

Processeur MIPS R3000.

Architecture Interne Microprogrammée

Version 2.5

(Septembre 2002)

**Pirouz Bazargan
François Dromard
Alain Greiner**

A) INTRODUCTION

Ce document présente l'architecture interne de la version microprogrammée du processeur MIPS R3000 réalisée par l'équipe Architecture de l'Université Pierre et Marie Curie. L'architecture externe du processeur, et en particulier le jeu d'instructions sont décrits dans un autre document, dont la lecture est indispensable pour la compréhension de celui-ci.

La réalisation décrite ici n'est pas optimale du point de vue des performances ; elle a essentiellement pour but de présenter, au niveau d'un cours de licence, les principes de la microprogrammation.

Une version plus performante, mais plus complexe, du processeur MIPS R3000, utilisant les techniques de pipe-line synchrone, tire tout le bénéfice de la structure RISC du jeu d'instructions. Cette réalisation fait l'objet d'un document séparé, et fait partie du cours de maîtrise.

B) INTERFACE DU PROCESSEUR

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

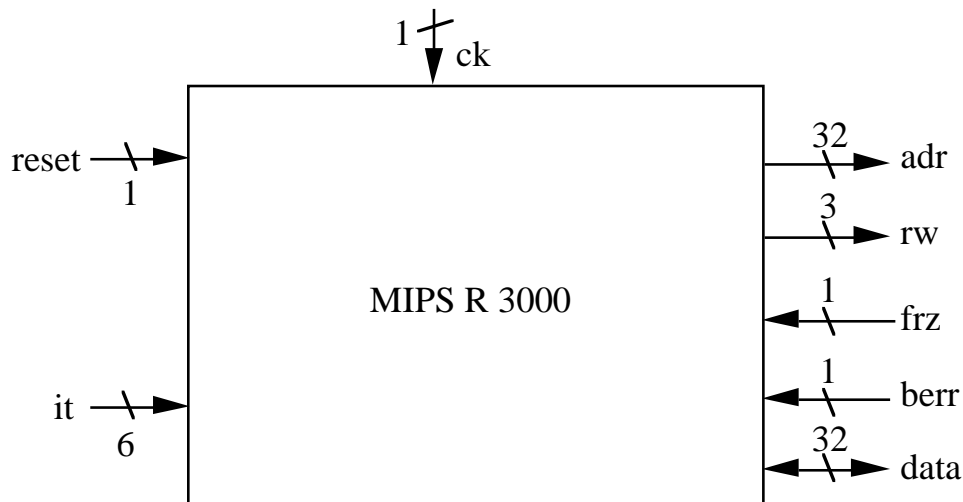
RW	REQUETE	
000	NO	ni écriture, ni lecture
001	WW	écriture d'un mot
010	WH	écriture d'un demi-mot
011	WB	écriture d'un octet
1**	RW	lecture d'un mot

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

Dans le cas RW, les deux bits de poids faible de ADR ne sont pas pris en compte, et la mémoire doit fournir sur le bus DATA un mot aligné. Dans le cas des instructions LBU et LHU, c'est le processeur qui effectue le recadrage et l'extension de signe.

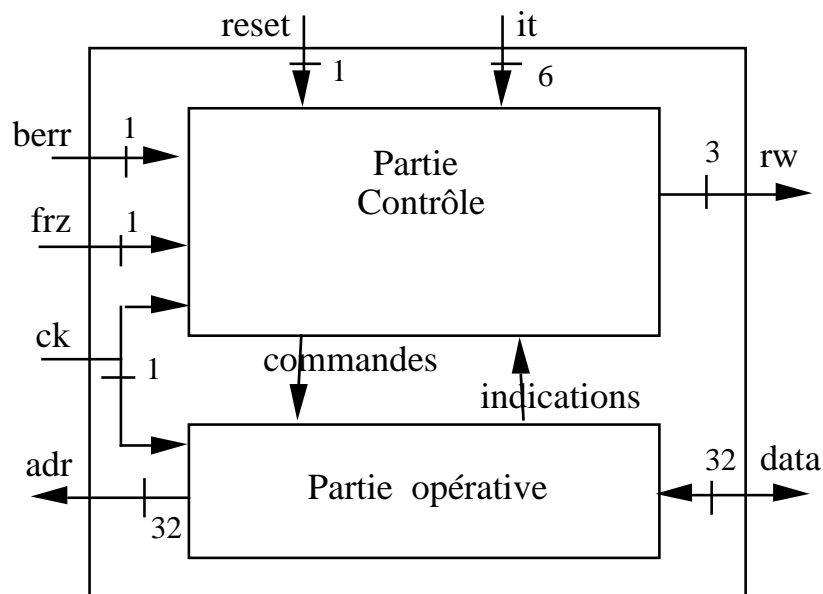
Dans le cas où le système mémoire ne peut pas satisfaire en un cycle la requête d'écriture ou de lecture (par exemple en cas de MISS dans le cache), le signal FRZ doit être activé : le processeur maintient sa requête tant que le signal FRZ est actif.

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



C) ARCHITECTURE INTERNE

L'architecture interne du processeur se décompose en une partie opérative et une partie contrôle. La partie opérative (PO) contient les registres et les opérateurs. Sa structure est présentée au chapitre D. Elle réalise des transferts élémentaires de données entre un ou plusieurs registres sources et un registre destination. Un transfert élémentaire est exécuté en un cycle. La partie opérative est commandée par la partie contrôle (PC). La partie contrôle est chargée de définir, pour chaque cycle d'horloge, les transferts élémentaires qui doivent être réalisés par la partie opérative. La partie contrôle contient principalement un séquenceur décrit comme un automate d'états finis (automate de MOORE). La structure de cet automate est décrite dans le chapitre F.



L'interface entre la partie opérative et la partie contrôle est définie par deux nappes de fils. Dans le sens PC -> PO, cette nappe de fils est décomposée en six champs de commande qui définissent de fait le format de la micro-instruction. Ce format ainsi que les valeurs possibles des différents champs de commande constituent le "langage de micro-programmation", qui est décrit au chapitre E. Dans le sens PO -> PC, cette nappe de fils est constituée de signaux de compte-rendu permettant au séquenceur de prendre des décisions.

D) CHEMIN DE DONNEES

En plus des registres visibles du logiciel, le processeur possède 3 registres internes :

- IR Registre instruction qui mémorise l'instruction lue en mémoire.
- DT Registre data qui reçoit la donnée provenant de la mémoire
- AD Registre d'adresse qui peut être utilisé comme registre temporaire.

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

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

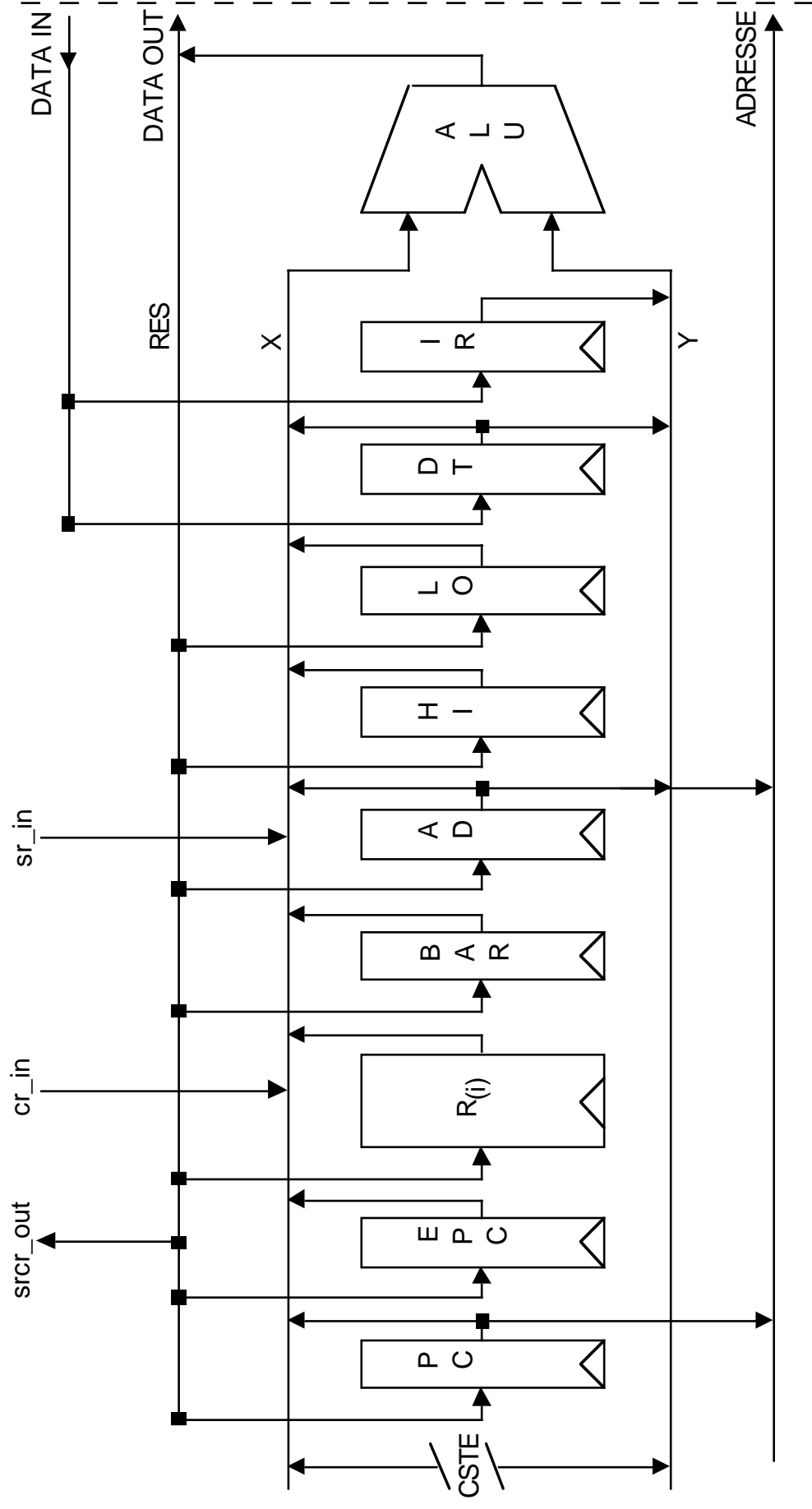
Un mot de 32 bits provenant de la mémoire peut être écrit dans les registres DT ou IR. Un mot de 32 bits provenant de la boîte à opérations peut être écrit dans les registres PC, EPC, SR, CR, BAR, HI, LO, AD, ou dans un registre général R(n). Ceci signifie que cette partie opérative est capable d'effectuer deux transferts élémentaires à chaque cycle : un transfert interne (qui utilise deux registres sources, la boîte à opérations et un registre résultat), et un transfert externe (lecture d'un mot en mémoire externe et écriture dans IR ou DT).

Le banc de registre R(n) ne possède que 2 accès simultanés (un en lecture et un en écriture).

Des pseudo-registres permettent de forcer des valeurs constantes sur les bus X et Y.

Les 32 bits du registre d'instruction IR sont envoyés à la partie contrôle pour lui permettre de prendre des décisions.

Par ailleurs, la partie opérative envoie vers la partie contrôle le signal NUL qui vaut 1 quand RES = Ø. Elle renvoie également vers la partie contrôle les bits de signe des deux opérands X et Y, ainsi que celui du résultat, la retenue de l'alu et les deux bits de poids faibles du résultat. Ces signaux, calculés par la micro-instruction i peuvent être utilisés par la partie contrôle pour choisir la micro-instruction i+1.



MIPS - CHEMIN DE DONNEES

E) LANGAGE DE LA MICRO-PROGRAMMATION

A chaque cycle, la partie contrôle envoie vers la partie opérative des signaux de commande qui constituent la micro-instruction.

La micro-instruction possède 6 champs dont les valeurs possibles définissent le langage de microprogrammation. Ces valeurs sont définies par des mnémoniques dans les tableaux ci-dessous :

- Le champ OPX définit l'opérande X en entrée de la boîte à opérations.
- Le champ OPY définit l'opérande Y en entrée de la boîte à opérations.
- Le champ ALU définit l'opération à exécuter par la boîte à opérations.
- Le champ RES définit la destination du résultat calculé par la boîte à opérations.
- Le champ ADRW définit le type d'échange entre le processeur et la mémoire.
- Le champ EXCP contrôle les mécanismes de détection d'exception et de trappes.

