amberMind Conseil et création technologiques

Langage machine des objets VIOLET Lowcost

RevA	19/01/2005	Création du document	Sylvain Huet
RevB	15/12/2005	Retouches	SH

1 Introduction

1.1 Aperçu sur le document

Ce document décrit le langage machine de la machine virtuelle intégrée dans les objets VIOLET Lowcost.

2 Généralités

L'objet VIOLET Lowcost est un objet qui :

- contient un peu d'intelligence, grâce à un micro-contrôleur 8 bits, et un peu de mémoire vive
- est connecté à une plate-forme distante, initialement via Wifi. Cette plate-forme pilote en fait l'objet à distance en lui indiquant la manière dont il interagit
- peut se manifester en contrôlant quelques leds multicolores, un ou deux moteurs, du son
- peut capter son environnement grâce à des capteurs binaires (0 ou 1)
- ne peut pas mettre son firmware à jour à distance

Pour des raisons de technologie de connexion, et de coût d'exploitation, il n'y a pas de streaming entre la plate-forme et l'objet. La plate-forme transmet régulièrement des données à l'objet, sur requête de ce dernier.

On s'intéresse dans ce document au format des données transmises par la plate-forme à l'objet, et notamment au langage machine de la machine virtuelle.

3 Stratégie

On se fixe comme objectif de :

- limiter le traitement effectué par l'objet : les données reçues doivent être très proches des manifestations à produire
- garder une grande souplesse, pour ne pas enfermer l'objet dans des comportements semblables
- permettre une continuité de fonctionnement de l'objet en cas de perte de connexion, pendant au moins quelques minutes

De ce fait, la solution retenue consiste à définir une architecture de type 'machine virtuelle' avec un jeu d'instruction réduit et adapté au problème. Les données transmises seront simplement le 'langage machine' utilisé par l'objet. On les appellera « le programme » de l'objet.

Lorsque l'objet reçoit un programme différent du précédent, la question de la transition est réglée en fonction de l'état de l'objet et des indications fournies par le nouveau programme. Il est à noter que la plate-forme a évidemment la connaissance des programmes successifs transmis à l'objet (puisque c'est elle qui les produit), et c'est donc à elle que revient le travail de régler ces transitions.

4 Machine virtuelle

4.1 Etats

La machine virtuelle contient les éléments suivants :

- une mémoire programme, qui contient le programme reçu depuis la plate-forme. La taille maximale est de 64ko (16 bits).
- une ram de 256 octets, dédiées à la machine virtuelle, et utilisée également comme pile système
- une ram de « sources » de 64 octets, en read-only pour la machine virtuelle et en write-only pour le serveur
- un compteur de programme 16 bits, qui pointe vers l'instruction à exécuter : PC
- un compteur d'attente : W
- 16 accumulateurs 8 bits : R0, R1, …, R15
- CC : un bit de retenu
- cmp1, cmp2 : deux registres d'état
- pour chaque interruption (il peut y en avoir 16)
    - un pointeur programme « input » qui indique le code à effectuer en cas d'interruption (-1 si l'interruption doit être ignorée) : input.0, …, input.15

Les valeurs initiales sont :

- PC : 17
- W : 0
- R0, … R15 : 0
- Pour les interruptions :
    - Input.i=-1

4.2 Fonctionnement régulier

Le fonctionnement de la machine virtuelle est le même à chaque pas. Ces pas ont lieu régulièrement, environ 20 fois par seconde.

- On examine W :
    - si W<>0, on décrémente W
    - sinon, on lit la prochaine instruction du programme, et on l'interprète
        - on recommence de manière à lire au plus 1.000.000 instructions, tant que W=0

L'objet fonctionne en boucle séquentielle, et exécuté également entre chaque pas des opérations de gestion des communications réseau, des lectures audio, des leds, …
La limite à 1.000.000 d'instructions par pas est donc une sécurité, qui assure que la machine virtuelle rend la main pour les autres traitements, mais elle ne devrait jamais être atteinte : le programme en langage machine doit lui-même régulièrement indiquer qu'il est arrivé à la fin du pas, en positionnant la valeur W au moins à 1 (voir fonctions WAIT).

4.3 Pile système

La pile système est utilisée pour :

- stocker la valeur du compteur programme lors de l'appel d'un sous-programme
- stocker la valeur du compteur programme, des registres d'état et de la retenue lors d'une interruption
- stocker temporairement différents registres (fonctions PULL et PUSH)

La pile système est stockée dans la Ram de la machine virtuelle. Le pointeur de pile système est le registre R15.

