TD – **Architecture de référence**

nollinge@cmi.univ-mrs.fr

année 2004/2005

☞ *Le présent document décrit l'architecture de référence utilisée en TD. Cette architecture simplifiée s'inspire d'architectures réelles. Ainsi le processeur Ant-32 s'inspire du processeur MIPS R1000.*

1 Vue d'ensemble

Une partie du système d'exploitation dépend de l'ordinateur sur lequel il s'exécute. D'une part parce que le système d'exploitation doit gérer les ressources comme la mémoire et les périphériques (disques, clavier, carte graphique, interface réseau, etc) et d'autre part parce que la mise en oeuvre de certains services (gestion des processus, de la mémoire virtuelle, des appels système, etc) demande une interaction forte avec le processeur.

Bien entendu, il serait trop complexe d'étudier une véritable architecture en détail. D'un autre côté, il peut être intéressant et formateur de se demander comment sont mis en oeuvre certains détails du fonctionnement des systèmes d'exploitation et comment ils sont rendus possibles par la structure des architectures modernes. Aussi, nous allons présenter une architecture simplifiée qui nous servira de référence lorsque nous aborderons ces aspects.

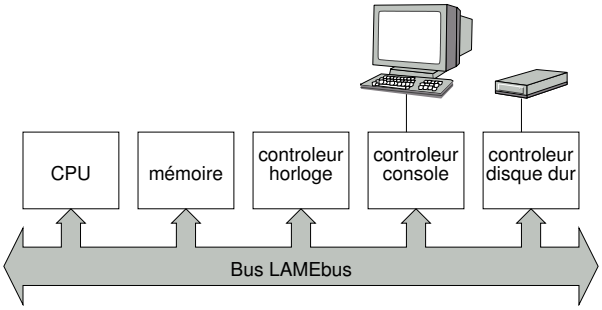

Fig. 1 – interaction entre les composants

Notre architecture de référence est composée d'un processeur Ant-32, d'une mémoire et d'un ensemble de contrôleurs de périphériques qui intéragissent par l'intermédiaire d'un bus de type LAMEbus, comme représenté sur la figure 1.

2 Le processeur Ant-32

2.1 Présentation générale

Le processeur Ant-32 est un processeur RISC moderne, inspiré du processeur MIPS R1000. C'est un processeur grand boutiste (*i.e.* big indian, octet de poids fort en tête) qui travaille avec des registres et des mots mémoire de 32 bits. Les adresses mémoire sont elles aussi représentées sur 32 bits. Les accès mémoire à des mots de 32 bits doivent être alignés (c'est-à-dire que les adresses utilisées doivent être divisibles par 4). Les instructions sont codées sur un mot mémoire dont l'octet de poids fort contient le numéro de l'instruction à exécuter.

Le processeur dispose d'un certain nombre de registres identifiés par une valeur sur 8 bits. Seuls 64 registres généraux, les registres `r0` à `r63`, sont disponibles pour programmer. Le registre `r0` est en lecture seule et contient la constante `0`. Une écriture dans ce registre est sans effet.

L'accès à la mémoire se fait par l'intermédiaire d'une unité de gestion de la mémoire virtuelle (*i.e.* une MMU) qui sera présentée en 2.3. Les adresses de la mémoire physique sont représentées sur 30 bits (soit 1Go maximum de mémoire).

Le processeur possède deux modes de fonctionnements : un mode superviseur qui est le mode de fonctionnement du noyau du système d'exploitation et du gestionnaire d'exceptions qui traite les comportements exceptionnels (division par zéro, accès à la mémoire impossible, etc) et un mode utilisateur qui est le mode de fonctionnement des programmes utilisateur et des outils du système d'exploitation. L'appel à un service du système d'exploitation se fait par passage en mode superviseur par le biais de l'instruction `trap` qui lève une exception.

L'interaction avec les périphériques a lieu en mode superviseur par l'intermédiaire de certaines zones de la mémoire. Les périphériques peuvent demander une interaction avec le système par le biais d'une interruption matérielle (*i.e.* IRQ) qui provoque la levée d'une exception.

