1 downto 0);

        -- define default values
        constant PC_DEFL : ADDR := conv_std_logic_vector(0,ADDR'length);

------------------------------------------------------------------
-- SOFTWARE definitions
------------------------------------------------------------------
        -- define the basic ALU operations
        --      Le fait qu'une operation soit signee ou non sera indique a
l'ALU par un signal
        --      supplementaire, ceci dit cela n'affecte que les bits d'etat.
        type ALU_OPS is
(ALU_ADD,ALU_SUB,ALU_AND,ALU_OR,ALU_NOR,ALU_XOR,ALU_SLT,ALU_LSL,ALU_LSR);


        -- define the size of datas during memory access
        type MEM_DS is (MEM_8,MEM_16,MEM_32,MEM_64);

        -- definition des champs d'instructions
        subtype OPCODE          is std_logic_vector(31 downto 26);
        subtype RS       is std_logic_vector(25 downto 21);
        subtype RT                  is std_logic_vector(20 downto 16);
        subtype RD       is std_logic_vector(15 downto 11);
        subtype VALDEC   is std_logic_vector(10 downto 6);
        subtype FCODE             is std_logic_vector(5 downto 0);
        subtype BCODE             is std_logic_vector(20 downto 16);
        subtype IMM                 is std_logic_vector(15 downto 0);
        subtype JADR               is std_logic_vector(25 downto 0);

        ------------------------------------------------------------------
        -- definition des constantes de type des instructions

        ------------------------------------------------------------------
        -- Instructions de type R : Registre Inst[OPCODE'range]
        constant TYPE_R : std_logic_vector := "000000" ;

        -- Fonctions associes au TYPE_R Inst[FCODE'range]
        constant ADD            : std_logic_vector := "100000" ;
        constant ADDU   : std_logic_vector := "100001" ;
        constant SUB            : std_logic_vector := "100010" ;
        constant SUBU   : std_logic_vector := "100011" ;
        constant iAND   : std_logic_vector := "100100" ; -- i pour
differencier l'operateur du meme nom
        constant iOR            : std_logic_vector := "100101" ; -- i pour
differencier l'operateur du meme nom
```

```
        constant iNOR   : std_logic_vector := "100111" ; -- i pour
differencier l'operateur du meme nom
        constant iXOR   : std_logic_vector := "100110" ; -- i pour
differencier l'operateur du meme nom
        constant SLT            : std_logic_vector := "101010" ;
        constant SLTU   : std_logic_vector := "101011" ;
        constant LSL            : std_logic_vector := "000000" ;
        constant LSR            : std_logic_vector := "000010" ;
        constant JR              : std_logic_vector := "001000" ;
        constant JALR   : std_logic_vector := "001001" ;

        ---------------------------------------------------------------
        -- Instruction de Type B : Branchement
        constant TYPE_B : std_logic_vector := "000001" ;
        -- Branch associes au TYPE_B Inst[BCODE'range]
        constant BLTZ            : std_logic_vector := "00000" ;
        constant BGEZ            : std_logic_vector := "00001" ;
        constant BLTZAL         : std_logic_vector := "10000" ;
        constant BGEZAL : std_logic_vector := "10001" ;

        ---------------------------------------------------------------
        -- Instructions de type J : Saut
        constant J                      : std_logic_vector := "000010" ;
        constant JAL     : std_logic_vector := "000011" ;

        ---------------------------------------------------------------
        -- Instruction de type I : Immediat
        constant ADDI           : std_logic_vector := "001000" ;
        constant ADDIU  : std_logic_vector := "001001" ;
        constant SLTI           : std_logic_vector := "001010" ;
        constant SLTIU  : std_logic_vector := "001011" ;
        constant ANDI           : std_logic_vector := "001100" ;
        constant ORI             : std_logic_vector := "001101" ;
        constant XORI           : std_logic_vector := "001110" ;
        constant LUI             : std_logic_vector := "001111" ;
        constant LB               : std_logic_vector := "100000" ;
        constant LH               : std_logic_vector := "100001" ;
        constant LW               : std_logic_vector := "100011" ;
        constant LBU             : std_logic_vector := "100100" ;
        constant LHU             : std_logic_vector := "100101" ;
        constant SB               : std_logic_vector := "101000" ;
        constant SH               : std_logic_vector := "101001" ;
        constant SW               : std_logic_vector := "101011" ;
        constant BEQ             : std_logic_vector := "000100" ;
        constant BNE             : std_logic_vector := "000101" ;
        constant BLEZ           : std_logic_vector := "000110" ;
        constant BGTZ           : std_logic_vector := "000111" ;

-------------------------------------------------------------------
-------------------------------------------------------------------
-- HARDWARE Multiplexer and Pipelines registers definitions
-------------------------------------------------------------------
-------------------------------------------------------------------

        --================================================
        -- Definition des multiplexeurs dans les etages
        --================================================

  ------------------------Mux_aleas--------------------------------
        type MUX_ALU_XA   is (ALU_A,EX_MEM_A,MEM_ER_A, MEM_Q);
        type MUX_ALU_XB   is (ALU_B,EX_MEM_B,MEM_ER_B, MEM_Q);
```

```
        type MUX_mem_data   is (Data_ual_s, Data_rt_read);

  ------------------------------------------------------------------
        type MUX_ALU_A    is (REGS_QA,REGS_QB,IMMD);
        type MUX_ALU_B    is (REGS_QB,IMMD,VAL_DEC, VAL_16, VAL_0);
        type MUX_REG_DST  is (REG_RD,REG_RT,R31);
        type MUX_REGS_D   is (ALU_S,MEM_Q,NextPC);
        type MUX_BRANCH   is (B_beq,B_bne,B_blez,B_bgtz,B_bltz,B_bgez);

--      constant VAL16    : std_logic_vector := X"00000010" ;
--      constant VAL0     : std_logic_vector := X"00000000" ;

  --================================================
  -- Definitions des structures de controles des etages
  --================================================

        ---------------1>>> Structure des signaux de control de l'etage EI
<<<---------------
        -- No signal de control in this stage! Halt and Flus is use by branch
harzard control and LW harzard control.


  ---------------2>>> Structure des signaux de control de l'etage DI
<<<---------------
        type mxDI is record
                SIGNED_EXT              : std_logic;               --
extension signee ou non donnee immediate
                J_j          : std_logic ;                        -- saut
immediate SAUT_IMMD= J_j
                J_jal     : std_logic;  -- SAUT   : J_jal
        end record;
        -- default DI control
        constant DI_DEFL : mxDI := ( SIGNED_EXT=>'0',others => '0');

-------------------------------------------------------------------------------------------

  ---------------3>>> Structure des signaux de control de l'etage EX
<<<---------------
        type mxEX is record
                ALU_OP                      : ALU_OPS;                --
operation sur l'ALU
                ALU_SIGNED              : std_logic;               --
operation ALU signee ou non
                ALU_SRCA                  : MUX_ALU_A;             -- mux
pour entree A de l'ALU
                ALU_SRCB                  : MUX_ALU_B;             -- mux
pour entree B de l'ALU
                REG_DST            : MUX_REG_DST;                -- mux pour
registre destinataire
                J_jr         : std_logic;     -- SAUT : J_jr
                J_jalr    : std_logic;  -- SAUT   : J_jalr
        end record;
        -- default EX control
  constant EX_DEFL : mxEX := ( ALU_OP=>ALU_OPS'low, ALU_SRCA=>MUX_ALU_A'low,
ALU_SRCB=>MUX_ALU_B'low,

                  REG_DST=>MUX_REG_DST'low, others=>'0' );

-------------------------------------------------------------------------------------------

  ---------------4>>> Structure des signaux de control de l'etage MEM
<<<---------------
```

```
        type mxMEM is record
                DC_DS             : MEM_DS;                -- DataCache
taille d'acces 8/16/32/64/...
                DC_RW             : std_logic;          -- DataCache signal
R/W*
                DC_AS             : std_logic;          -- DataCache signal
Address Strobe
                DC_SIGNED       : std_logic;            -- DataCache
operation signee ou non (lecture)
        BRANCH     : std_logic;         -- BRANCH active or not
          B_TYPE    : MUX_BRANCH;       --control de type de branchement
        end record;
        -- default MEM control
        constant MEM_DEFL : mxMEM := ( DC_DS=>MEM_DS'low, DC_RW=>'1', B_TYPE
=>MUX_BRANCH'low, others=>'0' );

-------------------------------------------------------------------------------

  ---------------5>>> Structure des signaux de control de l'etage ER
<<<---------------
        type mxER is record
                REGS_W           : std_logic;    -- signal d'ecriture W* du
banc de registres
                REGS_SRCD       : MUX_REGS_D;   -- mux vers bus de donnee D
du banc de registres
        end record;
        -- default ER control
        constant ER_DEFL : mxER := ( REGS_W=>'1', REGS_SRCD=>MUX_REGS_D'low,
others=>'0' );

-------------------------------------------------------------------------------

  --=================================================
  -- Definition des strucures des registres pipeline
  --=================================================

        --1>>> Structure du registre EI/DI
        type EI_DI is record
                -- === Data ===
                pc_next : std_logic_vector (PC'range);          -- cp
incremente
                inst              : std_logic_vector (INST'range);      --
instruction extraite
                flush    : std_logic ; --PV-05-02-2013
                code_op         : std_logic_vector (OPCODE'length-1 downto
0);
                code_func : std_logic_vector(FCODE'length-1 downto 0);
                rs                  : std_logic_vector (REGS'range);     --
champ rs
                rt                  : std_logic_vector (REGS'range);     --
champ rt
                rd                  : std_logic_vector (REGS'range);     --
champ rd

                -- === Control ===
                --di_ctrl               : mxDI;         -- signaux de control
de l'etage EX
        end record;

        --2>>> Structure du registre DI/EX
        type DI_EX is record
```

```
                -- === Data ===
                pc_next         : std_logic_vector (PC'range);              --
cp incremente propage
                rs                  : std_logic_vector (REGS'range);     --
champ rs
                rt                  : std_logic_vector (REGS'range);     --
champ rt
                rd                  : std_logic_vector (REGS'range);     --
champ rd
                val_dec         : std_logic_vector (VALDEC'range);      --
valeur de decalage
                imm_ext         : std_logic_vector (DATA'range);  -- valeur
immediate etendue
                jump_adr: std_logic_vector (JADR'RANGE);   -- champ adresse de
sauts
                rs_read         : std_logic_vector (DATA'range);  -- donnee
du registre lu rs
                rt_read         : std_logic_vector (DATA'range);  -- donnee
du registre lu rt
                code_op         : std_logic_vector (OPCODE'length-1 downto
0);
                -- === Control ===
                ex_ctrl         : mxEX;         -- signaux de control de
l'etage EX
                mem_ctrl : mxMEM;       -- signaux de control de l'etage MEM
                er_ctrl : mxER;         -- signaux de control de l'etage ER
        end record;

        --3>>> Structure du registre EX/MEM
        type EX_MEM is record
                -- === Data ===
                pc_next          : std_logic_vector (PC'range);           --
cp incremente propage
                ual_S              : std_logic_vector (DATA'range);
                ual_N              : std_logic ;
                ual_V              : std_logic ;
                ual_C              : std_logic ;
                ual_Z              : std_logic ;

                rs                   : std_logic_vector (REGS'range);     --
champ rs
                rt                   : std_logic_vector (REGS'range);     --
champ rt
                imm_ext         : std_logic_vector (DATA'range);  -- valeur
immediate etendue
                -- Remove pc_branch
          --pc_branch   : std_logic_vector (PC'range);            -- adresse
de branchement
                reg_dst         : std_logic_vector (REGS'range);  -- registre
destination (MUX_REG_DST)
                rt_read         : std_logic_vector (DATA'range);
                code_op         : std_logic_vector (OPCODE'length-1 downto
0);
                -- === Control ===
                mem_ctrl : mxMEM;                                  -- signaux de
control de l'etage MEM
                er_ctrl : mxER;                                    -- signaux de
control de l'etage ER
        end record;

        --4>>> Structure du registre MEM/ER
```

```
        type MEM_ER is record
                -- Signaux
                pc_next         : std_logic_vector (PC'range);           --
cp incremente propage
                mem_Q           : std_logic_vector (DATA'range);  -- sortie
memoire
                ual_S           : std_logic_vector (DATA'range);   --
resultat ual propage
                reg_dst         : std_logic_vector (REGS'range); -- registre
destination propage
          -- code_op            : std_logic_vector (OPCODE'length-1 downto
0);

                -- Signaux de control
                er_ctrl         : mxER;                                  --
signaux de control de l'etage ER propage
        end record;

-- ============================================================
-- DEFINITION DE FONCTIONS/PROCEDURES
-- ============================================================

-- Si on ne specifie rien devant les parametres...il considere que c'est une
variable
-- exemple : procedure adder_cla (A,B: in std_logic_vector;...)
-- ici A et B sont consideres comme etant des variables...
-- Sinon il faut : procedure adder_cla (signal A,B: in std_logic_vector;...)

        -- Fonction "+" --> procedure adder_cla
        -- function "+" (A,B: in std_logic_vector) return std_logic_vector;

        -- Procedure adder_cla
        procedure adder_cla (   A,B: in std_logic_vector; C_IN : in
std_logic;
                                                S : out std_logic_vector;
C_OUT : out std_logic; V : out std_logic);


        -- Procedure alu
        --      on notera l'utilisation d'un signal comme parametres formels
de type OUT
        procedure alu ( A,B: in std_logic_vector; signal S: out
std_logic_vector;
                                  signal N,V,Z,C: out std_logic; SIGNED_OP:
in std_logic; CTRL_ALU: in ALU_OPS);


        -- Procedure control
        --              permet de positionner les signaux de control pour
chaque etage (EX MEM ER)
        --              en fonction de l'instruction identifiee soit par son
code op, soit par
        --              son code fonction, soit par son code branchement.
        procedure control (     flush : in std_logic ;
                                OP    : in std_logic_vector(OPCODE'length-1
downto 0);
                                          F     : in
std_logic_vector(FCODE'length-1 downto 0);
                                          B     : in
std_logic_vector(BCODE'length-1 downto 0);
                                         signal DI_ctrl  : out mxDI;
        -- signaux de controle de l'etage DI
```

```
                                                signal EX_ctrl  : out mxEX;
        -- signaux de controle de l'etage EX
                                                signal MEM_ctrl : out mxMEM;
-- signaux de controle de l'etage MEM
                                                signal ER_ctrl  : out mxER );
-- signaux de controle de l'etage ER

        -- Procedure envoi
        procedure envoi (   DI_EX_code_op  : in std_logic_vector
(OPCODE'length-1 downto 0);

                            EX_MEM_code_op  : in std_logic_vector
(OPCODE'length-1 downto 0);

                                 DI_EX_Register_Rs   : in std_logic_vector
(REGS'range);
                        DI_EX_Register_Rt   : in std_logic_vector (REGS'range);
                        EX_MEM_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                        MEM_ER_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                        EX_MEM_Regs_W     : in std_logic;
                        MEM_ER_Regs_W     : in std_logic;

                        signal mem_halt : in std_logic;
                        signal mem_data   : out MUX_mem_data;
                        signal ALU_XA    : out MUX_ALU_XA;
                        signal ALU_XB    : out MUX_ALU_XB );

  -- Procedure aleaLW
  procedure aleaLW (  EX_MEM_Register_Rt : in std_logic_vector (REGS'range);
                      DI_EX_Register_Rs : in std_logic_vector (REGS'range);
                      DI_EX_Register_Rt : in std_logic_vector (REGS'range);
                      EX_MEM_Regs_W      : in std_logic;
                      EX_MEM_DC_RW      : in std_logic;
                      EX_MEM_DC_AS      : in std_logic;
                      signal halt       : out std_logic );


end cpu_package;

--
-------------------------------------------------------------------------------
-- the package contains types, constants, and function prototypes
--
-------------------------------------------------------------------------------
package body cpu_package is

  -- ============================================================
  -- DEFINITION DE FONCTIONS/PROCEDURES
  -- ============================================================

  -->>> fonction log2
--  function log2 (I: in natural) return natural is
--    variable ip : natural := 1; -- valeur temporaire
--      variable iv : natural := 0; -- nb de bits
--  begin
--      while ip < i loop
--              ip := ip + ip; -- ou ip := ip * 2
--              iv := iv + 1;
--      end loop;
```

```
--       -- renvoie le nombre de bits
--       return iv;
--   end log2;

  -->>> fonction "+" --> procedure adder_cla
--   function "+" (A,B: in std_logic_vector) return std_logic_vector is
--       variable tmp_S : std_logic_vector(A'range);
--       variable tmp_COUT,tmp_V : std_logic;
--   begin
--       adder_cla(A,B,'0',tmp_S,tmp_COUT,tmp_V);
--       return tmp_S;
--   end "+";

  -- Le drapeau overflow V ne sert que lors d'operations signees !!!
  -- Overflow V=1 si operation signee et :
  --             addition de deux grands nombres positifs dont le resultat < 0
  --             addition de deux grands nombres negatifs dont le resultat >= 0
  --             soustraction d'un grand nombre positif et d'un grand nombre
negatif dont le resultat < 0
  --             soustraction d'un grand nombre negatif et d'un grand nombre
positif dont le resultat >= 0
  --    Reviens a faire V = C_OUT xor <carry entrante du dernier bit>
  -->>> procedure adder_cla
  procedure adder_cla ( A,B: in std_logic_vector;C_IN : in std_logic;
                                                        S : out
std_logic_vector;C_OUT : out std_logic;
                                                        V : out std_logic)
is
                            variable G_CLA,P_CLA : std_logic_vector(A'length-1
downto 0);
                            variable C_CLA                      :
std_logic_vector(A'length downto 0);
  begin
        -- calcul de P et G
        G_CLA:= A and B;
        P_CLA:= A or B;
        C_CLA(0):=C_IN;
        for I in 0 to (A'length-1) loop
                C_CLA(I+1):= G_CLA(I) or (P_CLA(I) and C_CLA(I));
        end loop;
        -- mise a jour des sorties
        S:=(A Xor B) xor C_CLA(A'length-1 downto 0);
        C_OUT:=C_CLA(A'length);
    V:= C_CLA(A'length) xor C_CLA(A'length - 1);
  end adder_cla;

  -- procedure alu
  procedure alu (  A,B: in std_logic_vector;signal S: out std_logic_vector;
                                                  signal N,V,Z,C: out
std_logic;SIGNED_OP: in std_logic;
                                                  CTRL_ALU: in ALU_OPS) is
                        variable DATA_WIDTH : positive := A'length;
                         variable b_in       : std_logic_vector(DATA_WIDTH-1
downto 0);
                        variable c_in       : std_logic;
                        variable tmp_S           :
std_logic_vector(DATA_WIDTH-1 downto 0);
                        variable tmp_V           : std_logic;
                        variable tmp_N           : std_logic;
                        variable tmp_C           : std_logic;
                         variable tmp_CLA_C      : std_logic;
```

```
                        variable tmp_CLA_V       : std_logic;
  begin
        -- raz signaux
        tmp_V := '0';
        tmp_N := '0';
        tmp_C := '0';
        -- case sur le type d'operation
        case CTRL_ALU is
                when ALU_ADD | ALU_SUB | ALU_SLT =>
                        b_in := B;
                        c_in := '0';
                        if (CTRL_ALU /= ALU_ADD) then
                                b_in := not(B);
                                c_in := '1';
                        end if;
                        adder_cla(A,b_in,c_in,tmp_S,tmp_C,tmp_V);
                        if (CTRL_ALU = ALU_SLT) then
                                tmp_S := conv_std_logic_vector( (SIGNED_OP
and (tmp_V xor tmp_S(DATA_WIDTH-1))) or (not(SIGNED_OP) and not(tmp_C)) ,
S'length );
                                -- remize à 0 des flags selon definition
                                tmp_C := '0';
                                tmp_V := '0';
                        else
                                tmp_C := not(SIGNED_OP) and tmp_C;
                                tmp_N := SIGNED_OP and tmp_S(DATA_WIDTH-1);
                                tmp_V := SIGNED_OP and tmp_V;
                        end if;
                when ALU_AND =>
                        tmp_S := A and B;
                when ALU_OR =>
                        tmp_S := A or B;
                when ALU_NOR =>
                        tmp_S := A nor B;
                when ALU_XOR =>
                        tmp_S := A xor B;
                when ALU_LSL =>
                        tmp_S := shl(A,B);
                when ALU_LSR =>
                        tmp_S := shr(A,B);
                when others =>
        end case;
        -- affectation de la sortie
        S <= tmp_S;
        -- affectation du drapeau Z (valable dans tous les cas)
        if (tmp_S=conv_std_logic_vector(0,DATA_WIDTH)) then Z <= '1';
        else Z <= '0';
        end if;
        -- affectation des autres drapeaux N,V,C
        C <= tmp_C;
        N <= tmp_N;
        V <= tmp_V;
  end alu;

  -->>> === Procedure control ========================================
  --            Permet de positionner les signaux de control pour chaque etage
(EX MEM ER)
  --            en fonction de l'instruction identifiee soit par son code op,
soit par
  --            son code fonction, soit par son code branchement.
  --  If the signal control has also value X (don't care), write the code in
(if .... end if), not (if ... elsif... else ... end if)
```

```
  --  because we the signal will take the last time value for the value X.
  procedure control ( flush : in std_logic ;
                      OP    : in std_logic_vector(OPCODE'length-1 downto 0);
                                                                  F     : in
std_logic_vector(FCODE'length-1 downto 0);
                                                                  B     :in
std_logic_vector(BCODE'length-1 downto 0);
                                                                  signal
DI_ctrl : out mxDI;             -- signaux de controle de l'etage DI
                                                                  signal
EX_ctrl : out mxEX;             -- signaux de controle de l'etage EX
                                                                  signal
MEM_ctrl: out mxMEM;    -- signaux de controle de l'etage MEM
                                                                  signal
ER_ctrl : out mxER ) is -- signaux de controle de l'etage ER
  begin
        -- Initialisation

        DI_ctrl <= DI_DEFL;
        EX_ctrl <= EX_DEFL;
        MEM_ctrl <= MEM_DEFL;
        ER_ctrl <= ER_DEFL;

    if (flush = '0') then

    --==============================
    ---- ControlDI :
    -->>> SIGNED_EXT: signe-t-on ou non l'extension de la valeur immediate
    -->>> SAUT @IMMD: J_j
    --==============================

        -->>> SIGNED_EXT
        if (  (OP=TYPE_B) or (OP=ADDI) or (OP=ADDIU) or (OP=SLTI) or
(OP=SLTIU) or
              (OP=LB) or (OP=LH) or (OP=LW) or (OP=LBU) or (OP=LHU) or (OP=SB)
or
              (OP=SH) or (OP=SW) or (OP=BEQ) or (OP=BNE) or (OP=BLEZ) or
(OP=BGTZ)  )then
            DI_ctrl.SIGNED_EXT <= '1';
        else
            DI_ctrl.SIGNED_EXT <= '0';
        end if;

             -->>> SAUT @IMMD: J_j
             if (  (OP=J)) then
              DI_ctrl.J_j <= '1';
             end if ;

        -->>> J_jal
        if ( (OP= JAL)) then
                  DI_ctrl.J_jal <= '1';
        end if;

    --==============================
    ---- Control EX
    -->>> SAUT @Reg: J_jr
    -->>> REG_DST
    -->>> ALU_SRCB
    -->>> ALU_SRCA
    -->>> ALU_SIGNED
    -->>> ALU_OP
```

```
    -->>> J_jal
    -->>> J_jalr
    --==============================

        -->>> SAUT @Reg: J_jr
             if ((OP=TYPE_R) and (F=JR)) then
              EX_ctrl.J_jr <= '1';
             end if ;

        -->>> J_jalr
        if ( (OP= TYPE_R) and (F=JALR) ) then
                  EX_ctrl.J_jalr <= '1';
        end if;

        -->>> REG_DST
        if (  (OP=ADDI)or (OP=ADDIU) or (OP=SLTI)or (OP=SLTIU)or (OP=ANDI)or
(OP=ORI)or (OP=XORI)
              or (OP=LUI)or(OP= LB)or (OP= LH)or (OP= LW)or (OP= LBU)or (OP=
LHU) ) then
              EX_ctrl.REG_DST <= REG_RT;
        end if;
        if (((OP= TYPE_B)and ((B=BLTZAL)or(B=BGEZAL)))or(OP=JAL))then
                  EX_ctrl.REG_DST <= R31;
        end if;
        if ((OP= TYPE_R)) then
                  EX_ctrl.REG_DST <= REG_RD;
        end if;

        -->>> ALU_SRCB
        if ((OP=TYPE_B)) then
              EX_ctrl.ALU_SRCB <= VAL_0;
        end if;
        if ((OP=LUI)) then
              EX_ctrl.ALU_SRCB <= VAL_16;
        end if;
        if ( (OP=ADDI) or (OP=ADDIU)or (OP=SLTI)or (OP=SLTIU)or (OP=ANDI) or
(OP=ORI)or (OP=XORI) or
             (OP= LB)or (OP= LH) or (OP= LW) or (OP= LBU)or (OP= LHU)or (OP=
SB)or (OP= SH)or(OP= SW) )then
            -- EX_ctrl.ALU_SRCA <= REGS_QA;
              EX_ctrl.ALU_SRCB <= IMMD;
        end if;
        if ( (OP= TYPE_R) and ((F=LSL) or (F=LSR)) )then
              EX_ctrl.ALU_SRCB <= VAL_DEC;
        end if;
        if ( ( (OP= TYPE_R) and ((F=ADD) or (F=ADDU) or (F=SUB) or (F=SUBU) or
(F=iAND) or (F=iOR) or (F=iNOR) or
                                  (F=iXOR) or (F=SLT) or (F=SLTU) or (F=JR) or
(F=JALR)) )
                or (OP= BEQ) or (OP= BNE) or (OP= BLEZ) or (OP= BGTZ)
              ) then
              EX_ctrl.ALU_SRCB <= REGS_QB;
        end if;

        -->>> ALU_SRCA
        if ( (OP=LUI)) then
              EX_ctrl.ALU_SRCA <= IMMD;
        end if;
        if ( (OP= TYPE_R) and ((F=LSL) or (F=LSR)) )then
              EX_ctrl.ALU_SRCA <= REGS_QB;
        end if;
```

```
        if ( (OP= TYPE_R) or (OP= TYPE_B) or (OP=ADDI) or (OP=ADDIU) or
(OP=SLTI) or (OP=SLTIU) or (OP=ANDI) or (OP=ORI)or (OP=XORI) or
             (OP= LB) or (OP= LH) or (OP= LW) or (OP= LBU) or (OP= LHU) or
(OP= SB) or (OP= SH) or(OP= SW) or
             (OP= BEQ) or(OP= BNE) or(OP= BLEZ) or(OP= BGTZ) ) then
              EX_ctrl.ALU_SRCA <= REGS_QA;
        end if;

        -->>> ALU_SIGNED
        if ( ( (OP=TYPE_R) and ( (F=ADDU) or (F=SUBU) or (F=iAND) or (F=iOR)
or (F=iNOR) or (F=iXOR) or (F=SLTU) or (F=LSL) or (F=LSR) ) ) or
              (OP=ADDIU) or (OP=SLTIU) or (OP=ANDI) or (OP=ORI) or (OP=XORI) )
then
              EX_ctrl.ALU_SIGNED <= '0';
        end if;
        if ( ( (OP=TYPE_R) and ( (F=ADD) or (F=SUB) or (F=SLT) ) ) or (OP=
TYPE_B) or (OP=ADDI) or (OP=SLTI) or (OP=LUI) or
                (OP= LB) or (OP= LH) or (OP= LW) or (OP= LBU) or (OP= LHU) or
(OP= SB) or (OP= SH) or(OP= SW) or (OP=BEQ) or
                (OP=BNE) or (OP=BLEZ) or (OP=BGTZ) ) then
              EX_ctrl.ALU_SIGNED <= '1';
        end if;

        -->>> ALU_OP
        if ( ((OP=TYPE_R)and (F=iAND)) or (OP=ANDI) ) then
              EX_ctrl.ALU_OP <= ALU_AND;
         end if;

             if (((OP= TYPE_R) and (F=iOR))or (OP=ORI)) then
                  EX_ctrl.ALU_OP <= ALU_OR;
             end if;

             if (((OP= TYPE_R) and (F=iXOR))or (OP=XORI)) then
                 EX_ctrl.ALU_OP <= ALU_XOR;
             end if;

             if ((OP= TYPE_R) and (F=iNOR)) then
                 EX_ctrl.ALU_OP <= ALU_NOR;
             end if;

             if ( ((OP= TYPE_R) and ((F=SUB)or (F=SUBU)))or ((OP= TYPE_B)and
((B=BLTZ)or(B=BGEZ)or(B=BLTZAL)or(B=BGEZAL))) or
              (OP=BEQ)or(OP=BNE)or(OP=BLEZ)or(OP=BGTZ)) then
             EX_ctrl.ALU_OP      <= ALU_SUB ;
             end if ;

        if ( ((OP= TYPE_R) and ((F=ADD)or (F=ADDU)))or   (OP= ADDI) or (OP=
ADDIU) or (OP= LB) or (OP= LH) or (OP= LW) or
              (OP= LBU) or (OP= LHU) or (OP= SB) or (OP= SH) or (OP= SW) )
then
             EX_ctrl.ALU_OP      <= ALU_ADD ;
        end if;

        if ( ((OP= TYPE_R) and (F=LSL)) or (OP= LUI) ) then
                 EX_ctrl.ALU_OP <= ALU_LSL;
             end if;

             if ( ((OP= TYPE_R) and (F=LSR)) ) then
                 EX_ctrl.ALU_OP <= ALU_LSR;
             end if;
```

```
             if ( ((OP= TYPE_R) and ((F=SLT) or (F=SLTU))) or (OP= SLTI) or
(OP= SLTIU) ) then
                 EX_ctrl.ALU_OP <= ALU_SLT;
             end if;

    --==============================
    ---- Control MEM
    -->>> BRANCH
    -->>> B_TYPE
    -->>> DC_RW
    -->>> DC_DS
    -->>> DC_SIGNED
    -->>> DC_AS
    --==============================


        -->>> BRANCH
        if ( ((OP= TYPE_B)and ((B=BLTZ)or(B=BGEZ))) or
(OP=BEQ)or(OP=BNE)or(OP=BLEZ)or(OP=BGTZ)) then
             MEM_ctrl.BRANCH <= '1';
             end if ;