Une micro-instruction est définie par le choix d'une valeur pour chacun des six champs.

OPX	OPY	ALU	RES	ADRW	EXCP
-----	-----	-----	-----	------	------

FORMATION OPERANDE X : CHAMP OPX

- | | |
|---------|-------------------------------------|
| • X_RS | Registre pointé par IR [25:21] |
| • X_RT | Registre pointé par IR [20:16] |
| • X_PC | Registre PC |
| • X_AD | Registre AD |
| • X_HI | Registre HI |
| • X_LO | Registre LO |
| • X_EPC | Registre EPC |
| • X_BAR | Registre BAR |
| • X_SR | Registre SR |
| • X_CR | Registre CR |
| • X_DT | Registre DT |
| • X_PC4 | PC[31:28] 0(28) |
| • X_CØ | Constante 00000000 |
| • X_CZ | Constante BFC00000 (reset handler) |
| • X_CH | Constante 0000FFFF (masque 16 bits) |

FORMATION OPERANDE Y : CHAMP OPY

• Y_I 16	IR15 (16) IR [15:0]
• Y_I 18	IR15 (14) IR [15:0] 0(2)
• Y_IU 28	0(4) IR [25:0] 0(2)
• Y_I SH	0(27) IR [10:6]
• Y_DT	registre DT
• Y_AD	registre AD
• Y_CØ	Constante 00000000
• Y_C4	Constante 00000004
• Y_C8	Constante 00000008
• Y_C16	Constante 00000010
• Y_C24	Constante 00000018
• Y_CX	Constante 80000080 (exception handler)

OPERATION : CHAMP ALU

• A_SUM	RES <--- $X + Y$	Addition
• A_DIF	RES <--- $X - Y$	Soustraction
• A_AND	RES <--- $X \cdot Y$	et
• A_OR	RES <--- $X Y$	ou
• A_XOR	RES <--- $X \oplus Y$	ou exclusif
• A_NOR	RES <--- $\neg X \cdot \neg Y$	non-ou
• A_SLL	RES <--- $X \ll Y [4:0]$	X est décalé à gauche de Y bits (on complète avec des Ø)
• A_SRL	RES <--- $X \gg Y [4:0]$	X est décalé à droite de Y bits (on complète avec des Ø)
• A_SRA	RES <--- $X * \gg Y [4:0]$	X est décalé à droite de Y bits (on complète par extension de signe)
• A_SLTU	RES <--- 1 si $(X < Y)$ RES <--- 0 si $(X \geq Y)$	comparaison entre nombres non-signés.
• A_SLT	RES <--- 1 si $(X < Y)$ RES <--- 0 si $(X \geq Y)$	comparaison entre nombres signés
• A_MUL	RES <--- $(X * Y)[31:0]$	multiplication non signée, sortie 32 bits bas
• A_MUH	RES <--- $(X * Y)[63:32]$	multiplication non signée, sortie 32 bits haut

- A_MSL RES <--- (X * Y)[31:0] multiplication signée, sortie 32 bits bas
- A_MSH RES <--- (X * Y)[63:32] multiplication signée, sortie 32 bits haut
- A_DUQ RES <--- (X / Y)Q division non signée, quotient
- A_DUR RES <--- (X / Y)R division non signée, reste
- A_DSQ RES <--- (X / Y)Q division signée, quotient
- A_DSR RES <--- (X / Y)R division signée, reste

DESTINATION RESULTAT : CHAMP RES

- R_NOP Pas d'écriture
- R_PC registre PC
- R_AD registre AD
- R_BAR registre BAR
- R_LO registre LO
- R_HI registre HI
- R_EPC registre EPC
- R_CR registre CR
- R_SR registre SR
- R_RT registre général pointé par IR[20:16]
- R_RD registre général pointé par IR[15:11]
- R_R31 registre général R(31)
- R_ERQ écriture nouvel état dans SR et écriture CR
- R_RFE restitution ancien état dans SR

Les codes RFE et ERQ sont des codes spéciaux. Le code RFE provoque un décalage de 2 bits vers la droite sur les 6 bits de poids faibles du registre SR. Le code ERQ provoque un décalage de 2 bits vers la gauche sur les six bits de poids forts du registre SR, et autorise l'écriture dans les 6 bits de poids faible du registre CR.

ACCES MEMOIRE : CHAMP ADRW

- M_NOP NO <--- M (AD) pas d'accès mémoire .
- M_IF IR <--- M (PC) fetch instruction
- M_WW M(AD) <--- RES écriture mémoire d'un mot
- M_WH M(AD) <--- RES [15:0] écriture mémoire d'un demi-mot
- M_WB M(AD) <--- RES [7:0] écriture mémoire d'un octet
- M_LW DT <--- M(AD) lecture mémoire d'un mot

DETECTION DES EXCEPTIONS : CHAMP EXCP

• E_NOP	pas de détection d'exception	
• E_RI	détection d'un codop illégal	(RI)
• E_OVF	détection d'overflow	(OVF)
• E_CPU	détection de violation de privilège	(CPU)
• E_SYS	instruction SYSCALL	(SYS)
• E_BRK	instruction BREAK	(BP)
• E_WW	détection erreur d'adressage sur écriture mot	(ADES)
• E_WH	détection erreur d'adressage sur écriture demi-mot	(ADES)
• E_WB	détection erreur d'adressage sur écriture octet	(ADES)
• E_LW	détection erreur d'adressage sur lecture mot	(ADEL)
• E_LH	détection erreur d'adressage sur lecture demi-mot	(ADEL)
• E_LB	détection erreur d'adressage sur lecture octet	(ADEL)
• E_IF	détection erreur d'adressage sur lecture instruction	(ADEL)
• E_IBE	détection erreur bus instruction	(IBE)
• E_DBE	détection erreur bus donnée	(DBE)
• E_CLR	met à zéro toutes les bascules d'exception	

F) STRUCTURE DE L'AUTOMATE DE CONTROLE

Le séquenceur chargé de contrôler la partie opérative est un automate de MOORE dont les entrées sont les signaux indicateurs envoyés par la partie opérative, et dont les sorties sont les signaux de commande constituant la micro-instruction.

Cet automate est décrit de façon graphique. Les noeuds (ellipses) représentent les états, les flèches représentent les transitions entre états, qui peuvent dépendre des signaux provenant de la partie opérative. Chaque état est identifié par un nom. A chaque état de l'automate est associée une micro-instruction, qui porte le nom de cet état.

Toute instruction du langage machine se décompose en une séquence de micro-instructions. Toutes les instructions commencent par la même micro-instruction, qui réalise l'incrément du compteur ordinal. Ensuite le comportement de l'automate dépend de la valeur du code opération (IR[31:26]), qui fait partie des entrées de l'automate. Enfin, par convention, la dernière micro-instruction d'une instruction i effectue la lecture de l'instruction $i+1$.

G) TRAITEMENT DES EXCEPTIONS ET INTERRUPTIONS

La partie contrôle contient neuf bascules 1 bit correspondant aux sept types d'exception définis dans l'architecture du processeur MIPS R3000 (ADEL, ADES, DBE, IBE, OVF, RI, CPU) et aux deux appels systèmes correspondant aux instructions SYSCALL et BREAK (SYS,BRK). Ces bascules sont mises à un en cas d'exception lorsque la détection de l'exception correspondante est autorisée par le champ EXCP de la micro-instruction. Elles sont toutes remises à zéro si le champ EXCP contient la commande E_CLR, ce qui est normalement fait par la première micro-instruction du micro-programme de branchement au gestionnaire d'exceptions.

En cas d'exception, le microprogramme de l'instruction fautive se déroule "normalement", jusqu'à la dernière micro-instruction. Cependant, toutes les écritures dans les registres visibles du logiciel et les accès à la mémoire sont inhibés dès qu'une bascule d'exception est à 1.

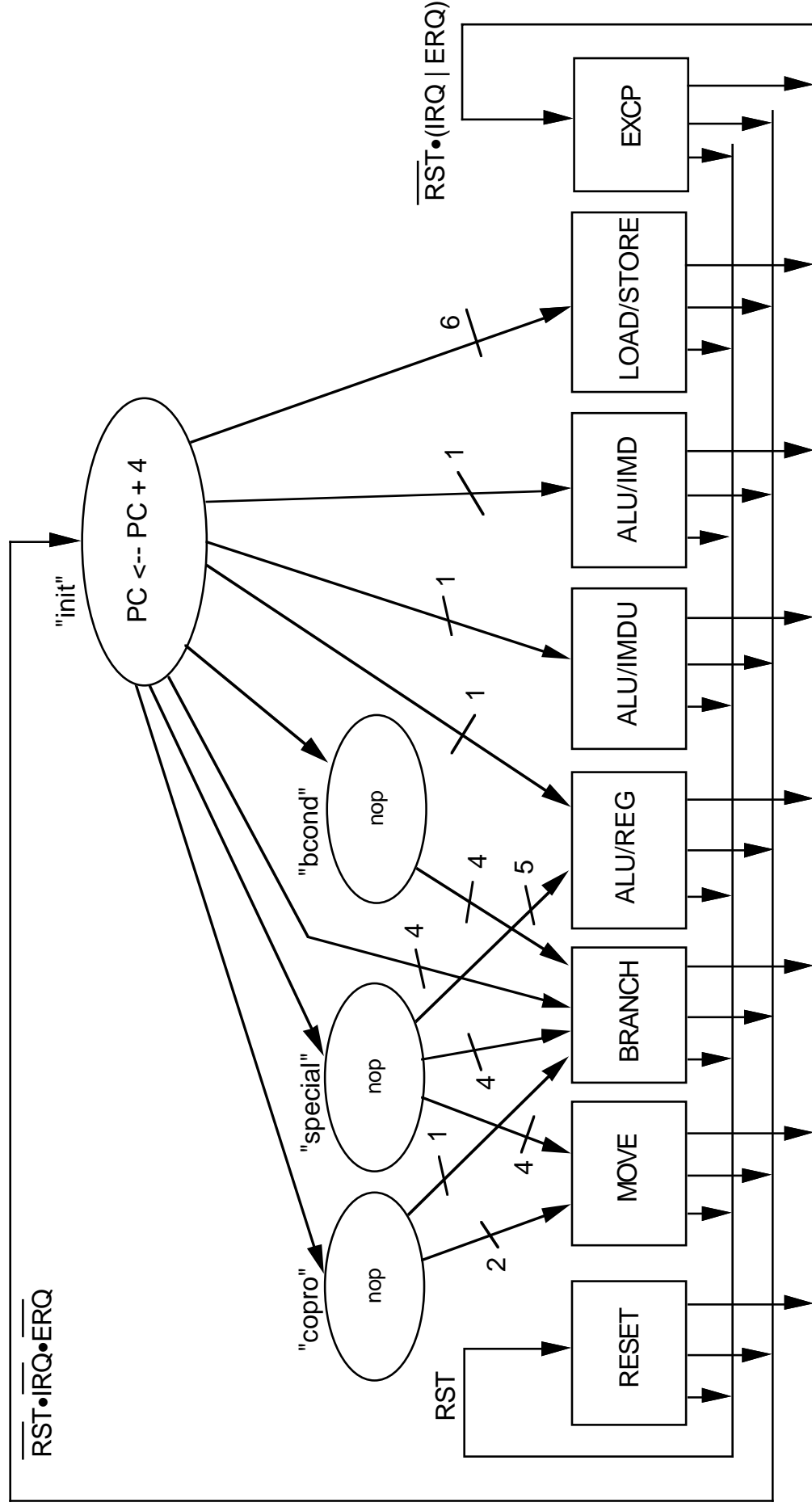
La dernière micro-instruction d'une instruction teste s'il n'y a pas une interruption ou une exception pendante. Une interruption est pendante si l'une des lignes IT5 à IT0 est active. Une exception est pendante si une des neuf bascules d'exception est à 1. Si c'est le cas, au lieu de démarrer l'exécution de l'instruction suivante, l'automate exécute la séquence de micro-instructions permettant de se brancher au gestionnaire d'exceptions. Ce micro-programme comporte quatre micro-instructions distinctes et effectue les actions suivantes:

- passage en mode superviseur et masquage des interruptions dans SR.
- écriture de la cause du déroutement dans CR et CLR des bascules d'exception.
- sauvegarde du PC dans EPC.
- chargement de l'adresse 80000080 dans PC.
- sauvegarde de AD dans BAR.
- chargement du registre instruction.