L'état du processeur à un instant donné dépend : du contenu des registres, du compteur ordinal (ou pointeur d'instruction, qui désigne l'instruction en cours de décodage, noté `pc` dans la suite), de l'adresse du gestionnaire d'exception (`exh`), des drapeaux indiquant si les interruptions sont autorisées (`fli`), si les exceptions sont autorisées (`fle`) et si le processeur est en mode utilisateur (`flm`). Enfin le processeur dispose d'une horloge d'instruction (`timer`) qui, lorsqu'elle contient une valeur positive, est décrémentée à chaque exécution d'instruction et lève une exception lorsqu'elle atteint la valeur 0.

2.2 Gestion des exceptions

Lorsqu'une exception est levée, les deux drapeaux d'exception et d'interruption (`fle` et `fli`) sont mis à la valeur 1 et le contrôle est transféré à l'adresse du gestionnaire d'exception (modifiable par l'instruction `leh`). Si cette adresse n'est pas définie ou n'est pas correctement alignée le comportement du processeur n'est pas défini.

Les drapeaux d'exception et d'interruption se comprennent comme suit. Si le drapeau d'interruption est levé (valeur 1) alors les interruptions levées par le bus ne lèvent pas d'exception IRQ mais dès que ce drapeau est rabaissé (valeur 0), une exception IRQ sera levée, quelque soit l'état du bus à cet instant. Si le drapeau d'exception est levé et qu'une exception doit être levée, alors le processeur se réinitialise !

Pour traiter correctement l'exception, on peut s'aider des registres suivant qui ne sont accessibles qu'en mode superviseur :

- `k0` à `k1` (*i.e.* `r248` à `r251`) qui sont libres pour toute utilisation ;
- `e0` (*i.e.* `r252`) en lecture seule qui contient une copie de `pc` mise à jour à chaque cycle si le drapeau d'exception `fle` vaut 0 ;
- `e1` (*i.e.* `r253`) en lecture seule qui contient une copie de `fli` dans son bit 0 mise à jour à chaque cycle si le drapeau d'exception `fle` vaut 0 ;
- `e2` (*i.e.* `r254`) en lecture seule qui contient la dernière adresse envoyée à la MMU mise à jour si le drapeau d'exception `fle` vaut 0 ;
- `e3` (*i.e.* `r255`) en lecture seule qui contient des informations sur l'exception levée.

Le registre `e3` est composé de quatre bits de poids faible de drapeaux suivis de 28 bits codant l'exception levée. Les valeurs des drapeaux sont les suivantes :

SU bit 0, vaut 1 si le processeur était en mode superviseur au moment de la levée d'exception ;

IF bit 1, vaut 1 si l'exception a été levée lors du chargement d'une instruction par le processeur ;

DS bit 2, vaut 1 si l'exception a été levée lors du stockage d'une valeur en mémoire ;

DF bit 3, vaut 1 si l'exception a été levée lors de la lecture d'une valeur en mémoire.

Enfin, les numéros d'exceptions sont interprétés comme suit :

(1) IRQ la ligne d'interruption du bus est active ;

(3) bus error un accès à la mémoire physique a échoué ;

(4) illegal instruction erreur lors du décodage d'une instruction ;

(5) privileged instruction tentative d'exécution d'une instruction privilégiée en mode utilisateur ;

(6) trap appel système ;

(7) divide by zero division par zéro ;

(8) alignment error tentative d'accès mémoire non-aligné ;

(9) segment privilege tentative d'accès aux segments 1 à 3 en mode utilisateur ;

(10) register violation tentative d'écriture dans un registre en lecture seule (différent de `r0`) ou d'accès à un registre superviseur en mode utilisateur ;

(11) TLB miss pas de correspondance dans la TLB ;

(12) TLB protection opération mémoire incompatible avec les permissions déclarées dans la TLB ;

(13) TLB multiple match plusieures entrées de la TLB correspondent ;

(14) TLB invalid index entrée de TLB non existante ;

(20) timer l'horloge d'instructions a expirée.

2.3 Accès à la mémoire et MMU

Le processeur accède à la mémoire à travers un composant de gestion mémoire appelé MMU. Ce composant interprète les requêtes du processeur et les transforme le cas échéant en adresses de la mémoire physique (ou lève une exception ad hoc). C'est ce composant qui permet la gestion de la mémoire paginée.

Les deux premiers bits des adresses, spécifiées sur 32 bits, indiquent un numéro de segment parmi 4, chaque segment représentant au maximum 1Go de mémoire. Seul le segment 0 est accessible en mode utilisateur. Les segments 0 et 1 sont des segments de mémoire paginée accessible via le mécanisme de TLB. Les segments 2 et 3 sont des segments d'accès direct à la mémoire physique, le segment 3 permettant un accès direct sans cache.