        -->>> B_TYPE
        if ( (OP= TYPE_B)and ((B=BLTZ) or (B=BLTZAL) )) then
                 MEM_ctrl.B_TYPE <= B_bltz;
        end if;

        if ( (OP= TYPE_B)and ((B=BGEZ) or (B=BGEZAL))) then
            MEM_ctrl.B_TYPE <= B_bgez;
        end if;

        if (OP=BEQ) then
                 MEM_ctrl.B_TYPE <= B_beq;
        end if;

        if (OP=BNE) then
            MEM_ctrl.B_TYPE <= B_bne;
        end if;

        if (OP=BLEZ) then
            MEM_ctrl.B_TYPE <= B_blez;
        end if;

        if (OP=BGTZ) then
            MEM_ctrl.B_TYPE <= B_bgtz;
        end if;

            -->>> DC_RW
        if ( (OP = LB) or (OP = LH) or (OP = LW) or (OP = LBU) or (OP = LHU) )
then
                 MEM_ctrl.DC_RW <= '1' ;
         end if ;
         if ( (OP = SB) or (OP = SH) or (OP = SW) ) then
                MEM_ctrl.DC_RW<= '0' ;
         end if ;

        -->>> DC_DS
        if ( (OP = LB) or (OP = LBU) or (OP = SB) ) then
                 MEM_ctrl.DC_DS <= MEM_8 ;
           end if ;
        if ( (OP = LH) or (OP = LHU) or (OP = SH) ) then
```

```
                MEM_ctrl.DC_DS <= MEM_16 ;
           end if ;
        if ( (OP = LW) or (OP = SW) ) then
                MEM_ctrl.DC_DS <= MEM_32 ;
           end if ;

        -->>> DC_SIGNED
        if ( (OP = LH) or (OP = LHU) or (OP = SH) or(OP = LB) or (OP = LBU) or
(OP = SB) ) then
                MEM_ctrl.DC_SIGNED <= '1' ;
           end if ;
        if ( (OP = LW) or (OP = SW) ) then
                MEM_ctrl.DC_SIGNED <= '0' ;
           end if ;

        -->>> DC_AS
        if ( (OP = LH) or (OP = LHU) or (OP = SH) or (OP = LB) or (OP = LBU)
or (OP = SB) or
             (OP = LW) or (OP = SW) ) then
                MEM_ctrl.DC_AS<=  '1' ;
              else
                   MEM_ctrl.DC_AS<=  '0' ;
           end if ;

    --==============================
    -- Control ER
    -->>> REGS_SRCD
    -->>> REGS_W : active à l'état 0

    --==============================

         -->>> REGS_SRCD
        if (  (OP= LB) or (OP= LH) or (OP= LW) or (OP= LBU) or (OP= LHU)) then
                  ER_ctrl.REGS_SRCD <= MEM_Q ;
              end if;
             if ( ((OP= TYPE_B) and ((B=BGEZAL)or(B=BLTZAL))) or ((OP=TYPE_R)
and (F=JALR)) or (OP=JAL) ) then
                  ER_ctrl.REGS_SRCD <= NextPC;
              end if;
             if ( ((OP= TYPE_R) and ((F=ADD) or (F=ADDU) or (F=SUB) or
(F=SUBU) or (F=iAND) or (F=iOR) or (F=iNOR) or
                  (F=iXOR) or (F=SLT) or (F=SLTU) or (F=LSL) or (F=LSR))) or
(OP=ADDI) or (OP=ADDIU) or (OP=SLTI) or
                  (OP=SLTIU) or (OP=ANDI) or (OP=ORI)or (OP=XORI) or (OP=LUI)
) then
                  ER_ctrl.REGS_SRCD <= ALU_S;
             end if;

             -->>> REGS_W : active à l'état 0
             if ( ((OP= TYPE_B) and ((B=BLTZ)or(B=BGEZ))) or (OP=J) or (OP =
SB)
                   or (OP = SH) or (OP = SW) or (OP=BEQ) or (OP=BNE) or
(OP=BLEZ) or (OP=BGTZ) )  then
                    ER_ctrl.REGS_W <= '1'; -- inactif
        else
              ER_ctrl.REGS_W <= '0'; -- actif
        end if;

    end if; -- if flush
```

```
  end procedure control;

  -- === Procedure envoi ========================================
  -->>> MUX: ALU_XA
  -->>> MUX: ALU_B

  procedure envoi (   DI_EX_code_op  : in std_logic_vector (OPCODE'length-1
downto 0);

                      EX_MEM_code_op  : in std_logic_vector (OPCODE'length-1
downto 0); -- Aléas LW suivi par add sans suspendre étage MEM

                      DI_EX_Register_Rs   : in std_logic_vector (REGS'range);
                      DI_EX_Register_Rt   : in std_logic_vector (REGS'range);

                      EX_MEM_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                      MEM_ER_Register_Reg_dst  : in std_logic_vector
(REGS'range);
                      EX_MEM_Regs_W     : in std_logic; -- active à l'état '0'
                      MEM_ER_Regs_W     : in std_logic; -- active à l'état '0'

                      signal mem_halt : in std_logic;
                      signal mem_data   : out MUX_mem_data;
                      signal ALU_XA    : out MUX_ALU_XA;
                      signal ALU_XB    : out MUX_ALU_XB
                       ) is
   begin

         -- Initialisation

          mem_data <= Data_rt_read;
         ALU_XA <= ALU_A;
         ALU_XB <= ALU_B;


         --=================
         -->>> MUX: ALU_XA
         --=================

         -- Aléas étage MEM
         IF ( (MEM_ER_Regs_W = '0') AND
              (MEM_ER_Register_Reg_dst /= X"00000000") AND -- Register_Reg_dst
-> R0 Non envoi
              (EX_MEM_Register_Reg_dst /= DI_EX_Register_Rs) AND
              (MEM_ER_Register_Reg_dst = DI_EX_Register_Rs)) THEN

              ALU_XA <= MEM_ER_A; --MEM HAZARD

         -- Aléas étage EX
         ELSIF ( (EX_MEM_Regs_W = '0') AND
                 (EX_MEM_Register_Reg_dst /= X"00000000") AND --
Register_Reg_dst -> R0 Non envoi
                 (EX_MEM_Register_Reg_dst = DI_EX_Register_Rs)) THEN

             -- Aléas LW suivi par add sans suspendre étage MEM
             -- lw suivi par add. À l'instruction add d'étage DI_EX, S'il y a
de halt, entré de ALU_XA prend la valeur de MEM_Q pour addition.
             if ((mem_halt='1') AND ( EX_MEM_code_op=LB or EX_MEM_code_op=LH or
EX_MEM_code_op=LW or EX_MEM_code_op=LBU or
                EX_MEM_code_op=LHU ) ) then
```

```
                ALU_XA <= MEM_Q;
            else
                ALU_XA <= EX_MEM_A; --EX HAZARD
            end if;

        -- Pas d'Aléas
        ELSE
            ALU_XA <= ALU_A; --NO HAZARD
        END IF;

        --================
        -->>> MUX: ALU_B
        --================

        -- Aléas étage MEM
        IF (  (MEM_ER_Regs_W = '0') AND
                (MEM_ER_Register_Reg_dst /= X"00000000") AND --
Register_Reg_dst -> R0 Non envoi
                (EX_MEM_Register_Reg_dst /= DI_EX_Register_Rt) AND
                (MEM_ER_Register_Reg_dst = DI_EX_Register_Rt) ) THEN
          -- Pour instruction
          -- lw R1, 4(R0) & sw R1, 2(R0)
          -- lw R1, 4(R0) & lw R1, 2(R0)
          -- lw R1, 4(R0) & addi R2, R1, 2
          -- MEM_ER_Register_Reg_dst = DI_EX_Register_Rt; mais pour sw il faut
que la sortie QA ou Rs (R0) + Valeur IMMD (2)
          -- Donc on doit mettre la condition ci-dessous pour dire que on ne
mets pas la valeur entrée de B de ALU = la sortie de mémoire.
          IF (
                DI_EX_code_op/= SW AND DI_EX_code_op/= SB AND DI_EX_code_op/=
SH AND
                DI_EX_code_op/= LB AND DI_EX_code_op/= LH AND DI_EX_code_op/=
LW AND DI_EX_code_op/= LBU AND
                DI_EX_code_op/= LHU AND

                DI_EX_code_op/= BGEZ AND DI_EX_code_op/= BLTZAL AND
DI_EX_code_op/= BGEZAL AND

                DI_EX_code_op/= ADDI AND DI_EX_code_op/=ADDIU AND
DI_EX_code_op/=SLTI AND DI_EX_code_op/=SLTIU AND
                DI_EX_code_op/=ANDI AND DI_EX_code_op/=ORI AND
DI_EX_code_op/=XORI AND
                DI_EX_code_op/=LUI
              ) THEN
            ALU_XB <= MEM_ER_B; --MEM
          END IF;

        -- Aléas étage EX
        ELSIF (  (EX_MEM_Regs_W = '0') AND
                  (EX_MEM_Register_Reg_dst /= X"00000000") AND --
Register_Reg_dst -> R0 Non envoi
                  (EX_MEM_Register_Reg_dst = DI_EX_Register_Rt) ) THEN
          -- Pour instruction
          -- addi R1, R0, 1
          -- sw R1, 2(R0)
          -- EX_MEM_Register_Reg_dst = DI_EX_Register_Rt; mais pour sw il faut
que la sortie QA ou Rs (R0) + Valeur IMMD (2)
          -- Donc on doit mettre la condition ci-dessous pour dire que on ne
mets pas la valeur entrée de B de ALU = la sortie de ALU_S avant.
          -- addi R1, R0, 1
          -- lw R1, 4(R0)
```

```
          -- Pareil que sw!
           IF (
               DI_EX_code_op/= SW AND DI_EX_code_op/= SB AND DI_EX_code_op/=
SH AND
               DI_EX_code_op/= LB AND DI_EX_code_op/= LH AND DI_EX_code_op/=
LW AND DI_EX_code_op/= LBU AND
               DI_EX_code_op/= LHU AND

               DI_EX_code_op/= BGEZ AND DI_EX_code_op/= BLTZAL AND
DI_EX_code_op/= BGEZAL AND

               DI_EX_code_op/= ADDI AND DI_EX_code_op/=ADDIU AND
DI_EX_code_op/=SLTI AND DI_EX_code_op/=SLTIU AND
               DI_EX_code_op/=ANDI AND DI_EX_code_op/=ORI AND
DI_EX_code_op/=XORI AND
               DI_EX_code_op/=LUI
              ) THEN

                -- Aléas LW suivi par add sans suspendre étage MEM
                -- lw suivi par add. À l'instruction add d'étage DI_EX, S'il y
a de halt, entré de ALU_XA prend la valeur de MEM_Q pour addition.
                if ((mem_halt='1') AND ( EX_MEM_code_op=LB or
EX_MEM_code_op=LH or EX_MEM_code_op=LW or EX_MEM_code_op=LBU or
                     EX_MEM_code_op=LHU ) ) then
                    ALU_XB <= MEM_Q;
                else
                    ALU_XB <= EX_MEM_B; --EX HAZARD
                end if;

          END IF;

          IF ((DI_EX_code_op = SW) or (DI_EX_code_op = SB) or (DI_EX_code_op =
SH)) THEN
              mem_data <= Data_ual_s;
          END IF;

       -- Pas d'Aléas
       ELSE
            ALU_XB <= ALU_B; --NO HAZARD
       END IF;

  end procedure envoi;

  -- === Procedure Alea LW ========================================
  ------ >> Alea LW suivi par une instruction arithmetique ---------
  procedure aleaLW (  EX_MEM_Register_Rt : in std_logic_vector (REGS'range);
                      DI_EX_Register_Rs : in std_logic_vector (REGS'range);
                      DI_EX_Register_Rt : in std_logic_vector (REGS'range);
                      EX_MEM_Regs_W      : in std_logic;
                      EX_MEM_DC_RW      : in std_logic; -- PV 05-02-2013
                      EX_MEM_DC_AS      : in std_logic; -- PV 05-02-2013
                      signal halt       : out std_logic ) is

  begin
        -- Initialisation
        halt <= '0';
     --DETECT A LOAD/USE HAZARD e.g. LW $2, 20($1) followed by ADD $4, $2, $1
      IF ( (EX_MEM_Regs_W = '0') AND EX_MEM_DC_AS = '1' AND EX_MEM_DC_RW = '1'
AND
           (   (EX_MEM_Register_Rt = DI_EX_Register_Rs and DI_EX_Register_Rs
/= conv_std_logic_vector(0,REGS'length))
```

```
            OR (EX_MEM_Register_Rt = DI_EX_Register_Rt and DI_EX_Register_Rt
/= conv_std_logic_vector(0,REGS'length)) ) THEN
          halt <= '1'; --LOAD/USE HAZARD
      ELSE
          halt <= '0'; --NO HAZARD
      END IF;

  end procedure aleaLW;

end cpu_package;
```

```
-----------------------------------
-- Processeur RISC
-- THIEBOLT Francois le 09/12/04
-----------------------------------

------------------------------------------------------
-- Lors de la phase RESET, permet la lecture d'un fichier
-- instruction et un fichier donnees passe en parametre
--      generique.
------------------------------------------------------

-- Definition des librairies
library IEEE;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.cpu_package.all;
--use ieee.numeric_std.all;
--use WORK.all;

-- Definition de l'entite
entity risc is

        -- definition des parametres generiques
        generic (
                -- fichier d'initialisation cache instruction
                IFILE : string := "";

                -- fichier d'initialisation cache donnees
                DFILE : string := "" );

        -- definition des entrees/sorties
        port    (
                -- signaux de controle du processeur
                RST                     : in std_logic; -- actifs a l'etat
bas
                CLK                     : in std_logic );

end risc;

-- Definition de l'architecture du banc de registres
architecture behavior of risc is

        -- definition de constantes

        -- definitions de types/soustypes

        -- definition des ressources internes

        -- Registres du pipeline
        signal reg_EI_DI : EI_DI;               -- registre pipeline EI/DI
        signal reg_DI_EX : DI_EX;               -- registre pipeline DI/EX
        signal reg_EX_MEM       : EX_MEM;       -- registre pipeline EX/MEM
        signal reg_MEM_ER       : MEM_ER;       -- registre pipeline MEM/ER

        --1>>> Ressources de l'etage EI
        signal reg_PC                 : ADDR;                           --
compteur programme format octet
        signal ei_pc_next          : PC;                                --
pointeur sur prochaine instruction
```

```
        signal ei_inst               : INST;                          --
instruction en sortie du cache instruction
  -- signal ei_halt                  : std_logic;         -- suspension etage
pipeline
  --    signal ei_flush             : std_logic;  -- vidange de l'etage --PV:
Sans utiliser

        --2>>> Ressources de l'etage DI
        signal di_qa                          : DATA;
-- sortie QA du banc de registres
        signal di_qb                          : DATA;                      --
sortie QB du banc de registres
        signal di_imm_ext                 : DATA;
-- valeur immediate etendue
        signal di_ctrl_di                 : mxDI;
-- signaux de controle de l'etage DI
        signal di_ctrl_ex                 : mxEX;
-- signaux de controle de l'etage EX
        signal di_ctrl_mem        : mxMEM;                          -- signaux
de controle de l'etage MEM
        signal di_ctrl_er                 : mxER;
-- signaux de controle de l'etage ER
        signal di_flush             : std_logic; -- vidange de l'etage
  --  signal di_halt                 : std_logic;         -- suspension etage
pipeline

        --3>>> Ressources de l'etage EX
        signal ex_ALU_Z       : std_logic;
        signal ex_ALU_N       : std_logic;
        signal ex_ALU_V       : std_logic;
        signal ex_ALU_C       : std_logic;
  signal ex_ALU_S       : DATA;     -- sortie S de l'alu
        signal ex_ALU_A       : DATA;     -- sortie ALU_A de MUX_AUL_A
        signal ex_ALU_B       : DATA;     -- sortie ALU_B de MUX_AUL_B
        signal ex_ALU_XA      : DATA;     -- PV: ex_envoie_A === ex_ALU_XA
        signal ex_ALU_XB      : DATA;     -- PV: ex_envoie_B === ex_ALU_XB
        signal ex_mem_data    : DATA;

        signal ex_reg_dst                  : REGS;     -- PV: registres
destination

        signal ex_MUX_mem_data : MUX_mem_data;
  signal ex_MUX_ALU_XA  : MUX_ALU_XA;
        signal ex_MUX_ALU_XB  : MUX_ALU_XB;

  signal ex_pc_branch       : PC;                 -- adresse de branchement

        signal ex_flush             : std_logic;         -- vidange de
l'etage
  -- signal ex_halt                 : std_logic;         -- suspension etage
pipeline

        --4>>> Ressources de l'etage MEM
        signal MEM_mem_Q    : DATA;   -- PV: 08-02-2013
  signal mem_branch   : std_logic ;   -- PV: mem_branch_bol === mem_branch
  signal mem_b_type   : std_logic;      -- PV: mem_branch_type_out ===
mem_b_type

  signal mem_flush               : std_logic;

  signal mem_halt                   : std_logic;         -- suspension etage
pipeline
```

```
        --5>>> Ressources de l'etage ER
        signal er_regd              : DATA;                     -- donnees a
ecrire dans le banc de registre
        signal er_adrw              : REGS;                     -- adresse du
registre a ecrire dans le banc


begin


-- ============================================================
-- === Etage EI ===============================================
-- ============================================================

------------------------------------------------------------------
-- instanciation et mapping du composant cache instruction
icache : entity work.memory(behavior)
                        generic map ( DBUS_WIDTH=>CPU_DATA_WIDTH,
ABUS_WIDTH=>CPU_ADR_WIDTH, MEM_SIZE=>L1_ISIZE,

ACTIVE_FRONT=>L1_FRONT, FILENAME=>IFILE )
                        port map ( RST=>RST, CLK=>CLK, RW=>'1', DS=>MEM_32,
Signed=>'0', AS=>'1', Ready=>open,
                                                        Berr=>open,
ADR=>reg_PC, D=>(others => '0'), Q=>ei_inst );


------------------------------------------------------------------
-- Affectations dans le domaine combinatoire de l'etage EI
-->>> Incrementation du PC (format mot)
ei_pc_next <=  ex_pc_branch when mem_branch = '1' else         -- mem_branch:
Branch & Branch type & (Z, N,C,V)
               reg_DI_EX.rs_read (PC'length-1 downto 0)  when
reg_DI_EX.ex_ctrl.J_jr = '1' or reg_DI_EX.ex_ctrl.J_jalr = '1' else     --
J_jr comme from reg_DI_EX.ex_ctrl.J_jr
               reg_EI_DI.inst(JADR'range)    when di_ctrl_di.J_j = '1' or
di_ctrl_di.J_jal = '1'  else  -- reg_EI_DI.di_ctrl.J_j <-> di_ctrl_di.J_j
                                reg_PC(PC'range) +'1';

--ei_halt <= '0' ;


-->>> Appel de la procedure aleaLW
UA: aleaLW (  reg_EX_MEM.reg_dst, --reg_DI_EX.rd,
              reg_DI_EX.rs,      -- reg_EI_DI.inst(RS'range)
                    reg_DI_EX.rt,      -- reg_EI_DI.inst(RT'range)
                reg_EX_MEM.er_ctrl.Regs_W, -- er_ctrl.regs_W: active '0'
                reg_EX_MEM.mem_ctrl.DC_RW,
                reg_EX_MEM.mem_ctrl.DC_AS,
                mem_halt );

------------------------------------------------------------------

-- Process Etage Extraction de l'instruction et mise a jour de
--      l'etage EI/DI et du PC
EI: process(CLK)

begin
```

```
  -- test du front actif d'horloge
  if (rising_edge (CLK)) then
     -- test du reset
           if (RST='0') then
                    -- reset du PC
                    reg_PC <= PC_DEFL;
           else
               --if (ei_halt = '0') then

                 if ( di_flush='1' or ex_flush ='1' or mem_flush ='1' ) then
                                          reg_EI_DI.flush   <= '1';
                 else
                                          reg_EI_DI.flush   <= '0';
                 end if ;
                           reg_PC(PC'range)<= ei_pc_next;

                        -- Mise a jour du registre inter-etage EI/DI
             reg_EI_DI.code_op  <= ei_inst(OPCODE'range) ;
             reg_EI_DI.code_func  <= ei_inst(FCODE'range) ;

                          reg_EI_DI.rs                        <=
ei_inst(RS'range);
                  reg_EI_DI.rt                      <= ei_inst(RT'range);
                  reg_EI_DI.rd                      <= ei_inst(RD'range);


                          reg_EI_DI.pc_next <= ei_pc_next;
                          reg_EI_DI.inst    <= ei_inst;

       --end if; --ei_halt
    end if;
  end if;
end process EI;

-- ============================================================
-- === Etage DI ===============================================
-- ============================================================

-----------------------------------------------------------------
-- instantiation et mapping du composant registres
regf : entity work.registres(behavior)
                        generic map ( DBUS_WIDTH=>CPU_DATA_WIDTH,
ABUS_WIDTH=>REG_WIDTH, ACTIVE_FRONT=>REG_FRONT )
                        port map ( CLK=>CLK, W=>reg_MEM_ER.er_ctrl.regs_W,
RST=>RST, D=>er_regd,

ADR_A=>reg_EI_DI.inst(RS'range), ADR_B=>reg_EI_DI.inst(RT'range),
                                                        ADR_W=>er_adrw,
QA=>di_qa, QB=>di_qb );


-----------------------------------------------------------------
-- Affectations dans le domaine combinatoire de l'etage DI

-- Calcul de l'extension de la valeur immediate
di_imm_ext(IMM'range) <= reg_EI_DI.inst(IMM'range);
di_imm_ext(DATA'high downto IMM'high+1) <= (others => '0') when
di_ctrl_di.signed_ext='0' else
                                                          (others =>
reg_EI_DI.inst(IMM'high));
```

```
---- Appel de la procedure contol
UC: control(    reg_EI_DI.flush,
                reg_EI_DI.inst(OPCODE'range),
                                        reg_EI_DI.inst(FCODE'range),
                                        reg_EI_DI.inst(BCODE'range),
                                        di_ctrl_di,
                                        di_ctrl_ex,
                                        di_ctrl_mem,
                                        di_ctrl_er );

di_flush <= '1' when di_ctrl_di.J_j  = '1' or di_ctrl_di.J_jal  = '1' else
            '0';
-----------------------------------------------------------------

-- Process Etage Extraction de l'instruction et mise a jour de
--      l'etage DI/EX
DI: process(CLK)
begin

-- test du front actif d'horloge
if (rising_edge (CLK)) then
         -- test du reset et signal flush
         if (RST='0' or ex_flush ='1' or mem_flush ='1') then
                  -- reset des controle du pipeline
                reg_DI_EX.ex_ctrl       <= EX_DEFL;
                reg_DI_EX.mem_ctrl <= MEM_DEFL;
                reg_DI_EX.er_ctrl       <= ER_DEFL;

         else
                  -- Mise a jour du registre inter-etage DI/EX
                reg_DI_EX.pc_next               <= reg_EI_DI.pc_next;
                reg_DI_EX.rs                        <=
reg_EI_DI.inst(RS'range);
                reg_DI_EX.rt                        <=
reg_EI_DI.inst(RT'range);
                reg_DI_EX.rd                        <=
reg_EI_DI.inst(RD'range);
                reg_DI_EX.val_dec               <=
reg_EI_DI.inst(VALDEC'range);
                reg_DI_EX.imm_ext               <= di_imm_ext;
                reg_DI_EX.jump_adr      <= reg_EI_DI.inst(JADR'range);
                reg_DI_EX.rs_read               <= di_qa;
                reg_DI_EX.rt_read               <= di_qb;
                  reg_DI_EX.code_op  <= reg_EI_DI.inst(OPCODE'range) ;

                -- Mise a jour des signaux de controle
                reg_DI_EX.ex_ctrl               <= di_ctrl_ex;
                reg_DI_EX.mem_ctrl      <= di_ctrl_mem;
                reg_DI_EX.er_ctrl               <= di_ctrl_er;

         end if;
end if ;

end process DI;


-- ============================================================
-- === Etage EX ===============================================
-- ============================================================
-- Process
--l'etage  EX/MEM
```

```
-- affectation combinatoire dans l'etage exécution

ex_reg_dst <= reg_DI_EX.rd when reg_DI_EX.ex_ctrl.reg_dst=REG_RD else
                                    reg_DI_EX.rt when
reg_DI_EX.ex_ctrl.reg_dst=REG_RT else
                                    (others => '1'); -- R31

-- Appel de l'unité d'envoie
UE: envoi (  reg_DI_EX.code_op,
             reg_EX_MEM.code_op,

             reg_DI_EX.rs,      -- Registe Lecture 1 RS
                   reg_DI_EX.rt,      -- Registre Lecture 2 RT
                   reg_EX_MEM.reg_dst,
                   reg_MEM_ER.reg_dst,
                   reg_EX_MEM.er_ctrl.regs_W,
                   reg_MEM_ER.er_ctrl.regs_W,

                   mem_halt,
                   ex_MUX_mem_data,
                   ex_MUX_ALU_XA,
                   ex_MUX_ALU_XB);


ex_ALU_A  <= reg_DI_EX.rs_read when reg_DI_EX.ex_ctrl.ALU_SRCA = REGS_QA  else
             reg_DI_EX.rt_read when reg_DI_EX.ex_ctrl.ALU_SRCA = REGS_QB  else
                               reg_DI_EX.imm_ext ;

ex_ALU_XA <= ex_ALU_A          when ex_MUX_ALU_XA = ALU_A                 else
             reg_EX_MEM.ual_s  when ex_MUX_ALU_XA = EX_MEM_A              else
             MEM_mem_Q when ex_MUX_ALU_XA = MEM_Q              else
             er_regd ;

ex_ALU_B  <= reg_DI_EX.rt_read                        when
reg_DI_EX.ex_ctrl.ALU_SRCB = REGS_QB  else
             reg_DI_EX.imm_ext                                when
reg_DI_EX.ex_ctrl.ALU_SRCB = IMMD     else
             X"000000" & "000" & reg_DI_EX.val_dec  when
reg_DI_EX.ex_ctrl.ALU_SRCB = VAL_DEC  else
             X"00000010"                              when
reg_DI_EX.ex_ctrl.ALU_SRCB = VAL_16   else
                                   (others => '0');

ex_ALU_XB <= ex_ALU_B          when ex_MUX_ALU_XB =  ALU_B                 else
             reg_EX_MEM.ual_s  when ex_MUX_ALU_XB =  EX_MEM_B              else
             MEM_mem_Q when ex_MUX_ALU_XB = MEM_Q              else
             er_regd ;

ex_mem_data  <= reg_EX_MEM.ual_s  when ex_MUX_mem_data =  Data_ual_s     else
              reg_DI_EX.rt_read; -- Need for store instruction

--P_ALU:alu
P_ALU:  alu(  ex_ALU_XA, ex_ALU_XB, ex_ALU_S, ex_ALU_N, ex_ALU_V, ex_ALU_Z,
ex_ALU_C, reg_DI_EX.ex_ctrl.ALU_SIGNED, reg_DI_EX.ex_ctrl.ALU_OP);


-- calcul d'adresse de branch
   ex_pc_branch <= reg_EX_MEM.imm_ext (PC_WIDTH -1 downto 0) +
reg_EX_MEM.pc_next ;
```

```
-- Flush
ex_flush <= '1' when reg_DI_EX.ex_ctrl.J_jr = '1' or reg_DI_EX.ex_ctrl.J_jalr
= '1' else
            '0';


EX: process(CLK)
begin

        -- test du front actif d'horloge
        if (rising_edge (CLK)) then
                -- test du reset
    if (RST='0' or mem_flush ='1') then
                    -- reset des controle du pipeline
                    reg_EX_MEM.mem_ctrl <= MEM_DEFL;
                    reg_EX_MEM.er_ctrl  <= ER_DEFL ;
          else
                -- Mise a jour du registre inter-etage DI/EX
                reg_EX_MEM.pc_next                  <= reg_DI_EX.pc_next;
                reg_EX_MEM.ual_S        <= ex_ALU_S ;
                reg_EX_MEM.ual_N        <= ex_ALU_N ;
                reg_EX_MEM.ual_V        <= ex_ALU_V ;
                reg_EX_MEM.ual_Z        <= ex_ALU_Z ;
                reg_EX_MEM.ual_C        <= ex_ALU_C ;
                reg_EX_MEM.reg_dst      <= ex_reg_dst;
                reg_EX_MEM.rt                       <= reg_DI_EX.rt;
                reg_EX_MEM.rs                       <= reg_DI_EX.rs;

    -- reg_EX_MEM.rt_read      <= reg_DI_EX.rt_read; -- Need for store
instruction
    reg_EX_MEM.rt_read      <= ex_mem_data;
    reg_EX_MEM.imm_ext      <= reg_DI_EX.imm_ext;
                reg_EX_MEM.code_op  <= reg_DI_EX.code_op ;

                -- Mise a jour des signaux de controle
                reg_EX_MEM.mem_ctrl         <= reg_DI_EX.mem_ctrl ;
                reg_EX_MEM.er_ctrl      <= reg_DI_EX.er_ctrl ;
          end if;
  end if;

end process EX ;

-- ============================================================
-- === Etage MEM ==============================================
-- ============================================================

--instanciation et mapping du composant cache donnée
dcache : entity work.memory(behavior)

                        generic map ( DBUS_WIDTH=>CPU_DATA_WIDTH,
ABUS_WIDTH=>CPU_ADR_WIDTH, MEM_SIZE=>L1_ISIZE,

ACTIVE_FRONT=>L1_FRONT, FILENAME=>DFILE )
                        port map    ( RST=>RST, CLK=>CLK,
RW=>reg_EX_MEM.mem_ctrl.DC_RW , DS=>reg_EX_MEM.mem_ctrl.DC_DS,
                                      Signed=>reg_EX_MEM.mem_ctrl.DC_SIGNED,
AS=> reg_EX_MEM.mem_ctrl.DC_AS, Ready=>open,

Berr=>open, ADR=>reg_EX_MEM.ual_s, D=>reg_EX_MEM.rt_read , Q=> MEM_mem_Q );

-- Branchement
```

```
mem_b_type  <=  reg_EX_MEM.ual_Z
                                                          when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BEQ   else
                not(reg_EX_MEM.ual_Z)