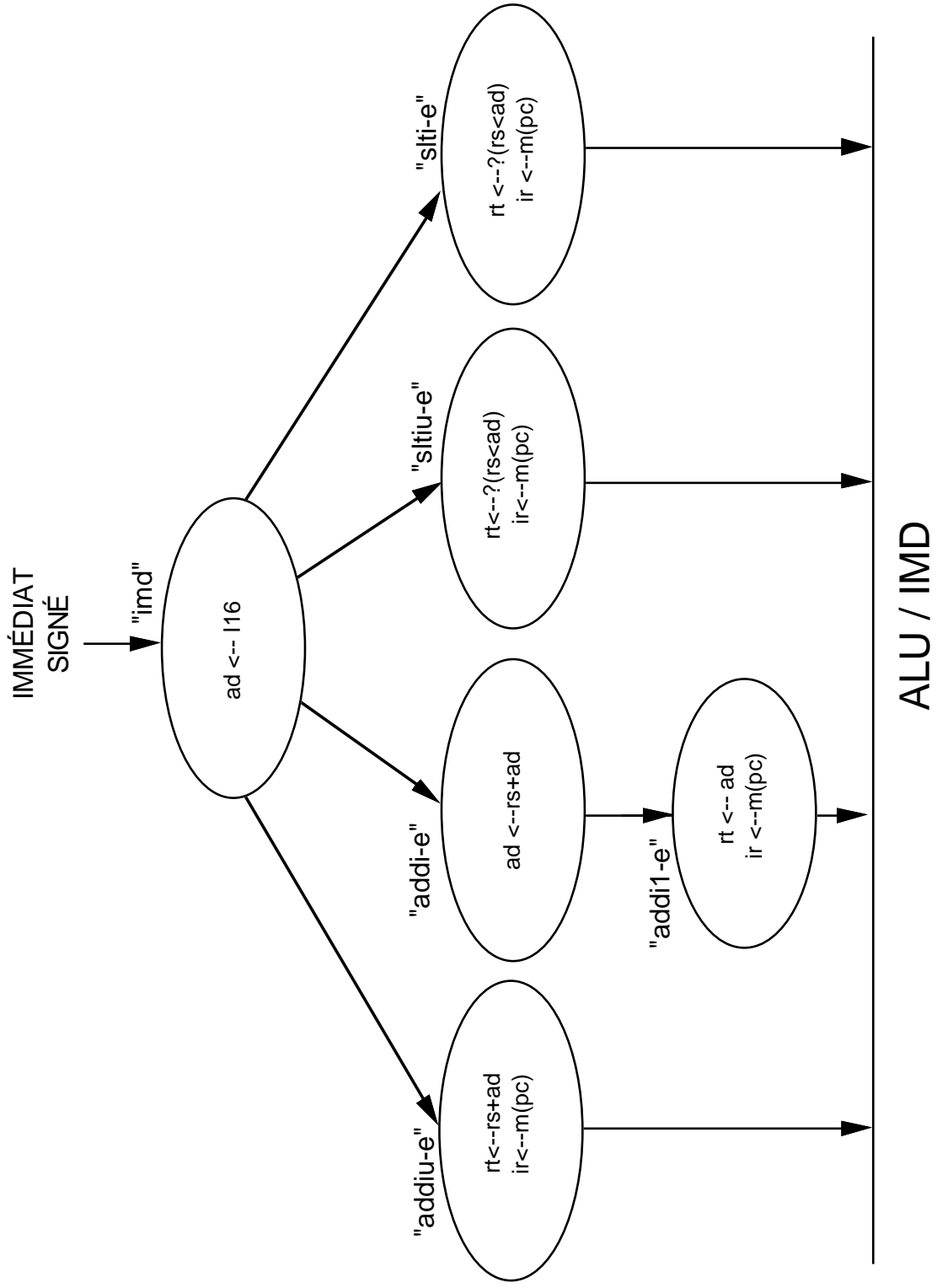
Dans le cas où une exception et une interruption sont pendantes simultanément, l'exception est traitée en premier.

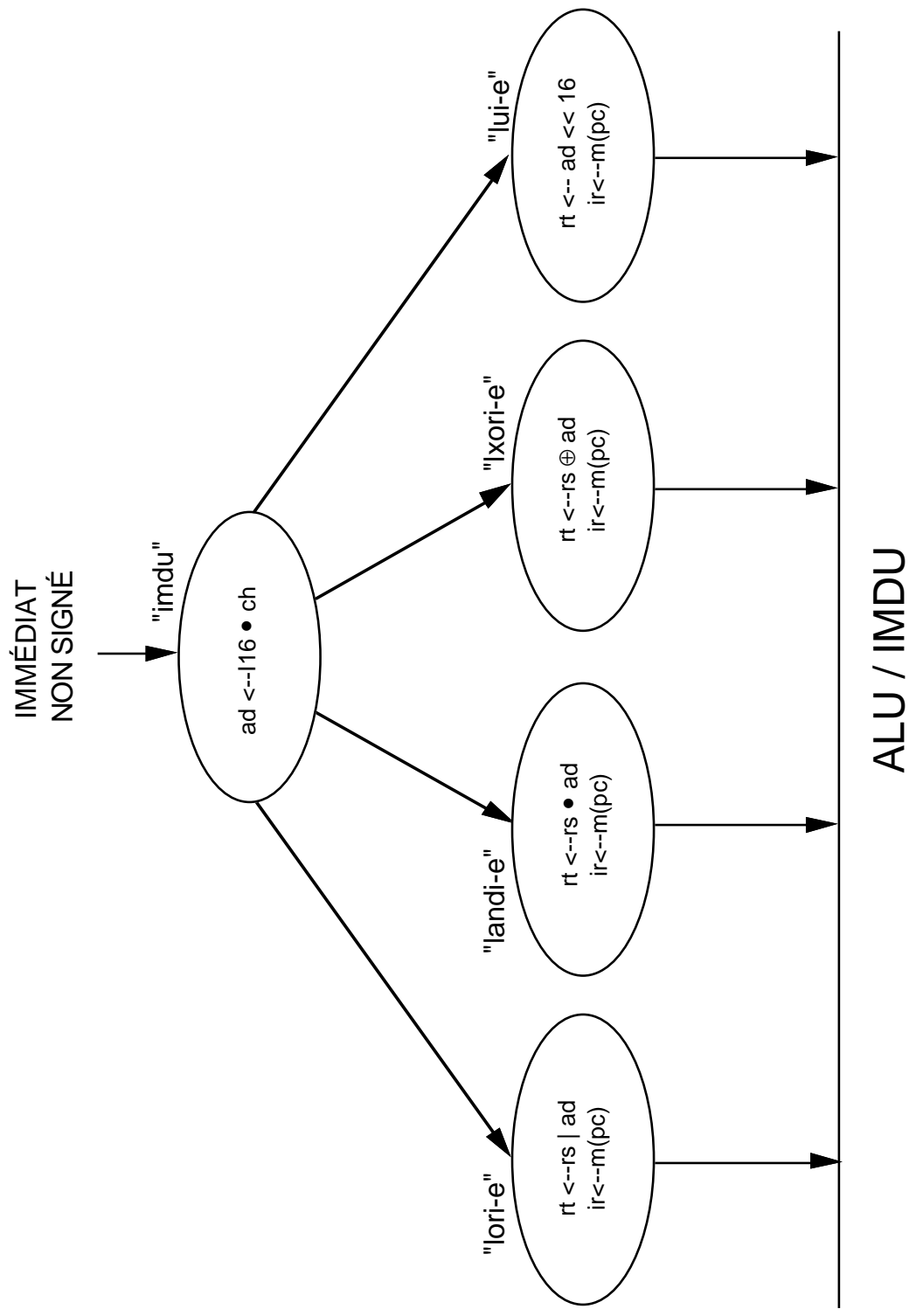
Le RESET est traité de la même manière, mais le microprogramme de branchement au gestionnaire d'initialisation est plus simple (trois micro-instructions).

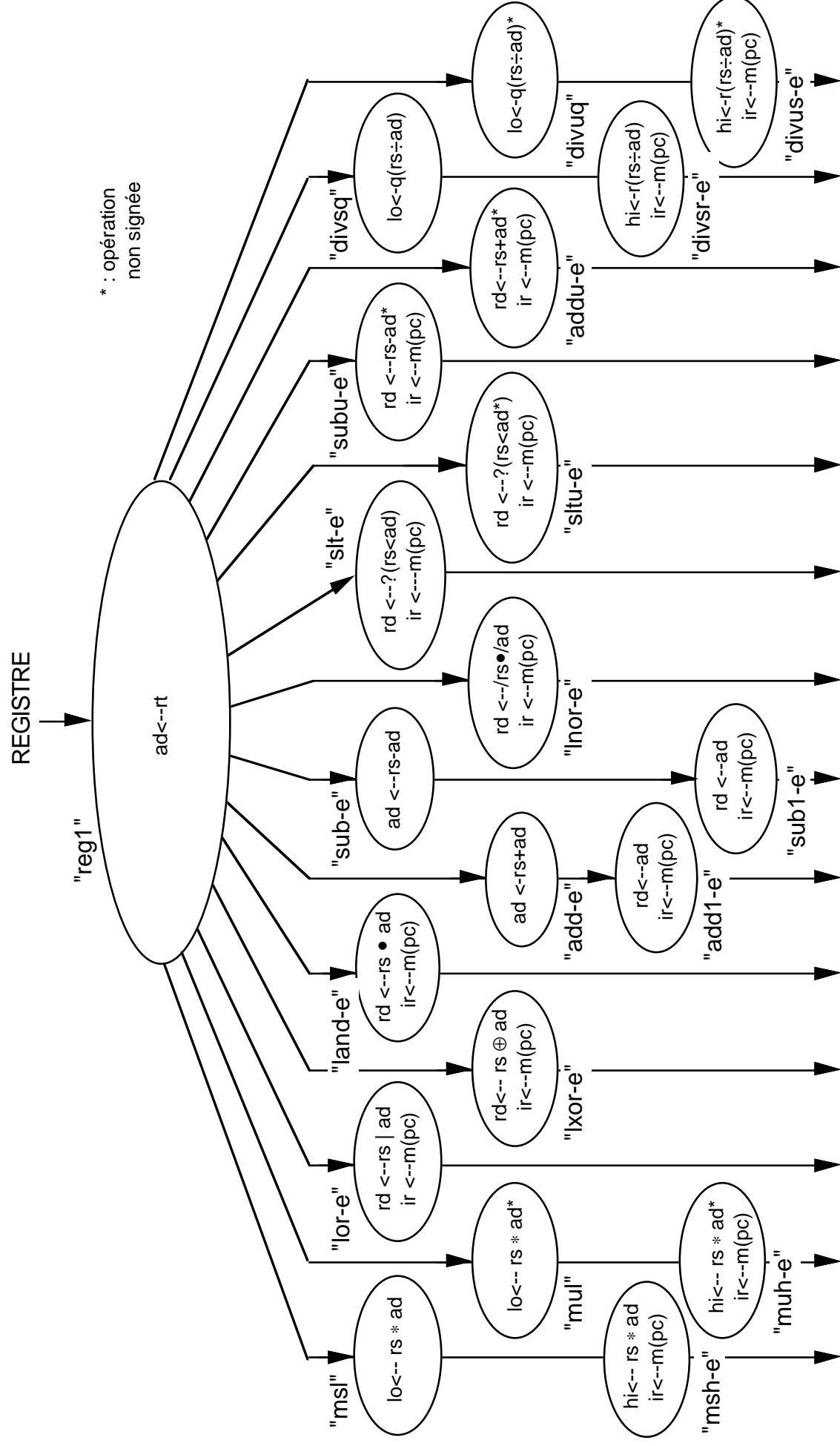
L'automate complet comporte 106 états. Pour améliorer la lisibilité, le graphe a été découpé en plusieurs sous-graphes. Les pages suivantes présentent le découpage en sous-graphes ainsi que chacun des sous-graphes.