L'empilement consiste à :
- décrémenter R15
- stocker la valeur à empiler dans la Ram, à l'adresse R15

Le dépilement consiste à :
- lire la valeur à dépiler dans la Ram, à l'adresse R15
- incrémenter R15

4.4 Interruptions

Lorsque, entre deux pas, la machine virtuelle détecte un événement débouchant sur l'exécution d'une interruption (par exemple l'interruption 'i'), le fonctionnement est le suivant :
- Si input.i = -1, ne rien faire
- Sinon, empiler PC, CC, CMP1, CMP2, puis positionner PC sur la valeur input.i

Pour le Nabaztag, les interruptions sont les suivantes :
- 0 : appui sur le bouton poussoir
- 1 : relâchement du bouton poussoir
- 2 : changement de position du bouton 3 états
- 14 : exception (opcode inconnu)
- 15 : timer (tous les 20 pas)

Les autres interruptions sont utilisables par le programme en tant qu'interruptions software.

4.5 Envoi de données vers le serveur

La machine virtuelle peut transmettre des données à la plate-forme. Ces données sont des mots de 16 bits. Le fonctionnement est le suivant :
- la machine virtuelle envoie un message 16 bits : le buffer d'émission est réglé sur « plein »
- la couche réseau va transmettre le message, puis régler le buffer d'émission sur « vide »
- la machine virtuelle devrait attendre que le buffer soit vide avant de ré-émettre un message.

4.6 Instruction inconnue

Lorsque la machine virtuelle tombe sur un Opcode inconnu, elle appelle l'interruption 14, puis effectue un Reset soft.

4.7 Registres d'état

Le mécanisme de registre d'état de la machine virtuelle est simplifié par rapport aux micro-processeurs classiques.
Il est composé de trois registres d'état :
- CC : bit de retenue
- CMP1 : opérateur 1 (8 bits) de la dernière opération
- CMP2 : opérateur 2 (8 bits) de la dernière opération

Pour toutes les opérations :

- CMP1 est chargé avec le résultat de l'opération
- CMP2 est chargé avec 0

Il y a une exception : l'opération CMP, qui charge CMP1 et CMP2 avec les deux opérandes. Dans le cas où le résultat de l'opération est sur 16 bits, CMP1 est réglé avec le résultat du OU logique sur l'octet de poids fort et l'octet de poids faible. Ceci ne permet alors que les tests BEQ et BNE.

4.8 Modes d'adressage

Les instructions ont la structure suivante :

- un opérateur, qui contient notamment le mode d'adressage
- zéro, un ou plusieurs octets d'opérandes

4.8.1 – (inhérent) 0 octet

Aucun argument : par exemple RTI

4.8.2 i

Un entier 8 bits. L'entier est stocké dans l'octet de l'opérande.

4.8.3 r

Un registre. Le registre est stocké dans les 4 bits de poids faible de l'opérande.

4.8.4 r, i

Un registre et un entier. Ce mode est utilisé d'une manière spéciale avec fusion de l'opcode et du registre : l'opcode n'utilise que les 4 bits de poids fort ; on utilise les 4 bits de poids faible pour stocker le registre. Puis on utilise un octet d'opérande pour l'entier.

4.8.5 r, r'

Deux registres. Le premier utilise les 4 bits de poids fort de l'opérande ; le second en utilise les 4 bits de poids faible.

4.8.6 i, i'

Deux entiers. Le premier se trouve dans le premier octet de l'opérande, le second dans le second octet.

4.8.7 r, i, r'

Deux registres et un entier. Le premier octet de l'opérande regroupe les deux registres, le second contient l'entier

4.8.8 w

Une adresse 16 bits, stockée dans l'ordre BigEndian.

4.8.9 r,w

Un registre et une adresse 16 bits. Le registre est stocké dans les 4 bits de poids faible du premier octet de l'opérande, l'adresse dans les deux suivants.

4.8.10 r,r,w

Deux registres et une adresse 16 bits. Le premier octet de l'opérande regroupe les deux registres, les deux suivants contiennent l'adresse.

4.9 Jeu d'instructions

On trouve ci-après le jeu d'instruction de la machine virtuelle. La forme est :

- mnémonique_adressage (0xopcode).

Dans le cas de l'adressage « r,i », l'opcode réel est obtenu en additionnant le numéro du registre à l'opcode de base.
Par exemple, ADD r2,$47 se traduit par deux octets : $12, $47

4.9.1 LD_ri (0x0)

Charge le registre avec la valeur de l'entier
Modifie CMP1, CMP2.

