

ASM, LE manuel

Volume III

Les mnémoniques

Version **1999.5.16 BETA**

Par Sébastien Munch (BBossByll), bbossbyll@i-france.com

Ce document est un volume du manuel intitulé "ASM, LE manuel". Il ne peut être distribué que conjointement au reste de ce manuel.

Sommaire

- A. Registres
- B. Champs
- C. Explications du classement
- D. Les différentes versions du Saturn
- E. Symboles et notations utilisés
- F. Conseils et astuces
- G. Mnémoniques
 - Instructions de saut
 - Instructions de sous-programmes
 - Instructions de test
 - Tests des registres
 - Tests du pointeur P
 - Tests des drapeaux
 - Tests des bits Hardware Status
 - Tests des bits des registres
 - Instructions de pointeur
 - Manipulation de bits
 - Instructions de statut
 - Drapeaux
 - Hardware status*
 - Etat système

Ce volume liste les mnémoniques du processeur Saturn.

Source:

La documentation de SASM (assembleur Saturn sur PC)

Guide to the Saturn processor par Matthew Mastracci

A. Registres

Registres de travail:

Par ordre d'importance: *C, A, B, D*: 64 bits; divisés en champs (voir ci-dessous)

Registres de sauvegarde:

R0, R1, R2, R3, R4: 64 bits

Pointeurs:

D0, D1: 20 bits (pointeurs mémoire); Le contenu de l'adresse pointée est lisible à *DAT0* et *DAT1*

P: 4 bits (pointeur registres)
PC: pointeur programme (program counter)

Entrée/sortie:
IN: 16 bits
OUT: 12 bits

Bits:

C: Carry

HST: Hardware status bits (voir plus bas)

ST: 16 drapeaux (program status bits) (ne pas utiliser les drapeaux C à F: ils sont utilisés par la routine de traitement des interruptions) Note: Le flag F sert à interdire les interruptions.

On peut utiliser les drapeaux 0 à B comme un seul registre 12 bits (instructions $C=ST$, $ST=C$, $CLRST$). Pour raison de commodité, je noterai ce "registre" $ST(R)$. Rappelez-vous que ce n'est qu'une notation de mon invention.

Hardware Status bits

Les bits "hardware status" sont activés, mais pas désactivés, par des événements liés au matériel.

Ils doivent pas conséquemment être remis à 0 avant de pouvoir être lus.

Fonctions de chaque bit:

MP (*module pulled*) est activé chaque fois que "*NINTX CPU" est désactivé. (Que celui qui comprend ça me contacte...)

SB (*sticky bit*) est activé quand un bit ou un quartet non-nul est décalé à droite, lors d'une opération... de décalage

SR (*service request*) est activé comme résultat de l'instruction $SREQ?$... euh... pas bien compris... "if any hardware service request is pending"... (merci de bien vouloir m'aider)

XM (*external module missing*) est activé à l'exécution de l'instruction 00h / $RTNSXM$

Hardware status:

Bit	Symbole	Nom
3	MP	Module pulled
2	SR	Service request
1	SB	Sticky bit
0	XM	External module missing

B. Champs

Cette notion est à connaître dès que l'on veut assembler sur Saturn.

Par rapport aux autres processeurs de sa génération, le Saturn a un avantage certain: ses registres sont de 64 bits.

Par contre, il devient très difficile de manipuler ces registres dans leur intégralité (vous me comprendrez).

Ils ont été par conséquent subdivisés en champs, dont voici la liste:

Champ	Nom et description	Opcode (a)	Opcode (b)	Nombre de quartets (d)
P	Pointer: Pointeur. Quartet spécifié par le registre P (0 à F)	0	8	1
WP	Champ allant du quartet spécifié par P jusqu'au quartet 0	1	9	P+1
XS	Exponent sign: Signe de l'exposant	2	A	1
X	Exponent: Exposant	3	B	3
S	Sign: Signe	4	C	1
M	Mantissa: Mantisse	5	D	12
B	Byte: Octet	6	E	2
W	Word: Mot (registre entier)	7	F	16
A	Address: Adresse	F	-	5

Diagramme des champs (excepté P et WP) dans un registre (A, B, C ou D):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
S	M	M	M	M	M	M	M	M	M	M	M	M	X	X	X
											A	A	XS	B	B
													A	A	A

C. Explication du classement

Les mnémoniques sont classés par catégories de fonctions.