                                                          when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BNE   else
                     (reg_EX_MEM.ual_Z) or ((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V))
                                                              when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BLEZ  else
                     (not(reg_EX_MEM.ual_Z)) and not((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V))
                                                              when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BGTZ  else
                     (not(reg_EX_MEM.ual_Z)) and ((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V))
                                                              when
reg_EX_MEM.mem_ctrl.B_TYPE = B_BLTZ  else
                     ((reg_EX_MEM.ual_Z)) or not((reg_EX_MEM.ual_N) xor
(reg_EX_MEM.ual_V)) ;
                                                        --when
reg_EX_MEM.mem_ctrl.BRAN_TYPE = B_BGEZ

mem_branch    <= (reg_EX_MEM.mem_ctrl.BRANCH and mem_b_type) ;

-- Flush
mem_flush <= '1' when mem_branch= '1' else
             '0';

MEM: process(CLK)
begin

    if (rising_edge (CLK)) then
        if (RST='0') then
                      reg_MEM_ER.er_ctrl         <= ER_DEFL ;
           else
                    -- Mise a jour du registre inter-etage DI/EX
                    reg_MEM_ER.mem_Q <= MEM_mem_Q;
                    reg_MEM_ER.pc_next           <= reg_EX_MEM.pc_next;
                    reg_MEM_ER.reg_dst   <= reg_EX_MEM.reg_dst;
                    reg_MEM_ER.ual_S     <= reg_EX_MEM.ual_S;

                    -- reg_MEM_ER.code_op  <= reg_EX_MEM.code_op ;

                    -- Mise a jour des signaux de controle
                    reg_MEM_ER.er_ctrl   <= reg_EX_MEM.er_ctrl;
           end if;
    end if ;
end process MEM ;

-- ============================================================
-- === Etage ER ===============================================
-- ============================================================

er_adrw <= reg_MEM_ER.reg_dst    ;

er_regd <= reg_MEM_ER.ual_S  when reg_MEM_ER.er_ctrl.REGS_SRCD = ALU_S  else
           reg_MEM_ER.mem_Q  when reg_MEM_ER.er_ctrl.REGS_SRCD = MEM_Q   else
                       reg_MEM_ER.pc_next;
```

```
end behavior;
```

```
--------------------------------------------------------------------
-- Fichier de test du processeur RISC
-- THIEBOLT Francois le 16/12/02
-- PHOR Vicheka
--------------------------------------------------------------------

-- Definition des librairies
library IEEE;
library WORK;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.cpu_package.all;

-- Definition de l'entite
entity test_risc is
end test_risc;

-- Definition de l'architecture
architecture behavior of test_risc is

-- definition des constantes de test
        constant FILE_I : string        := "logique.i.0.txt";
        constant FILE_D : string        := "logique.d.0.txt";
        constant WFRONT : std_logic     := CPU_WR_FRONT;
        constant TIMEOUT        : time          := 300 ns; -- timeout de la
simulation


-- definition de constantes
constant clkpulse : Time := 5 ns; -- 1/2 periode horloge

-- definition de types

-- definition de ressources internes

-- definition de ressources externes
signal E_CLK                                                    : std_logic;
signal E_RST                                                    : std_logic;
-- actifs a l'etat bas


begin

--------------------------------------------------------------------
-- definition de l'horloge
P_E_CLK: process
begin
        E_CLK <= '1';
        wait for clkpulse;
        E_CLK <= '0';
        wait for clkpulse;
end process P_E_CLK;

--------------------------------------------------------------------
-- definition du timeout de la simulation
P_TIMEOUT: process
begin
        wait for TIMEOUT;
```

```
        assert FALSE report "SIMULATION TIMEOUT!!!" severity FAILURE;
end process P_TIMEOUT;

-------------------------------------------------------------------
-- instantiation et mapping du composant processeur
r3k : entity work.risc(behavior)
                        generic map ( IFILE=>FILE_I, DFILE=>FILE_D )
                        port map ( CLK=>E_CLK, RST=>E_RST);

-------------------------------------------------------------------
-- debut sequence de test
P_TEST: process
begin

        -- initialisations
        E_RST <= '0';

        -- sequence RESET
        E_RST <= '0';
        wait for clkpulse*3;
        E_RST <= '1';
        wait for clkpulse;

        -- ADD NEW SEQUENCE HERE

        -- LATEST COMMAND (NE PAS ENLEVER !!!)
        wait until (E_CLK=(WFRONT)); wait for clkpulse/2;
        --assert FALSE report "FIN DE SIMULATION" severity FAILURE;
        wait; -- le processeur ne s'arrete pas, on attend le timeout

end process P_TEST;

end behavior;
```

```
-----------------------------------
-- Banc Memoire pour processeur RISC
-- THIEBOLT Francois le 01/12/05
-- PHOR Vicheka
-----------------------------------

------------------------------------------------------
-- Lors de la phase RESET, permet la lecture d'un fichier
-- passe en parametre generique.
------------------------------------------------------

-------------------------------------------------------------------
-- Ne s'agissant pas encore d'un cache, le signal Ready est cable
-- a 1 puisque toute operation s'execute en un seul cycle.
--      Ceci est la version avec lecture ASYNCHRONE pour une
--      integration plus simple dans le pipeline.
-- Si la lecture du fichier d'initialisation ne couvre pas tous
--      les mots memoire, ceux-ci seront initialises a 0
-------------------------------------------------------------------

-- Definition des librairies
library IEEE;
library STD;
library WORK;

-- Definition des portee d'utilisation
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use WORK.cpu_package.all;
use WORK.cpu_package2_1.all;

-- Definition de l'entite
entity memory is

        -- definition des parametres generiques
        generic (
                -- largeur du bus de donnees par defaut
                DBUS_WIDTH : natural := 32;

                -- largeur du bus adr par defaut
                ABUS_WIDTH : natural := 32;

                -- nombre d'elements dans le cache exprime en nombre de mots
                MEM_SIZE : natural := 16;

                -- front actif par defaut
                ACTIVE_FRONT : std_logic := '1';

                -- fichier d'initialisation
                FILENAME : string := "" );

        -- definition des entrees/sorties
        port    (
                -- signaux de controle du cache
                RST                     : in std_logic; -- actifs a l'etat
bas
                CLK,RW          : in std_logic; -- R/W*
                DS                              : in MEM_DS;            --
acces octet, demi-mot, mot...
```

```
                Signed          : in std_logic;  -- extension de signe
                AS                              : in std_logic;  -- Address
Strobe (sorte de CS*)
                Ready                   : out std_logic;-- indicateur
HIT/MISS
                Berr                    : out std_logic;-- bus error (acces
non aligne par exemple), active low


                -- bus d'adresse du cache
                ADR                     : in std_logic_vector(ABUS_WIDTH-1
downto 0);


                -- Ports entree/sortie du cache
                D                               : in
std_logic_vector(DBUS_WIDTH-1 downto 0);
                Q                               : out
std_logic_vector(DBUS_WIDTH-1 downto 0) );


end memory;

-- Definition de l'architecture du banc de registres
architecture behavior of memory is

        -- definition de constantes
        constant BITS_FOR_BYTES : natural := log2(DBUS_WIDTH/8) ; -- nb bits
adr pour acceder aux octets d'un mot
        constant BITS_FOR_WORDS : natural := log2( MEM_SIZE) ; -- nb bits adr
pour acceder aux mots du cache
        constant BYTES_PER_WORD : natural := DBUS_WIDTH/8; -- nombre d'octets
par mot


        -- definitions de types (index type default is integer)
        subtype BYTE is std_logic_vector(7 downto 0); -- definition d'un octet
        type WORD is array (BYTES_PER_WORD-1 downto 0) of BYTE; -- definition
d'un mot composï¿½ d'octets


        type FILE_REGS is array (MEM_SIZE-1 downto 0) of WORD;
        subtype I_ADR is std_logic_vector((BITS_FOR_WORDS + BITS_FOR_BYTES)-1
downto BITS_FOR_BYTES); -- internal ADR au format mot du cache


        subtype B_ADR is std_logic_vector( BITS_FOR_BYTES-1 downto 0); -- byte
ADR pour manipuler les octets dans le mot
        subtype byte_adr is natural range 0 to 7; -- manipulation d'octets
dans les mots


        -- definition de la fonction de chargement d'un fichier
        --              on peut egalement mettre cette boucle dans le process
qui fait les ecritures
        --function LOAD_FILE (F : in string) return FILE_REGS is
        --      variable temp_REGS : FILE_REGS;
        --      file mon_fichier : TEXT open READ_MODE is STRING'(F); --
VHDL93 compliant
```

```
                --      file mon_fichier : TEXT is in STRING'(F); -- older
implementation
        --      variable line_read : line := null;
        --      variable line_value : std_logic_vector (DBUS_WIDTH-1 downto
0);
        --      variable index,i : natural := 0;
--      begin
                -- lecture du fichier
        --      index:=0;
        --      while (not ENDFILE(mon_fichier) and (index < MEM_SIZE))
        --      loop
        --              readline(mon_fichier,line_read);
        --              read(line_read,line_value);
        --              for i in 0 to BYTES_PER_WORD-1 loop
        --                      temp_REGS(index)(i):=line_value(((i+1)*8)-1
downto i*8);
        --              end loop;
--                      temp_REGS(index):=line_value;
        --              index:=index+1;
        --      end loop;
                -- test si index a bien parcouru toute la memoire
        --      if (index < MEM_SIZE) then
        --              temp_REGS( MEM_SIZE-1 downto index ):=(others =>
(others => (others => '0')));
        --      end if;
                -- renvoi du resultat
        --      return temp_REGS;
        --end LOAD_FILE;

        -- definition des ressources internes
        signal REGS : FILE_REGS; -- le banc memoire

        -- l'adressage de la memoire se faisant par element de taile
DBUS_WIDTH, par rapport
        -- au bus d'adresse au format octet il faut enlever les bits d'adresse
de poids faible
        -- (octets dans le mot), puis prendre les bits utiles servant a
l'acces des mots du cache.
        -- ex.: mots de 32 bits => 2 bits de poids faible our les octets dans
le mot
        --              16 mots memoire => 4 bits necessaire
        -- D'ou I_ADR = ADR (4+2-1 downto 2)

begin
-----------------------------------------------------------------------
-- Affectations dans le domaine combinatoire
--
--Q <=  REGS (conv_integer (ADR (I_ADR'range)))  when (AS ='1' and RW='1')
else
  --   (others => 'Z') ;
-- Indicateur acces MISS/HIT
Ready <= '1'; -- car pas encore un cache

-----------------------------------------------------------------------
-- Process P_CACHE
--      La lecture etant asynchrone c.a.d qu'elle ne depend que des
--              signaux d'entree, nous sommes obliges de les mettre dans la
--              liste de sensitivite du process
P_CACHE: process(CLK,RST,ADR,AS,RW,DS,Signed)
```

```
variable i: natural := 0 ;
variable temp_berr : std_logic ;
Variable  temp_signed : std_logic ;
begin

temp_signed := Signed    ;

  if RST='0' then
               --if (STRING'(FILENAME) /= "") then
               --      REGS <= LOAD_FILE(FILENAME);
               --else
           REGS <= (others => (others => (others => '0')));
               --end if;
               Q <= (others => 'Z') ;
  elsif (AS ='1') then
     temp_berr := '1';
                Berr <= '1';

               -- if rising_edge(CLK) then

                case DS is

                  when MEM_8  =>

                        -- lecture
          if (RW = '1' and temp_berr = '1'  ) then
               if (temp_signed = '0') then
                         Q <= (others => '0') ;
                        else
                         Q <= (others  => '1');
                        end if ;
                  for i in 0 to 1 loop
                    Q (((i+1)*8)-1 downto i*8)<= REGS(conv_integer
(ADR(I_ADR'range))) (conv_integer(ADR(B_ADR'range))+i);
                  end loop ;
          end if ;

               -- ecriture
           if (RW = '0' and temp_berr = '1' ) then
                 for i in 0 to 1 loop
                   REGS(conv_integer (ADR(I_ADR'range)))
(conv_integer(ADR(B_ADR'range)) + i ) <= D(((i+1)*8)-1 downto i*8) ;
                  end loop ;
          end if ;


                  when MEM_16  =>
                                      -- lecture
          if (RW = '1' and temp_berr = '1'  ) then
               if (temp_signed = '0') then
                         Q <= (others => '0') ;
                        else
                         Q <= (others  => '1');
                        end if ;
                  for i in 0 to  2 loop
                    Q (((i+1)*8)-1 downto i*8)<= REGS(conv_integer
(ADR(I_ADR'range))) (conv_integer(ADR(B_ADR'range))+i);
                  end loop ;
          end if ;

               -- ecriture
```

```
             if (RW = '0' and temp_berr = '1' ) then
                   for i in 0 to  2 loop
                     REGS(conv_integer (ADR(I_ADR'range)))
(conv_integer(ADR(B_ADR'range)) + i ) <= D(((i+1)*8)-1 downto i*8) ;
                    end loop ;
           end if ;
                     if ADR (0) /= '0' then
                        temp_berr := '0';
                     end if ;

                    when MEM_32 =>
                                              -- lecture
           if (RW = '1' and temp_berr = '1'  ) then
                if (temp_signed = '0') then
                          Q <= (others => '0') ;
                         else
                           Q <= (others  => '1');
                        end if ;
                   for i in 0 to  3 loop
                     Q (((i+1)*8)-1 downto i*8)<= REGS(conv_integer
(ADR(I_ADR'range))) (conv_integer(ADR(B_ADR'range))+i);
                    end loop ;
           end if ;

                -- ecriture
            if (RW = '0' and temp_berr = '1') then
                   for i in 0 to  3 loop
                     REGS(conv_integer (ADR(I_ADR'range)))
(conv_integer(ADR(B_ADR'range)) + i ) <= D(((i+1)*8)-1 downto i*8) ;
                    end loop ;
           end if ;
                      if ADR (1 downto 0) /= conv_std_logic_vector (0,2) then
                        temp_berr := '0';
                      end if ;
                    --  end if;
                         when others =>

         end case ;

                Berr <= temp_berr ;
         --end if;

   else
        Q <= (others=>'Z');
 end if ;



end process P_CACHE;

end behavior;
```

ANNEXE A4

```c
/* ============================================================================
    r3kasm2.h
    Fichier de definition de l'assembleur R3000
    ============================================================================ */
#ifndef _R3KASM2_H
#define _R3KASM2_H

// Definitions generales
#define MAX_LABELS   50
#define SIZE_LABELS  100
#define MAX_MACRO        50
#define SIZE_MACRO   100

#define OPCODE           32      // taille en bit d'une instruction
#define MAX_REGS     32
#define MAX_VALDEC   32
#define TYPE_R           0
#define TYPE_I                   1
#define TYPE_J                   2

//Structures des labels
typedef struct {
    char name[MAX_LABELS] [SIZE_LABELS];
    int cp[MAX_LABELS];
    int nb;
} TLabels;

//Strutures des macros
typedef struct {
    char name[SIZE_MACRO] [MAX_MACRO];
    char replace[SIZE_MACRO] [MAX_MACRO];
    int nb;
} TMacros;

// Declaration des fonctions
void Lread(FILE *s, char *dest);

void Nettoyage(char *dest);

void intTOsbin(int n, char *result, int length);

void strUP(char *s);

void addLabel(const char *name, int CP, TLabels *l);

int cpLabel(const char *name, TLabels *l);

void opMake(FILE *f, int type, const char *op, int rs, int rt, int rd, int valdec, const char *fct, int imm, int adr);

void replaceMacro(char *s,TMacros *m);

void prePARSE(FILE *source, FILE *dest, TLabels *l, TMacros *m);

void addMacro(TMacros *m, char *name, char *replace);

char *searchMacro(TMacros *m, const char *name);

// Definition OPCODES
#define OP_LSL       "000000"
#define OP_LSR       "000000"
#define OP_JR        "000000"
#define OP_ADD       "000000"
#define OP_ADDU  "000000"
#define OP_SUB       "000000"
#define OP_SUBU  "000000"
#define OP_AND       "000000"
#define OP_OR        "000000"
#define OP_XOR       "000000"
#define OP_NOR       "000000"
#define OP_SLT       "000000"
#define OP_SLTU  "000000"
#define OP_JALR  "000000"
#define OP_BLTZ  "000001"
#define OP_BGEZ  "000001"
#define OP_BLTZAL    "000001"
#define OP_BGEZAL    "000001"
#define OP_J         "000010"
#define OP_JAL       "000011"
#define OP_BEQ       "000100"
#define OP_BNE       "000101"
#define OP_BLEZ  "000110"
#define OP_BGTZ  "000111"
#define OP_ADDI  "001000"
#define OP_ADDIU "001001"
#define OP_SLTI  "001010"
#define OP_SLTIU "001011"
#define OP_ANDI  "001100"
#define OP_ORI       "001101"
#define OP_XORI  "001110"
#define OP_LUI       "001111"
#define OP_LB        "100000"
#define OP_LH        "100001"
#define OP_LW        "100011"
#define OP_LBU       "100100"
#define OP_LHU       "100101"
#define OP_SB        "101000"
#define OP_SH        "101001"
#define OP_SW        "101011"

// Definition FONCTIONS
```

```
#define F_LSL        "000000"
#define F_LSR        "000010"
#define F_JR         "001000"
#define F_ADD        "100000"
#define F_ADDU       "100001"
#define F_SUB        "100010"
#define F_SUBU       "100011"
#define F_AND        "100100"
#define F_OR         "100101"
#define F_XOR        "100110"
#define F_NOR        "100111"
#define F_SLT        "101010"
#define F_SLTU       "101011"
#define F_JALR       "001001"

// definition des BCOND
#define B_BLTZ       "00000"
#define B_BGEZ       "00001"
#define B_BLTZAL "10000"
#define B_BGEZAL "10001"

#endif

```

```c
/* ================================================================================
    r3kasm2.c
    Assembleur R3000
    ================================================================================ */

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "r3kasm2.h"

/* R3Kasm by FOX *****************************************************/


// One line read from file
void Lread(FILE *s, char *dest)
{
    int i=0;

    do
        dest[i++]= (char) fgetc(s);
    while( (!feof(s)) && (dest[i-1] != '\n') );
    dest[i]='\0';
    if (feof(s))
        dest[i-1]='\0';
}

// Nettoyage de la chaine de caractere lue
void Nettoyage(char *dest)
{
    char *p1,*p2;
    p1=dest;
    p2=p1;

    while ( *p2 != '\0' )
    {
        if ((*p2!=' ') && (*p2!='\n') && (*p2!='\t'))
            *p1++=*p2;
        p2++;
    }
    *p1='\0';
}

// Conversion d'un entier en un binaire de type char*
void intTOsbin(int n, char *result, int length)
{
    int tmp,i;
    strcpy(result,"");

    for(i=length-1;i>=0;i--)
    {
        tmp= n & (1<<i);
        if (tmp==0)
            strcat(result, "0");
        else
            strcat(result, "1");
    }
    //  printf("Conversion n %d sur %dbits = %s\n",n,length,result);
}

// Mise en MAJUSCULE
void strUP(char *s)
{
    int i;

    for(i=0;i<strlen(s);i++)
        s[i]=toupper(s[i]);
}

// Ajoute un label dans la liste
void addLabel(const char *name, int CP, TLabels *l)
{
    strcpy( (*l).name[(*l).nb], name );
    (*l).cp[(*l).nb]=CP;
    (*l).nb++;
}

// Retourne le CP du label correspondant
int cpLabel(const char *name, TLabels *l)
{
    int i;

    for(i=0;i<(*l).nb;i++)
    {
        if ( strcmp(name,(*l).name[i])==0 )
        return (*l).cp[i];
    }

    // si on est ici c'est que le label est inconnu !
    printf ("\tERROR label %s unknown !\n",name);
    exit (1);
}

// Ecriture de l'OPcode assemble
void opMake(FILE *f, int type, const char *op, int rs, int rt, int rd, int valdec,
                    const char *fct, int imm, int adr)
{
    static char tmp[200];
    char retop[OPCODE+1];
    strcpy(retop,"");

```

```
101         // Tests de conformite
102         if ( (rs<0) || (rs>=MAX_REGS) )
103         {
104             printf ("\tERROR registre RS invalide: R%d\n",rs);
105             exit(1);
106         }
107 
108         if ( (rt<0) || (rt>=MAX_REGS) )
109         {
110             printf ("\tERROR registre RT invalide: R%d\n",rt);
111             exit(1);
112         }
113 
114         if ( (rd<0) || (rd>=MAX_REGS) )
115         {
116             printf ("\tERROR registre RD invalide: R%d\n",rd);
117             exit(1);
118         }
119 
120         if ( (valdec<0) || (valdec>=MAX_VALDEC) )
121         {
122             printf ("\tERROR valdec incorrect: %d\n",valdec);
123             exit(1);
124         }
125 
126         switch(type)
127         {
128             case TYPE_R:
129             {
130                 strcat(retop,op); //code op
131                 intTOsbin(rs,tmp,5);
132                 strcat(retop,tmp); //rs
133                 intTOsbin(rt,tmp,5);
134                 strcat(retop,tmp); //rt
135                 intTOsbin(rd,tmp,5);
136                 strcat(retop,tmp); //rd
137                 intTOsbin(valdec,tmp,5); //valdec
138                 strcat(retop,tmp);
139                 strcat(retop,fct);
140             }
141             break;
142 
143             case TYPE_I:
144             {
145                 strcat(retop,op); //code op
146                 intTOsbin(rs,tmp,5);
147                 strcat(retop,tmp); //rs
148                 intTOsbin(rt,tmp,5);
149                 strcat(retop,tmp); //rt
150                 intTOsbin(imm,tmp,16);
151                 strcat(retop,tmp); //imm
152             }
153             break;
154 
155             case TYPE_J:
156             {
157                 strcat(retop,op); //code op
158                 intTOsbin(adr,tmp,26);
159                 strcat(retop,tmp); //adr
160             }
161             break;
162         }
163         fprintf(f,"%s\n",retop);
164     }
165 
166     // Remplacement des macros
167     void replaceMacro(char *s,TMacros *m)
168     {
169         char *ptr1,*ptr2,*tmp1;
170         char lgnLu[100],tmp2[100],name[100];
171 
172         strcpy(lgnLu,s);
173         tmp1=lgnLu;
174         strUP(lgnLu);
175         strcpy(tmp2,"");
176         ptr1=strchr(tmp1,'$');
177 
178         do
179         {
180             // printf("tmp1 %s **** tmp2 %s \n",tmp1,tmp2);
181             strncat(tmp2,tmp1,ptr1-tmp1);
182             if (*(ptr1-1)=='(')
183                 ptr2=strchr(ptr1,')');
184             else
185                 ptr2=strchr(ptr1,',');
186 
187             if (ptr2!=NULL)
188             {
189                 strncpy(name,ptr1,ptr2-ptr1);
190                 name[ptr2-ptr1]='\0';
191                 tmp1=ptr2;
192             }
193             else
194             {
195                 strcpy(name,ptr1);
196                 name[strlen(name)-1] = '\0';
197                 tmp1=ptr1+strlen(name);
198             }
199 
200             Nettoyage(name);
```

```
201             strcat(tmp2,searchMacro(m,name));
202
203             ptr1=strchr(tmp1,'$'); //repere le $ suivant
204         } while (ptr1!=NULL);
205
206         strcat(tmp2,tmp1);
207         strcpy(s,tmp2);
208     }
209
210     // Extraction des labels etiquettes macros...
211     void prePARSE(FILE *source, FILE *dest, TLabels *l, TMacros *m)
212     {
213         char tmp[255],tmp2[255],tmp3[255];
214         char *ptr;
215         int CP=0;
216         m->nb=0;
217         printf("Preparsing...\n");
218
219         //On repere les macros ainsi que les etiquettes
220         rewind(source);
221         l->nb=0;
222
223         while ( !feof(source) )
224         {
225             //fscanf(source,"%s",tmp);
226             Lread(source,tmp);
227             strUP(tmp); // majuscules
228             // Nettoyage(tmp);
229
230             // Labels
231             if (tmp[0]==':')
232             {
233                 Nettoyage(tmp);
234                 addLabel(tmp,CP,l);
235                 printf("\tLabel %s -> CP %d\n",l->name[l->nb-1],l->cp[l->nb-1]);
236                 continue;
237             }
238
239             // Macros
240             if (tmp[0]=='$')
241             {
242                 sscanf(tmp,"%s %s",tmp2,tmp3);
243                 Nettoyage(tmp2);
244                 addMacro(m,tmp2,tmp3);
245                 printf("\tMacro %s -> %s\n",m->name[m->nb-1],m->replace[m->nb-1]);
246                 continue;
247             }
248
249             // Commentaires ou ligne vide
250             Nettoyage(tmp);
251             if ( ((tmp[0]=='/') && (tmp[1]=='/')) || (tmp[0]=='\n') || (tmp[0]=='\0') )
252             continue;
253
254             printf("@CP %d, inst %s\n",CP,tmp);
255             CP++;
256         }
257
258         //on remplace les macros en enlevant les commentaires et etiquettes
259         rewind(source);
260         rewind(dest);
261
262         while( !feof(source) )
263         {
264             Lread(source,tmp);
265             strcpy(tmp2,tmp);
266             Nettoyage(tmp);
267             if ( (tmp[0]=='/') || (tmp[0]==':') || (tmp[0]=='$') || (tmp[0]=='\n') || (tmp[0]=='\0') )
268                 continue;
269             if ( (ptr=strchr(tmp2,'$'))!=NULL )
270             {
271                 // printf("avant: %s\n",tmp);
272                 replaceMacro(tmp2,m);
273                 // printf("apres: %s\n",tmp);
274             }
275             fprintf(dest,"%s",tmp2);
276         }
277     }
278
279     // Ajout d'une macro
280     void addMacro(TMacros *m, char *name, char *replace)
281     {
282         strcpy(m->name[m->nb],name);
283         strcpy(m->replace[m->nb],replace);
284         m->nb++;
285     }
286
287     // Recherche d'une macro
288     char *searchMacro(TMacros *m, const char *name)
289     {
290         int i,j;
291
292         //printf("Search macro name %s\n",name);
293         for(i=0;i<m->nb;i++)
294         {
295             j=strcmp( (*m).name[i],name);
296             if (j==0)
297             {
298                 printf ( "\tFound macro name %s replaced by %s\n",name,(*m).replace[i]);
299                 return (*m).replace[i];
300             }
```

```
301         }
302
303         // si on est ici c'est que la macro est inconnue !
304         printf("ERROR macro %s unknown !\n",name);
305         exit(1);
306     }
307
308     /* ------------------------------------------------
309         Programme Principal
310         ------------------------------------------------ */
311     int main (int argc, char *argv[])
312     {
313         FILE *source;
314         FILE *dest;
315         FILE *ftmp;
316         char name_ftmp[100], *tmp;
317         TLabels l;
318         TMacros m;
319         int CP=0;
320
321         // Test des arguments
322         if (argc!=3)
323         {
324             printf("Nombre de parametres invalide (%d)\n\tR3Kasm <source> <destination>\n",argc);
325             return -1;
326         }
327         printf("R3Kasm: source file %s, target %s\n",argv[1],argv[2]);
328
329         // ouverture du fichier source
330         source=fopen(argv[1],"rt");
331         if ( source==NULL )
332         {
333             printf("Impossible d'ouvrir %s \n",argv[1]);
334             exit(1);
335         }
336
337         // creation du fichier temporaire, preparsed
338         strncpy(name_ftmp,argv[1],sizeof(name_ftmp));
339         tmp = strrchr(name_ftmp,'.');
340         if ( !tmp )
341             strcat(name_ftmp,".pasm");
342         else
343             strcpy(tmp,".pasm");
344
345         ftmp=fopen(name_ftmp,"w");
346         if ( ftmp==NULL)
347         {
348             printf("Impossible d'ouvrir %s\n",name_ftmp);
349             exit(1);
350         }
351         prePARSE(source,ftmp,&l,&m);
352
353         //fermeture des fichiers
354         fclose(source);
355         fclose(ftmp);
356
357         //ouverture du fichier preparse
358         source=fopen(name_ftmp,"rt");
359         if ( source==NULL )
360         {
361             printf("Impossible d'ouvrir %s\n",name_ftmp);
362             exit(1);
363         }
364
365         //ouverture du fichier destination
366         dest=fopen(argv[2],"w");
367         if ( dest==NULL )
368         {
369             printf("Impossible d'ouvrir %s\n",argv[2]);
370             exit(1);
371         }
372
373         // Assemblage
374         printf("ASM in progress... \n");
375         rewind(source);
376         CP=0;
377
378         while ( !feof(source) )
379         {
380             static char op[30],reste[128],tmp2[SIZE_LABELS];
381             static int rs, rt, rd, imm, valdec, adr;
382             static char str_imm[30];
383
384             fscanf(source,"%s",op);
385             strUP(op); //majuscules
386             //      printf("Instruction %s\n",op);
387             CP++;
388             fscanf(source,"%s\n",reste);
389             strUP(reste);
390             printf("Instruction %s %s\n",op,reste);
391
392             if ( strcmp(op,"LSL")==0 )
393             {
394                 sscanf(reste,"R%d,R%d,%d\n",&rd,&rt,&valdec);
395                 opMake(dest,TYPE_R,OP_LSL,0,rt,rd,valdec,F_LSL,0,0);
396                 continue;
397             }
398
399             if ( strcmp(op,"LSR")==0 )
400             {
```