La mémoire paginée utilise des pages de taille fixe de 4Ko. Ainsi, une adresse exprimée sur 30 bits est décomposée en un index de page codé sur 18 bits et un déplacement dans cette page codé sur 12 bits. La correspondance entre les pages virtuelles et les pages de mémoire physique est gérée par TLB. La MMU dispose de 16 entrées (au moins) de TLB. Chaque entrée est constituée de deux mots mémoire qui s'interprétent comme suit :

bits 30 à 31	*bits 12 à 29*	*bits 0 à 11*						
0	index page physique	réservé	U	D	V	R	W	X
segment	index page virtuelle	libre						

avec les drapeaux suivants :

X accès en exécution (décodage d'instruction) ;

W accès en écriture ;

R accès en lecture ;

V entrée de TLB valide ;

D la page a été accédée en écriture ;

U ne pas utiliser le cache mémoire.

L'algorithme utilisé par la MMU pour transformer une adresse fournie par le processeur en adresse physique est le suivant :

- l'adresse est décomposée en segment, index de page et déplacement ;
- si le déplacement n'est pas bien aligné, l'exception alignement error est levée ;
- si le processeur est en mode utilisateur et que le segment n'est pas le segment 0 alors l'exception segment privilege est levée ;
- si le segment est 2 ou 3, l'adresse est traitée comme une adresse physique et l'algorithme termine (éventuellement par la levée de l'exception bus error) ;
- on recherche une entrée de TLB valide dont le segment et l'index de page correspondent. Le cas échéant les exceptions TLB miss ou TLB multiple match peuvent être levées ;
- si l'opération n'est pas permise, levée de l'exception TLB protection ;
- l'adresse physique est reconstruite à partir de l'entrée de la TLB et du déplacement et traitée comme une adresse physique (engendrant éventuellement la levée de l'exception bus error).

2.4 Initialisation

Lors de l'initialisation ou de la remise à zéro du processeur, son état est le suivant :

- mode superviseur (`flm=0;`) ;
- exceptions inhibées (`fle=1;`) ;
- interruptions inhibées (`fli=1;`) ;
- toutes les entrées de la TLB sont marquées invalides ;
- les registres généraux `r0` à `r63` valent 0 ;
- les registres superviseur `k0` à `k3` valent 0 ;
- l'horloge d'instructions est désactivée (`timer=0`) ;
- l'adresse du gestionnaire d'exception est indéfinie ;
- le contenu des registres d'exception est indéfini ;
- l'exécution commence au dernier mot de mémoire physique (`pc=0xfffffffc`).

2.5 Jeu d'instructions

Dans les tableaux qui suivent seules les exceptions spécifiques aux instructions ont été notées. À tout moment les exceptions IRQ, bus error, segment privilege, register violation, TLB miss, TLB protection, TLB multiple match ou encore timer peuvent être levées.

2.5.1 Opérations de comparaison

op	description	sémantique	codage				exceptions
`eq`	**égalité**	`r[des]=(r[src1]==r[src2])?1:0;`	`0xc2`	`des`	`src1`	`src2`	
`gts`	> **signé**	`r[des]=(r[src1]>r[src2])?1:0;`	`0xc1`	`des`	`src1`	`src2`	
`ges`	>= **signé**	`r[des]=(r[src1]>=r[src2])?1:0;`	`0xc3`	`des`	`src1`	`src2`	
`gtu`	> **non-signé**	`r[des]=(r[src1]>r[src2])?1:0;`	`0xc5`	`des`	`src1`	`src2`	
`geu`	>= **non-signé**	`r[des]=(r[src1]>=r[src2])?1:0;`	`0xc7`	`des`	`src1`	`src2`	