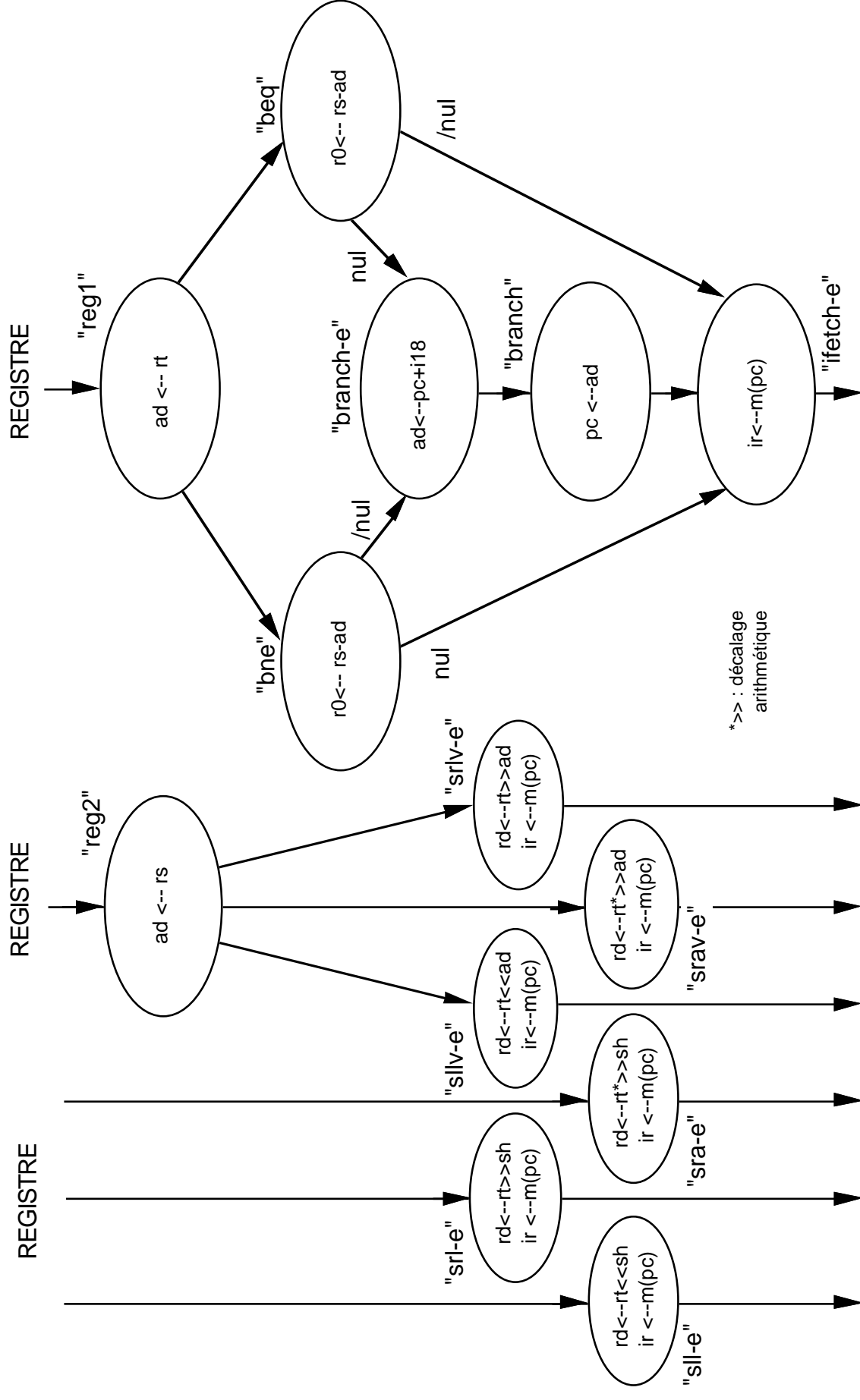


ORGANISATION DU MICROPROGRAMME DU MIPS R3000

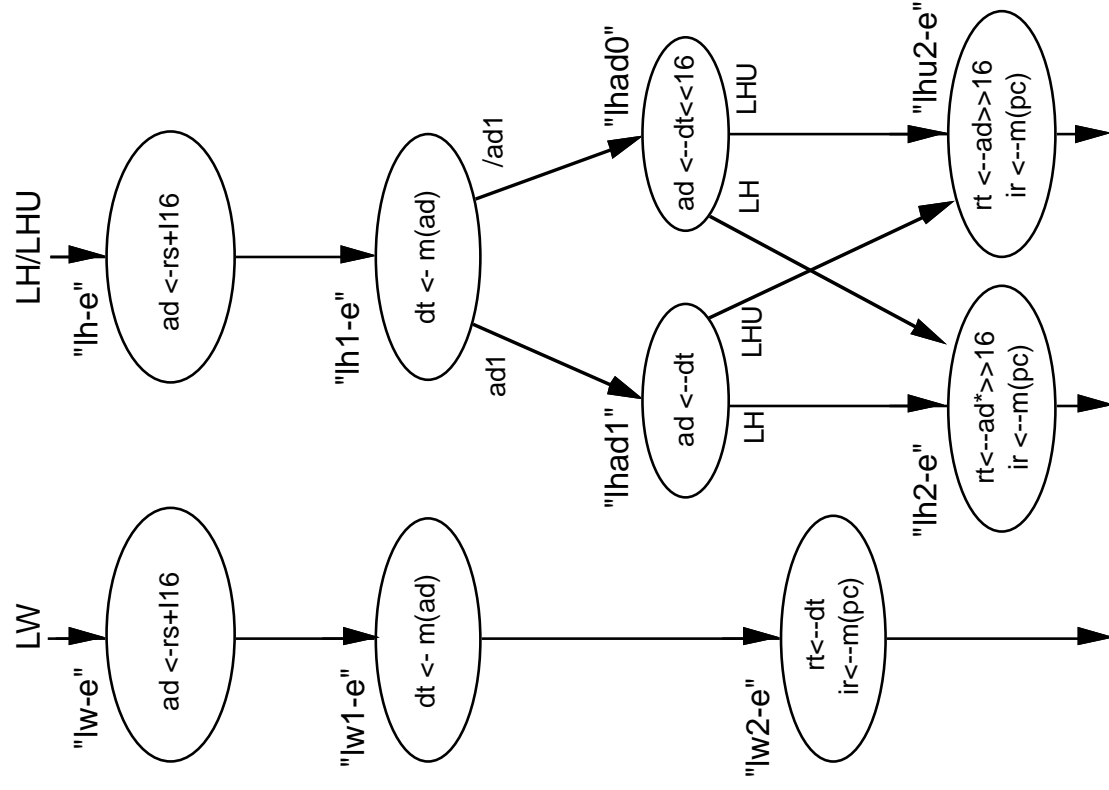




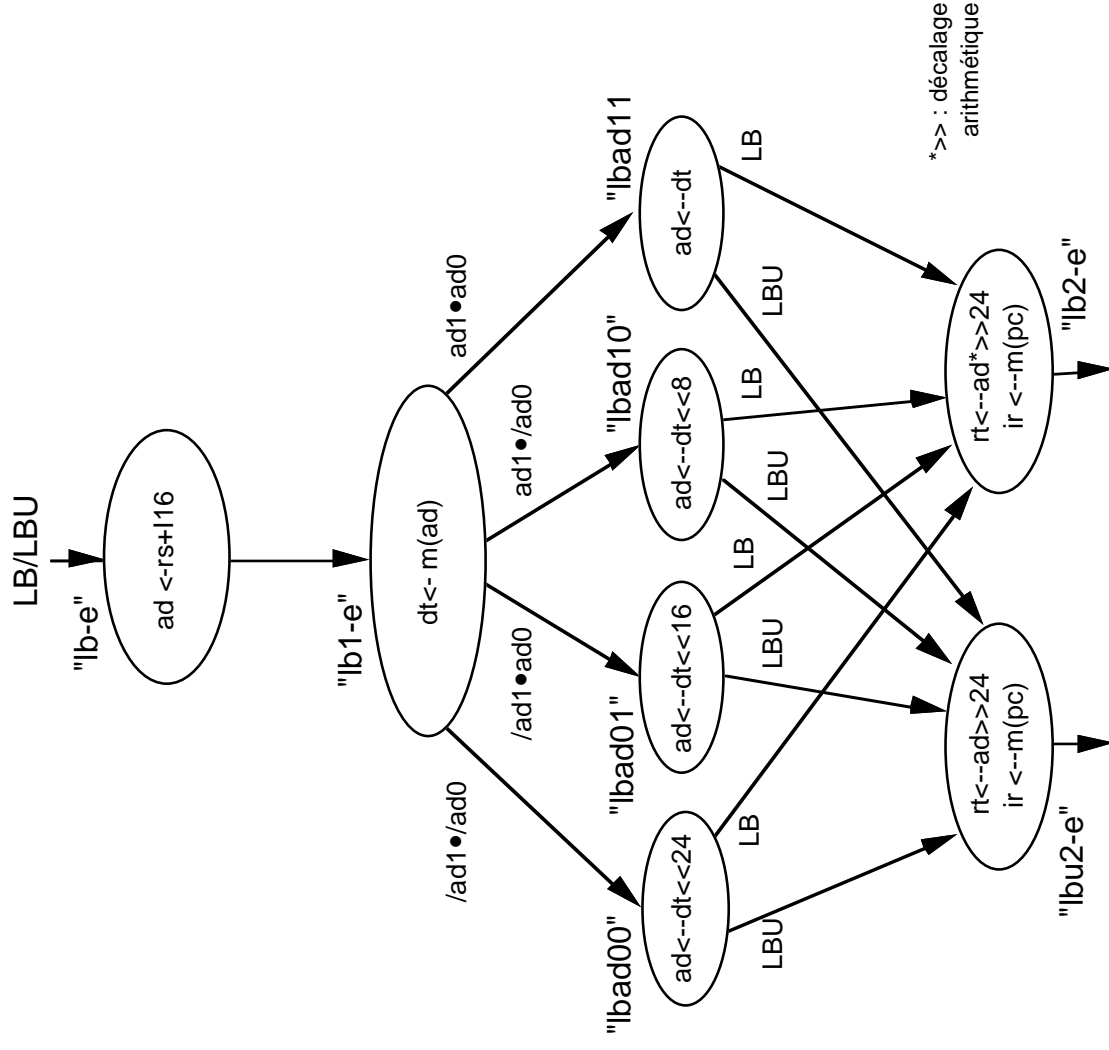


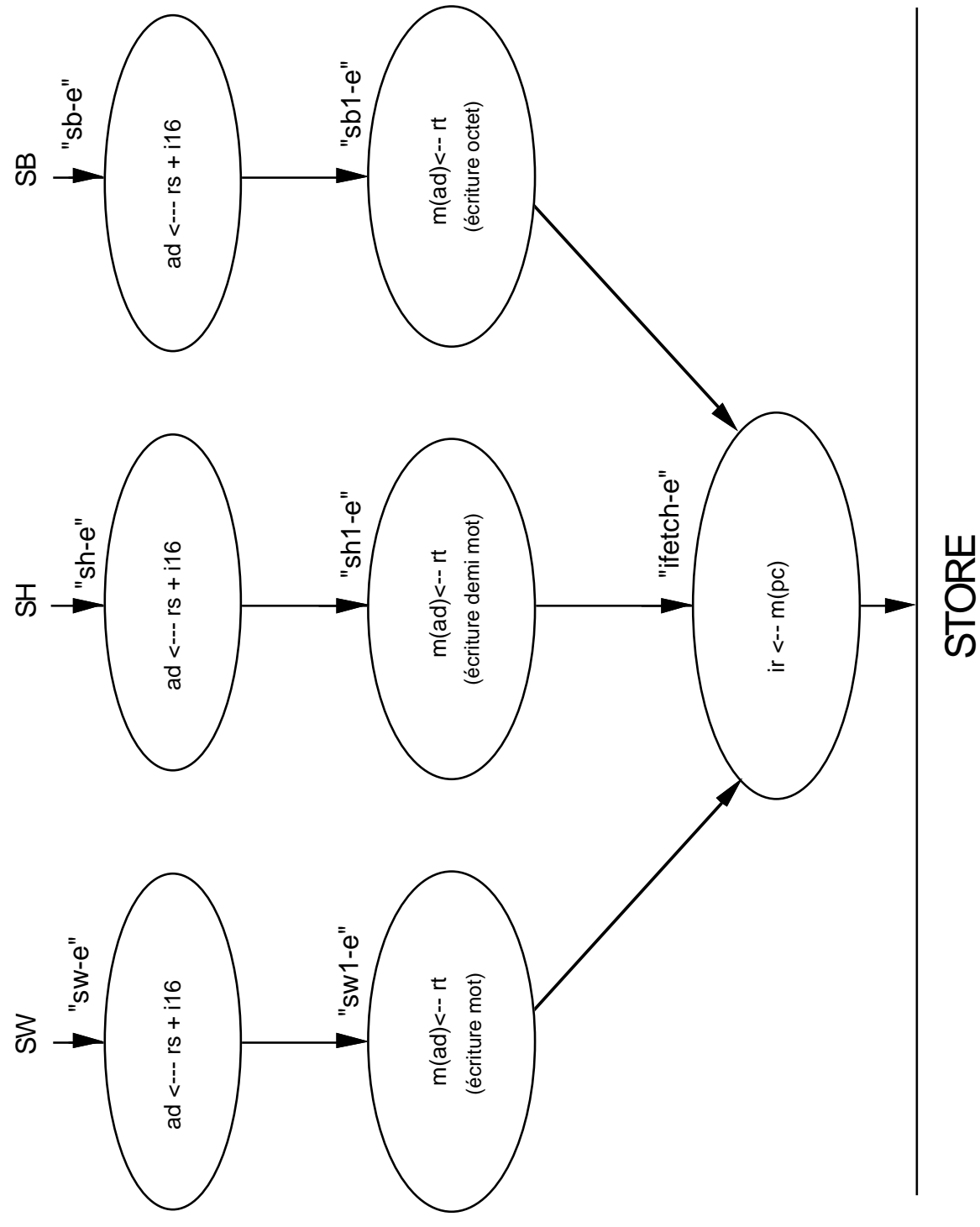


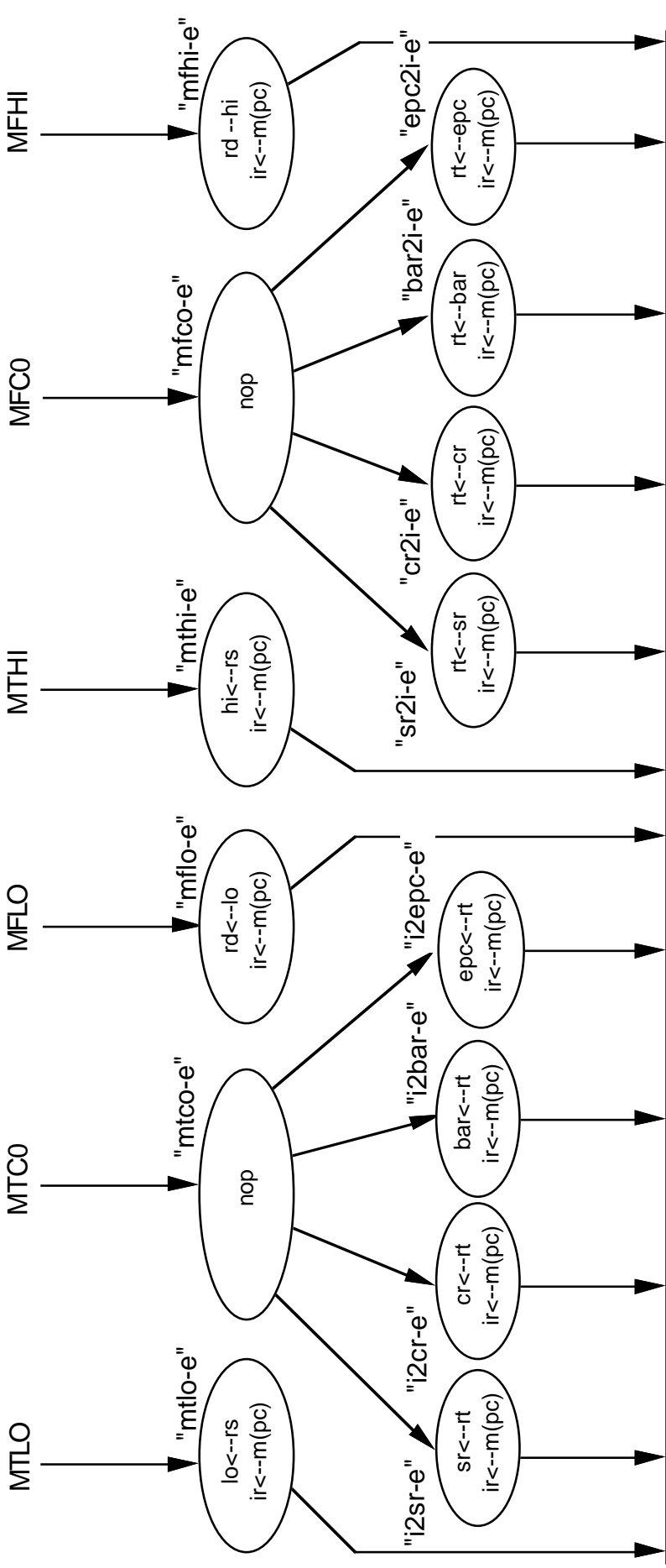
ALU / REG



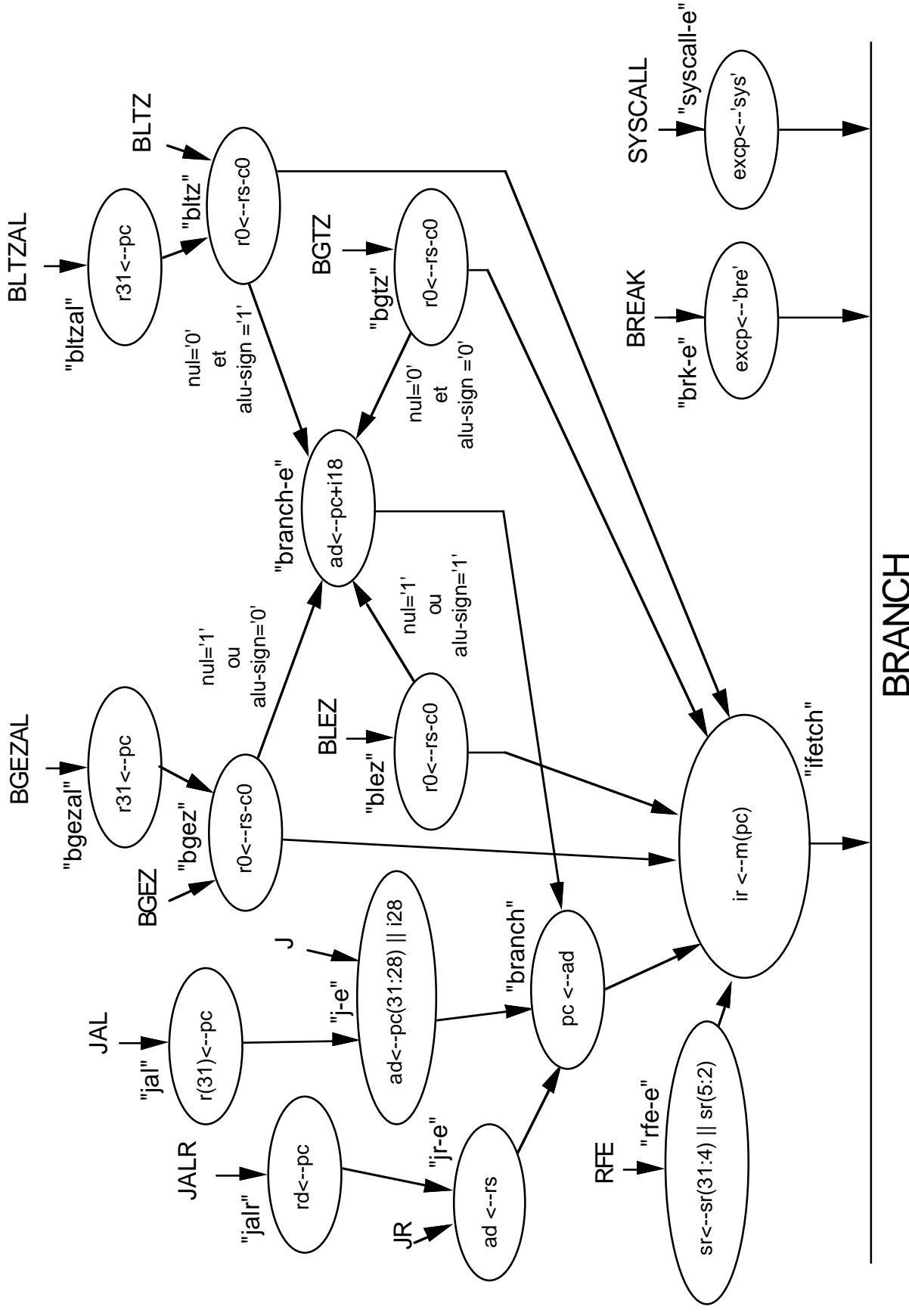
LOAD

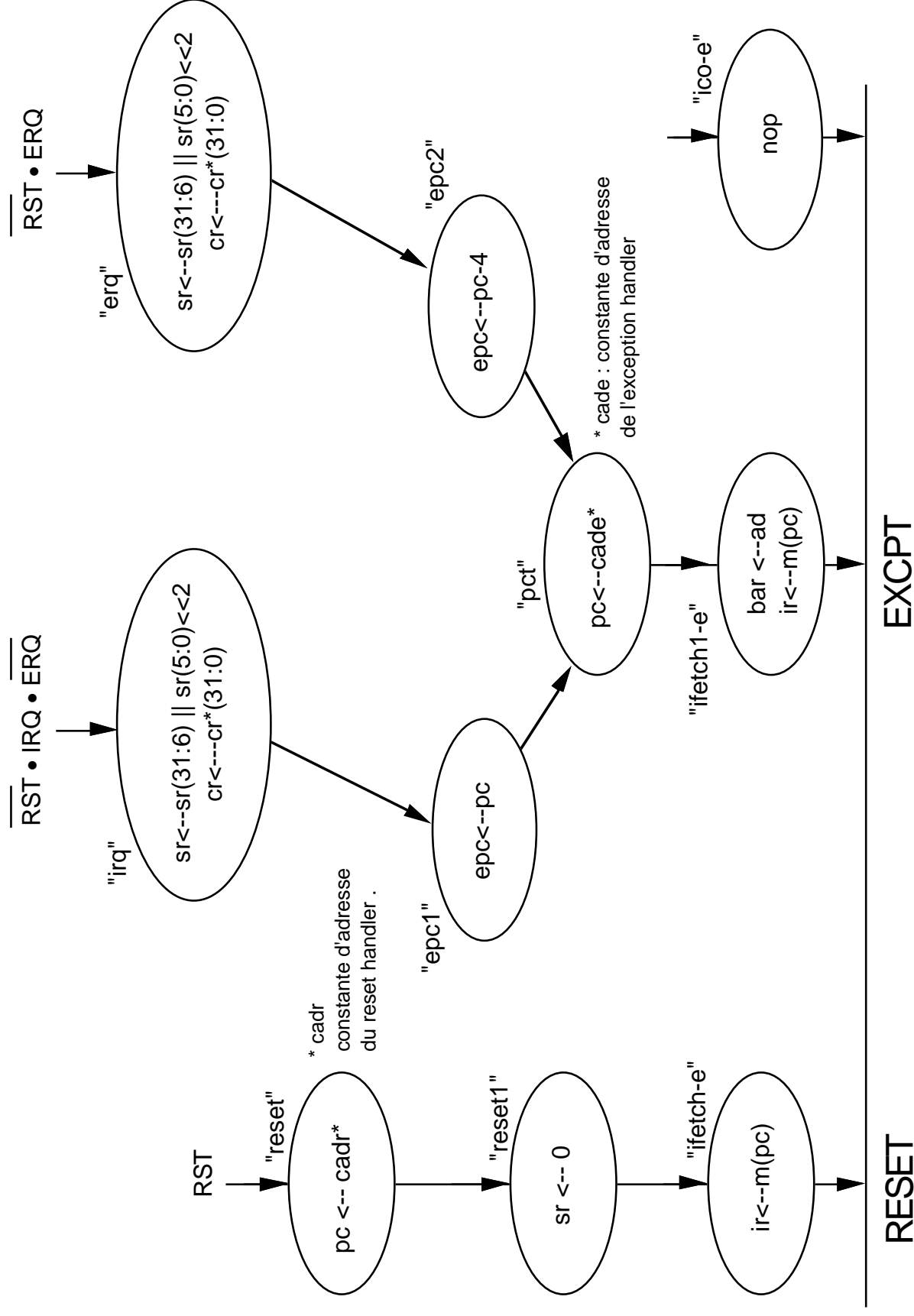






MOVE





EP	OPX	OPY	ALU	RES	ADRW	EXCP
add-e	X_RS	Y_AD	A_SUM	R_AD	M_NOP	E_OVF
addl-e	X_C0	Y_AD	A_SUM	R_RD	M_IF	E_IBE
addi-e	X_RS	Y_AD	A_SUM	R_AD	M_NOP	E_OVF
addil-e	X_C0	Y_AD	A_SUM	R_RT	M_IF	E_IBE
addiu-e	X_RS	Y_AD	A_SUM	R_RT	M_IF	E_IBE
addu-e	X_RS	Y_AD	A_SUM	R_RD	M_IF	E_IBE
bar2i-e	X_BAR	Y_C0	A_SUM	R_RT	M_IF	E_IBE
bcond	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_NOP
beq	X_RS	Y_AD	A_DIF	R_NOP	M_NOP	E_NOP
bgez	X_RS	Y_C0	A_DIF	R_NOP	M_NOP	E_NOP
bgezal	X_PC	Y_C0	A_DIF	R_R31	M_NOP	E_NOP
bgtz	X_RS	Y_C0	A_DIF	R_NOP	M_NOP	E_NOP
blez	X_RS	Y_C0	A_DIF	R_NOP	M_NOP	E_NOP
bltz	X_RS	Y_C0	A_DIF	R_NOP	M_NOP	E_NOP
bltzal	X_PC	Y_C0	A_DIF	R_R31	M_NOP	E_NOP
bne	X_RS	Y_AD	A_DIF	R_NOP	M_NOP	E_NOP
branch	X_AD	Y_C0	A_SUM	R_PC	M_NOP	E_NOP
branch-e	X_PC	Y_I18	A_SUM	R_AD	M_NOP	E_IF
brk-e	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_BRK
copro	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_NOP
cr2i-e	X_CR	Y_C0	A_SUM	R_RT	M_IF	E_IBE
divsq	X_RS	Y_AD	A_DSQ	R_LO	M_NOP	E_NOP
divsr-e	X_RS	Y_AD	A_DSR	R_HI	M_IF	E_IBE
divuq	X_RS	Y_AD	A_DUQ	R_LO	M_NOP	E_NOP
divur-e	X_RS	Y_AD	A_DUR	R_HI	M_IF	E_IBE
epc1	X_PC	Y_C0	A_SUM	R_EPC	M_NOP	E_NOP
epc2	X_PC	Y_C4	A_DIF	R_EPC	M_NOP	E_NOP
epc2i-e	X_EPC	Y_C0	A_SUM	R_RT	M_IF	E_IBE
erq	X_C0	Y_C0	A_SUM	R_ERQ	M_NOP	E_CLR

EP	OPX	OPY	ALU	RES	ADRW	EXCP
i2bar-e	X_RT	Y_C0	A_SUM	R_BAR	M_IF	E_IBE