```
            sscanf(reste,"R%d,R%d,%d\n",&rd,&rt,&valdec);
            opMake(dest,TYPE_R,OP_LSR,0,rt,rd,valdec,F_LSR,0,0);
            continue;
        }

        if ( strcmp(op,"ADD")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_ADD,rs,rt,rd,0,F_ADD,0,0);
            continue;
        }

        if ( strcmp(op,"ADDU")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_ADDU,rs,rt,rd,0,F_ADDU,0,0);
            continue;
        }

        if ( strcmp(op,"SUB")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_SUB,rs,rt,rd,0,F_SUB,0,0);
            continue;
        }

        if ( strcmp(op,"SUBU")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_SUBU,rs,rt,rd,0,F_SUBU,0,0);
            continue;
        }

        if ( strcmp(op,"AND")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_AND,rs,rt,rd,0,F_AND,0,0);
            continue;
        }

        if ( strcmp(op,"OR")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_OR,rs,rt,rd,0,F_OR,0,0);
            continue;
        }

        if ( strcmp(op,"XOR")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_XOR,rs,rt,rd,0,F_XOR,0,0);
            continue;
        }

        if ( strcmp(op,"NOR")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_NOR,rs,rt,rd,0,F_NOR,0,0);
            continue;
        }

        if ( strcmp(op,"SLT")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_SLT,rs,rt,rd,0,F_SLT,0,0);
            continue;
        }

        if ( strcmp(op,"SLTU")==0 )
        {
            sscanf(reste,"R%d,R%d,R%d\n",&rd,&rs,&rt);
            opMake(dest,TYPE_R,OP_SLTU,rs,rt,rd,0,F_SLTU,0,0);
            continue;
        }

        if ( strcmp(op,"JR")==0 )
        {
            sscanf(reste,"R%d\n",&rs);
            opMake(dest,TYPE_R,OP_JR,rs,0,0,0,F_JR,0,0);
            continue;
        }

        if ( strcmp(op,"JALR")==0 )
        {
            sscanf(reste,"R%d,R%d\n",&rs,&rd);
            opMake(dest,TYPE_R,OP_JALR,rs,0,rd,0,F_JALR,0,0);
            continue;
        }

        if ( strcmp(op,"BLTZ")==0 )
        {
            sscanf(reste,"R%d,%s\n",&rs,tmp2);
            if (tmp2[0]==':')
            {
                printf("\tlabel %s\n",tmp2);
                imm=(cpLabel(tmp2,&l)-CP);
            }
            else
                sscanf(tmp2,"%d",&imm);
            opMake(dest,TYPE_I,OP_BLTZ,rs,0,0,0,NULL,imm,0);
```

```
501                 continue;
502             }
503
504             if ( strcmp(op,"BGEZ")==0 )
505             {
506                 sscanf(reste,"R%d,%s\n",&rs,tmp2);
507                 if (tmp2[0]==':')
508                 {
509                     printf("\tlabel %s\n",tmp2);
510                     imm=(cpLabel(tmp2,&l)-CP);
511                 }
512                 else
513                     sscanf(tmp2,"%d",&imm);
514                 opMake(dest,TYPE_I,OP_BGEZ,rs,1,0,0,NULL,imm,0);
515                 continue;
516             }
517
518             if ( strcmp(op,"BLTZAL")==0 )
519             {
520                 sscanf(reste,"R%d,%s\n",&rs,tmp2);
521                 if (tmp2[0]==':')
522                 {
523                     printf("\tlabel %s\n",tmp2);
524                     imm=(cpLabel(tmp2,&l)-CP);
525                 }
526                 else
527                     sscanf(tmp2,"%d",&imm);
528                 opMake(dest,TYPE_I,OP_BLTZAL,rs,16,0,0,NULL,imm,0);
529                 continue;
530             }
531
532             if (strcmp(op,"BGEZAL")==0 )
533             {
534                 sscanf (reste,"R%d,%s\n",&rs,tmp2);
535                 if (tmp2[0]==':')
536                 {
537                     printf("\tlabel %s\n",tmp2);
538                     imm=(cpLabel(tmp2,&l)-CP);
539                 }
540                 else
541                     sscanf(tmp2,"%d",&imm);
542                 opMake(dest,TYPE_I,OP_BGEZAL,rs,17,0,0,NULL,imm,0);
543                 continue;
544             }
545
546             if ( strcmp(op,"J")==0 )
547             {
548                 if (reste[0]==':')
549                 {
550                     adr=cpLabel(reste,&l);
551                     printf("\tlabel %s\n",reste);
552                 }
553                 else
554                     sscanf(reste,"%d\n",&adr);
555                 opMake(dest,TYPE_J,OP_J,0,0,0,0,NULL,0,adr);
556                 continue;
557             }
558
559             if ( strcmp(op,"JAL")==0 )
560             {
561                 if (reste[0]==':')
562                 {
563                     adr=cpLabel(reste,&l);
564                     printf("\tlabel %s\n",reste);
565                 }
566                 else
567                     sscanf (reste,"%d\n",&adr);
568                 opMake(dest,TYPE_J,OP_JAL,0,0,0,0,NULL,0,adr);
569                 continue;
570             }
571
572             if ( strcmp(op,"BEQ")==0 )
573             {
574                 sscanf (reste,"R%d,R%d,%s\n",&rs,&rt,tmp2);
575                 if (tmp2[0]==':')
576                 {
577                     printf("\tlabel %s\n",tmp2);
578                     imm=(cpLabel(tmp2,&l)-CP);
579                 }
580                 else
581                     sscanf(tmp2,"%d",&imm);
582                 opMake(dest,TYPE_I,OP_BEQ,rs,rt,0,0,NULL,imm,0);
583                 continue;
584             }
585
586             if ( strcmp(op,"BNE")==0 )
587             {
588                 sscanf(reste,"R%d,R%d,%s\n",&rs,&rt,tmp2);
589                 if (tmp2[0]==':')
590                 {
591                     printf("\tlabel %s\n",tmp2);
592                     imm=(cpLabel(tmp2,&l)-CP);
593                 }
594                 else
595                     sscanf(tmp2,"%d",&imm);
596                 opMake(dest,TYPE_I,OP_BNE,rs,rt,0,0,NULL,imm,0);
597                 continue;
598             }
599
600             if ( strcmp(op,"BLEZ")==0 )
```

```
601             {
602                 sscanf(reste,"R%d,%s\n",&rs,tmp2);
603                 if (tmp2[0]==':')
604                 {
605                     printf("\tlabel %s\n",tmp2);
606                     imm=(cpLabel(tmp2,&l)-CP);
607                 }
608                 else
609                     sscanf(tmp2,"%d",&imm);
610                 opMake(dest,TYPE_I,OP_BLEZ,rs,0,0,0,NULL,imm,0);
611                 continue;
612             }
613 
614             if ( strcmp(op,"BGTZ")==0 )
615             {
616                 sscanf(reste,"R%d,%s\n",&rs,tmp2);
617                 if (tmp2[0]==':')
618                 {
619                     printf("\tlabel %s\n",tmp2);
620                     imm=(cpLabel(tmp2,&l)-CP);
621                 }
622                 else
623                     sscanf(tmp2,"%d",&imm);
624                 opMake(dest,TYPE_I,OP_BGTZ,rs,0,0,0,NULL,imm,0);
625                 continue;
626             }
627 
628             if ( strcmp(op,"ADDI")==0 )
629             {
630                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
631                 opMake(dest,TYPE_I,OP_ADDI,rs,rt,0,0,0,imm,0);
632                 continue;
633             }
634 
635             if ( strcmp(op,"ADDIU")==0 )
636             {
637                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
638                 opMake(dest,TYPE_I,OP_ADDIU,rs,rt,0,0,0,imm,0);
639                 continue;
640             }
641 
642             if ( strcmp(op,"SLTI")==0 )
643             {
644                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
645                 opMake(dest,TYPE_I,OP_SLTI,rs,rt,0,0,0,imm,0);
646                 continue;
647             }
648 
649             if ( strcmp(op,"SLTIU")==0 )
650             {
651                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
652                 opMake(dest,TYPE_I,OP_SLTIU,rs,rt,0,0,0,imm,0);
653                 continue;
654             }
655 
656             if ( strcmp(op,"ANDI")==0 )
657             {
658                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
659                 opMake(dest,TYPE_I,OP_ANDI,rs,rt,0,0,0,imm,0);
660                 continue;
661             }
662 
663             if ( strcmp(op,"ORI")==0 )
664             {
665                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
666                 opMake(dest,TYPE_I,OP_ORI,rs,rt,0,0,0,imm,0);
667                 continue;
668             }
669 
670             if ( strcmp(op,"XORI")==0 )
671             {
672                 sscanf(reste,"R%d,R%d,%d\n",&rt,&rs,&imm);
673                 opMake(dest,TYPE_I,OP_XORI,rs,rt,0,0,0,imm,0);
674                 continue;
675             }
676 
677             if ( strcmp(op,"LUI")==0 )
678             {
679                 sscanf(reste,"%d(R%d)\n",&imm,&rt);
680                 opMake(dest,TYPE_I,OP_LUI,0,rt,0,0,0,imm,0);
681                 continue;
682             }
683 
684             if ( strcmp(op,"LB")==0 )
685             {
686                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
687                 opMake(dest,TYPE_I,OP_LB,rs,rt,0,0,0,imm,0);
688                 continue;
689             }
690 
691             if ( strcmp(op,"LH")==0 )
692             {
693                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
694                 opMake(dest,TYPE_I,OP_LH,rs,rt,0,0,0,imm,0);
695                 continue;
696             }
697 
698             if ( strcmp(op,"LW")==0 )
699             {
700                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
```

```
701                 opMake(dest,TYPE_I,OP_LW,rs,rt,0,0,0,imm,0);
702                 continue;
703             }
704 
705             if ( strcmp(op,"LBU")==0 )
706             {
707                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
708                 opMake(dest,TYPE_I,OP_LBU,rs,rt,0,0,0,imm,0);
709                 continue;
710             }
711 
712             if ( strcmp(op,"LHU")==0 )
713             {
714                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
715                 opMake(dest,TYPE_I,OP_LHU,rs,rt,0,0,0,imm,0);
716                 continue;
717             }
718 
719             if ( strcmp(op,"SB")==0 )
720             {
721                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
722                 opMake(dest,TYPE_I,OP_SB,rs,rt,0,0,0,imm,0);
723                 continue;
724             }
725 
726             if ( strcmp(op,"SH")==0 )
727             {
728                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
729                 opMake(dest,TYPE_I,OP_SH,rs,rt,0,0,0,imm,0);
730                 continue;
731             }
732 
733             if ( strcmp(op,"SW")==0 )
734             {
735                 sscanf(reste,"R%d,%d(R%d)\n",&rt,&imm,&rs);
736                 opMake(dest,TYPE_I,OP_SW,rs,rt,0,0,0,imm,0);
737                 continue;
738             }
739 
740             // Mnemonique inconnu
741             printf("\tERROR UNKNOWN INSTRUCTION\n");
742             exit(1);
743         }
744         // fermeture des fichiers
745         fclose(source);
746         fclose(dest);
747     }
748 
```

ANNEXE A5

```
1   //************PROGRAMME DE TEST*****************
2   //testadd.asm
3   //Realise l'addition de A*B
4
5   $A 12
6   $B 16
7   $C 641
8   $D 256
9
10  addi R1,R0,$A
11  // R1 = R0 + 12                  => R1 = 12
12  addi R2,R0,$B
13  // R2 = R0 + 16                  => R2 = 16
14  addi R3,R0,$C
15  // R3 = R0 + 641                 => R3 = 641
16  addi R4,R0,$D
17  // R4 = R0 + 256                 => R4 = 256
18  lw R5,4(R1)
19  // R5 = MEM[4+R1]                => R5 = 4
20  lw R6,20(R1)
21  // R6 = MEM[20+R1]               => R6 = 8
22  sw R3,0(R2)
23  // MEM[0+R2] = R3                    => MEM[16] = 641
24  sw R4,8(R1)
25  // MEM[8+R1] = R4                    => MEM[20] = 256
26  sw R5,-601(R3)
27  // MEM[-601+R3] = R5     => MEM[40] = 4
28  add R20,R1,R1
29  add R21,R1,R1
30  add R22,R1,R1
31  // R4 = R6 + R3          => R4 = 649
32  lw R10,16(R0)
33  // R10 = MEM[16+R0]      => R10 = MEM[16] = 641
34  add R7,R5,R6
35  // R7 = R5 + R6          => R7 = 12
36  add R8,R2,R3
37  // R8 = R2 + R3          => R8 = 657
38  add R9,R4,R4
39  // R9 = R4 + R4          => R9 = 512
40  add R11,R10,R4
41  // R11 = R10 + R4            => R11 = 897
42  add R4,R6,R3
43  add R20,R1,R1
44  add R21,R1,R1
45  add R22,R1,R1
46
47
48
```

testadd.txt

```
 1   00100000000000010000000000001100
 2   00100000000000100000000000010000
 3   00100000000000110000001010000001
 4   00100000000001000000000100000000
 5   10001100001001010000000000000100
 6   10001100001001100000000000010100
 7   10101100010000110000000000000000
 8   10101100001001000000000000001000
 9   10101100011001011111110110100111
10   00000000001000011010000000100000
11   00000000001000011010100000100000
12   00000000001000011011000000100000
13   10001100000010100000000000010000
14   00000000101001100011100000100000
15   00000000010000110100000000100000
16   00000000100001000100100000100000
17   00000001010001000101100000100000
18   00000000110000110010000000100000
19   00000000001000011010000000100000
20   00000000001000011010100000100000
21   00000000001000011011000000100000
22
```

```
//************PROGRAMME DE TEST*****************
//bench_1.asm
//;Aleas de type irresolvable

//;Aleas de rangement
addi  R1,R0,1
sw R1,2(R0)

//;Aleas de chargement
addi  R1,R0,1
sw R1,4(R0)
lw R1,4(R0)
addi R2,R1,2
add R10,R2,R1
//; aleas resolvable

//;TypeR
//; dependance ds l etage EX/MEM
addi R1,R0,2
addi R2,R1,3

//; dependance ds l etage MEM/ER
addi R1,R0,2
addi R2,R0,3
addi R3,R1,3

//; dependance ds l etage EX/MEM et MEM/ER
addi R1,R0,2
addi R2,R0,3
add R3,R1,R2

//;Type JR ou JALR
//;dependance ds l etage DI/EX
addi R1,R0,2
jr R1
addi R2,R0,0
addi R2,R1,3
addi R2,R1,4
addi R2,R1,5

//;dependance ds l etage EX/MEM
addi R1,R0,2
addi R1,R0,3
jr R1
addi R2,R0,0
addi R2,R1,4
addi R2,R1,5
addi R2,R1,5
//; aleas resolvable

//; dependance ds l etage EX/MEM
addi R1,R0,2
addi R2,R1,3

//; dependance ds l etage MEM/ER
addi R1,R0,2
addi R2,R0,3
addi R3,R1,3

//; dependance ds l etage EX/MEM et MEM/ER
addi R1,R0,2
addi R2,R0,3
add R3,R1,R2

```

bench_1.txt

```
00100000000000010000000000000001
10101100000000010000000000000010
00100000000000010000000000000001
10101100000000010000000000000100
10001100000000010000000000000100
00100000010001000000000000000010
00000000010000010101000000100000
00100000000000010000000000000010
00100000010001000000000000000011
00100000000000010000000000000010
00100000000000010000000000000011
00100000010001100000000000000011
00100000000000010000000000000010
00100000000000010000000000000011
00000000001000100001100000100000
00100000000000010000000000000010
00000000001000000000000000001000
00100000000000010000000000000000
00100000010001000000000000000011
00100000010001000000000000000100
00100000001000100000000000000101
00100000000000010000000000000010
00100000000000010000000000000011
00000000001000000000000000001000
00100000000000010000000000000000
00100000001000100000000000000100
00100000010001000000000000000101
00100000001000100000000000000101
00100000000000010000000000000010
00100000010001000000000000000011
00100000000000010000000000000010
00100000000000010000000000000011
00100000010001100000000000000011
00100000000000010000000000000010
00100000000000010000000000000011
00000000001000100001100000100000
```

```
//************PROGRAMME DE TEST*****************
//test_aleas_plus.asm
//
:debut
// on met 10 dans R1
ADDI R1,R0,10

// test aleas simple R2 <= 10
ADD R2,R1,R0

// test aleas double R3 <= 20
ADD R3,R2,R1

// debut test du choix entre l'etage mem et er R4 <= 1
// initialisation de R4 a 0
ADDI R4,R0,1
//R4 <= R4+1 4 fois a la suite
ADDI R4,R4,1
ADDI R4,R4,1
ADDI R4,R4,1
ADDI R4,R4,1

// test si on peut faire acces simultanement au meme etage
ADD R9,R4,R4
ADD R10,R4,R4
BNE R9,R10,:erreur

// on affecte 5 a R5 qui doit etre le resultat attendu de R4
:test_suite
ADDI R5,R0,5

// on sauvegarde la valeur de R5 a l@ 8 pour tester aleas de lecture suivie d une operation
SW R5,8(R0)

//verifie si le resultat de R4 correspond a ce qui est attendu et test si le halt est prioritaire sur le flush du branch
BEQ R5,R4,:suite

//ceci ne doit pas etre execute si l'aleas et le halt sont bien traite
LW R6,8(R0)
ADD R7,R6,R0
ADD R8,R6,R0
j :erreur
//si c'est mal traite pas de rebouclage du programme

//sinon on verifie la bonne insertion de la bulle pour une lecture suivie d'une operation
:suite
LW R6,8(R0)
ADD R7,R6,R0

BEQ R6,R7,:debut
//on reboucle si tous les resultats sont ceux attendu
:erreur

```

test_aleas_plus.txt