Ils sont détaillés dans un tableau à six colonnes: *mnémonique*, *opérande*, *fonction*, *code opération*, *nombre de cycles* et *commentaires*.

La première colonne donne le nom officiel du mnémonique, utilisé dans tous les assembleurs.

La deuxième colonne donne le type d'opérande utilisée avec l'instruction (voir paragraphe suivant)

La troisième commande explique à quoi sert l'instruction

La quatrième donne le code opération (Opcode) de l'instruction, c'est-à-dire ce à quoi elle correspond en langage machine.

La cinquième donne le nombre de cycles utilisés par l'instruction (pour éventuellement calculer le temps d'exécution d'un programme, et aussi pour pouvoir optimiser sa vitesse, en utilisant certaines instruction plutôt que d'autres.

Et enfin la dernière colonne donne des commentaires sur l'instruction.

D. Les différentes versions du Saturn

Le Saturn a connu trois différentes versions:

1LF2:

01/02/1984 HP71B (Premières versions)

1LK7:

??/??/??? HP71B (Versions suivantes)

01/06/1986 HP18C

05/01/1987 HP28C

1LT8:

04/01/1988 HP17B Lewis

04/01/1988 HP19B Lewis

04/01/1988 HP27S Lewis

04/01/1988 HP28S Lewis

16/03/1991 HP48SX Clarke

02/04/1991 HP48S Clarke

01/06/1993 HP48GX Yorke

01/06/1993 HP48G Yorke

??/09/1995 HP38G Yorke

??/??/1998 HP48G+ Yorke

Les instructions présentes sur toutes les versions sont appelées instructions de niveau 0

Les instructions absentes du 1LF2 sont appelées instructions de niveau 1

Les instructions spécifiques au 1LT8 sont appelées instructions de niveau 2

J'indiquerai quand les instructions sont de niveau 1 ou 2

E. Symboles et notations utilisés

J'utiliserai dans cette liste de mnémoniques un certain nombre de symboles.

En voici la liste:

chp Symbole d'un champ

d Nombre de quartets d'un champ (voir tableau des champs)

h Un chiffre / caractère hexadécimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E ou F

label Le nom d'un label présent dans le programme en cours d'exécution. Un label permet passer de l'instruction en cours d'exécution à une instruction placée complètement autre part. Au moment d'assembler le programme, l'assembleur traduit les opérandes "label" en l'adresse visée (saut absolu), ou en le nombre d'adresses séparant l'instruction courante et l'instruction visée (saut relatif). Si vous entriez vous-même l'instruction en ML (avec son opcode) dans la machine, il vous faudrait compter le nombre d'adresse séparant les instructions, sachant que, chaque adresse étant en 4 bits, une adresse correspond à un chiffre hexadécimal. (Par exemple, l'instruction *GOTO* entière s'étend sur 4 adresses. (4 caractères hexa. composent sont opcode).

Les *h* dans les opcodes correspondent TOUJOURS à la valeur des opérandes des instructions.

(x)y veut dire "le champ y du registre x"

C correspond à *carry* et non pas au registre C

Pour simplifier l'écriture des nombres, j'utiliserai la syntaxe de la HP lorsque cela sera ambigu, c'est-à-dire le nombre suivi de h au cas d'une notation hexa, et suivi de b au cas d'une notation binaire.

Vocabulaire: j'utilise *activer* pour *mettre à 1* et *désactiver* pour *mettre à 0*. Si vous avez des mots plus adaptés, prévenez-moi.

F. Conseils et astuces

- Programmez proprement (alignez les instructions sur la même colonne par exemple)
- Abusez des commentaires pour vos sources (de toute façon les commentaires ne sont pas codés avec le programme, donc ils n'alourdissent que la source, et pas le programme lui-même). Vous trouverez des informations à propos des commentaires dans les documentations des assembleurs...
- Un truc à faire bien attention:
Le Saturn peut aussi bien fonctionner en décimal qu'en hexadécimal. Méfiez-vous et faites attention.
Personnellement, je tente de n'utiliser QUE l'hexa, car certaines instructions fonctionnent de toute façon en hexa. Alors tant qu'à faire, autant TOUT faire en hexa...