i2cr-e	X_RT	Y_C0	A_SUM	R_CR	M_IF	E_IBE
i2epc-e	X_RT	Y_C0	A_SUM	R_EPC	M_IF	E_IBE
i2sr-e	X_RT	Y_C0	A_SUM	R_SR	M_IF	E_IBE
ico-e	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_RI
ifetch-e	X_C0	Y_C0	A_SUM	R_NOP	M_IF	E_IBE
ifetchl-e	X_C0	Y_AD	A_SUM	R_BAR	M_IF	E_IBE
imd	X_C0	Y_I16	A_SUM	R_AD	M_NOP	E_NOP
imdu	X_CH	Y_I16	A_AND	R_AD	M_NOP	E_NOP
init	X_PC	Y_C4	A_SUM	R_PC	M_NOP	E_NOP
irq	X_C0	Y_C0	A_SUM	R_ERQ	M_NOP	E_CLR
j-e	X_PC4	Y_IU28	A_OR	R_AD	M_NOP	E_IF
jal	X_PC	Y_C0	A_SUM	R_R31	M_NOP	E_NOP
jalr	X_PC	Y_C0	A_SUM	R_RD	M_NOP	E_NOP
jr-e	X_RS	Y_C0	A_SUM	R_AD	M_NOP	E_IF
land-e	X_RS	Y_AD	A_AND	R_RD	M_IF	E_IBE
landi-e	X_RS	Y_C0	A_AND	R_RT	M_IF	E_IBE
lb-e	X_RS	Y_I16	A_SUM	R_AD	M_NOP	E_LB
lb1-e	X_C0	Y_C0	A_SUM	R_NOP	M_LW	E_DBE
lb2-e	X_AD	Y_C24	A_SRA	R_RT	M_IF	E_IBE
lbad00	X_DT	Y_C24	A_SLL	R_AD	M_NOP	E_NOP
lbad01	X_DT	Y_C16	A_SLL	R_AD	M_NOP	E_NOP
lbad10	X_DT	Y_C8	A_SLL	R_AD	M_NOP	E_NOP
lbad11	X_DT	Y_C0	A_SUM	R_AD	M_NOP	E_NOP
lbu2-e	X_AD	Y_C24	A_SRL	R_RT	M_IF	E_IBE
lh-e	X_RS	Y_I16	A_SUM	R_AD	M_NOP	E_LH
lh1-e	X_C0	Y_C0	A_SUM	R_NOP	M_LW	E_DBE
lh2-e	X_AD	Y_C16	A_SRA	R_RT	M_IF	E_IBE
lhad0	X_DT	Y_C16	A_SLL	R_AD	M_NOP	E_NOP

EP	OPX	OPY	ALU	RES	ADRW	EXCP
lhadd1	X_C0	Y_DT	A_SUM	R_AD	M_NOP	E_NOP
lhu2-e	X_AD	Y_C16	A_SRL	R_RT	M_IF	E_IBE
lnor-e	X_RS	Y_AD	A_NOR	R_RD	M_IF	E_IBE
lor-e	X_RS	Y_AD	A_OR	R_RD	M_IF	E_IBE
lori-e	X_RS	Y_AD	A_OR	R_RT	M_IF	E_IBE
lui-e	X_AD	Y_C16	A_SLL	R_RT	M_IF	E_IBE
lw-e	X_RS	Y_I16	A_SUM	R_AD	M_NOP	E_LW
lw1-e	X_C0	Y_C0	A_SUM	R_NOP	M_LW	E_DBE
lw2-e	X_C0	Y_DT	A_SUM	R_RT	M_IF	E_IBE
lxor-e	X_RS	Y_AD	A_XOR	R_RD	M_IF	E_IBE
lxori-e	X_RS	Y_AD	A_XOR	R_RT	M_IF	E_IBE
mfc0-e	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_CPU
mfhi-e	X_HI	Y_C0	A_SUM	R_RD	M_IF	E_IBE