```
1   00100000000000010000000000001010
2   00000000001000000001000000100000
3   00000000100000100011100000100000
4   00100000000001000000000000000001
5   00100000100001000000000000000001
6   00100000100001000000000000000001
7   00100000100001000000000000000001
8   00100000100001000000000000000001
9   00000000100001000100100000100000
10  00000000100001000101000000100000
11  00010101001010100000000000001010
12  00100000000001010000000000000101
13  10101100000001010000000000001000
14  00010000101001000000000000000100
15  10001100000001100000000000001000
16  00000000110000000011100000100000
17  00000000110000000100000000100000
18  00001000000000000000000000010101
19  10001100000001100000000000001000
20  00000000110000000011100000100000
21  00010000110001111111111111101011
22
```

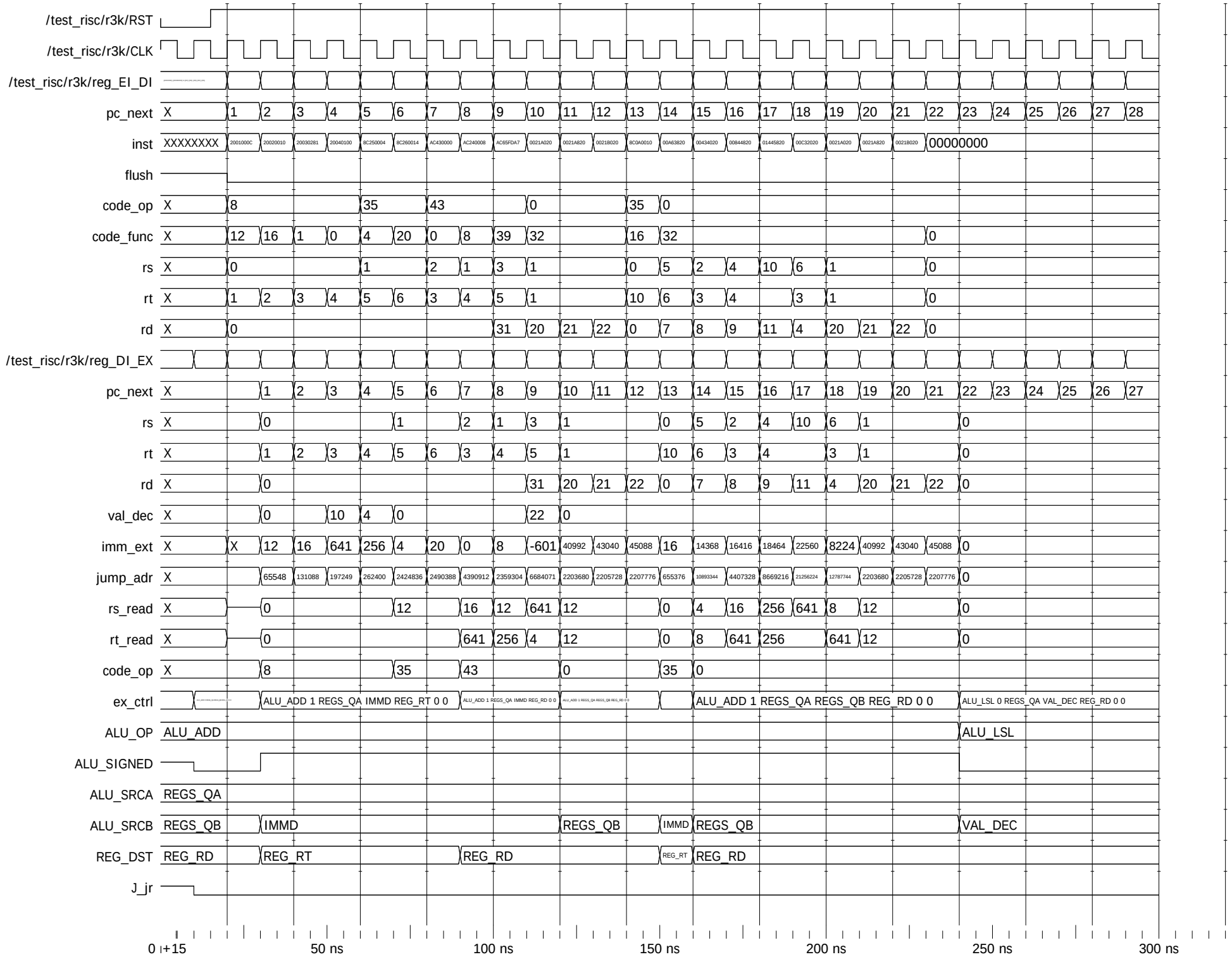
ANNEXE A6

ANNEXE A6 I

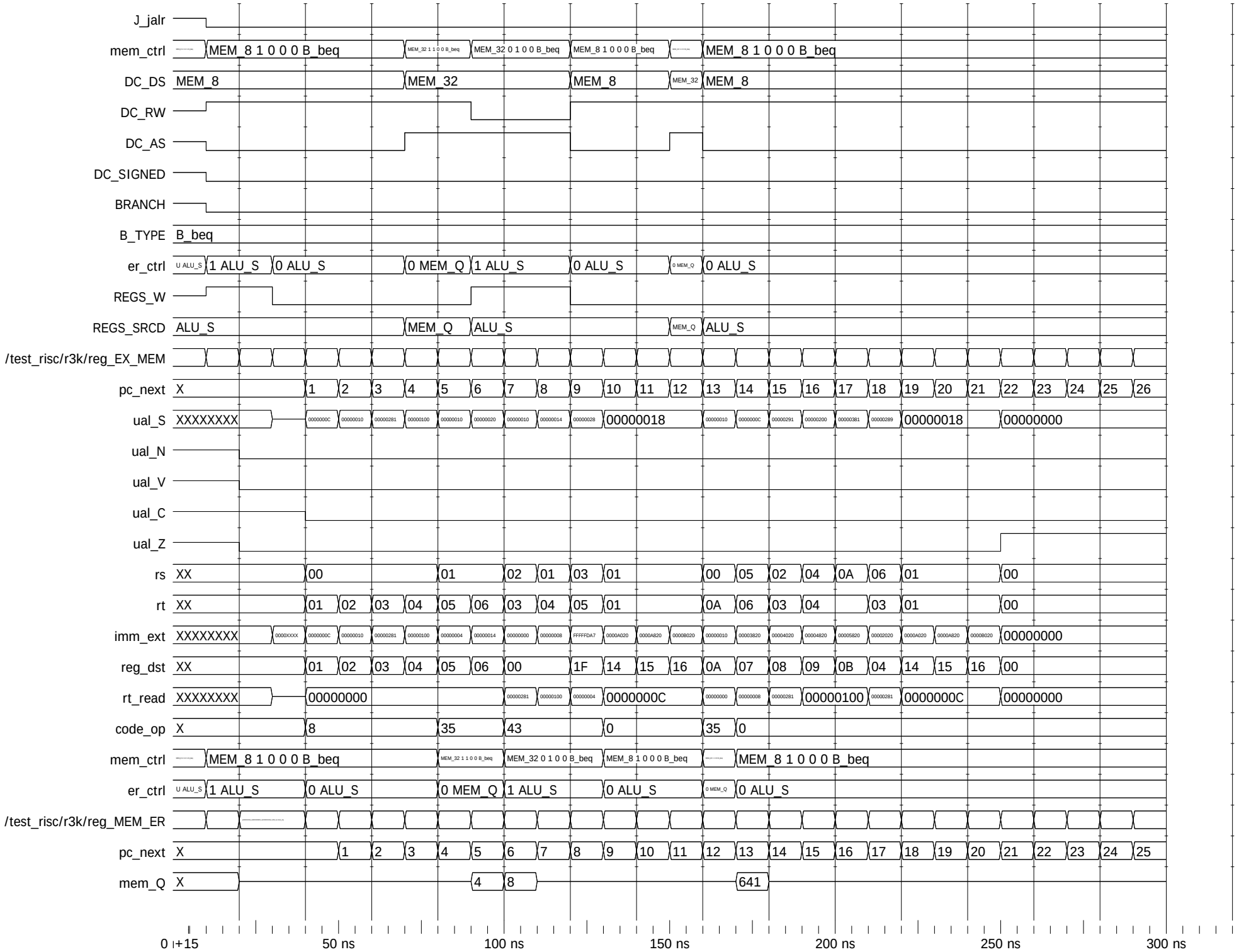
Résultats de simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd*

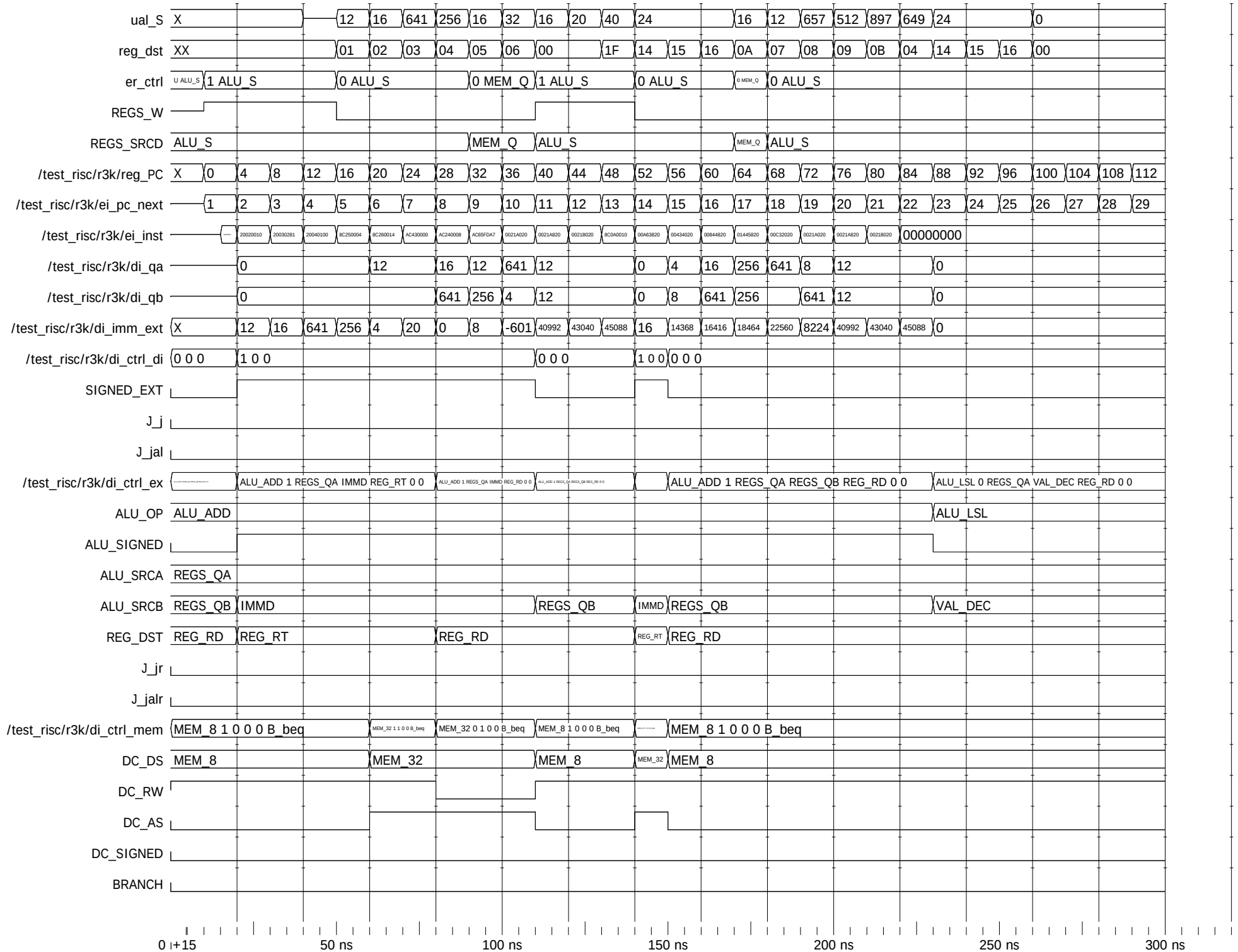
ANNEXE A6 I.1

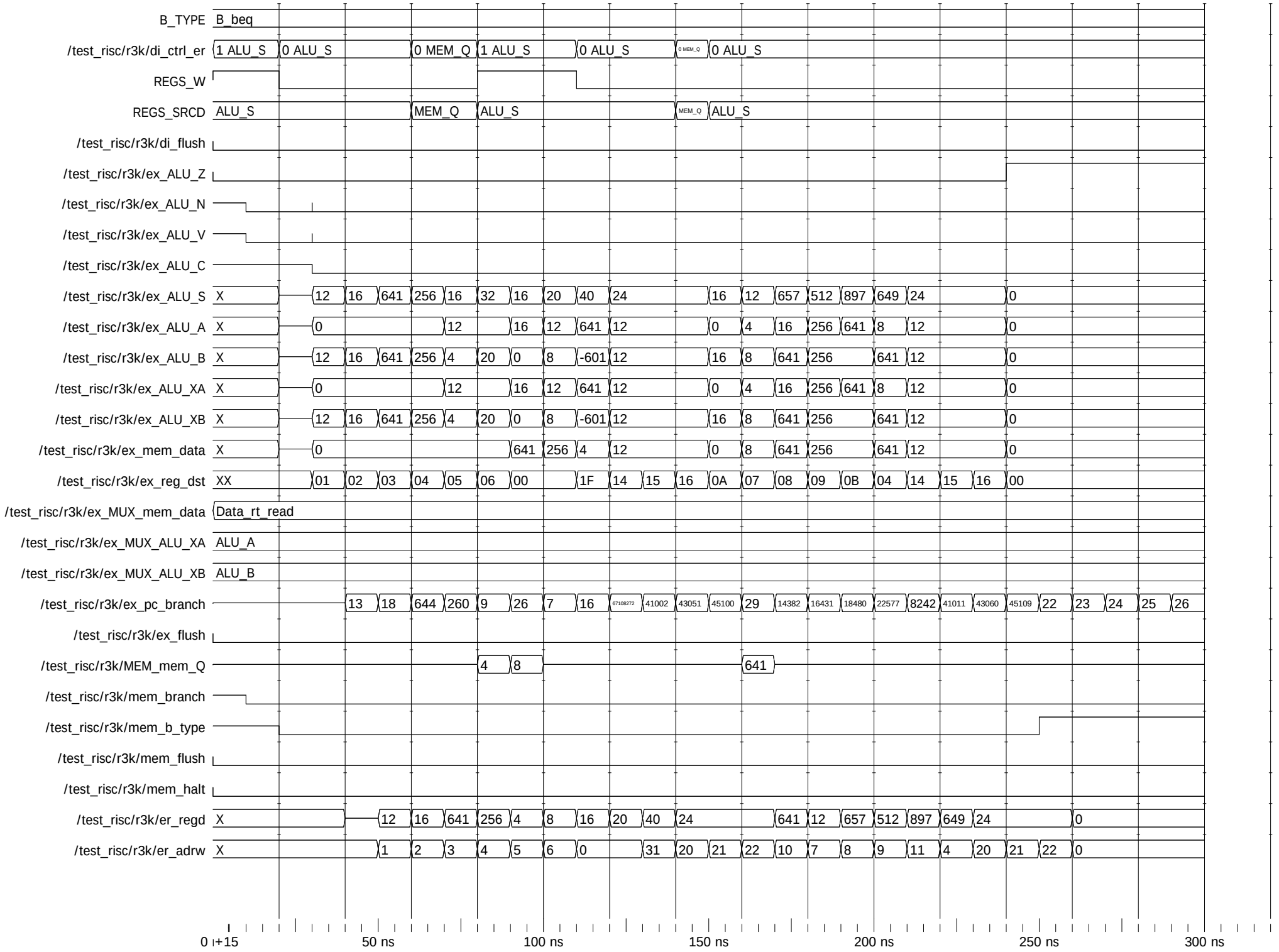
Résultats de la simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd*
simulés avec fichier *testadd.txt*



Entity:test_risc Architecture:behavior Date: Mon Feb 18 09:54:08 PM Romance Standard Time 2013 Row: 1 Page: 1

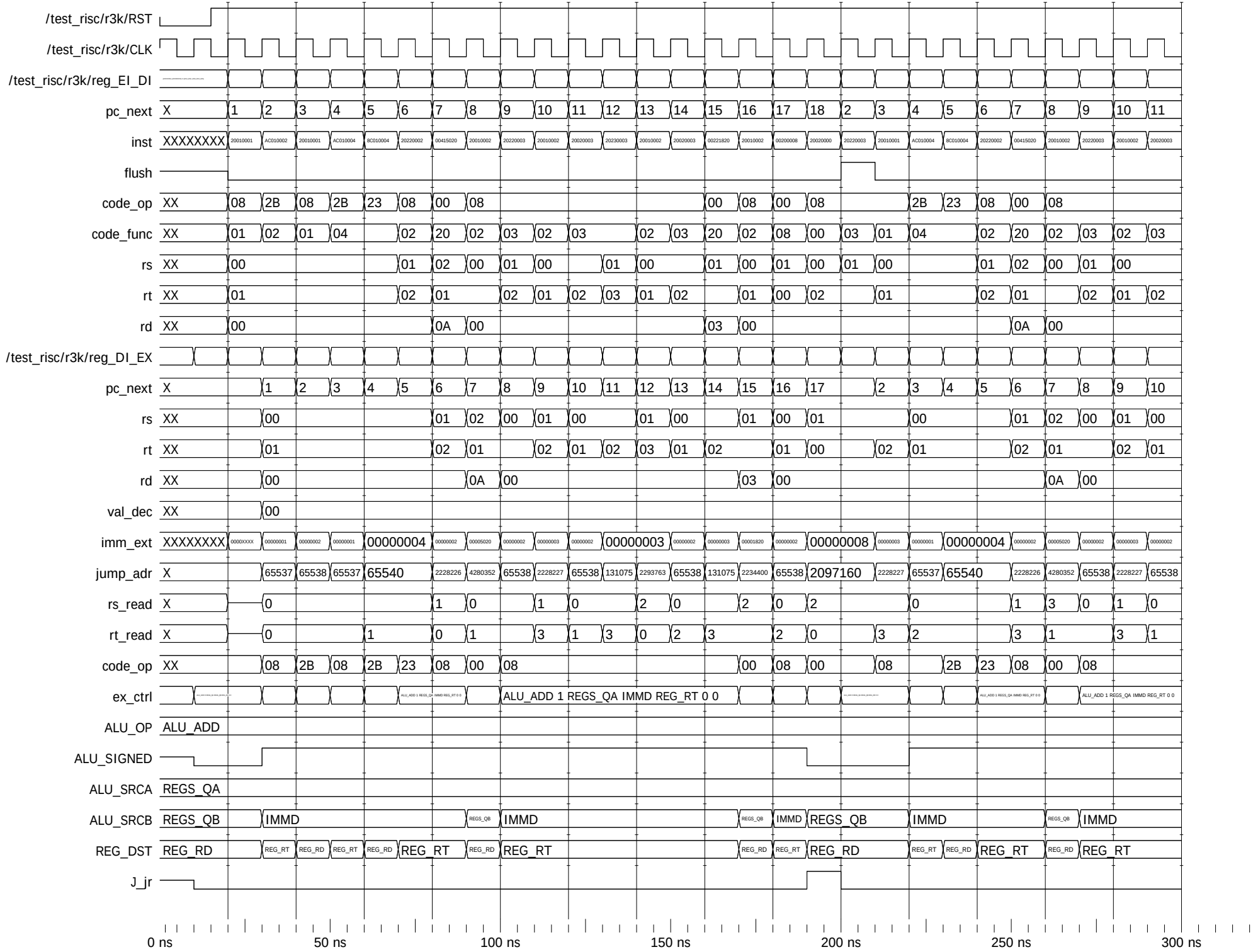

J_jalr
mem_ctrl
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_EX_MEM
pc_next
ual_S
ual_N
ual_V
ual_C
ual_Z
rs
rt
imm_ext
reg_dst
rt_read
code_op
mem_ctrl
er_ctrl
/test_risc/r3k/reg_MEM_ER
pc_next
mem_Q
MEM_8 1 0 0 0 B_beq
MEM_32 1 1 0 0 B_beq
MEM_32 0 1 0 0 B_beq
MEM_8
MEM_32
B_beq
U ALU_S
1 ALU_S
0 ALU_S
0 MEM_Q
ALU_S
MEM_Q
0 ns
+15
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


ual_S
reg_dst
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_PC
/test_risc/r3k/ei_pc_next
/test_risc/r3k/ei_inst
/test_risc/r3k/di_qa
/test_risc/r3k/di_qb
/test_risc/r3k/di_imm_ext
/test_risc/r3k/di_ctrl_di
SIGNED_EXT
J_j
J_jal
/test_risc/r3k/di_ctrl_ex
ALU_OP
ALU_SIGNED
ALU_SRCA
ALU_SRCB
REG_DST
J_jr
J_jalr
/test_risc/r3k/di_ctrl_mem
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
ALU_ADD 1 REGS_QA IMMD REG_RT 0 0
ALU_ADD 1 REGS_QA IMMD REG_RD 0 0
ALU_ADD 1 REGS_QA REGS_QB REG_RD 0 0
ALU_LSL 0 REGS_QA VAL_DEC REG_RD 0 0
MEM_8 1 0 0 0 B_beq
MEM_32 0 1 0 0 B_beq
0 +15
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

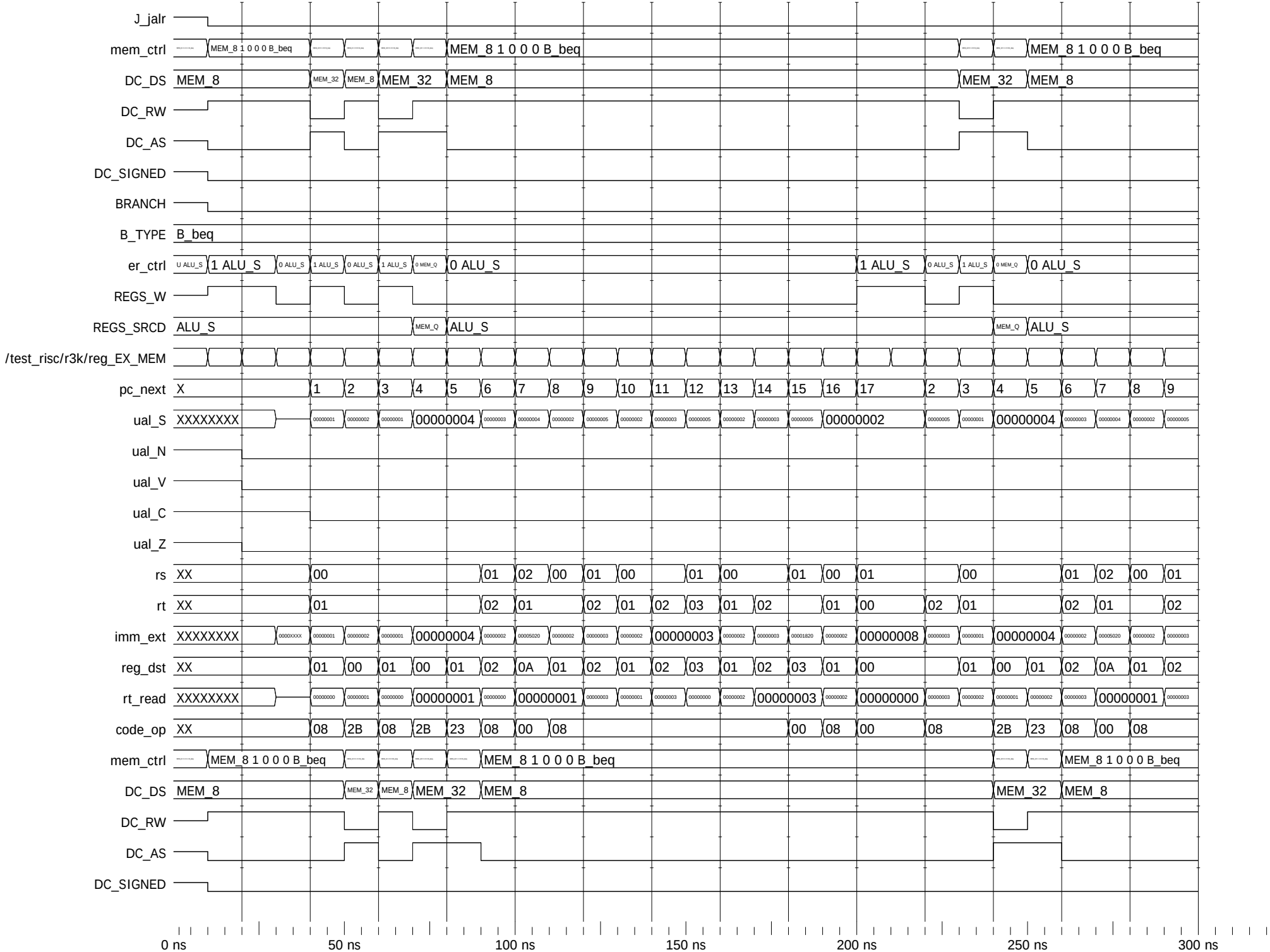

B_TYPE
B_beq
/test_risc/r3k/di_ctrl_er
/test_risc/r3k/REGS_W
/test_risc/r3k/REGS_SRCD
/test_risc/r3k/di_flush
/test_risc/r3k/ex_ALU_Z
/test_risc/r3k/ex_ALU_N
/test_risc/r3k/ex_ALU_V
/test_risc/r3k/ex_ALU_C
/test_risc/r3k/ex_ALU_S
/test_risc/r3k/ex_ALU_A
/test_risc/r3k/ex_ALU_B
/test_risc/r3k/ex_ALU_XA
/test_risc/r3k/ex_ALU_XB
/test_risc/r3k/ex_mem_data
/test_risc/r3k/ex_reg_dst
/test_risc/r3k/ex_MUX_mem_data
/test_risc/r3k/ex_MUX_ALU_XA
/test_risc/r3k/ex_MUX_ALU_XB
/test_risc/r3k/ex_pc_branch
/test_risc/r3k/ex_flush
/test_risc/r3k/MEM_mem_Q
/test_risc/r3k/mem_branch
/test_risc/r3k/mem_b_type
/test_risc/r3k/mem_flush
/test_risc/r3k/mem_halt
/test_risc/r3k/er_regd
/test_risc/r3k/er_adrw
0 +15
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

ANNEXE A6 I.2

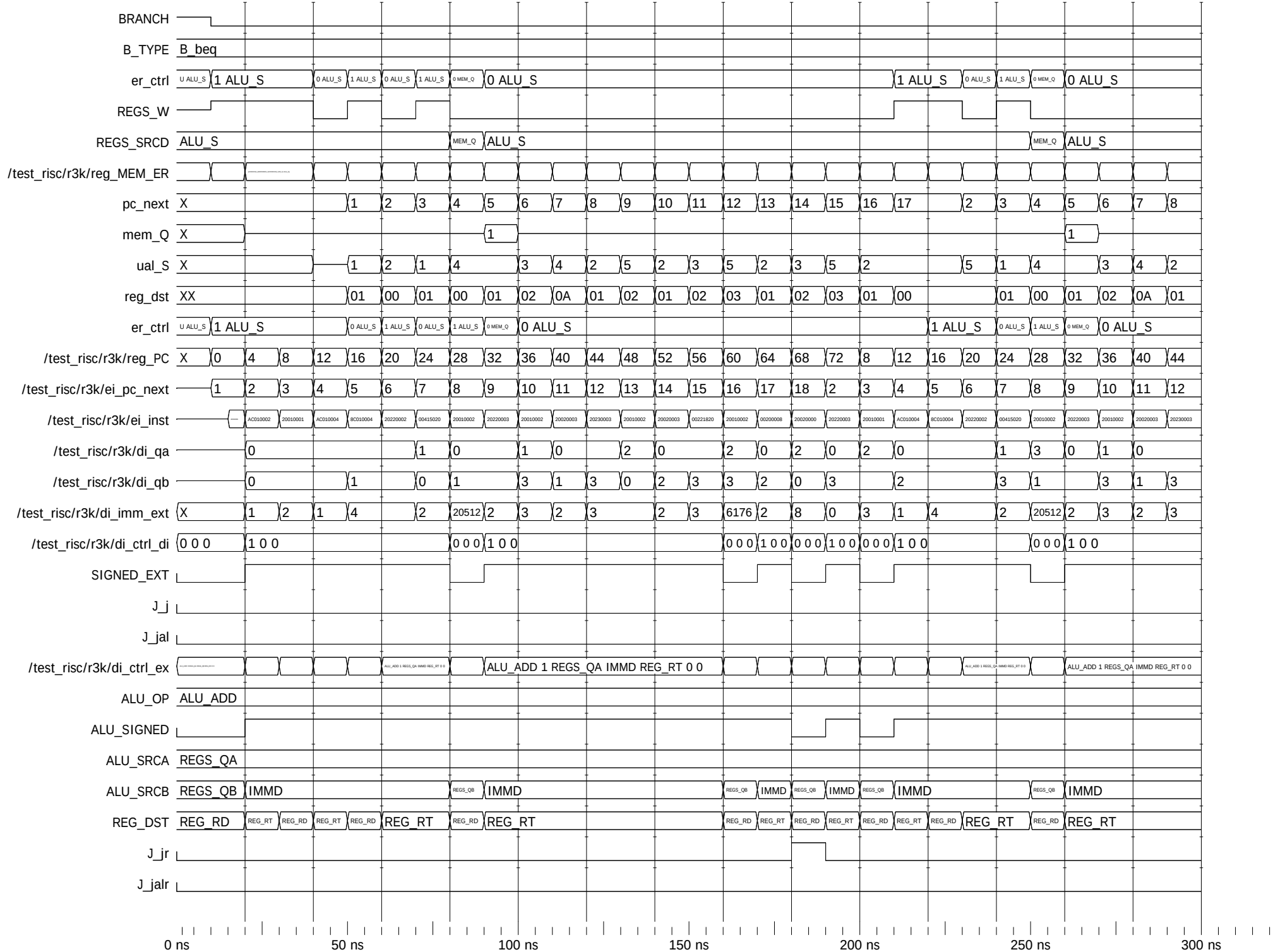
Résultats de la simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd*
simulés avec fichier *bench_1.txt*

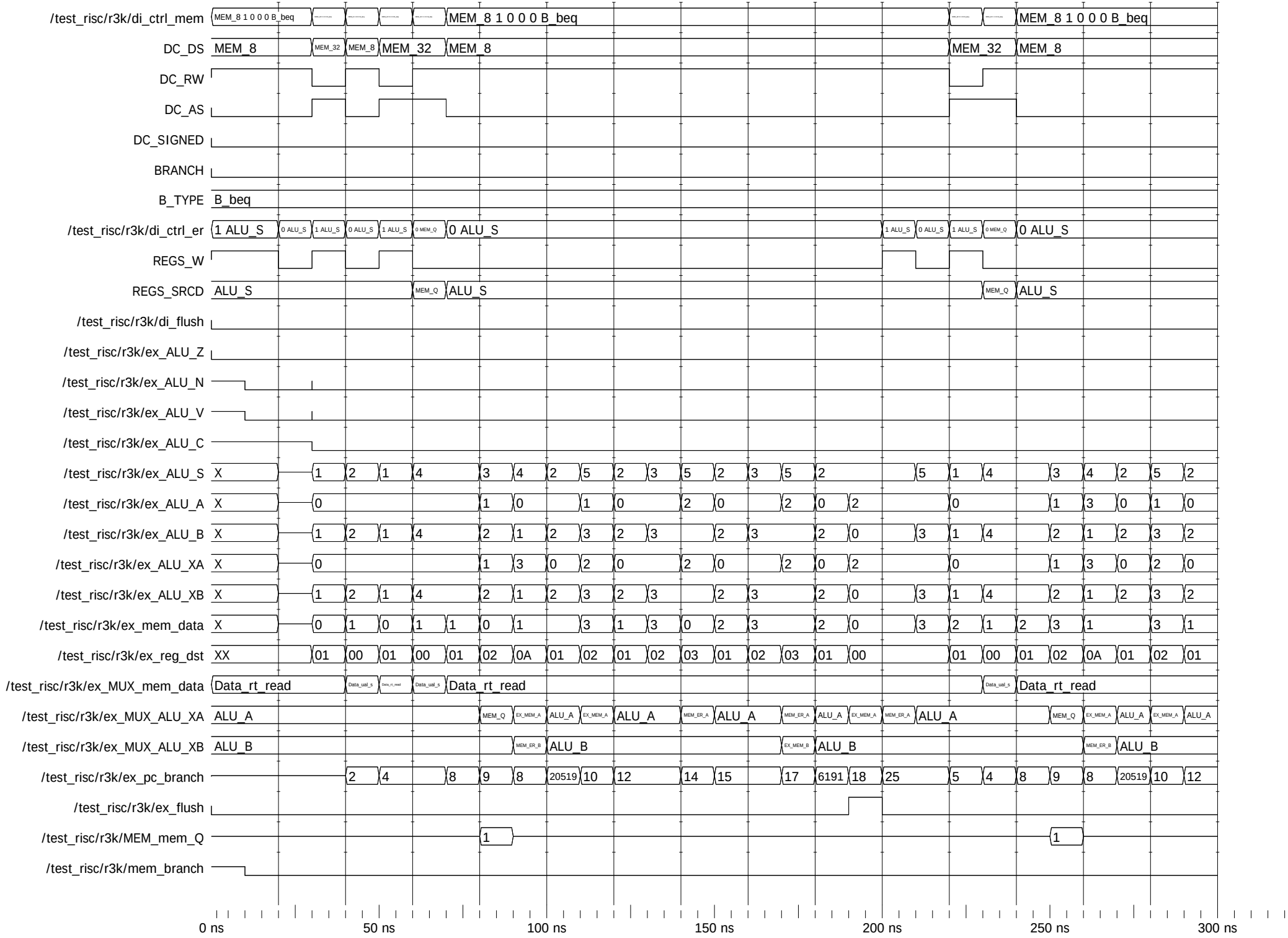


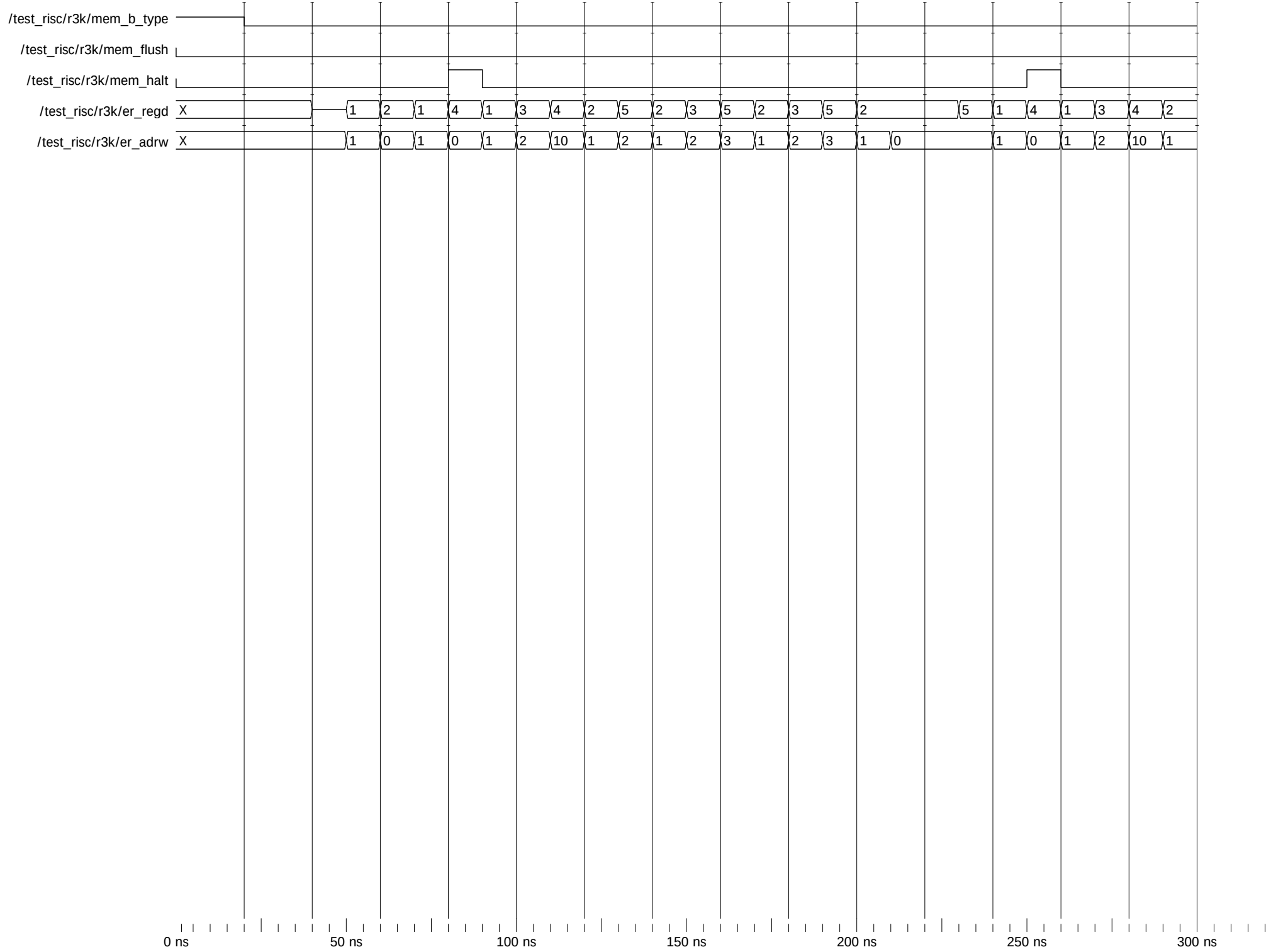
Entity:test_risc Architecture:behavior Date: Mon Feb 18 09:59:30 PM Romance Standard Time 2013 Row: 1 Page: 1



Entity:test_risc Architecture:behavior Date: Mon Feb 18 09:59:30 PM Romance Standard Time 2013 Row: 1 Page: 2

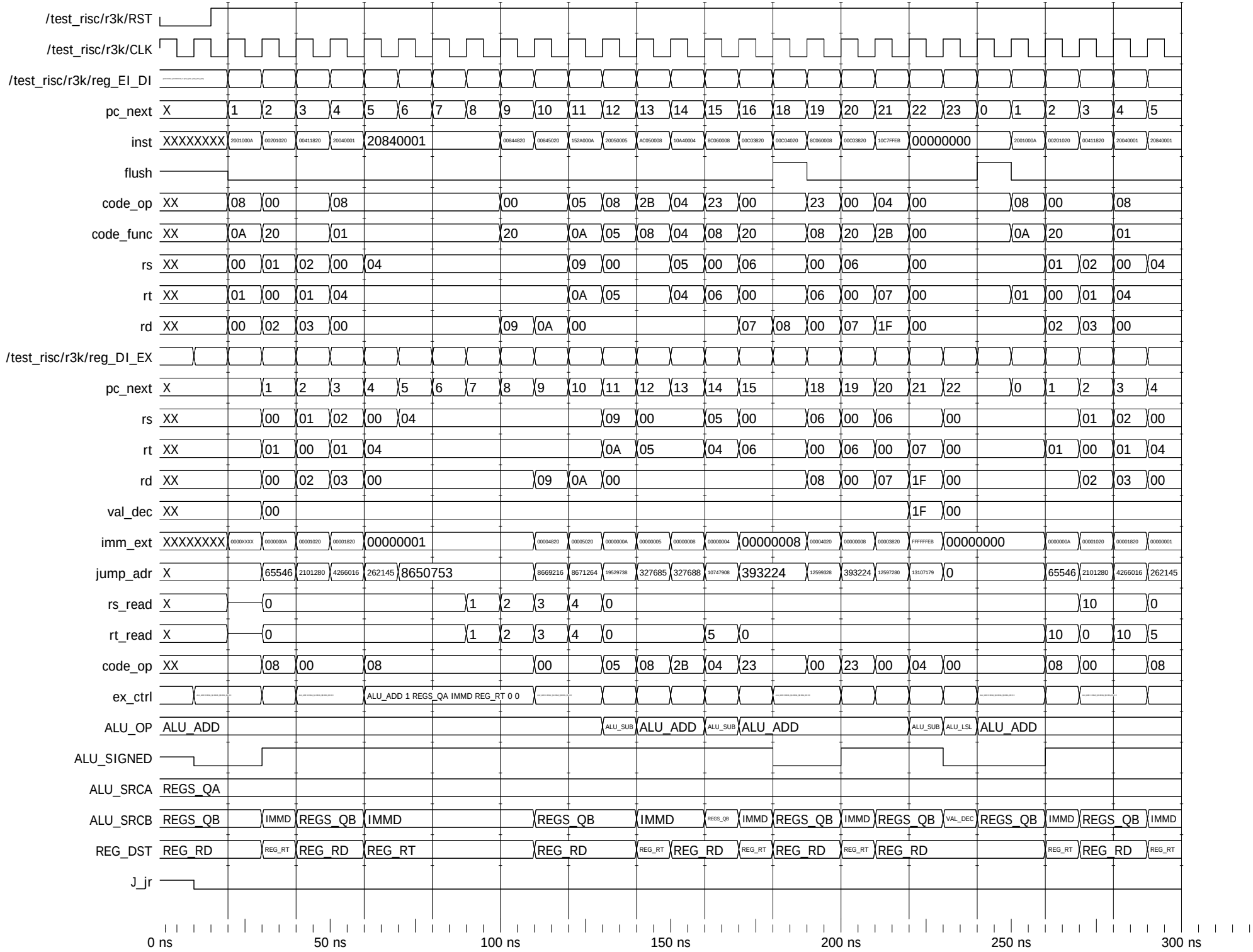

BRANCH
B_TYPE
B_beq
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_MEM_ER
pc_next
mem_Q
ual_S
reg_dst
er_ctrl
/test_risc/r3k/reg_PC
/test_risc/r3k/ei_pc_next
/test_risc/r3k/ei_inst
/test_risc/r3k/di_qa
/test_risc/r3k/di_qb
/test_risc/r3k/di_imm_ext
/test_risc/r3k/di_ctrl_di
SIGNED_EXT
J_j
J_jal
/test_risc/r3k/di_ctrl_ex
ALU_OP
ALU_SIGNED
ALU_SRCA
ALU_SRCB
REG_DST
J_jr
J_jalr
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

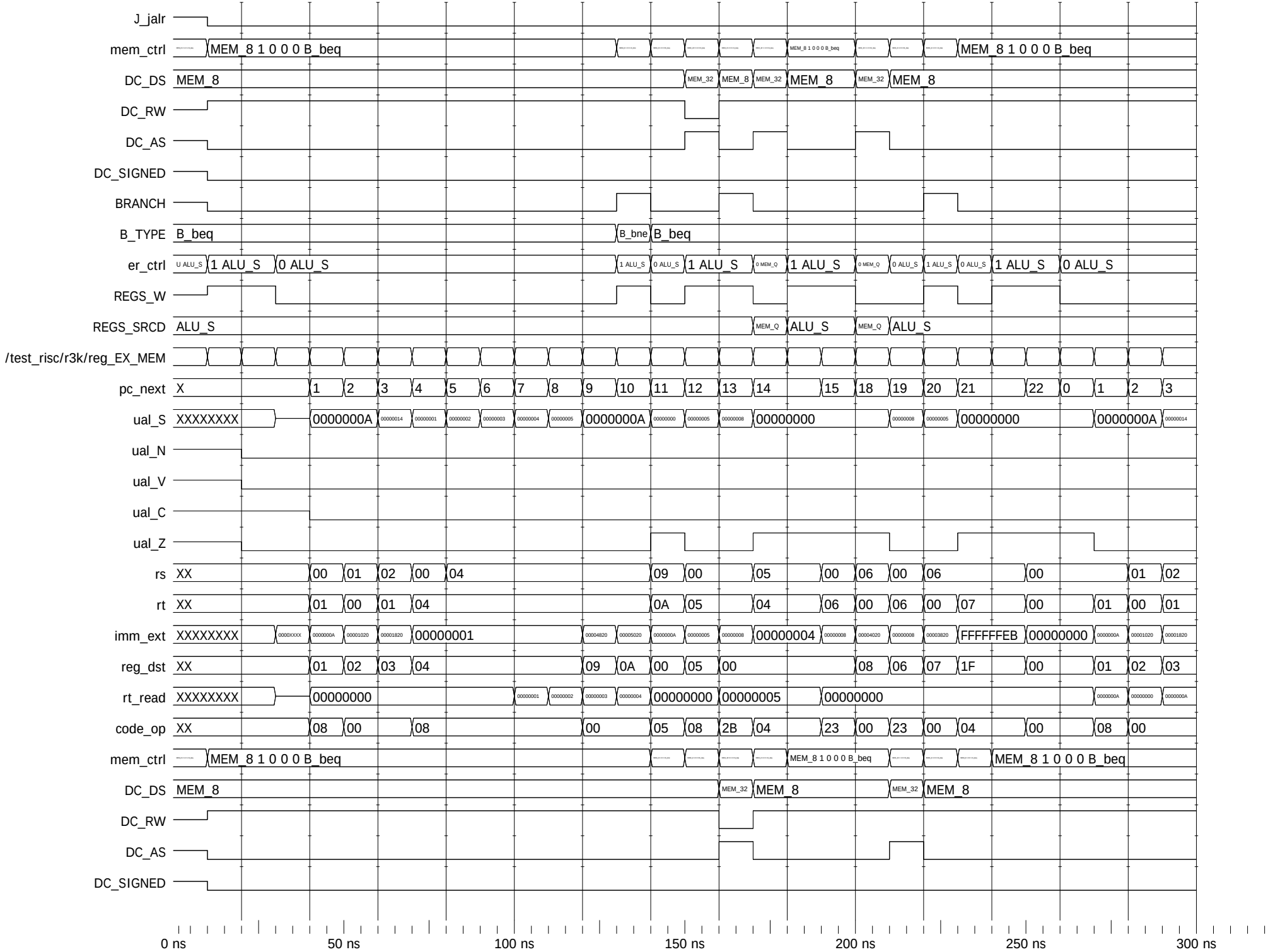

/test_risc/r3k/di_ctrl_mem
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
/test_risc/r3k/di_ctrl_er
REGS_W
REGS_SRCD
/test_risc/r3k/di_flush
/test_risc/r3k/ex_ALU_Z
/test_risc/r3k/ex_ALU_N
/test_risc/r3k/ex_ALU_V
/test_risc/r3k/ex_ALU_C
/test_risc/r3k/ex_ALU_S
/test_risc/r3k/ex_ALU_A
/test_risc/r3k/ex_ALU_B
/test_risc/r3k/ex_ALU_XA
/test_risc/r3k/ex_ALU_XB
/test_risc/r3k/ex_mem_data
/test_risc/r3k/ex_reg_dst
/test_risc/r3k/ex_MUX_mem_data
/test_risc/r3k/ex_MUX_ALU_XA
/test_risc/r3k/ex_MUX_ALU_XB
/test_risc/r3k/ex_pc_branch
/test_risc/r3k/ex_flush
/test_risc/r3k/MEM_mem_Q
/test_risc/r3k/mem_branch
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

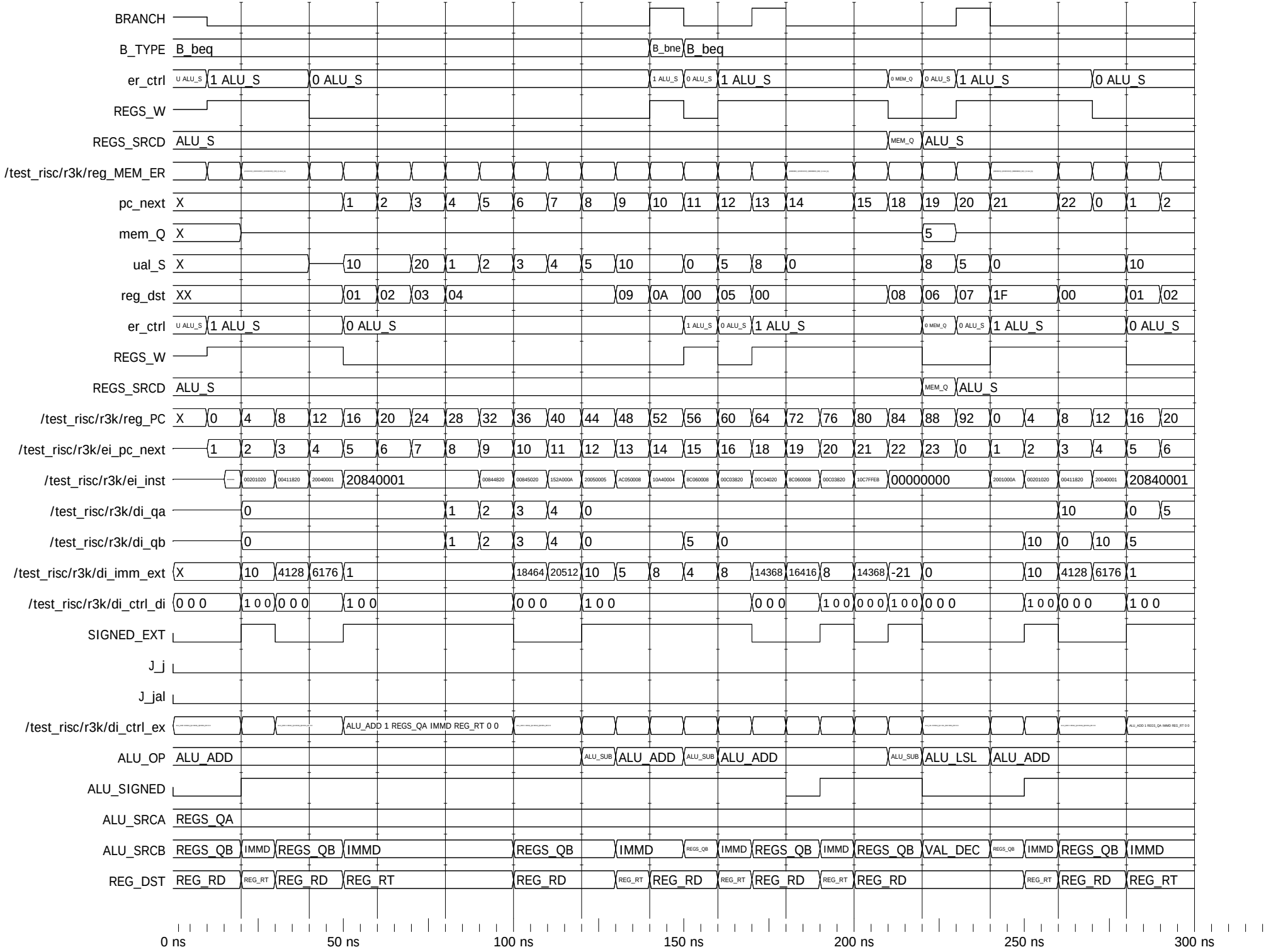

/test_risc/r3k/mem_b_type
/test_risc/r3k/mem_flush