4.9.2 ADD_ri (0x10)

Ajoute au registre la valeur de l'entier
Modifie CMP1, CMP2.

4.9.3 SUB_ri (0x20)

Soustrait du registre la valeur de l'entier
Modifie CMP1, CMP2.

4.9.4 AND_ri (0x30)

Effectue un ET logique entre le registre et l'entier, et stocke le résultat dans le registre.
Modifie CMP1, CMP2.

4.9.5 OR_ri (0x40)

Effectue un OU logique entre le registre et l'entier, et stocke le résultat dans le registre.
Modifie CMP1, CMP2.

4.9.6 LDR_ri (0x50)

Charge le registre avec le i-ème octet de la Ram (où i est l'entier passé en paramètre).
Modifie CMP1, CMP2.

4.9.7 STR_ri (0x60)

Stocke le registre dans le i-ème octet de la Ram (où i est l'entier passé en paramètre)

4.9.8 NOP_o (0x70)

Ne fait rien

4.9.9 RTI_o (0x72)

Fin d'interruption : dépile PC, CC, CMP1, CMP2

4.9.10 CLRCC_o (0x73)

Remet à zéro le bit CC

4.9.11 SETCC_o (0x74)

Met à 1 le bit CC

4.9.12 ADDCC_rr (0x75)

Effectue l'addition de deux registres et de la retenue et met le résultat dans le premier registre.
Modifie CMP1, CMP2.

4.9.13 SUBCC_rr (0x76)

Effectue la soustraction de deux registres et de la retenue et met le résultat dans le premier registre.
Modifie CMP1, CMP2.

4.9.14 INCW_rr (0x77)

Effectue une incrémentation sur un compteur 16 bits qui serait constitué du premier registre pour le poids fort, et du second pour le poids faible.
Modifie CMP1, CMP2.

4.9.15 DECW_rr (0x78)

Effectue une décrémentation sur un compteur 16 bits qui serait constitué du premier registre pour le poids fort, et du second pour le poids faible.
Modifie CMP1, CMP2.

4.9.16 MULW_rr (0x79)

Effectue la multiplication non signée de deux registres, et stocke le poids fort du résultat dans le premier, et le poids faible dans le second.
Modifie CMP1, CMP2.

4.9.17 INPUTRST_r (0x7a)

Réinitialise l'interruption dont le numéro est contenu dans le registre.

4.9.18 INT_r (0x7b)

Provoque une interruption software dont le numéro est contenu dans le registre.

4.9.19 WAIT_r (0x7d)

Régle la valeur W de la machine virtuelle avec la valeur du registre.

4.9.20 WAIT_i (0x7e)

Régle la valeur W de la machine virtuelle avec la valeur de l'entier.

4.9.21 RND_r (0x7f)

Retourne dans le registre une valeur aléatoire sur 8 bits.
Modifie CMP1, CMP2.

4.9.22 DEC_r (0x80)

Décrémente le registre.
Modifie CMP1, CMP2.

4.9.23 INC_r (0x81)

Incrémente le registre.
Modifie CMP1, CMP2.

4.9.24 CLR_r (0x82)

Règle le registre avec la valeur 0.
Modifie CMP1, CMP2.

4.9.25 NEG_r (0x83)

Inverse le signe du registre.
Modifie CMP1, CMP2.

4.9.26 NOT_r (0x84)

Effectue une transformation du registre : les bits 0 deviennent des bits 1 et inversement.
Modifie CMP1, CMP2.

4.9.27 TST_r (0x85)

Teste la valeur d'un registre.
Modifie CMP1, CMP2.

4.9.28 LD_rr (0x86)

Charge le premier registre avec la valeur du second.
Modifie CMP1, CMP2.

4.9.29 ADD_rr (0x87)

Ajoute au premier registre la valeur du second.
Modifie CMP1, CMP2.

4.9.30 SUB_rr (0x88)

Soustrait du premier registre la valeur du second.
Modifie CMP1, CMP2.

4.9.31 MUL_rr (0x89)

Multiplie deux registres et stocke le résultat dans le premier.
Modifie CMP1, CMP2.

4.9.32 AND_rr (0x8a)

Effectue un ET logique entre deux registres et stocke le résultat dans le premier.
Modifie CMP1, CMP2.

4.9.33 OR_rr (0x8b)

Effectue un OU logique entre deux registres et stocke le résultat dans le premier.
Modifie CMP1, CMP2.