mflo-e	X_LO	Y_C0	A_SUM	R_RD	M_IF	E_IBE
mulsh-e	X_RS	Y_AD	A_MSH	R_HI	M_IF	E_IBE
mulsl	X_RS	Y_AD	A_MSL	R_LO	M_NOP	E_NOP
mtc0-e	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_CPU
mthi-e	X_RS	Y_C0	A_SUM	R_HI	M_IF	E_IBE
mtlo-e	X_RS	Y_C0	A_SUM	R_LO	M_IF	E_IBE
muluh-e	X_RS	Y_AD	A_MUH	R_HI	M_IF	E_IBE
mulul	X_RS	Y_AD	A_MUL	R_LO	M_NOP	E_NOP
pct	X_C0	Y_CX	A_SUM	R_PC	M_NOP	E_NOP
reg1	X_RT	Y_C0	A_SUM	R_AD	M_NOP	E_NOP
reg2	X_RS	Y_C0	A_SUM	R_AD	M_NOP	E_NOP
reset	X_CZ	Y_C0	A_SUM	R_PC	M_NOP	E_CLR
reset1	X_C0	Y_C0	A_SUM	R_SR	M_NOP	E_NOP
rfe-e	X_C0	Y_C0	A_SUM	R_RFE	M_NOP	E_CPU
sb-e	X_RS	Y_I16	A_SUM	R_AD	M_NOP	E_WB
sb1-e	X_RT	Y_C0	A_SUM	R_NOP	M_WB	E_DBE
sh-e	X_RS	Y_I16	A_SUM	R_AD	M_NOP	E_WH

EP	OPX	OPY	ALU	RES	ADRW	EXCP
shl-e	X_RT	Y_C0	A_SUM	R_NOP	M_WH	E_DBE
sll-e	X_RT	Y_ISH	A_SLL	R_RD	M_IF	E_IBE
sllv-e	X_RT	Y_AD	A_SLL	R_RD	M_IF	E_IBE
slt-e	X_RS	Y_AD	A_SLT	R_RD	M_IF	E_IBE
slti-e	X_RS	Y_AD	A_SLT	R_RD	M_IF	E_IBE
sltiu-e	X_RS	Y_AD	A_SLTU	R_RT	M_IF	E_IBE
sltu-e	X_RS	Y_AD	A_SLTU	R_RD	M_IF	E_IBE
special	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_NOP
sr2i-e	X_SR	Y_C0	A_SUM	R_RT	M_IF	E_IBE
sra-e	X_RT	Y_ISH	A_SRA	R_RD	M_IF	E_IBE
srav-e	X_RT	Y_AD	A_SRA	R_RD	M_IF	E_IBE
srl-e	X_RT	Y_ISH	A_SRL	R_RD	M_IF	E_IBE
srlv-e	X_RT	Y_AD	A_SRL	R_RD	M_IF	E_IBE
sub-e	X_RS	Y_AD	A_DIF	R_AD	M_NOP	E_OVF
subl-e	X_C0	Y_AD	A_SUM	R_RD	M_IF	E_IBE
subu-e	X_RS	Y_AD	A_DIF	R_RD	M_IF	E_IBE
sw-e	X_RS	Y_I16	A_SUM	R_AD	M_NOP	E_WW
swl-e	X_RT	Y_C0	A_SUM	R_NOP	M_WW	E_DBE
syscall-e	X_C0	Y_C0	A_SUM	R_NOP	M_NOP	E_SYS

H) VERSION AVEC MEMOIRE DE MICRO-INSTRUCTIONS

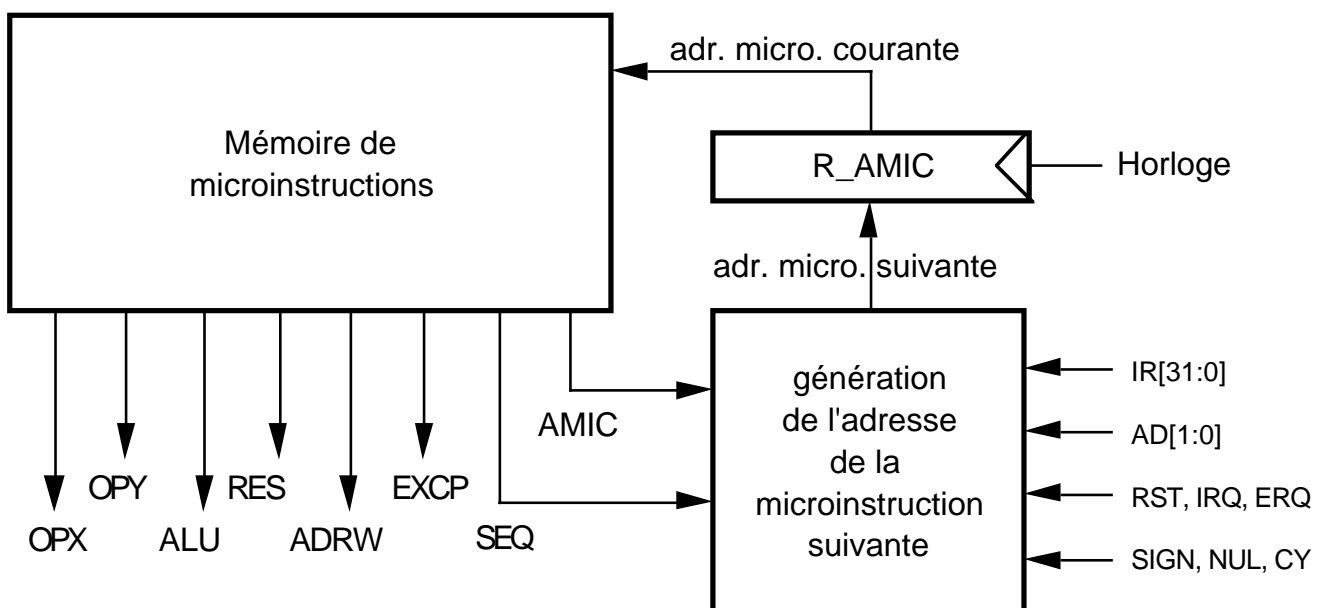
Dans cette version les micro-instructions comportent deux champs supplémentaires SEQ et AMIC, qui sont exploités par la partie contrôle.