/test_risc/r3k/mem_halt
/test_risc/r3k/er_regd
/test_risc/r3k/er_adrw
X
1 2 1 4 1 3 4 2 5 2 3 5 2 3 5 2 5 1 4 1 3 4 2
X
1 0 1 0 1 2 10 1 2 1 2 3 1 2 3 1 0 1 0 1 2 10 1
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

ANNEXE A6 I.3

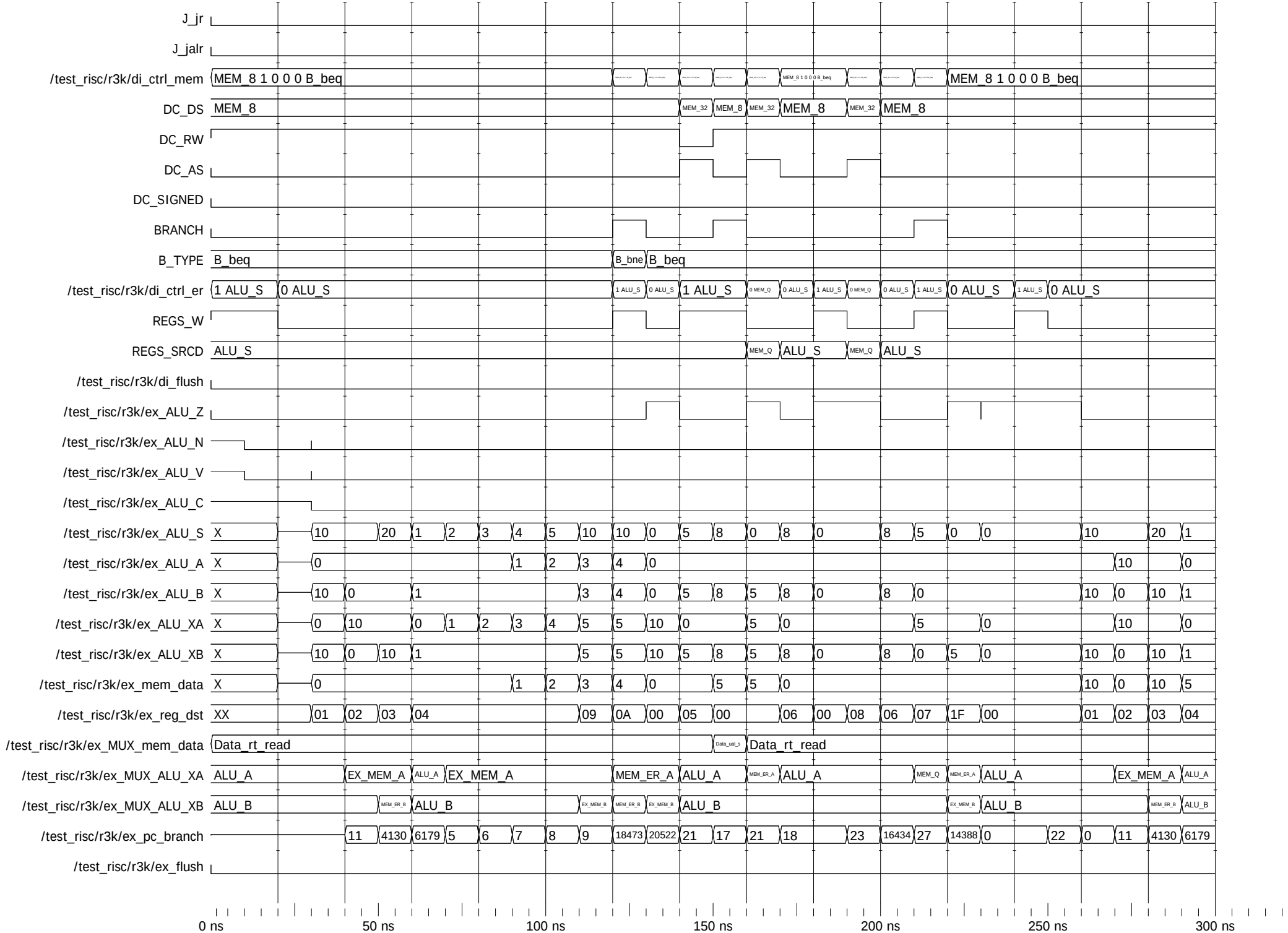
Résultats de la simulation des fichiers *V6cpu_package.2.vhd* et *V6risc.0.vhd*
simulés avec fichier *test_aleas_plus.asm*

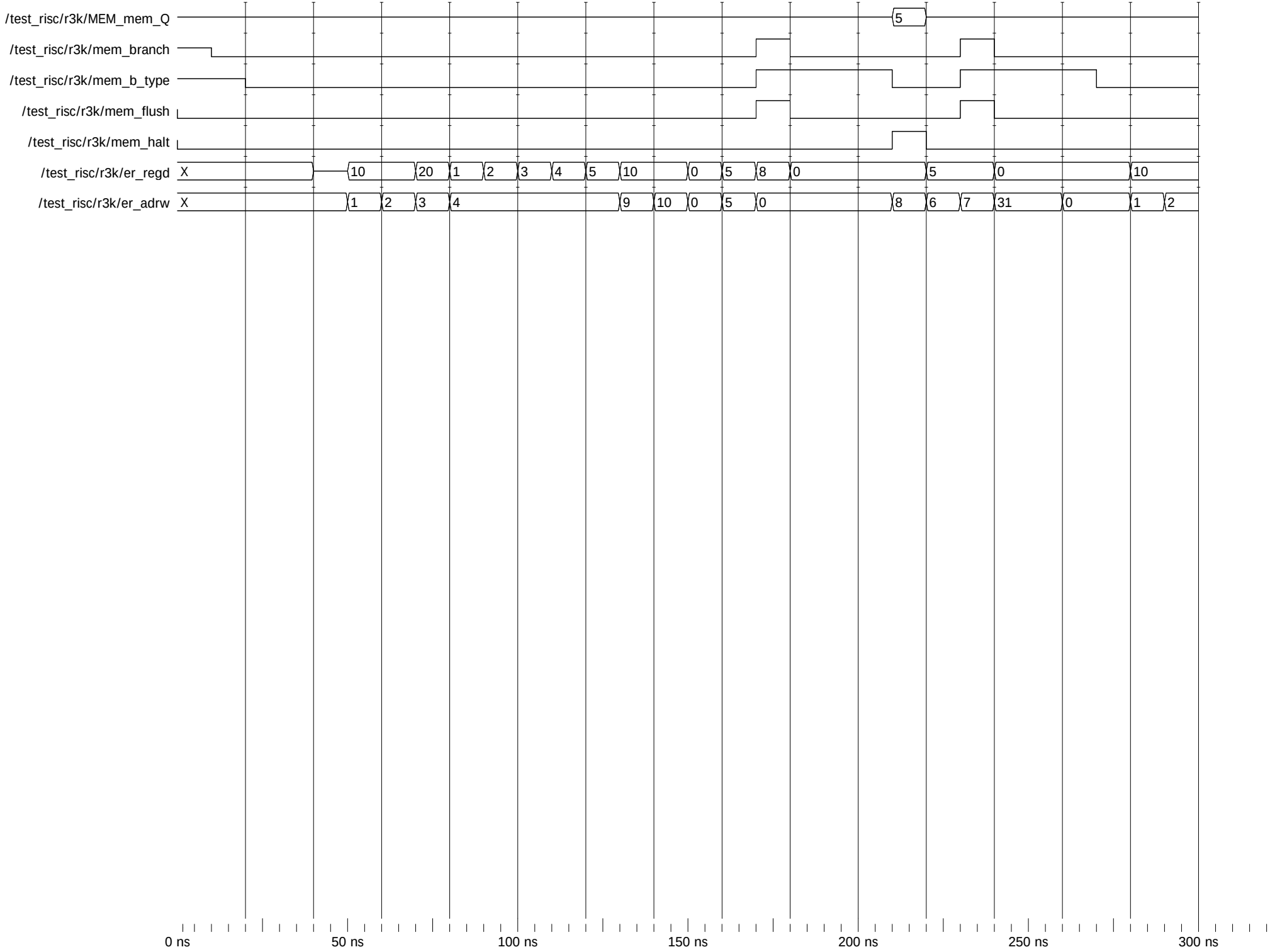

/test_risc/r3k/RST
/test_risc/r3k/CLK
/test_risc/r3k/reg_EI_DI
pc_next
inst
flush
code_op
code_func
rs
rt
rd
/test_risc/r3k/reg_DI_EX
pc_next
rs
rt
rd
val_dec
imm_ext
jump_adr
rs_read
rt_read
code_op
ex_ctrl
ALU_OP
ALU_SIGNED
ALU_SRCA
ALU_SRCB
REG_DST
J_jr
ALU_ADD 1 REGS_QA IMMD REG_RT 0 0
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


J_jalr
mem_ctrl
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_EX_MEM
pc_next
ual_S
ual_N
ual_V
ual_C
ual_Z
rs
rt
imm_ext
reg_dst
rt_read
code_op
mem_ctrl
DC_DS
DC_RW
DC_AS
DC_SIGNED
MEM_8 1 0 0 0 B_beq
MEM_8
MEM_32
B_beq
B_bne
1 ALU_S
0 ALU_S
ALU_S
MEM_Q
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns



Entity:test_risc Architecture:behavior Date: Mon Feb 18 10:04:10 PM Romance Standard Time 2013 Row: 1 Page: 3


J_jr
J_jalr
/test_risc/r3k/di_ctrl_mem
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
/test_risc/r3k/di_ctrl_er
REGS_W
REGS_SRCD
/test_risc/r3k/di_flush
/test_risc/r3k/ex_ALU_Z
/test_risc/r3k/ex_ALU_N
/test_risc/r3k/ex_ALU_V
/test_risc/r3k/ex_ALU_C
/test_risc/r3k/ex_ALU_S
/test_risc/r3k/ex_ALU_A
/test_risc/r3k/ex_ALU_B
/test_risc/r3k/ex_ALU_XA
/test_risc/r3k/ex_ALU_XB
/test_risc/r3k/ex_mem_data
/test_risc/r3k/ex_reg_dst
/test_risc/r3k/ex_MUX_mem_data
/test_risc/r3k/ex_MUX_ALU_XA
/test_risc/r3k/ex_MUX_ALU_XB
/test_risc/r3k/ex_pc_branch
/test_risc/r3k/ex_flush
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns



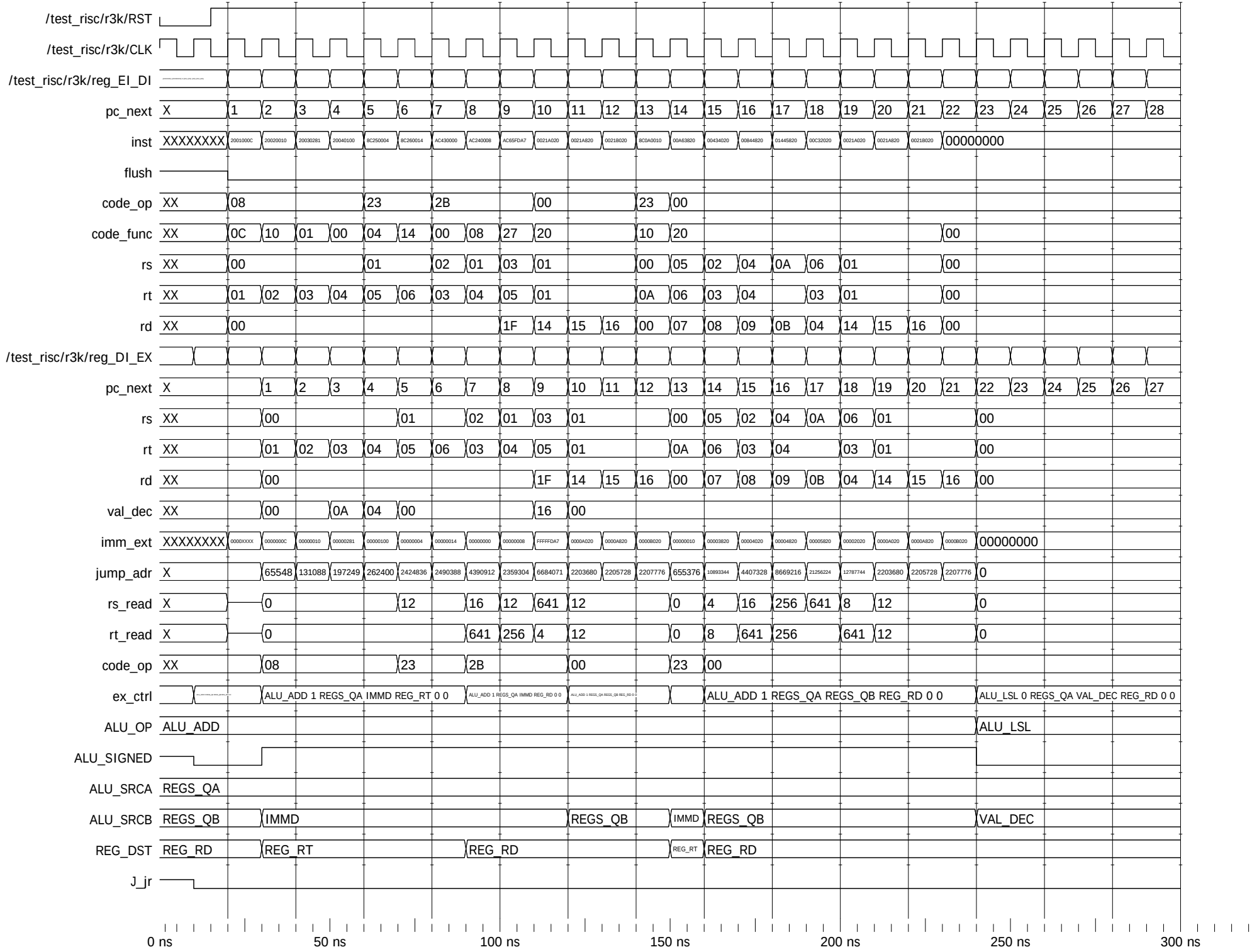
Entity:test_risc Architecture:behavior Date: Mon Feb 18 10:04:10 PM Romance Standard Time 2013 Row: 1 Page: 5

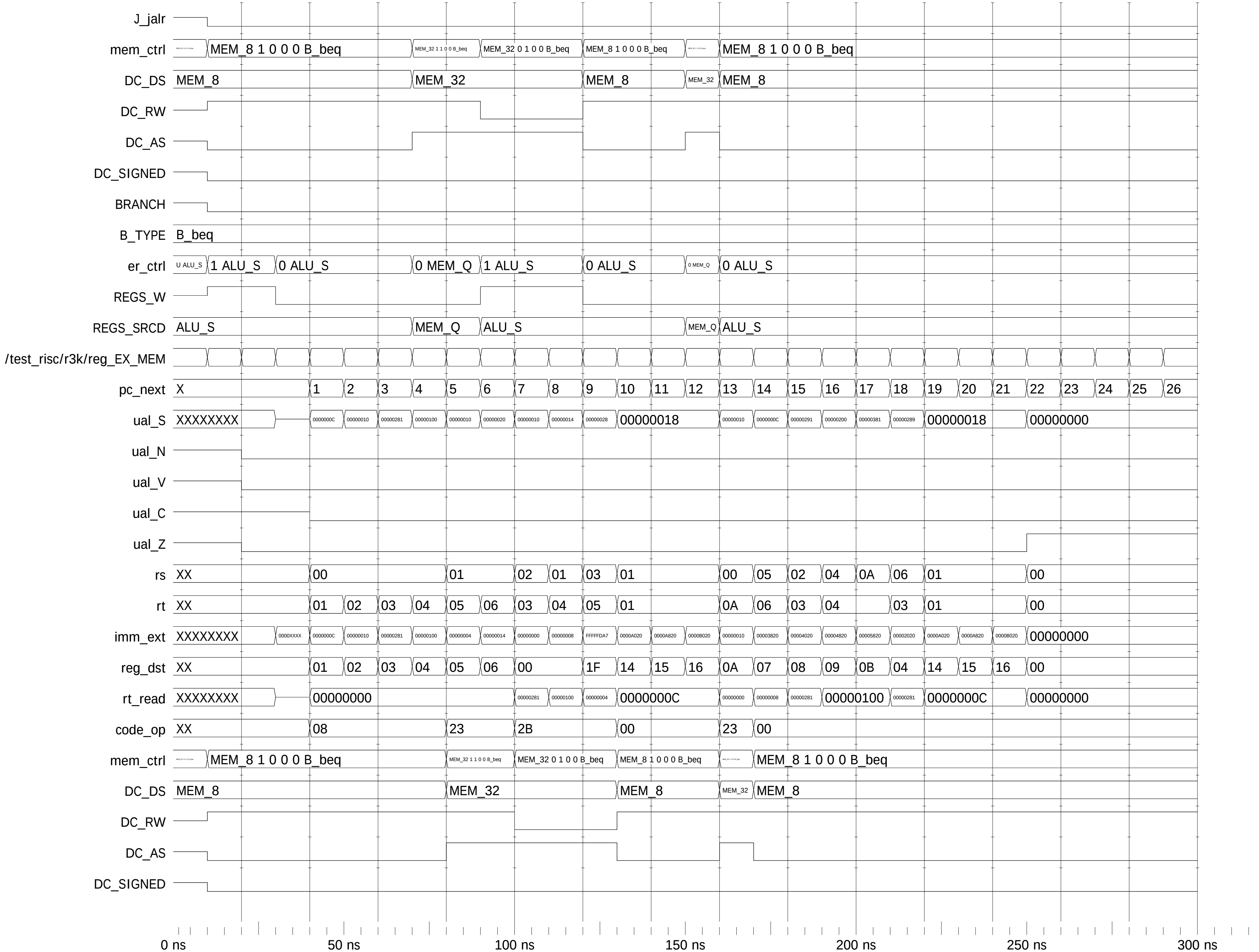
ANNEXE A6 II

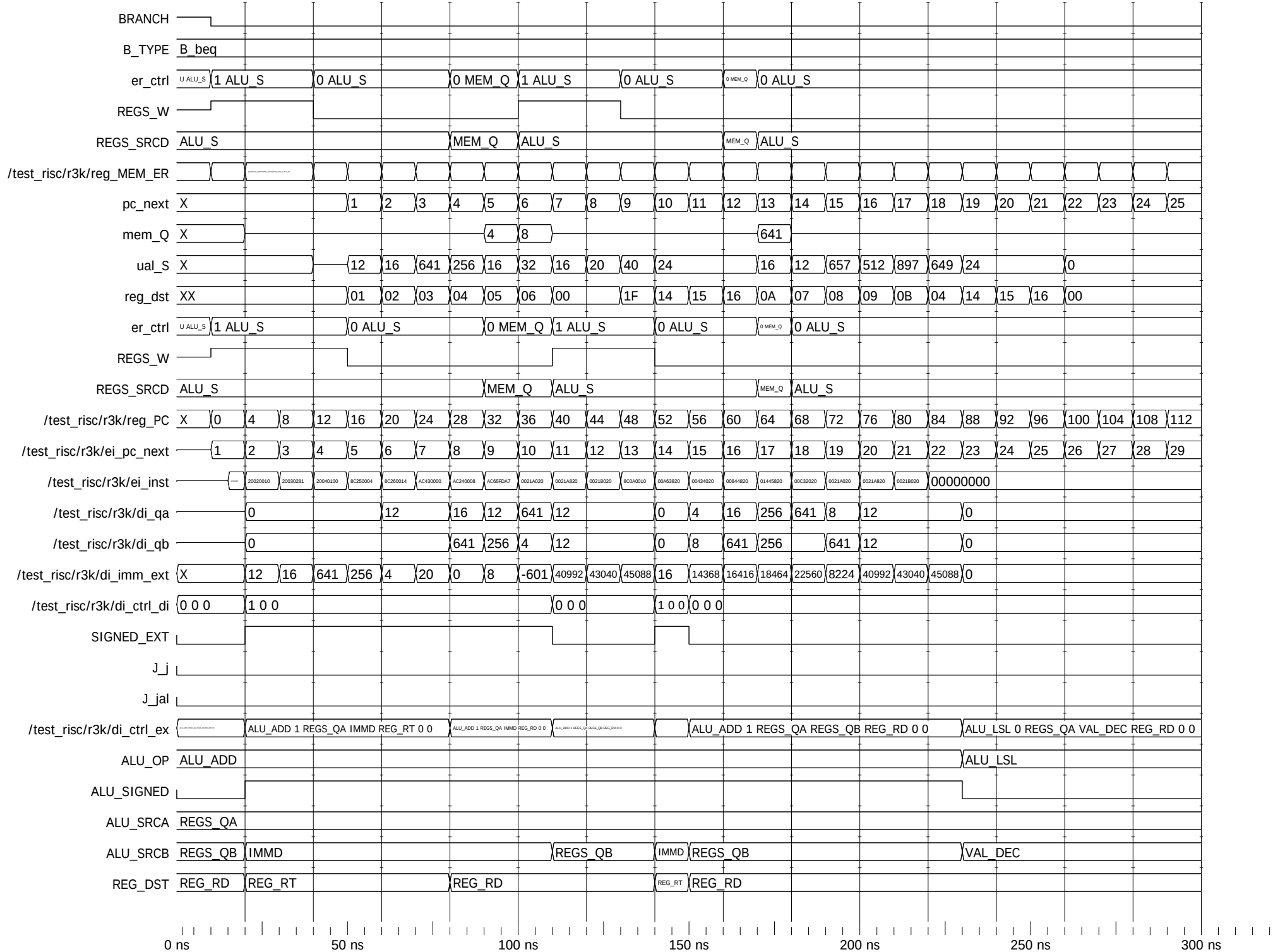
Résultats de simulation des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*

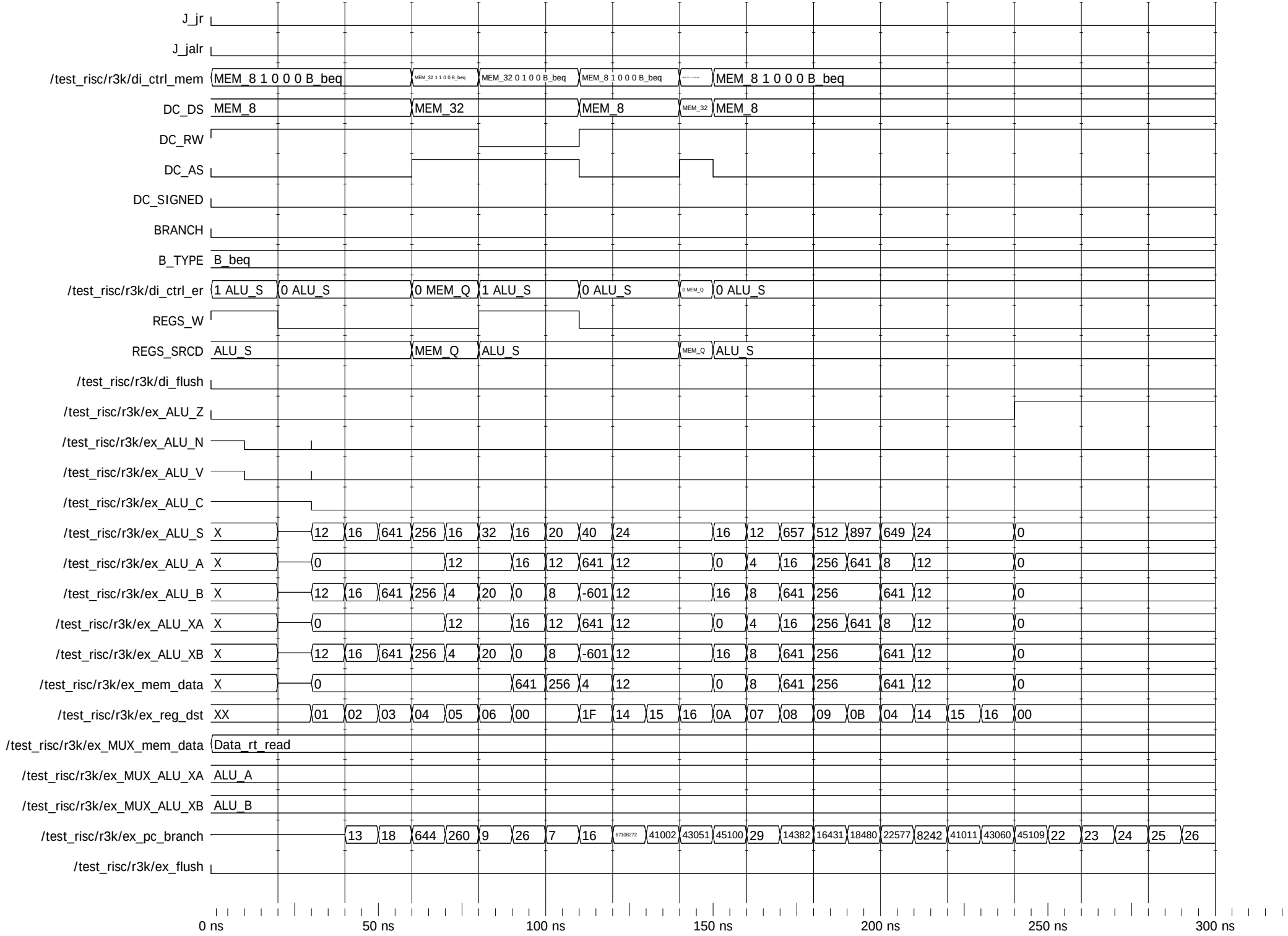
ANNEXE A6 II.1

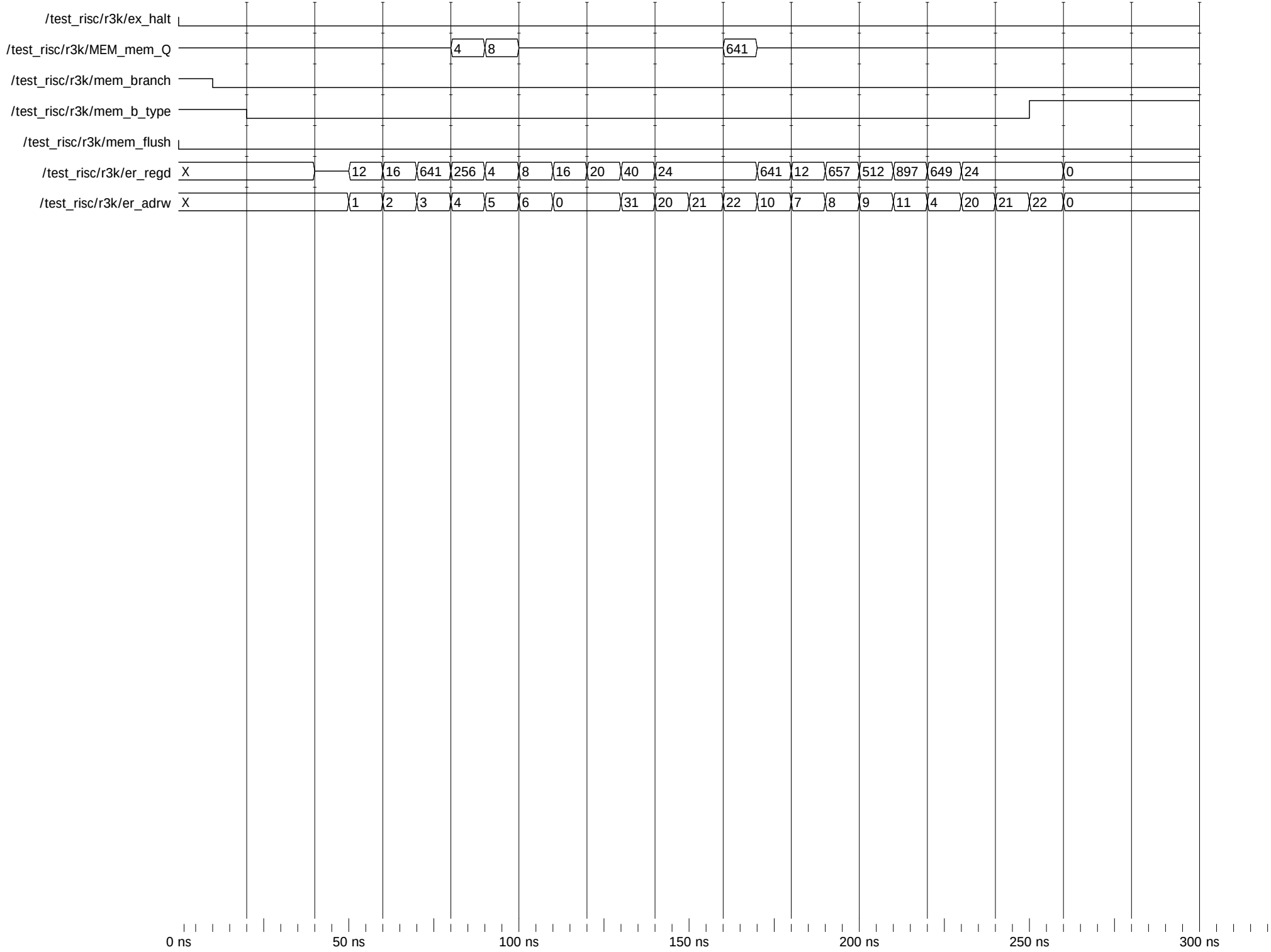
Résultats de la simulation des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*
simulés avec fichier *testadd.txt*


/test_risc/r3k/RST
/test_risc/r3k/CLK
/test_risc/r3k/reg_EI_DI
pc_next
inst
flush
code_op
code_func
rs
rt
rd
/test_risc/r3k/reg_DI_EX
pc_next
rs
rt
rd
val_dec
imm_ext
jump_adr
rs_read
rt_read
code_op
ex_ctrl
ALU_OP
ALU_SIGNED
ALU_SRCA
ALU_SRCB
REG_DST
J_jr
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


J_jalr
mem_ctrl
MEM_8 1 0 0 0 B_beq
MEM_32 0 1 0 0 B_beq
DC_DS
MEM_8
MEM_32
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
B_beq
er_ctrl
1 ALU_S
0 ALU_S
0 MEM_Q
REGS_W
REGS_SRCD
ALU_S
MEM_Q
/test_risc/r3k/reg_EX_MEM
pc_next
X
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
ual_S
XXXXXXXX
00000018
00000000
ual_N
ual_V
ual_C
ual_Z
rs
XX 00 01 02 01 03 01 00 05 02 04 0A 06 01 00
rt
XX 01 02 03 04 05 06 03 04 05 01 0A 06 03 04 03 01 00
imm_ext
XXXXXXXX
00000000
reg_dst
XX 01 02 03 04 05 06 00 1F 14 15 16 0A 07 08 09 0B 04 14 15 16 00
rt_read
XXXXXXXX
00000000
0000000C
00000100
code_op
XX 08 23 2B 00 23 00
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

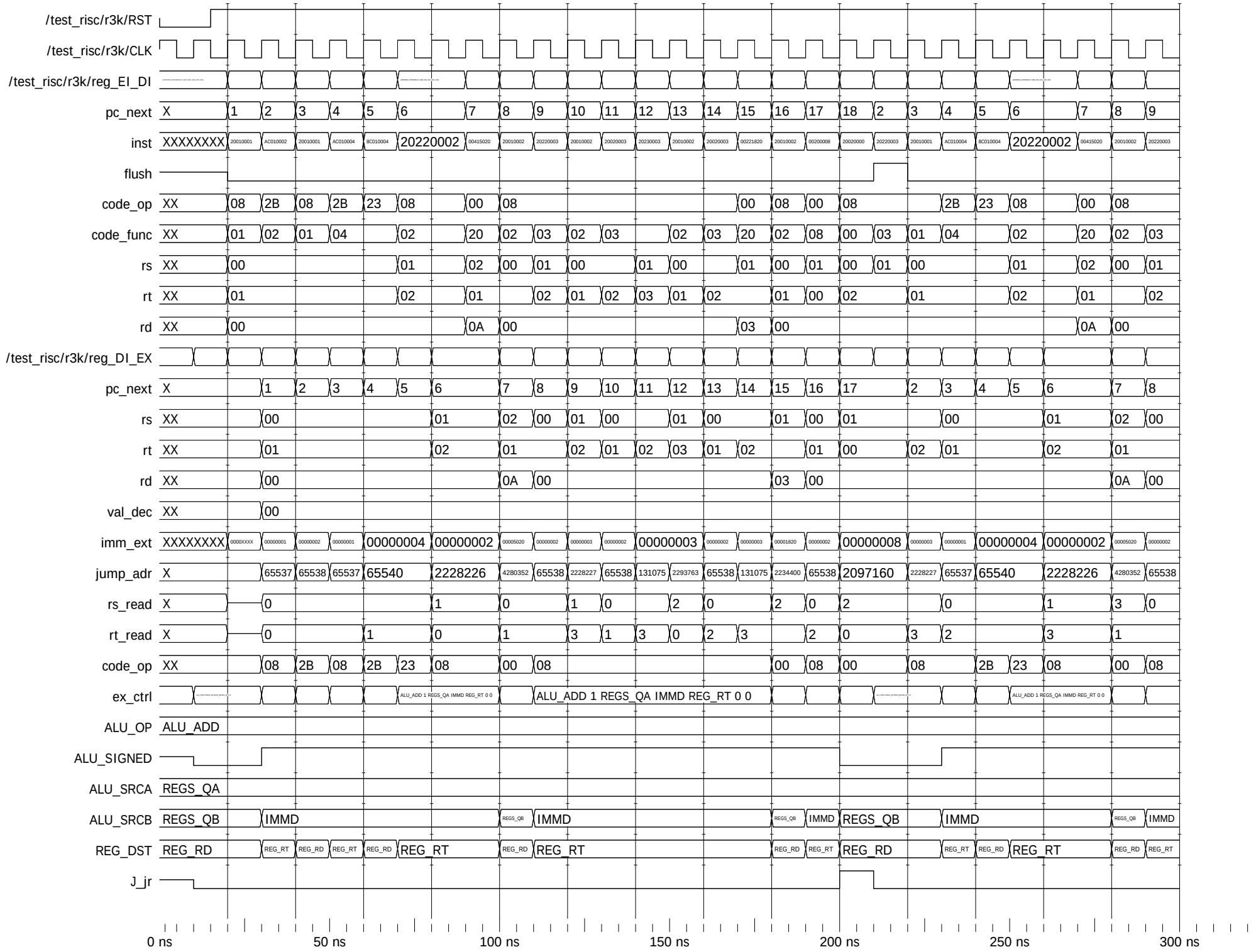

BRANCH
B_TYPE
B_beq
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_MEM_ER
pc_next
mem_Q
ual_S
reg_dst
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_PC
/test_risc/r3k/ei_pc_next
/test_risc/r3k/ei_inst
/test_risc/r3k/di_qa
/test_risc/r3k/di_qb
/test_risc/r3k/di_imm_ext
/test_risc/r3k/di_ctrl_di
SIGNED_EXT
J_j
J_jal
/test_risc/r3k/di_ctrl_ex
ALU_OP
ALU_SIGNED
ALU_SRCA
ALU_SRCB
REG_DST
ALU_ADD 1 REGS_QA IMMD REG_RT 0 0
ALU_ADD 1 REGS_QA IMMD REG_RD 0 0
ALU_ADD 1 REGS_QA REGS_QB REG_RD 0 0
ALU_LSL 0 REGS_QA VAL_DEC REG_RD 0 0
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

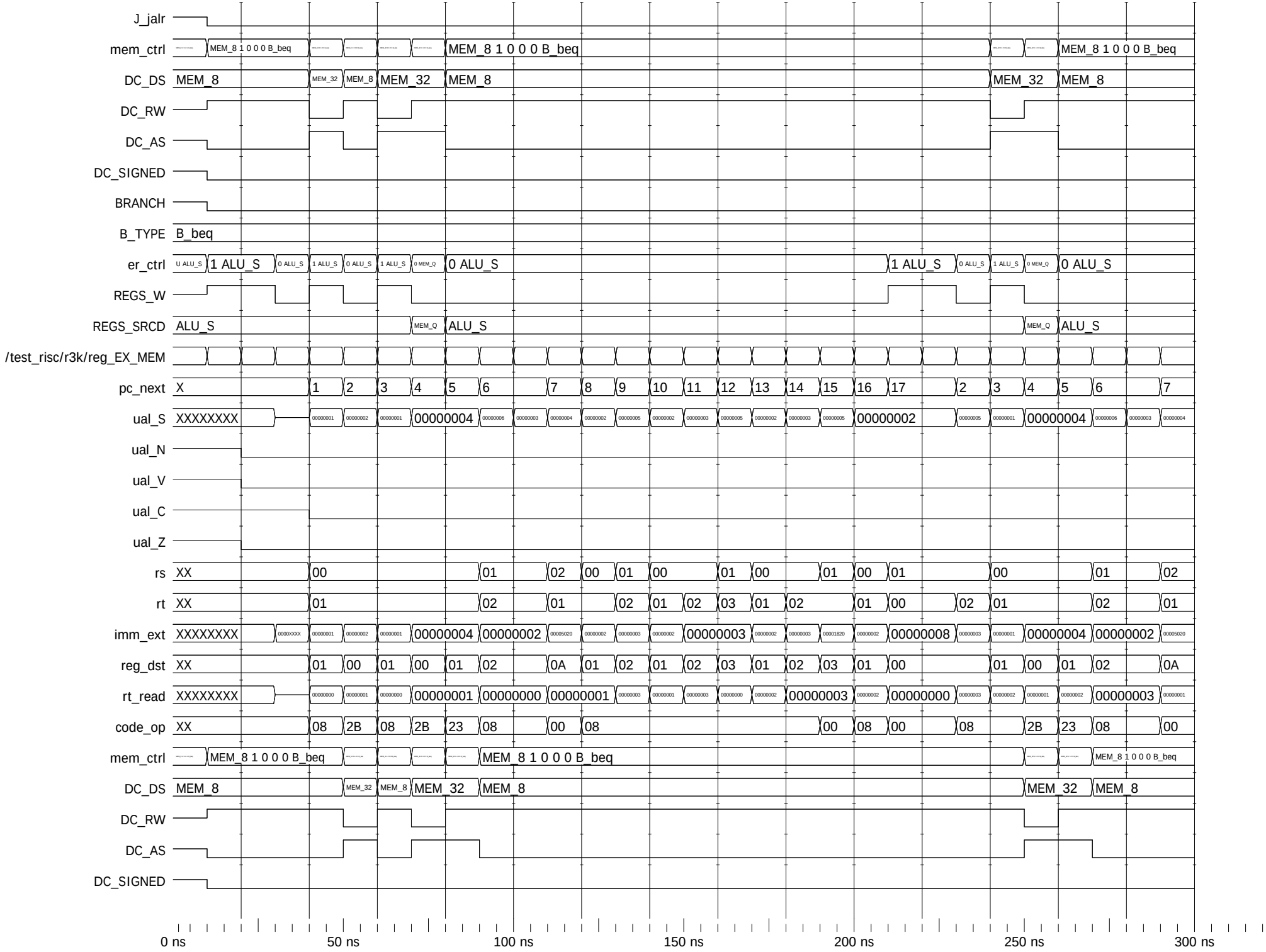

J_jr
J_jalr
/test_risc/r3k/di_ctrl_mem
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
/test_risc/r3k/di_ctrl_er
REGS_W
REGS_SRCD
/test_risc/r3k/di_flush
/test_risc/r3k/ex_ALU_Z
/test_risc/r3k/ex_ALU_N
/test_risc/r3k/ex_ALU_V
/test_risc/r3k/ex_ALU_C
/test_risc/r3k/ex_ALU_S
/test_risc/r3k/ex_ALU_A
/test_risc/r3k/ex_ALU_B
/test_risc/r3k/ex_ALU_XA
/test_risc/r3k/ex_ALU_XB
/test_risc/r3k/ex_mem_data
/test_risc/r3k/ex_reg_dst
/test_risc/r3k/ex_MUX_mem_data
/test_risc/r3k/ex_MUX_ALU_XA
/test_risc/r3k/ex_MUX_ALU_XB
/test_risc/r3k/ex_pc_branch
/test_risc/r3k/ex_flush
MEM_8 1 0 0 0 B_beq
MEM_32 1 1 0 0 B_beq
MEM_32 0 1 0 0 B_beq
MEM_8 1 0 0 0 B_beq
MEM_8 1 0 0 0 B_beq
MEM_8
MEM_32
MEM_8
MEM_32
MEM_8
B_beq
1 ALU_S
0 ALU_S
0 MEM_Q
1 ALU_S
0 ALU_S
0 MEM_Q
0 ALU_S
ALU_S
MEM_Q
ALU_S
MEM_Q
ALU_S
X 12 16 641 256 16 32 16 20 40 24 16 12 657 512 897 649 24 0