G. Mnémoniques

Mnémonique	Opérande	Fonction	opcode	Nombre de cycles	Commentaires
------------	----------	----------	--------	------------------	--------------

Instructions de saut

Un saut relatif est un saut où l'on point non pas vers une adresse, mais où l'on spécifie une longueur de saut.

Un saut absolu est un saut où est directement déterminée l'adresse pointée.

GOTO	label	Saut relatif inconditionnel	6hhh	11	adr-2048<label<adr+2047
GOC	label	Saut relatif si la retenue est à 1	4hh	10 (si C=1) 3 (si C=0)	adr-128<label<adr+127
GONC	label	Saut relatif si la retenue est à 0	5hh	10 (si C=0) 3 (si C=1)	adr-128<label<adr+127
GOLONG	label	Saut relatif long inconditionnel	8Chhhh	14	adr-32768<label<adr+32767
GOVLNG	label	Saut absolu	8Dhhhhh	14	
PC=(A)		Saut à l'adresse contenue dans l'adresse pointée par A(A)	808C	23	instruction de niveau 1
PC=(C)		Saut à l'adresse contenue dans l'adresse pointée par C(A)	?	?	instruction de niveau 2
PC=A		Saut à l'adresse contenue dans A(A)	?	?	instruction de niveau 2
PC=C		Saut à l'adresse contenue dans C(A)	?	?	instruction de niveau 2
APCEX		Sauvegarde de PC dans A(A) et saut à l'adresse contenue dans A(A)	?	?	instruction de niveau 2
CPCEX		Sauvegarde de PC dans C(A) et saut à l'adresse contenue dans C(A)	?	?	instruction de niveau 2
GOYES	label	Saut si le test donne un résultat juste	hh	inclus dans le test accompagné	Accompagne TOUJOURS un test. adr-128<label<adr+127

Instructions de sous-programmes

Un sous-programme est un bout de programme placé souvent à la fin du fichier, permettant de ne pas réécrire 36 fois la même suite d'instructions, et également de gagner de la place. L'adresse de laquelle le programme a été appelé est incrémentée au moment de l'appel, et stockée dans la *pile*. Cette pile n'a aucun rapport avec la pile de la HP que vous connaissez. Celle-ci sert, comme dit, à stocker les adresses desquelles les sous-programmes sont appelés. Il faut faire attention en appelant un sous-programme, car la plie ne contient que 8 niveaux, dont un déjà utilisé par la HP, ce qui veut dire que vous ne pouvez imbriquer que **7 appels de sous-programmes**. Les instructions de retour lisent dans la pile pour savoir où retourner, puis effacent la valeur lue. Vous devez également faire attention avec ces instructions de retour de sous-programme, car, si vous faites trop de retours, vous atteindrez le bout de la pile. La pile étant suivie de valeurs nulles, vous ferez à un moment ou à un autre un "retour" à l'adresse 00000h, et rappelez-vous que ce n'est rien d'autre que l'adresse de... *memory reset*.

Appel de sous-programme

GOSUB	label	Appel relatif	7hhh	12	adr-2048<label<adr+2047
GOSUBL	label	Appel relatif long	8Ehhhhh	15	adr-32768<label<adr+32767
GOSBVL	label	Appel absolu	8Fhhhhh	15	

Retour de sous-programme

RTN	Retour de sous-programme inconditionnel	01	9	
RTNSC	Retour de sous-programme et active C	02	9	
RTNCC	Retour de sous-programme et désactive C	03	9	
RTNSXM	Retour de sous-programme et active XM	00	9	
RTI	Retour de sous-programme et active les interruptions	0F	9	
RTNC	Retour de sous-programme si C=1	400	10 (si C=1) 3 (si C=0)	
RTNNC	Retour de sous-programme si C=0	500	10 (si C=0) 3 (si C=1)	
RTNYES	Retour de sous-programme si le résultat du test précédent est juste	00	inclus dans le test accompagné	Accompagne TOUJOURS un test.