OPX	OPY	ALU	RES	ADRW	EXCP	SEQ	AMIC
-----	-----	-----	-----	------	------	-----	------

- Le champ SEQ définit le calcul de l'adresse de la micro-instruction suivante.

- Le champ AMIC fournit une valeur sur 10 bits, utilisée pour le calcul de l'adresse de la micro-instruction suivante.

La partie contrôle comporte une mémoire contenant les micro-instructions, un registre contenant l'adresse de la micro-instruction à exécuter, et un bloc combinatoire calculant l'adresse de la micro-instruction suivante en fonction des champs SEQ, AMIC et des signaux de compte rendu venant de la partie opérative.



SEQUENCEMENT : CHAMP SEQ

• S_NEXT	R_AMIC	<---	AMIC[9:0]
• S_DECOD	R_AMIC	<---	01 IR[31:26] 00
• S_SPEC	R_AMIC	<---	10 IR[5:0] 00
• S_MOVC0	R_AMIC	<---	AMIC[9:4] IR[14:11]
• S_RIE	R_AMIC	<---	AMIC[9:3] RST IRQ ERQ
• S_COPRO	R_AMIC	<---	AMIC[9:2] IR[25] IR[23]
• S_BCOND	R_AMIC	<---	AMIC[9:2] IR[20] IR[16]

- S_AD10 R_AMIC <--- AMIC[9:2] || AD[1] || AD[0]
- S_SINUL R_AMIC <--- AMIC[9:2] || SIGN || NUL
- S_AD1 R_AMIC <--- AMIC[9:1] || AD[1]
- S_NUL R_AMIC <--- AMIC[9:1] || NUL
- S_SIGN R_AMIC <--- AMIC[9:1] || SIGN
- S_CY R_AMIC <--- AMIC[9:1] || CY
- S_UNSI R_AMIC <--- AMIC[9:1] || IR[28]

Le listing suivant donne une implémentation du microprogramme dans la mémoire de micro-instructions. Chaque micro-instruction comporte les éléments suivants :

- adresse de la micro-instruction
- nom de l'état correspondant de l'automate, précédé de "P_"
- les huit champs de la micro-instruction

Les commentaires commencent par "//".