2.5.2 Opérations arithmétiques sur 32 bits

op	description	sémantique	codage				exceptions
add	addition	r[des]=r[src1]+r[src2];	0x80	des	src1	src2	
sub	soustraction	r[des]=r[src1]-r[src2];	0x81	des	src1	src2	
mul	multiplication	r[des]=r[src1]*r[src2];	0x82	des	src1	src2	
div	division	r[des]=r[src1]/r[src2];	0x83	des	src1	src2	divide by zero
mod	modulo	r[des]=r[src1]%r[src2];	0x84	des	src1	src2	divide by zero
or	or bit-à-bit	r[des]=r[src1]|r[src2];	0x85	des	src1	src2	
nor	nor bit-à-bit	r[des]=~(r[src1]|r[src2]);	0x86	des	src1	src2	
xor	xor bit-à-bit	r[des]=r[src1]^r[src2];	0x87	des	src1	src2	
and	and bit-à-bit	r[des]=r[src1]&r[src2];	0x88	des	src1	src2	
shr	décalage à droite	r[des]=r[src1]>>f(r[src2]);	0x8c	des	src1	src2	
shru	décalage non signé	r[des]=r[src1]>>f(r[src2]);	0x8d	des	src1	src2	
shl	décalage à gauche	r[des]=r[src1]<<f(r[src2]);	0x8e	des	src1	src2	
addi	addition	r[des]=r[src1]+const8;	0x90	des	src1	const8	
subi	soustraction	r[des]=r[src1]-const8;	0x91	des	src1	const8	
muli	multiplication	r[des]=r[src1]*const8;	0x92	des	src1	const8	
divi	division	r[des]=r[src1]/const8;	0x93	des	src1	const8	divide by zero
modi	modulo	r[des]=r[src1]%const8;	0x94	des	src1	const8	divide by zero
shri	décalage à droite	r[des]=r[src1]>>f(const8);	0x9c	des	src1	const8	
shrui	décalage non signé	r[des]=r[src1]>>f(const8);	0x9d	des	src1	const8	
shli	décalage à gauche	r[des]=r[src1]<<f(const8);	0x9e	des	src1	const8	

Notation. `f(x)` représente le nombre défini par les 5 bits de poids faible de `x`.

2.5.3 Opérations arithmétiques sur 64 bits

Remarque. Pour les instructions ci-dessous, l'exception illegal instruction est levée lorsque `des` est impair.

op	description	sémantique	codage				exceptions
addo	addition	r[des]=b(r[src1]+r[src2]); r[des+1]=d(r[src1]+r[src2]);	0xa0	des	src1	src2	illegal instruction
subo	soustraction	r[des]=b(r[src1]-r[src2]); r[des+1]=d(r[src1]-r[src2]);	0xa1	des	src1	src2	illegal instruction
mulo	multiplication	r[des]=b(r[src1]*r[src2]); r[des+1]=h(r[src1]*r[src2]);	0xa2	des	src1	src2	illegal instruction
addio	addition	r[des]=b(r[src1]+const8); r[des+1]=d(r[src1]+const8);	0xb0	des	src1	const8	illegal instruction
subio	soustraction	r[des]=b(r[src1]-const8); r[des+1]=d(r[src1]-const8);	0xb1	des	src1	const8	illegal instruction
mulio	multiplication	r[des]=b(r[src1]*const8); r[des+1]=h(r[src1]*const8);	0xb2	des	src1	const8	illegal instruction

Notation. `b(x)` représente le nombre défini par les 32 bits de poids faible de `x`, `h(x)` représente le nombre défini par les 32 bits de poids fort de `x`, `d(x)` vaut `-1`, `0` ou `1` selon que le stockage de `x` sur 32 bits produit un débordement vers le bas, aucun débordement ou un débordement vers le haut.

2.5.4 Opérations de branchement et de saut

Remarque. Pour les instructions ci-dessous, l'exception alignment error est levée lorsque une adresse non-divisible par 4 est assignée à `pc`.

op	description	sémantique	codage				exceptions
bez	**branche si zéro**	`r[des]=pc;` `if (r[src1]==0) pc+=r[src2]-4;`	0xd0	des	src1	src2	alignment error
jez	**saute si zéro**	`r[des]=pc;` `if (r[src1]==0) pc=r[src2]-4;`	0xd1	des	src1	src2	alignment error
bnz	**branche si non-zéro**	`r[des]=pc;` `if (r[src1]!=0) pc+=r[src2]-4;`	0xd2	des	src1	src2	alignment error
jnz	**saute si non-zéro**	`r[des]=pc;` `if (r[src1]!=0) pc=r[src2]-4;`	0xd3	des	src1	src2	alignment error
bezi	**branche si zéro**	`if (r[src1]==0) pc+=4*const16-4;`	0xd4	src1	const16		
bnzi	**branche si non-zéro**	`if (r[src1]!=0) pc+=4*const16-4;`	0xd6	src1	const16		