Instructions de test

Toutes les instructions de test DOIVENT être suivies de l'instruction GOYES ou RTNYES. Si le résultat du test s'avère juste, le bit carry (C) est mis à 1 et mis à 0 si le résultat est faux. Les tests de comparaison ignorent le code complément à deux: ils testent en binaire naturel. Le test ne s'effectue que sur le champ sélectionné.

Les derniers *hh* (dans le opcode) est déterminé par le *GOYES* ou *RTNYES* suivant

Tests de registres

?A=B / ?B=A	chp	Teste si A(chp)=B(chp)	chp=A: 8A0hh chp≠A: 9A0hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B=C / ?C=B	chp	Teste si B(chp)=C(chp)	chp=A: 8A1hh chp≠A: 9A1hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A=C / ?C=A	chp	Teste si A(chp)=C(chp)	chp=A: 8A2hh chp≠A: 9A2hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C=D / ?D=C	chp	Teste si C(chp)=D(chp)	chp=A: 8A3hh chp≠A: 9A3hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A#B / ?B#A	chp	Teste si A(chp)≠B(chp)	chp=A: 8A4hh chp≠A: 9A4hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B#C / ?C#B	chp	Teste si B(chp)≠C(chp)	chp=A: 8A5hh chp≠A: 9A5hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A#C / ?C#A	chp	Teste si A(chp)≠C(chp)	chp=A: 8A6hh chp≠A: 9A6hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C#D / ?D#C	chp	Teste si C(chp)≠D(chp)	chp=A: 8A7hh chp≠A: 9A7hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A=0	chp	Teste si A(chp)=0	chp=A: 8A8hh chp≠A: 9A8hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B=0	chp	Teste si B(chp)=0	chp=A: 8A9hh chp≠A: 9A9hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C=0	chp	Teste si C(chp)=0	chp=A: 8AAhh chp≠A: 9AAhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?D=0	chp	Teste si D(chp)=0	chp=A: 8ABhh chp≠A: 9ABhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A#0	chp	Teste si A(chp)≠0	chp=A: 8ACHh chp≠A: 9ACHh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B#0	chp	Teste si B(chp)≠0	chp=A: 8ADhh chp≠A: 9ADhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C#0	chp	Teste si C(chp)≠0	chp=A: 8AEhh chp≠A: 9AEhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?D#0	chp	Teste si D(chp)≠0	chp=A: 8AFhh chp≠A: 9AFhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A>B	chp	Teste si A(chp)>B(chp)	chp=A: 8B0hh chp≠A: 9B0hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B>C	chp	Teste si B(chp)>C(chp)	chp=A: 8B1hh chp≠A: 9B1hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C>A	chp	Teste si C(chp)>A(chp)	chp=A: 8B2hh chp≠A: 9B2hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?D>C	chp	Teste si D(chp)>C(chp)	chp=A: 8B3hh chp≠A: 9B3hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A<B	chp	Teste si A(chp)<B(chp)	chp=A: 8B4hh chp≠A: 9B4hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B<C	chp	Teste si B(chp)<C(chp)	chp=A: 8B5hh chp≠A: 9B5hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C<A	chp	Teste si C(chp)<A(chp)	chp=A: 8B6hh chp≠A: 9B6hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?D<C	chp	Teste si D(chp)<C(chp)	chp=A: 8B7hh chp≠A: 9B7hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	

?A>=B	chp	Teste si A(chp)≥B(chp)	chp=A: 8B8hh chp≠A: 9B8hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B>=C	chp	Teste si B(chp)≥C(chp)	chp=A: 8B9hh chp≠A: 9B9hh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C>=A	chp	Teste si C(chp)≥A(chp)	chp=A: 8BAhh chp≠A: 9BAhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?D>=C	chp	Teste si D(chp)≥C(chp)	chp=A: 8BBhh chp≠A: 9BBhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?A<=B	chp	Teste si A(chp)≤B(chp)	chp=A: 8BChh chp≠A: 9BChh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?B<=C	chp	Teste si B(chp)≤C(chp)	chp=A: 8BDhh chp≠A: 9BDhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?C<=A	chp	Teste si C(chp)≤A(chp)	chp=A: 8BEhh chp≠A: 9BEhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	
?D<=C	chp	Teste si D(chp)≤C(chp)	chp=A: 8BFhh chp≠A: 9BFhh	13+d (si le résultat est juste) 6+d (si le résultat est faux)	

Tests du pointeur P

?P=	h	Teste si P=h	89hhh	13 (si le résultat est juste) 6 (si le résultat est faux)	le premier h du opcode est égal au h de l'opérande.
?P#	h	Teste si P≠h	88hhh	13 (si le résultat est juste) 6 (si le résultat est faux)	le premier h du opcode est égal au h de l'opérande.