X 0 12 16 12 641 12 0 4 16 256 641 8 12 0
X 12 16 641 256 4 20 0 8 -601 12 16 8 641 256 641 12 0
X 0 12 16 12 641 12 0 4 16 256 641 8 12 0
X 12 16 641 256 4 20 0 8 -601 12 16 8 641 256 641 12 0
X 0 641 256 4 12 0 8 641 256 641 12 0
XX 01 02 03 04 05 06 00 1F 14 15 16 0A 07 08 09 0B 04 14 15 16 00
Data_rt_read
ALU_A
ALU_B
13 18 644 260 9 26 7 16 67108272 41002 43051 45100 29 14382 16431 18480 22577 8242 41011 43060 45109 22 23 24 25 26
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

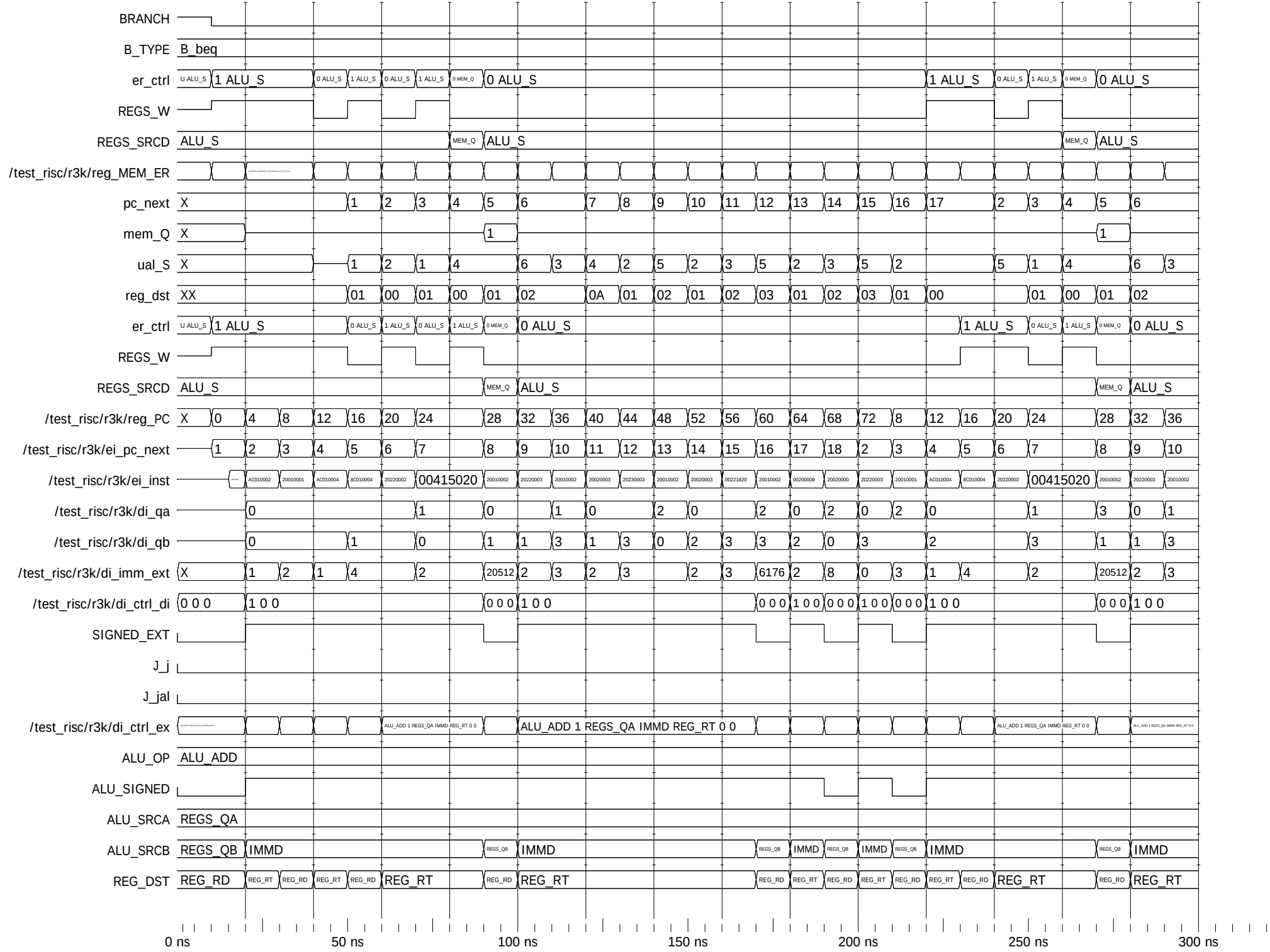

/test_risc/r3k/ex_halt
/test_risc/r3k/MEM_mem_Q
/test_risc/r3k/mem_branch
/test_risc/r3k/mem_b_type
/test_risc/r3k/mem_flush
/test_risc/r3k/er_regd
/test_risc/r3k/er_adrw
X 12 16 641 256 4 8 16 20 40 24 641 12 657 512 897 649 24 0
X 1 2 3 4 5 6 0 31 20 21 22 10 7 8 9 11 4 20 21 22 0
4 8 641
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

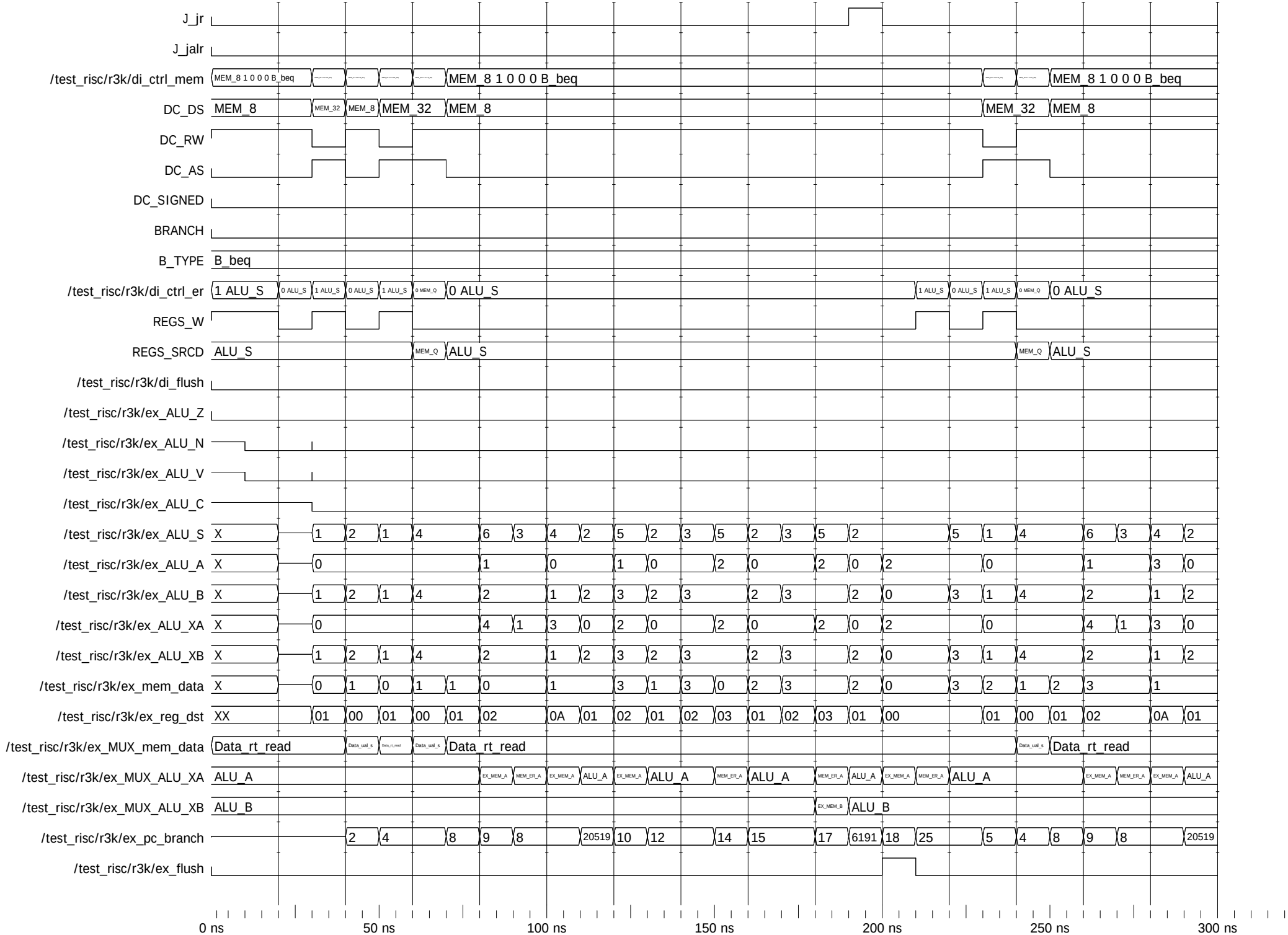
ANNEXE A6 II.2

Résultats de la simulation des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*
simulés avec fichier *bench_1.txt*

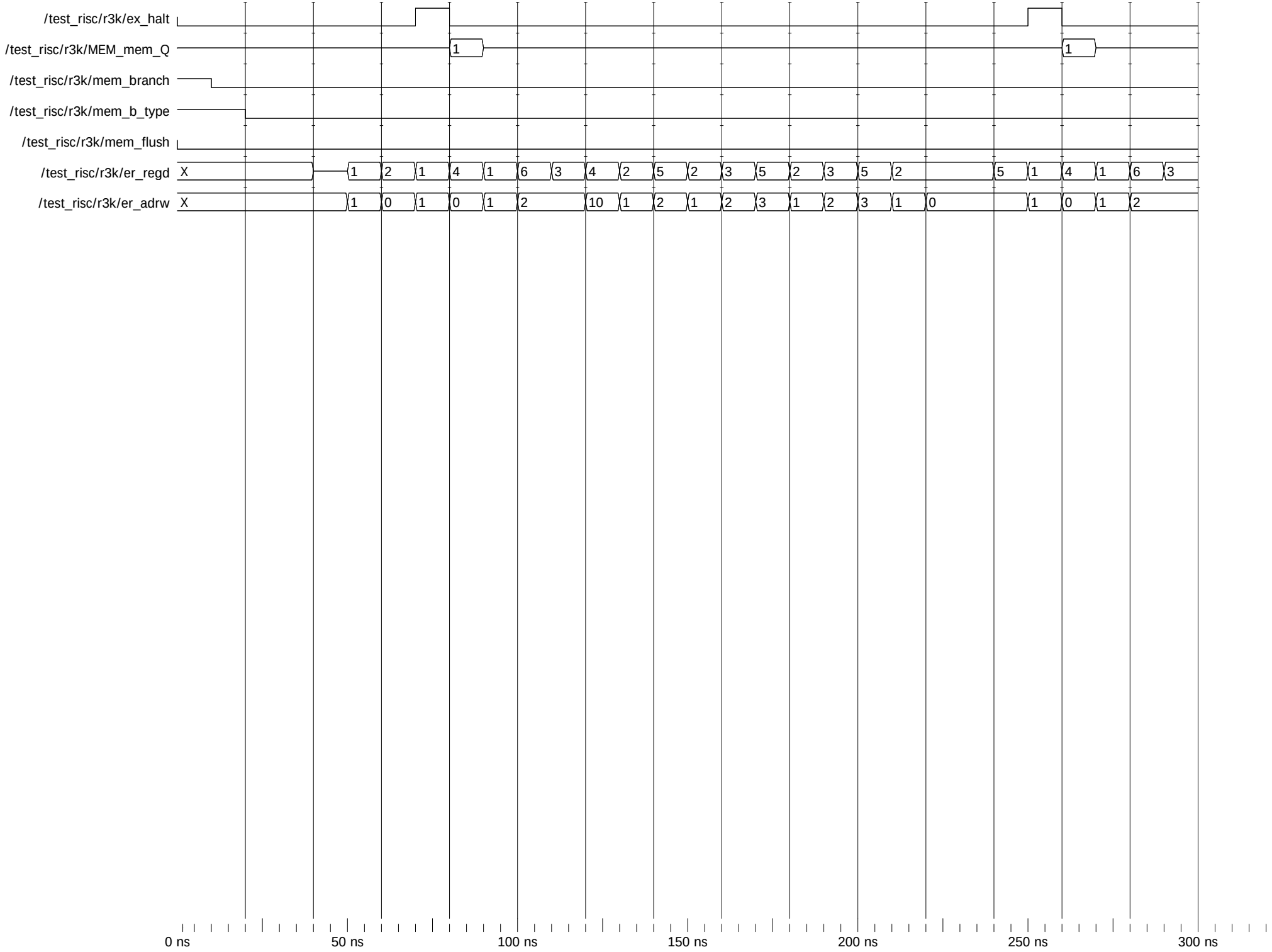

/test_risc/r3k/RST
/test_risc/r3k/CLK
/test_risc/r3k/reg_EI_DI
pc_next
inst
flush
code_op
code_func
rs
rt
rd
/test_risc/r3k/reg_DI_EX
pc_next
rs
rt
rd
val_dec
imm_ext
jump_adr
rs_read
rt_read
code_op
ex_ctrl
ALU_OP
ALU_SIGNED
ALU_SRCA
ALU_SRCB
REG_DST
J_jr
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


J_jalr
mem_ctrl
DC_DS
DC_RW
DC_AS
DC_SIGNED
BRANCH
B_TYPE
er_ctrl
REGS_W
REGS_SRCD
/test_risc/r3k/reg_EX_MEM
pc_next
ual_S
ual_N
ual_V
ual_C
ual_Z
rs
rt
imm_ext
reg_dst
rt_read
code_op
mem_ctrl
DC_DS
DC_RW
DC_AS
DC_SIGNED
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


BRANCH
B_TYPE
B_beq
er_ctrl
REGS_W
REGS_SRCD
ALU_S
MEM_Q
/test_risc/r3k/reg_MEM_ER
pc_next
mem_Q
ual_S
reg_dst
/test_risc/r3k/reg_PC
/test_risc/r3k/ei_pc_next
/test_risc/r3k/ei_inst
/test_risc/r3k/di_qa
/test_risc/r3k/di_qb
/test_risc/r3k/di_imm_ext
/test_risc/r3k/di_ctrl_di
SIGNED_EXT
J_j
J_jal
/test_risc/r3k/di_ctrl_ex
ALU_OP
ALU_ADD
ALU_SIGNED
ALU_SRCA
REGS_QA
ALU_SRCB
REGS_QB
IMMD
REG_DST
REG_RD
REG_RT
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

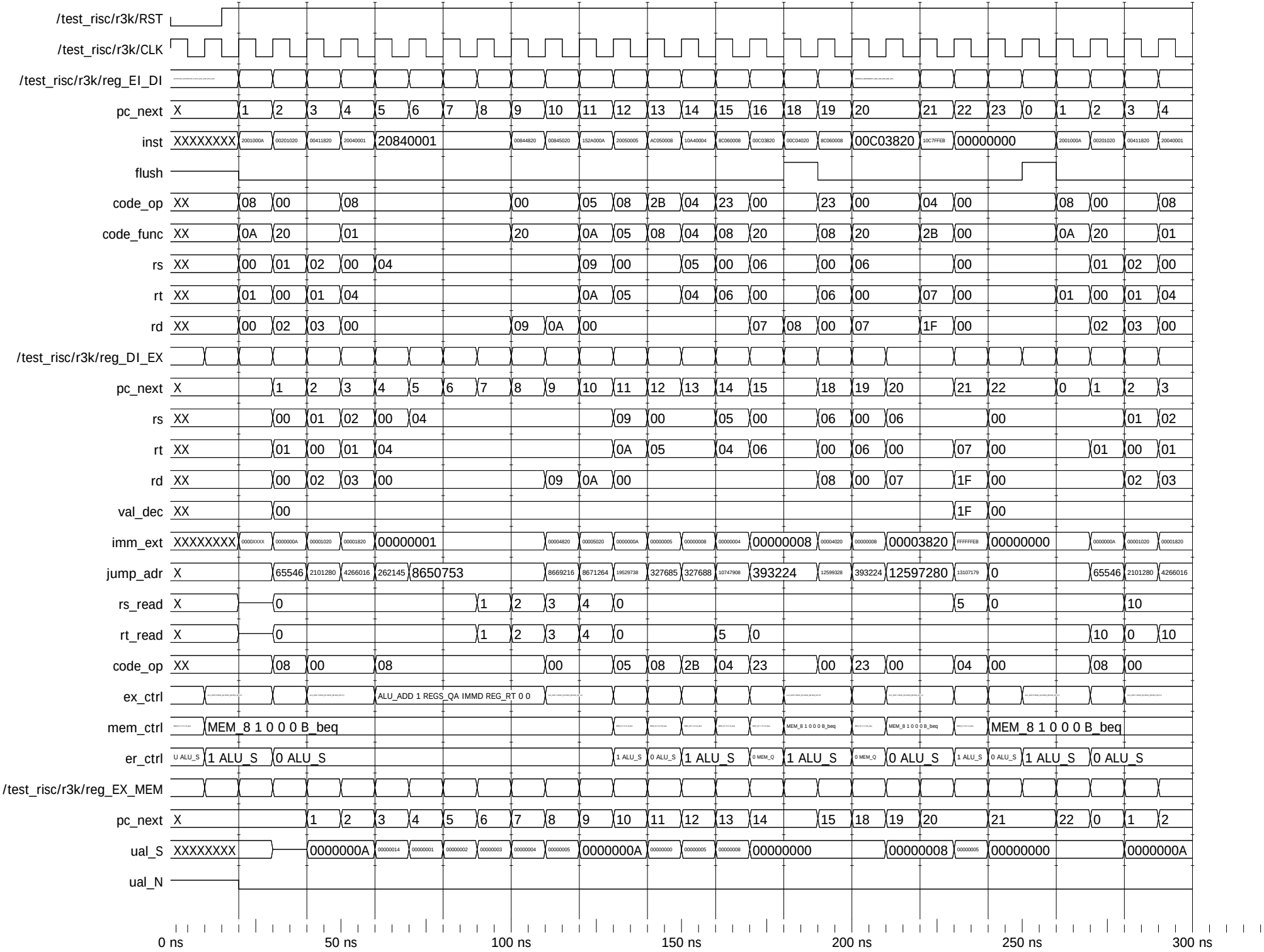


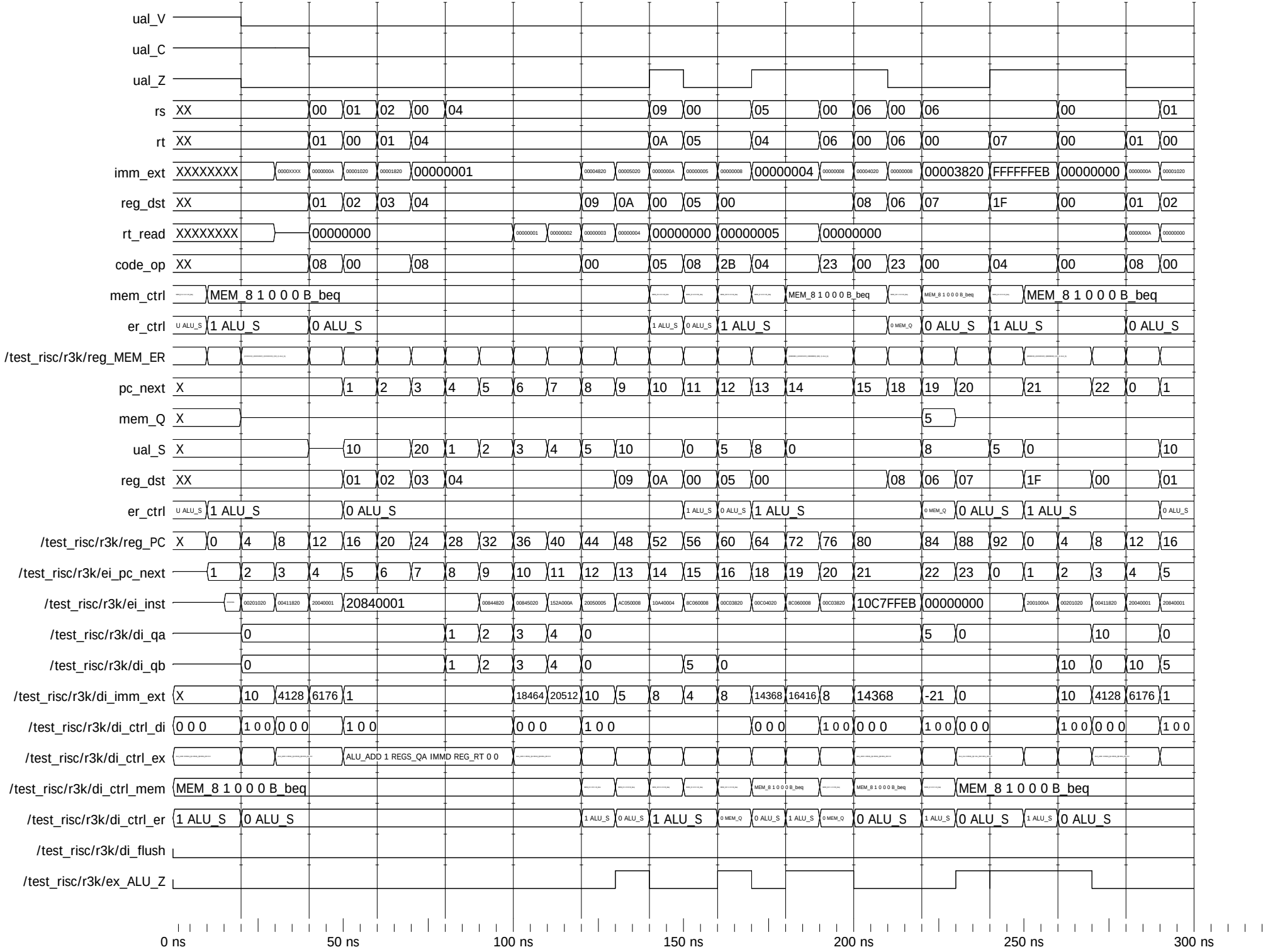
Entity:test_risc Architecture:behavior Date: Mon Feb 18 10:19:19 PM Romance Standard Time 2013 Row: 1 Page: 4

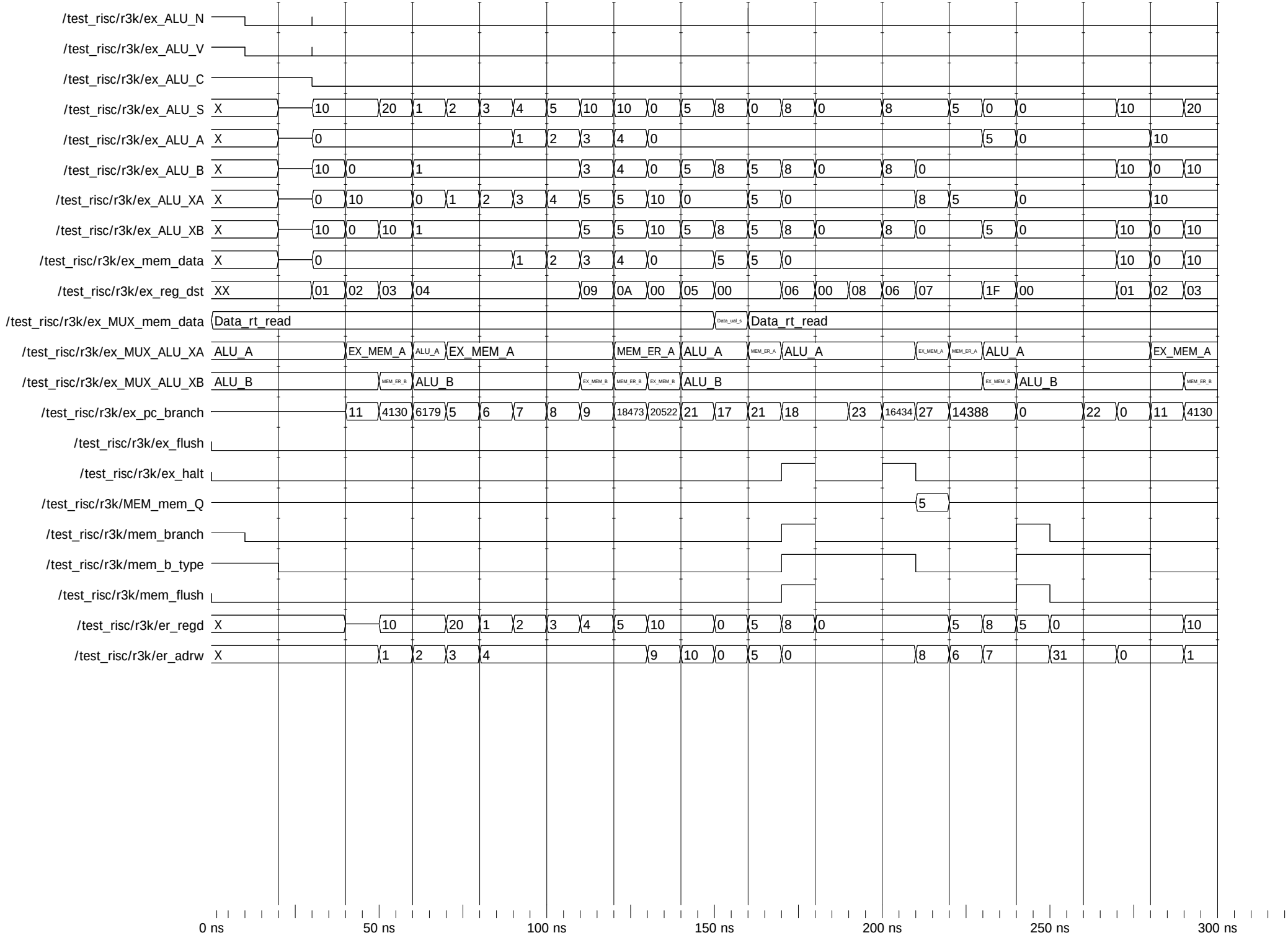

/test_risc/r3k/ex_halt
/test_risc/r3k/MEM_mem_Q
/test_risc/r3k/mem_branch
/test_risc/r3k/mem_b_type
/test_risc/r3k/mem_flush
/test_risc/r3k/er_regd
/test_risc/r3k/er_adrw
X 1 2 1 4 1 6 3 4 2 5 2 3 5 2 3 5 2 5 1 4 1 6 3
X 1 0 1 0 1 2 10 1 2 1 2 3 1 2 3 1 0 1 0 1 2
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

ANNEXE A6 II.3

Résultats de la simulation des fichiers *V5cpu_package.2.vhd* et *V5risc.0.vhd*
simulés avec fichier *test_aleas_plus.asm*


/test_risc/r3k/RST
/test_risc/r3k/CLK
/test_risc/r3k/reg_EI_DI
pc_next
inst
flush
code_op
code_func
rs
rt
rd
/test_risc/r3k/reg_DI_EX
pc_next
rs
rt
rd
val_dec
imm_ext
jump_adr
rs_read
rt_read
code_op
ex_ctrl
mem_ctrl
er_ctrl
/test_risc/r3k/reg_EX_MEM
pc_next
ual_S
ual_N
ALU_ADD 1 REGS_QA IMMD REG_RT 0 0
MEM_8 1 0 0 0 B_beq
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


ual_V
ual_C
ual_Z
rs
rt
imm_ext
reg_dst
rt_read
code_op
mem_ctrl
er_ctrl
/test_risc/r3k/reg_MEM_ER
pc_next
mem_Q
ual_S
reg_dst
er_ctrl
/test_risc/r3k/reg_PC
/test_risc/r3k/ei_pc_next
/test_risc/r3k/ei_inst
/test_risc/r3k/di_qa
/test_risc/r3k/di_qb
/test_risc/r3k/di_imm_ext
/test_risc/r3k/di_ctrl_di
/test_risc/r3k/di_ctrl_ex
/test_risc/r3k/di_ctrl_mem
/test_risc/r3k/di_ctrl_er
/test_risc/r3k/di_flush
/test_risc/r3k/ex_ALU_Z
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns



Entity:test_risc Architecture:behavior Date: Mon Feb 18 10:13:21 PM Romance Standard Time 2013 Row: 1 Page: 3

ANNEXE A7

Présentation de l'outil Synopsys Design Compiler pour la synthèse de circuits à partir de code VHDL

A mettre dans le rapport final :

- **Réaliser un script qui contient toutes les commandes que vous avez generé à travers le menu (voir log ou history)**

On va se placer dans le répertoire VHDL/SYNTHESE.
On lance l'environnement AMS qui installe le chemin d'accès vers le design kit AMS :

> **ams370** → **AMS370+CADENCE51)**

On lance l'environnement SYNOPSYS avec la commande :

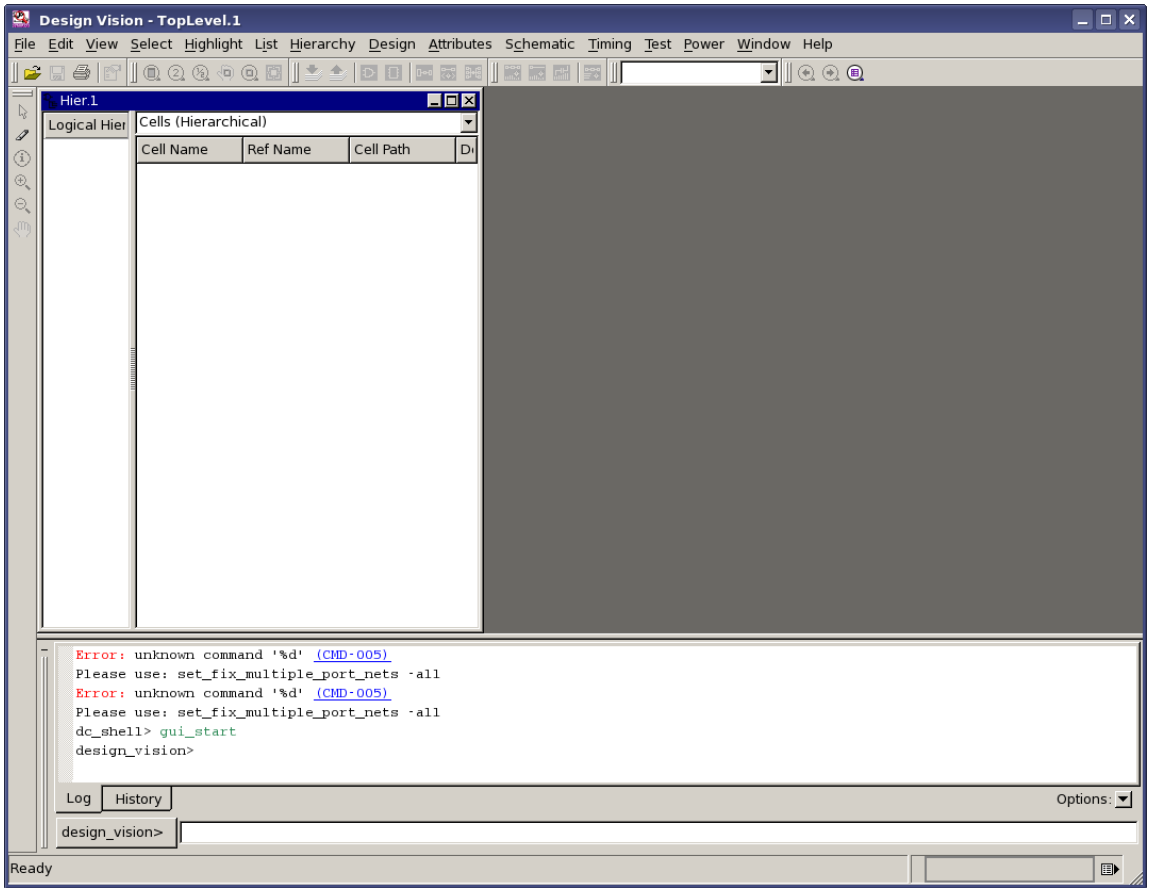
> **synopsys** → **synopsys11**

Ensuite on lance DESIGN COMPILER qui est l'outil de synthèse automatique des circuits à partir d'un modèle VHDL avec la commande :

design_vision-xg &

ou

dc_shell -gui



On va réaliser l'importation du fichier VHDL dont on veut faire la synthèse :

File → Read

On choisit le fichier VHDL (**counter.vhd ou ethernet.vhd)** qui se trouve dans le répertoire VHDL.

Si l'entité et l'architecture se trouve dans 2 fichiers sépares il faut faire :
File → Analyze