2.5.5 Opérations de chargement/sauvegarde

Remarque. Pour les instructions ci-dessous, l'exception alignment error est levée lors d'une tentative d'accès à une adresse non divisible par 4 pour des opérations sur 32 bits. L'exception bus error est levée lors d'une tentative d'accès incorrecte à la mémoire physique.

op	description	sémantique	codage				exceptions
ld1	**charge octet**	`r[des]=getmemb(r[src1]+const8);`	0xe0	des	src1	const8	bus error
ld4	**charge mot**	`r[des]=getmemw(r[src1]+const8);`	0xe2	des	src1	const8	bus error alignment error
st1	**sauve octet**	`setmemb(r[src2]+const8,r[src1]);`	0xe4	src1	src2	const8	bus error
st4	**sauve mot**	`setmemw(r[src2]+const8,r[src1]);`	0xe6	src1	src2	const8	bus error alignment error

ex4	**échange mots**	`tmp=r[des];`
		`r[des]=getmemw(r[src1]+const8);`
		`setmemw(r[src1]+const8,tmp);`
			0xea	des	src1	const8	bus error
							alignment error

lcl	**charge constante**	`r[des]=0xffff0000*(const16>>15);` `r[des]+=const16;`	0xf0	des	const16		
lch	**charge constante**	`r[des]&=0xffff;` `r[des]+=const16<<16;`	0xf1	des	const16		

Notation. La mémoire est accédée octet par octet à travers `getmemb` et `setmemb` et mot par mot à travers `getmemw` et `setmemw`.

2.5.6 Opération de passage en mode superviseur

op	description	sémantique	codage				exceptions
trap	**appel système**	`trap();`	0x10	?	?	?	trap

2.5.7 Opérations de gestion de la mémoire virtuelle

Remarque. Pour les instructions ci-dessous, l'exception privileged instruction est levée si l'instruction est utilisée en mode utilisateur. L'exception TLB multiple match est levée si plusieurs entrées de la TLB correspondent à l'adresse recherchée. L'exception TLB invalid index est levée si l'entrée de la TLB indiquée n'existe pas. Enfin, l'exception illegal instruction est levée si le numéro de registre de stockage d'une TLB est impair.

op	description	sémantique	codage				exceptions
tlbpi	**teste TLB**	`r[des]=tlbprobe(r[src1]);`	0x40	des	src1	0	privilege instruction TLB multiple match
tlble	**charge TLB**	`tlbload(r[src1],` `&r[des],&r[des+1]);`	0x41	des	src1	0	privilege instruction illegal instruction TLB invalid index
tlbse	**sauve TLB**	`tlbstore(r[src1],` `r[src2],r[src2+1]);`	0x42	0	src1	src2	privilege instruction illegal instruction TLB invalid index

Notation. La TLB est accédée à travers les fonctions `tlbprobe`, `tlbload` et `tlbstore`.

2.5.8 Autres opérations du mode superviseur

op	description	sémantique	codage				exceptions
leh	charge gestionnaire	`exh=r[src];`	0x44	0	src	0	privileged instruction
rfe	retour mode utilisateur	`fli=r[src2]&1; fle=0; flm=r[src3]&1; pc=r[src1]-4;`	0x45	src1	src2	src3	privileged instruction
sti	désactive int.	`fli=1;`	0x48	0	0	0	privileged instruction
cli	active int.	`fli=0;`	0x49	0	0	0	privileged instruction
ste	désactive exc.	`fle=1;`	0x4a	0	0	0	privileged instruction
cle	active exc.	`fle=0;`	0x4b	0	0	0	privileged instruction
timer	modifie timer	`r[des]=(timer>0)?timer:0; timer=r[src];`	0x4d	des	src	0	privileged instruction timer
idle	veille	`idle();`	0x4e	0	0	0	privileged instruction
halt	arrêt	`halt();`	0x4f	0	0	0	privileged instruction

Notation. Les drapeaux `fli`, `fle` et `flm` correspondent respectivement aux drapeaux de masquage d'interruption, de masquage d'exception et de sélection du mode utilisateur. La variable `exh` représente l'adresse du gestionnaire d'exceptions.