Tests des drapeaux

?ST#1 ou ?ST=1	h	Teste si le drapeau h est égal à 0	86hhh	14 (si le résultat est juste) 7 (si le résultat est faux)	le premier h du opcode est égal au h de l'opérande.
?ST#0 ou ?ST=1	h	Teste si le drapeau h est égal à 1	87hhh	14 (si le résultat est juste) 7 (si le résultat est faux)	le premier h du opcode est égal au h de l'opérande.

Test des bits Hardware Status

?XM=0		Teste si XM=0	831hh	13 (si le résultat est juste) 6 (si le résultat est faux)	
?SB=0		Teste si SB=0	832hh	13 (si le résultat est juste) 6 (si le résultat est faux)	
?SR=0		Teste si SR=0	834hh	13 (si le résultat est juste) 6 (si le résultat est faux)	
?MP=0		Teste si MP=0	838hh	13 (si le résultat est juste) 6 (si le résultat est faux)	

Apparemment, comme pour les instructions de remise à 0 de ces bits, ces instructions sont toutes dérivées de l'opcode 83h.

(je ne l'ai pas encore vérifié...)

Donc, théoriquement, 833 devrait tester en même temps XM et SB, avec le même nombre de cycles qu'un seul test.

(A essayer: l'instruction: ?HS=0 h, qui pourrait correspondre à 83h)

Test des bits des registres

?ABIT=0 / ?ABIT#1	hhhh	Teste si A(h...)=0	?	?	instruction de niveau 2
?ABIT=1 / ?ABIT#0	ou hhh	Teste si A(h...)=1	?	?	instruction de niveau 2
?CBIT=0 / ?CBIT#1	ou hh	Teste si C(h...)=0	?	?	instruction de niveau 2
?CBIT=1 / ?CBIT#0	ou h	Teste si C(h...)=1	?	?	instruction de niveau 2

Instructions de pointeur

P=	h	Place h en P	2h	2	
P=P+1		Incrémente P	0C	3	Affecte la retenue
P=P-1		Décrémente P	0D	3	Affecte la retenue
C+P+1 ou C=C+P+1		Ajoute P+1 à C(A)	809	8	
CPEX	h	Echange P et C(h)	80Fh	6	
P=C	h	Copie C(h) dans P	80Dh	6	
C=P	h	copie P dans C(h)	80Ch	6	

Manipulation de bits

ABIT=0	hhhh	Met le bit h... de A à 0	?	?	instruction de niveau 2
ABIT=1	ou hhh	Met le bit h... de A à 1	?	?	instruction de niveau 2
CBIT=0	ou hh	Met le bit h... de C à 0	?	?	instruction de niveau 2
CBIT=1	ou h	Met le bit h... de C à 1	?	?	instruction de niveau 2

Instructions de statut

Drapeaux

ST=0	h	Active le drapeau h	84h	4	
ST=1	h	Désactive le drapeau h	85h	4	
CSTEX		Echange ST(R) et C(X)	0B	6	
C=ST		Copie ST(R) dans C(X)	09	6	
ST=C		Copie C(X) dans ST(R)	0A	6	
CLRST		Met ST(R) à 000h	08	6	

Hardware status

XM=0		Met XM à 0	821	3	
SB=0		Met SB à 0	822	3	
SR=0		Met SR à 0	824	3	
MP=0		Met MP à 0	828	3	
CLRHS		Met HST à 0	82F	3	

En fait, toutes ces instructions sont dérivées de 82h, qui est un "masque" pour dire quels bits désactiver. Donc, théoriquement, 823 devrait mettre XM et SB à 0 en même temps.

(A essayer: l'instruction: $HS=0$ h, qui pourrait être 82h)

Etat du système

[illegible]