pour le fichier de l'entité et ensuite pour celui de l'architecture en choisissant la librairie du travail **WORK** et le **FILE FORMAT=VHDL.**
Ensuite il faut faire :
File → Elaborate
En choisissant dans la librairie WORK l'entité avec son architecture.

Ces 2 commandes (File→Analyze suivi de File→ Elaborate) sont équivalents à File→ Read .
La commande File→ Read s'utilise quand l'entité et l'architecture se trouve dans le même fichier .
Les commandes File→Analyze suivi de File→ Elaborate s'utilise quand l'entité et l'architecture se trouvent dans 2 fichiers sépares.

Donc nous venons de réaliser l'importation du fichier VHDL dont on veut réaliser la synthèse.

Il est impérative d'indiquer à l'outil de synthèse quel est l'horloge. De cette manière, l'outil de synthèse pourra ensuite faire de statistiques correctes en ce qui concerne la propagation des signaux et déterminer le chemin critique et donc la fréquence maximale de fonctionnement.

Attributes → Clock → Specify

Cliquez avec la souris sur le pin CK (le pin d'horloge), donner une périodes estimative de 10 ns (par exemple).
Cette commande n'est pas nécessaire pour la synthèse proprement dite du circuits mais pour que l'outil de synthèse soit capable de déterminer correctement les temps de propagations et donc la fréquence maximale de fonctionnement.
ATTENTION : Si cette commande n'est pas faite avant la synthèse le logiciel indiquera de chemins critiques faux.

Nous avons la possibilité d'optimiser la synthèse du circuit par rapport à sa taille ou au« timming » , On peut aussi imposer le fan-out de portes utiliser dans notre circuit .

Attributes → Design Constraints

Choisir l'optimisation voulue.

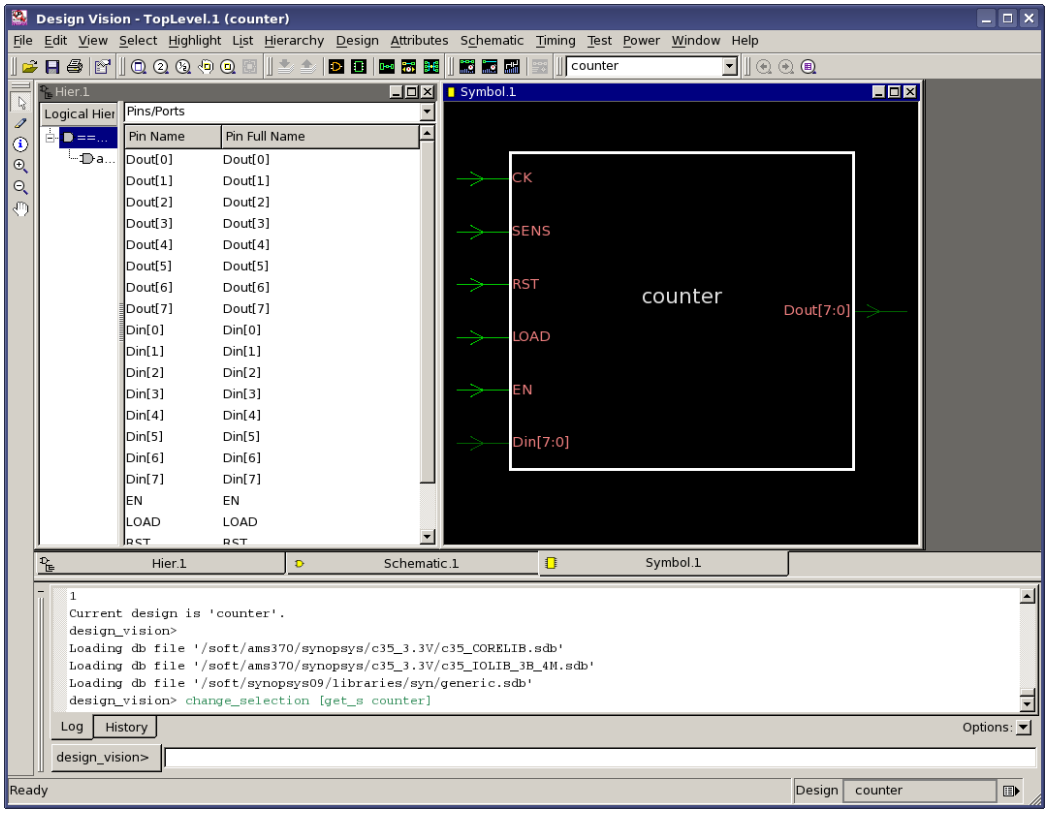
Pour faire la synthèse proprement dite:

Design → **Compile Design**

La synthèse automatique du circuit à partir d'un modèle VHDL est faite.

Pour un outil de synthèse plus performant et plus rapide vous devez utiliser la commande :
Design → Compile Ultra

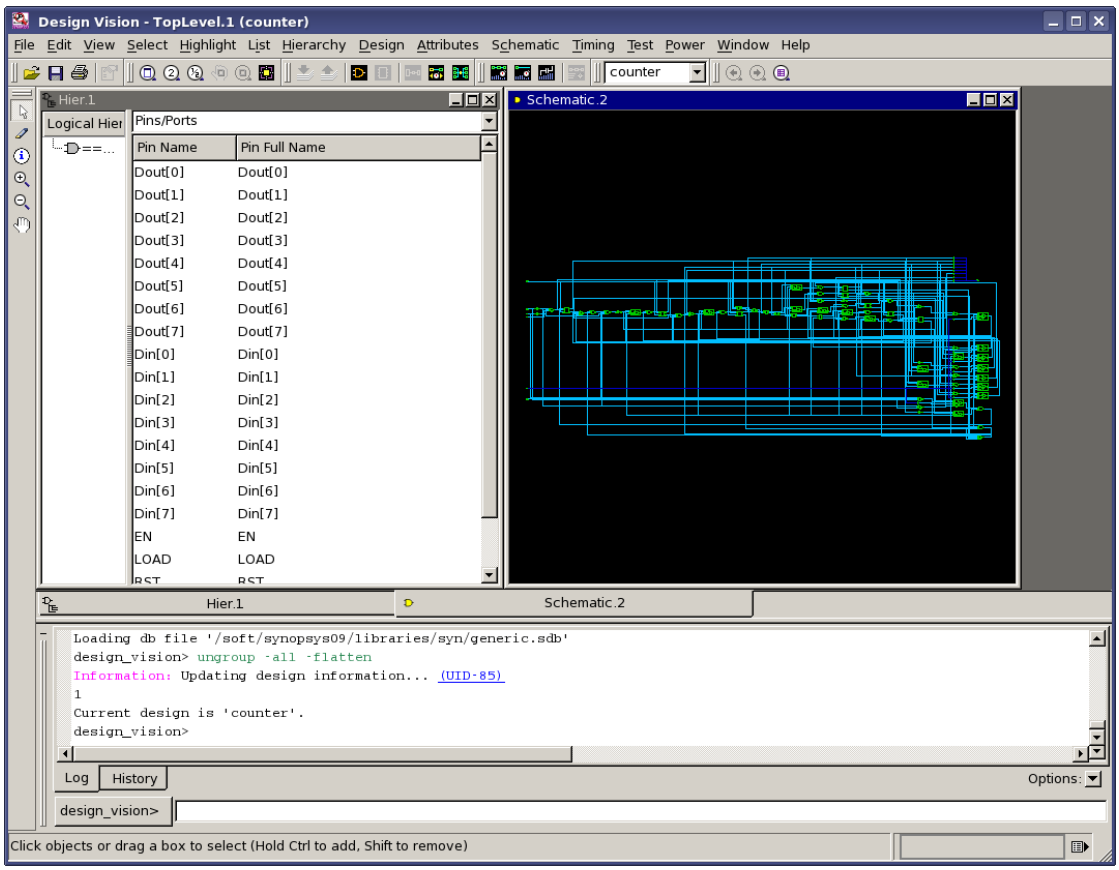
Pour voir le composant on clique sur le bouton qui représente un composant vert dans la barre d'outils.



Pour voir la description au niveau RTL on click sur le symbole de porte logique orange, toujours dans la meme barre. Nous remarquons que dans ce design nous avons des niveaux hiérarchiques.

Pour mettre le circuit « à plat », enlever les niveaux hiérarchiques on va taper dans la ligne de commande :

ungroup –all -flatten



Optimisation du chemin critique

On spécifie une période d'horloge assez faible compte tenu de notre circuit (par exemple 100 MHz pour le compteur)

Attributes → Specify Clock → 10 ns

On relance la synthèse a partir du circuit déjà synthétise.

Design → Compile Design

Map Effort → High

Incremental Mapping (on part du circuit déjà synthétise et on optimise)

OK → Synopsys Design Compiler va resynthétiser en essayant d'obtenir un chemin critique inferieur à la période d'horloge que nous avons spécifié.

On peut faire cette opération plusieurs fois afin d'obtenir le circuit optimise ayant une fréquence d'horloge maximale.
Par exemple pour un circuit plus complexe comme un microprocesseur on peut faire plusieurs itérations : T_{CLK}= 50 ns, T_{CLK}= 20 ns, T_{CLK}= 10 ns, T_{CLK}= 5 ns

OBS : Le chemin critique ne sera pas forcement le même. Pour T_{CLK}= 50 ns on aura un chemin critique que ensuite sera optimise pour arriver à fonctionner à T_{CLK}= 20 ns, donc c'est probable que à T_{CLK}= 20 ns on aura un autre chemin critique.

Préparation des fichiers de sortie après synthèse :

Pour préparer le format pour la sauvegarde du fichier de sortie (qui va être un fichier VHDL) on écrit aussi dans le **Command Window** :

Report_names –rules vhdl
Change_names –rules vhdl

On va sauvegarder le fichier en **TROIS** formats :

- VHDL (pour la simulation après synthèse),
- Verilog (pour rentrer sous SoC Encounter)
- Data base (pour pouvoir relire le fichier synthétisé sous Synopsys , si besoin).

File → Save as

Il faut sauvegarder maintenant les informations de timming : les retards de propagation des signaux dans le circuit dans un fichier de type SDF (Standard Delay Format). Ces retards dépendent de temps de propagation dans les portes logiques et les temps de set-up et hold des bascules.

Dans la console il faut taper la commande :

write_sdf counter.sdf pour le compteur

write_sdf ethernet.sdf pour le contrôleur Ethernet

Format SDF (Standard Delay Format)

Modification à effectuer dans le fichier .sdf

Les délais de propagation sur les interconnexions sont égaux à zéro dans le fichier .sdf obtenu après synthèse, car la synthèse créer le circuit digital et prend en compte les temps de propagation dans les portes logiques. Le routage n'étant pas faite, le temps de propagation sur les interconnexions n'est pas connu, donc il est mis à zéro.
Il faut effacer du fichier tous ces délais sur les interconnexions qui sont égales à zéro. Faites attention aux parenthèses (ouverture, fermeture).