3 Le bus LAMEbus

Le bus LAMEbus est un bus 32 bits simpliste mais suffisant pour ce qui nous intéresse. Sur ce bus sont disposés le processeur, la mémoire physique ainsi que divers contrôleurs de périphériques enfichés dans un des 32 emplacements disponibles.

L'interaction entre le processeur et le bus se fait par le biais de l'exception d'IRQ qui est levée lorsqu'une ligne d'interruption du bus est activée ainsi que par le biais d'une zone mémoire située dans un trou de la mémoire physique. Cette zone mémoire située à l'adresse `LAMEBASE` est composée de 32 morceaux successifs (un par emplacement) de 64Ko.

3.1 Le contrôleur de bus

Le dernier emplacement du bus est toujours utilisé par le contrôleur de bus. Sa mémoire est découpée en deux morceaux de 32Ko. La partie supérieure est occupée par une ROM de boot non spécifiée. La partie inférieure est découpée en 32 régions de 1Ko (une par emplacement du bus) qui contiennent chacune des informations sur le périphérique qui utilise cet emplacement.

L'adresse physique de la région de configuration de l'emplacement `N` est donc

`LAMEBASE+0x10000*31+0x400*N` .

Chaque région contient les champs suivants :

VID (0x00-0x03) identifiant du vendeur ;

DID (0x04-0x07) identifiant du périphérique ;

DRL (0x08-0x0b) version du périphérique.

Une valeur 0 dans le champ `VID` indique que l'emplacement est vide. Les identifiants de vendeur et de périphériques permettent l'identification des périphérique et le choix des gestionnaires adaptés par le système d'exploitation. La région du contrôleur de bus (`VID 1`, `DID 1`, `DRL 1`) contient les champs supplémentaires suivants :

RAMSZ (0x200-0x203) taille de la mémoire physique ;

IRQS (0x204-0x207) masque des emplacements levant une interruption ;

PWR (0x208-0x20b) registre de gestion de l'alimentation.

Pour éteindre l'ordinateur, on écrit la valeur `0xfee1dead` dans le registre de gestion d'alimentation. Toute autre valeur est sans effet.

3.2 Quelques périphériques

3.2.1 Contrôleur d'horloge

La mémoire du contrôleur d'horloge (`VID` 1, `DID` 2, `DRL` 1) contient les champs suivants :

0x00-0x03 heure courante (en secondes) ;

0x04-0x07 heure courante (en nanosecondes) ;

0x08-0x0b drapeau de bouclage ;

0x0c-0x0f vrai si interruption (la lecture remet à zéro) ;

0x10-0x13 compte à rebours (en microsecondes) ;

0x14-0x17 beeper (toute écriture produit un beep).

Si le drapeau de bouclage est activé, le compte à rebours est réactivé à chaque expiration.

3.2.2 Contrôleur de console

La mémoire du contrôleur de console (`VID` 1, `DID` 4, `DRL` 1) contient les champs suivants :

0x00-0x03 tampon de caractère ;

0x04-0x07 registre d'IRQ d'écriture ;

0x08-0x0b registre d'IRQ de lecture.

On lit et on écrit un caractère par le biais du tampon. Les deux registres ont chacun 2 bits. Si le bit 0 est mis une IRQ est générée quand le périphérique a effectué l'opération en question. Si le bit 1 est mis, c'est que le périphérique a en effet effectué cette opération.

3.2.3 Contrôleur de disque

La mémoire du contrôleur de disque (`VID` 1, `DID` 3, `DRL` 2) contient les champs suivants :

0x00-0x03 nombre de secteurs du disque ;

0x04-0x07 registre de status ;

0x08-0x0b index de secteur ;

0x0c-0x0f vitesse de rotation (RPM) ;

0x8000-0x81ff secteur de 512 octets.

Les bits du registre de status s'interprètent comme suit :

bit 0 operation en cours ;

bit 1 opération d'écriture ;

bit 2 opération effectuée ;

bit 3 index de secteur non valide ;

bit 4 erreur physique.

Le disque effectue une opération à la fois (lecture ou écriture). On active l'opération en mettant en place le bit opération en cours. Lorsque l'opération est terminée, une IRQ est levée et le bit opération effectuée est levé. L'écriture de 0 dans le registre de status abandonne l'opération en cours.

Références

[1] The Ant-32 Architecture. *Revision 3.1.0b*. http://ant.eecs.harvard.edu/.

[2] System/161 Manual. http://www.courses.fas.harvard.edu/ cs161/resources/.