Un même état de l'automate peut être représenté par plusieurs micro-instructions identiques.

```
// sequence Reset
0,P_reset,X_CZ,Y_C0,A_SUM,R_PC ,M_NOP,E_CLR,S_NEXT,1; // reset
1,P_reset1,X_C0,Y_C0,A_SUM,R_SR,M_NOP,E_NOP,S_NEXT,2; // reset1
2,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
//
15,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF, E_IBE,S_RIE,16; // ifetch-e
16,P_init,X_PC,Y_C4,A_SUM,R_PC ,M_NOP,E_NOP,S_DECOD,0; // init, decodage IR[31:26]
// interruption, exception
17,P_erq,X_C0,Y_C0,A_SUM,R_ERQ,M_NOP,E_CLR,S_NEXT,25; // erq
18,P_irq,X_C0,Y_C0,A_SUM,R_ERQ,M_NOP,E_NOP,S_NEXT,24; // irq
19,P_irq_erq,X_C0,Y_C0,A_SUM,R_ERQ,M_NOP,E_CLR,S_NEXT,25; // irq et erq
// sequence Reset
20,P_reset,X_CZ,Y_C0,A_SUM,R_PC ,M_NOP,E_CLR,S_NEXT,1; // reset
21,P_reset,X_CZ,Y_C0,A_SUM,R_PC ,M_NOP,E_CLR,S_NEXT,1; // reset
22,P_reset,X_CZ,Y_C0,A_SUM,R_PC ,M_NOP,E_CLR,S_NEXT,1; // reset
23,P_reset,X_CZ,Y_C0,A_SUM,R_PC ,M_NOP,E_CLR,S_NEXT,1; // reset
//
24,P_epc1,X_PC,Y_C0,A_SUM,R_EPC,M_NOP,E_NOP,S_NEXT,26; // epc1
25,P_epc2,X_PC,Y_C4,A_DIF,R_EPC,M_NOP,E_NOP,S_NEXT,26; // epc2
26,P_pct,X_C0,Y_CX,A_SUM,R_PC ,M_NOP,E_NOP,S_NEXT,27; // pct
27,P_ifetch1-e,X_C0,Y_AD,A_SUM,R_BAR,M_IF, E_IBE,S_RIE,16; // ifetch1-e
//-----
31,P_branch,X_AD,Y_C0,A_SUM,R_PC,M_NOP ,E_NOP,S_NEXT,15; // branch
//----- suite BEQ -----
32,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF, E_IBE,S_RIE,16; // ifetch-e
```

```
33,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
//----- suite BNE -----
34,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
35,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
//----- suite BLTZ -----
36,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
37,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
38,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
39,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
//----- suite BGEZ -----
40,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
41,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
42,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
43,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e No use
//----- suite BLEZ -----
44,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
45,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
46,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
47,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
//----- suite BGTZ -----
48,P_branch-e,X_PC,Y_I18,A_SUM,R_AD,M_NOP,E_IF,S_NEXT,31; // branch-e
49,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
50,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
51,P_ifetch-e,X_C0,Y_C0,A_SUM,R_NOP,M_IF,E_IBE,S_RIE,16; // ifetch-e
//----- suite LB/LBU -----
52,P_lbad00,X_DT,Y_C24,A_SLL,R_AD,M_NOP,E_NOP,S_UNSI,56; // lbad00
53,P_lbad01,X_DT,Y_C16,A_SLL,R_AD,M_NOP,E_NOP,S_UNSI,56; // lbad01
54,P_lbad10,X_DT,Y_C8,A_SLL,R_AD,M_NOP,E_NOP,S_UNSI,56; // lbad10
55,P_lbad11,X_DT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_UNSI,56; // lbad11
56,P_lb2-e,X_AD,Y_C24,A_SRA,R_RT,M_IF,E_IBE,S_RIE,16; // lb2-e
57,P_lbu2-e,X_AD,Y_C24,A_SRL,R_RT,M_IF,E_IBE,S_RIE,16; // lbu2-e
//----- suite LH/LHU -----
58,P_lhad0,X_DT,Y_C16,A_SLL,R_AD,M_NOP,E_NOP,S_UNSI,60; // lhad0
59,P_lhad1,X_DT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_UNSI,60; // lhad1
60,P_lh2-e,X_AD,Y_C16,A_SRA,R_RT,M_IF,E_IBE,S_RIE,16; // lh2-e
61,P_lhu2-e,X_AD,Y_C16,A_SRL,R_RT,M_IF,E_IBE,S_RIE,16; // lhu2-e
//-- COPRO -----
//IR[25,23] = 00, => @ 128 MFC0
128,P_mfco-e,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_CPU,S_MOVC0,144; // mfco-e
// decodage MOVC0
```

```
// IR[25,23] = 01, => @ 129  MTC0
129,P_mtco-e,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_CPU,S_MOVC0,160; // mtco-e
                                                                    // decodage MOVC0

// IR[25,23] = 1X, => @ 136  RFE
130,P_rfe-e,X_C0,Y_C0,A_SUM,R_RFE,M_NOP,E_CPU,S_NEXT,15; // rfe-e
131,P_rfe-e,X_C0,Y_C0,A_SUM,R_RFE,M_NOP,E_CPU,S_NEXT,15; // rfe-e
//----- suite MFC0 -----
144,P_ico-e,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_RI ,S_NEXT,15; // ico-e
152,P_bar2i-e,X_BAR,Y_C0,A_SUM,R_RT,M_IF, E_IBE,S_RIE,16; // bar2i-e (@8)
156,P_sr2i-e,X_SR ,Y_C0,A_SUM,R_RT,M_IF, E_IBE,S_RIE,16; // sr2i-e (@12)
157,P_cr2i-e,X_CR ,Y_C0,A_SUM,R_RT,M_IF, E_IBE,S_RIE,16; // cr2i-e (@13)
158,P_epc2i-e,X_EPC,Y_C0,A_SUM,R_RT,M_IF, E_IBE,S_RIE,16; // epc2i-e (@14)
//----- suite MTC0 -----
160,P_ico-e,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_RI ,S_NEXT,15; // ico-e
168,P_i2bar-e,X_RT,Y_C0,A_SUM,R_BAR,M_IF, E_IBE,S_RIE,16; // i2bar-e (@8)
172,P_i2sr-e,X_RT ,Y_C0,A_SUM,R_SR,M_IF, E_IBE,S_RIE,16; // i2sr-e (@12)
173,P_i2cr-e,X_RT ,Y_C0,A_SUM,R_CR,M_IF, E_IBE,S_RIE,16; // i2cr-e (@13)
174,P_i2epc-e,X_RT,Y_C0,A_SUM,R_EPC,M_IF, E_IBE,S_RIE,16; // i2epc-e (@14)
//-- BCOND -----
// IR[20,16] = 00, => @ 192  BLTZ
192,P_bltz,X_RS,Y_C0,A_DIF,R_NOP,M_NOP,E_NOP,S_SINUL,36; // bltz
// IR[20,16] = 01, => @ 196  BGEZ
193,P_bgez,X_RS,Y_C0,A_DIF,R_NOP,M_NOP,E_NOP,S_SINUL,40; // bgez
// IR[20,16] = 10, => @ 200  BLTZAL
194,P_bltzal,X_PC,Y_C0,A_SUM,R_R31,M_NOP,E_NOP,S_NEXT,192; // bltzal
// IR[20,16] = 11, => @ 204  BGEZAL
195,P_bgezal,X_PC,Y_C0,A_SUM,R_R31,M_NOP,E_NOP,S_NEXT,193; // bgezal
//-- OPCOD -----
// IR[31:26] = 000000, => @ 256  SPECIAL
256,P_special,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_NOP,S_SPEC,0; // special, decodage
// IR[31:26] = 000001, => @ 260  BCOND
260,P_bcond,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_NOP,S_BCOND,192; // bcond, decodage
// IR[31:26] = 000010, => @ 264  J
264,P_j-e,X_PC4,Y_IU28,A_OR,R_AD,M_NOP,E_IF,S_NEXT,31; // j-e
// IR[31:26] = 000011, => @ 268  JAL
268,P_jal,X_PC,Y_C0,A_SUM,R_R31,M_NOP,E_NOP,S_NEXT,264; // jal
// IR[31:26] = 000100, => @ 272  BEQ
272,P_reg1,X_RT,Y_C0,A_SUM,R_AD ,M_NOP,E_NOP,S_NEXT,273; // reg1
273,P_beq,X_RS,Y_AD,A_DIF,R_NOP,M_NOP,E_NOP,S_NUL,32; // beq
```

```
// IR[31:26] = 000101, => @ 276  BNE
276,P_reg1,X_RT,Y_C0,A_SUM,R_AD ,M_NOP,E_NOP,S_NEXT,277;    // reg1
277,P_bne,X_RS,Y_AD,A_DIF,R_NOP,M_NOP,E_NOP,S_NUL,34; // bne
// IR[31:26] = 000110, => @ 280  BLEZ
280,P_blez,X_RS,Y_C0,A_DIF,R_NOP,M_NOP,E_NOP,S_SINUL,44;    // blez
// IR[31:26] = 000111, => @ 284  BGTZ
284,P_bgtz,X_RS,Y_C0,A_DIF,R_NOP,M_NOP,E_NOP,S_SINUL,48;    // bgtz
// IR[31:26] = 001000, => @ 288  ADDI
288,P_imd,X_C0,Y_I16,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,289;    // imd
289,P_addi-e,X_RS,Y_AD,A_SUM,R_AD,M_NOP,E_OVF,S_NEXT,290;    // addi-e
290,P_addi1-e,X_C0,Y_AD,A_SUM,R_RT,M_IF, E_IBE,S_RIE,16;    // addi1-e
// IR[31:26] = 001001, => @ 292  ADDIU
292,P_imd,X_C0,Y_I16,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,293;    // imd
293,P_addiu-e,X_RS,Y_AD,A_SUM,R_RT,M_IF, E_IBE,S_RIE,16;    // addiu-e
// IR[31:26] = 001010, => @ 296  SLTI
296,P_imd,X_C0,Y_I16,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,297;    // imd
297,P_slti-e,X_RS,Y_AD ,A_SLT,R_RT,M_IF, E_IBE,S_RIE,16;    // slti-e
// IR[31:26] = 001011, => @ 300  SLTIU
300,P_imd,X_C0,Y_I16,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,301;    // imd
301,P_sltiu-e,X_RS,Y_AD ,A_SLTU,R_RT,M_IF, E_IBE,S_RIE,16; // sltiu-e
// IR[31:26] = 001100, => @ 304  ANDI
304,P_imdu,X_CH,Y_I16,A_AND,R_AD,M_NOP,E_NOP,S_NEXT,305;    // imdu
305,P_landi-e,X_RS,Y_AD ,A_AND,R_RT,M_IF, E_IBE,S_RIE,16;    // landi-e
// IR[31:26] = 001101, => @ 308  ORI
308,P_imdu,X_CH,Y_I16,A_AND,R_AD,M_NOP,E_NOP,S_NEXT,309;    // imdu
309,P_lori-e,X_RS,Y_AD ,A_OR ,R_RT,M_IF, E_IBE,S_RIE,16;    // lori-e
// IR[31:26] = 001110, => @ 312  XORI
312,P_imdu,X_CH,Y_I16,A_AND,R_AD,M_NOP,E_NOP,S_NEXT,313;    // imdu
313,P_lxori-e,X_RS,Y_AD ,A_XOR,R_RT,M_IF, E_IBE,S_RIE,16;    // lxori-e
// IR[31:26] = 001111, => @ 316  LUI
316,P_imdu,X_CH,Y_I16,A_AND,R_AD,M_NOP,E_NOP,S_NEXT,317;    // imdu
317,P_lui-e,X_AD,Y_C16,A_SLL,R_RT,M_IF, E_IBE,S_RIE,16;    // lui-e
// IR[31:26] = 010000, => @ 320  COPRO
320,P_copro,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_NOP,S_COPRO,128; // copro, decodage
// IR[31:26] = 011110, => @ 376  SGTI
//
// IR[31:26] = 100000, => @ 384  LB
384,P_lb-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_LB ,S_NEXT,385;    // lb-e
385,P_lb1-e,X_C0,Y_C0,A_SUM,R_NOP,M_LW,E_DBE,S_AD10,52;    // lb1-e
```

```
// IR[31:26] = 100001, => @ 388  LH
388,P_lh-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_LH ,S_NEXT,389; // lh-e
389,P_lh1-e,X_C0,Y_C0,A_SUM,R_NOP,M_LW, E_DBE,S_AD1,58; // lh1-e
// IR[31:26] = 100011, => @ 396  LW
396,P_lw-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_LW ,S_NEXT,397; // lw-e
397,P_lw1-e,X_C0,Y_C0,A_SUM,R_NOP,M_LW, E_DBE,S_NEXT,398; // lw1-e
398,P_lw2-e,X_C0,Y_DT,A_SUM,R_RT ,M_IF, E_IBE,S_RIE,16; // lw2-e
// IR[31:26] = 100100, => @ 400  LBU
400,P_lb-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_LB ,S_NEXT,385; // lb-e
// IR[31:26] = 100101, => @ 404  LHU
404,P_lh-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_LH ,S_NEXT,389; // lh-e
//
// IR[31:26] = 101000, => @ 416  SB
416,P_sb-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_WB ,S_NEXT,417; // sb-e
417,P_sb1-e,X_RT,Y_C0,A_SUM,R_NOP,M_WB, E_DBE,S_NEXT,15; // sb1-e
// IR[31:26] = 101001, => @ 420  SH
420,P_sh-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_WH ,S_NEXT,421; // sh-e
421,P_sh1-e,X_RT,Y_C0,A_SUM,R_NOP,M_WH, E_DBE,S_NEXT,15; // sh1-e
// IR[31:26] = 101011, => @ 428  SW
428,P_sw-e,X_RS,Y_I16,A_SUM,R_AD,M_NOP,E_WW ,S_NEXT,429; // sw-e
429,P_sw1-e,X_RT,Y_C0,A_SUM,R_NOP,M_WW, E_DBE,S_NEXT,15; // sw1-e
//-- SPECIAL -----
// IR[ 5: 0] = 000000, => @ 512  SLL
512,P_sll-e,X_RT,Y_ISH,A_SLL,R_RD,M_IF, E_IBE,S_RIE,16; // sll-e
// IR[ 5: 0] = 000010, => @ 520  SRL
520,P_srl-e,X_RT,Y_ISH,A_SRL,R_RD,M_IF, E_IBE,S_RIE,16; // srl-e
// IR[ 5: 0] = 000011, => @ 524  SRA
524,P_sra-e,X_RT,Y_ISH,A_SRA,R_RD,M_IF, E_IBE,S_RIE,16; // sra-e
// IR[ 5: 0] = 000100, => @ 528  SLLV
528,P_reg2,X_RS,Y_C0,A_SUM,R_AD ,M_NOP,E_NOP,S_NEXT,529; // reg2
529,P_sllv-e,X_RT,Y_AD,A_SLL,R_RD ,M_IF, E_IBE,S_RIE,16; // sllv-e
// IR[ 5: 0] = 000110, => @ 536  SRLV
536,P_reg2,X_RS,Y_C0,A_SUM,R_AD ,M_NOP,E_NOP,S_NEXT,537; // reg2
537,P_srlv-e,X_RT,Y_AD,A_SRL,R_RD ,M_IF, E_IBE,S_RIE,16; // srlv-e
// IR[ 5: 0] = 000111, => @ 540  SRAV
540,P_reg2,X_RS,Y_C0,A_SUM,R_AD ,M_NOP,E_NOP,S_NEXT,541; // reg2
541,P_srav-e,X_RT,Y_AD,A_SRA,R_RD ,M_IF, E_IBE,S_RIE,16; // srav-e
// IR[ 5: 0] = 001000, => @ 544  JR
544,P_jr-e,X_RS,Y_C0,A_SUM,R_AD,M_NOP,E_IF ,S_NEXT,31; // jr-e
```

```
// IR[ 5: 0] = 001001, => @ 548  JALR
548,P_jalr,X_PC,Y_C0,A_SUM,R_RD,M_NOP,E_NOP,S_NEXT,544;    // jalr
// IR[ 5: 0] = 001100, => @ 560  SYSCALL
560,P_syscall-e,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_SYS,S_NEXT,15;// syscall-e
// IR[ 5: 0] = 001101, => @ 564  BREAK
564,P_brk-e,X_C0,Y_C0,A_SUM,R_NOP,M_NOP,E_BRK,S_NEXT,15;    // brk-e
//
// IR[ 5: 0] = 010000, => @ 576  MFHI
576,P_mfhi-e,X_HI,Y_C0,A_SUM,R_RD ,M_IF, E_IBE,S_RIE,16;    // mfhi-e
// IR[ 5: 0] = 010001, => @ 580  MTHI
580,P_mthi-e,X_RS,Y_C0,A_SUM,R_HI ,M_IF, E_IBE,S_RIE,16;    // mthi-e
// IR[ 5: 0] = 010010, => @ 584  MFLO
584,P_mflo-e,X_LO,Y_C0,A_SUM,R_RD ,M_IF, E_IBE,S_RIE,16;    // mflo-e
// IR[ 5: 0] = 010011, => @ 588  MTLO
588,P_mtlo-e,X_RS,Y_C0,A_SUM,R_LO ,M_IF, E_IBE,S_RIE,16;    // mtlo-e
//
// IR[ 5: 0] = 011000, => @ 608  MULT
608,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,609;    // AD <- RT
609,P_muls1,X_RS,Y_AD,A_MSL,R_LO,M_NOP,E_NOP,S_NEXT,610;    // LO <- (RS*RT) and
                                                                0xFFFFFFFF
610,P_mulsh-e,X_RS,Y_AD,A_MSH,R_HI,M_IF,E_IBE,S_RIE,16;    // HI <- (RS*RT) >> 32,
                                                                IR < M(PC)

// IR[ 5: 0] = 011001, => @ 612  MULTU
612,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,613;    // AD <- RT
613,P_mulul,X_RS,Y_AD,A_MUL,R_LO,M_NOP,E_NOP,S_NEXT,614;    // LO <- (RS*RT) and
                                                                0xFFFFFFFF
614,P_muluh-e,X_RS,Y_AD,A_MUH,R_HI,M_IF,E_IBE,S_RIE,16;    // HI <- (RS*RT) >> 32,
                                                                IR < M(PC)

// IR[ 5: 0] = 011010, => @ 616  DIV
616,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,617;    // AD <- RT
617,P_divsq,X_RS,Y_AD,A_DSQ,R_LO,M_NOP,E_NOP,S_NEXT,618;    // LO <- Quo(RS/RT)
618,P_divsr-e,X_RS,Y_AD,A_DSR,R_HI,M_IF,E_IBE,S_RIE,16;    // HI <- Rem(Rs/Rt) ,
                                                                IR < M(PC)

// IR[ 5: 0] = 011011, => @ 620  DIVU
620,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,621;    // AD <- RT
621,P_divuq,X_RS,Y_AD,A_DUQ,R_LO,M_NOP,E_NOP,S_NEXT,622;    // LO <- Quo(RS/RT)
622,P_divur-e,X_RS,Y_AD,A_DUR,R_HI,M_IF,E_IBE,S_RIE,16;    // HI <- Rem(Rs/Rt) ,
                                                                IR < M(PC)

//
```

```
// IR[ 5: 0] = 100000, => @ 640  ADD
640,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,641;    // reg1
641,P_add-e,X_RS,Y_AD,A_SUM,R_AD,M_NOP,E_OVF,S_NEXT,642;    // add-e
642,P_addl-e,X_C0,Y_AD,A_SUM,R_RD,M_IF, E_IBE,S_RIE,16;      // addl-e
// IR[ 5: 0] = 100001, => @ 644  ADDU
644,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,645;    // reg1
645,P_addu-e,X_RS,Y_AD,A_SUM,R_RD,M_IF, E_IBE,S_RIE,16;      // addu-e
// IR[ 5: 0] = 100010, => @ 648  SUB
648,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,649;    // reg1
649,P_sub-e,X_RS,Y_AD,A_DIF,R_AD,M_NOP,E_OVF,S_NEXT,650;    // sub-e
650,P_subl-e,X_C0,Y_AD,A_SUM,R_RD,M_IF, E_IBE,S_RIE,16;      // subl-e
// IR[ 5: 0] = 100011, => @ 652  SUBU
652,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,653;    // reg1
653,P_subu-e,X_RS,Y_AD,A_DIF,R_RD,M_IF, E_IBE,S_RIE,16;      // subu-e
// IR[ 5: 0] = 100000, => @ 656  AND
656,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,657;    // reg1
657,P_land-e,X_RS,Y_AD,A_AND,R_RD,M_IF, E_IBE,S_RIE,16;      // land-e
// IR[ 5: 0] = 100001, => @ 660  OR
660,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,661;    // reg1
661,P_lor-e,X_RS,Y_AD,A_OR ,R_RD,M_IF, E_IBE,S_RIE,16;      // lor-e
// IR[ 5: 0] = 100010, => @ 664  XOR
664,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,665;    // reg1
665,P_lxor-e,X_RS,Y_AD,A_XOR,R_RD,M_IF, E_IBE,S_RIE,16;      // lxor-e
// IR[ 5: 0] = 100011, => @ 668  NOR
668,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,669;    // reg1
669,P_lnor-e,X_RS,Y_AD,A_NOR,R_RD,M_IF, E_IBE,S_RIE,16;      // lnor-e
//
// IR[ 5: 0] = 101010, => @ 680  SLT
680,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,681;    // reg1
681,P_slt-e,X_RS,Y_AD,A_SLT,R_RD,M_IF, E_IBE,S_RIE,16;      // slt-e
// IR[ 5: 0] = 101011, => @ 684  SLTU
684,P_reg1,X_RT,Y_C0,A_SUM,R_AD,M_NOP,E_NOP,S_NEXT,685;    // reg1
685,P_sltu-e,X_RS,Y_AD,A_SLTU,R_RD,M_IF, E_IBE,S_RIE,16;    // sltu-e
```