4.9.34 EOR_rr (0x8c)

Effectue un OU EXCLUSIF logique entre deux registres et stocke le résultat dans le premier.
Modifie CMP1, CMP2.

4.9.35 LSL_rr (0x8d)

Effectue un décalage à gauche sur le premier registre. Le nombre de bits de décalage est défini par la valeur du second.
Modifie CMP1, CMP2.

4.9.36 LSR_rr (0x8e)

Effectue un décalage logique à droite sur le premier registre. Le nombre de bits de décalage est défini par la valeur du second.
Modifie CMP1, CMP2.

4.9.37 ASR_rr (0x8f)

Effectue un décalage arithmétique à droite sur le premier registre. Le nombre de bits de décalage est défini par la valeur du second.
Modifie CMP1, CMP2.

4.9.38 ROL_rr (0x90)

Effectue une rotation à gauche sur le premier registre. Le nombre de bits de décalage est défini par la valeur du second.
Modifie CMP1, CMP2.

4.9.39 ROR_rr (0x91)

Effectue une rotation à droite sur le premier registre. Le nombre de bits de décalage est défini par la valeur du second.
Modifie CMP1, CMP2.

4.9.40 CMP_rr (0x92)

Compare deux registres. Charge CMP1 avec le premier, et CMP2 avec le second.

4.9.41 BIT_rr (0x93)

Effectue un ET logique entre deux registres, mais sans stocker le résultat.
Modifie CMP1, CMP2.

4.9.42 LDR_rr (0x94)

Charge le premier registre avec le i-ème octet de la Ram (où i est la valeur du second registre)
Modifie CMP1, CMP2.

4.9.43 LDR_rir (0x95)

Charge le premier registre avec le i-ème octet de la Ram (où i est la somme de l'entier et de la valeur du second registre)
Modifie CMP1, CMP2.

4.9.44 STR_rr (0x96)

Stocke le premier registre dans le i-ème octet de la Ram (où i est la valeur du second registre)

4.9.45 STR_rir (0x97)

Stocke le premier registre dans le i-ème octet de la Ram (où i est la somme de l'entier et de la valeur du second registre)

4.9.46 LDT_rrw (0x98)

Charge dans le premier registre la i-ème valeur d'une table (où i est la valeur du second registre). L'adresse du début de la table est passée en paramètre.
Modifie CMP1, CMP2.

4.9.47 LDTW_rw (0x99)

Charge dans le premier registre la i-ème valeur d'une table (où i est la valeur 16 bits constituée par les registres R3 (poids fort) et R4 (poids faible)). L'adresse du début de la table est passée en paramètre.
Modifie CMP1, CMP2.

4.9.48 INPUT_rw (0x9a)

Règle l'adresse d'une interruption. Le numéro de l'interruption est la valeur du registre.

4.9.49 RTIJ_w (0x9b)

Fin d'interruption, avec saut : dépile PC, CC, CMP1, CMP2, puis règle PC avec l'adresse passée en paramètre.

4.9.50 BRA_w (0x9c)

Règle PC avec l'adresse passée en paramètre (saut inconditionnel).

4.9.51 BEQ_w (0x9d)

Règle PC avec l'adresse passée en paramètre, si Cmp1=Cmp2

4.9.52 BNE_w (0x9e)

Règle PC avec l'adresse passée en paramètre, si Cmp1<>Cmp2

4.9.53 BGT_w (0x9f)

Règle PC avec l'adresse passée en paramètre, si Cmp1>Cmp2 (signé)

4.9.54 BGE_w (0xa0)

Règle PC avec l'adresse passée en paramètre, si Cmp1>=Cmp2 (signé)

4.9.55 BLT_w (0xa1)

Règle PC avec l'adresse passée en paramètre, si Cmp1<Cmp2 (signé)

4.9.56 BLE_w (0xa2)

Règle PC avec l'adresse passée en paramètre, si Cmp1<=Cmp2 (signé)

4.9.57 BHI_w (0xa3)

Règle PC avec l'adresse passée en paramètre, si Cmp1>Cmp2 (non signé)

4.9.58 BHS_w (0xa4)

Règle PC avec l'adresse passée en paramètre, si Cmp1>=Cmp2 (non signé)

4.9.59 BLO_w (0xa5)

Règle PC avec l'adresse passée en paramètre, si Cmp1<Cmp2 (non signé)

4.9.60 BLS_w (0xa6)

Règle PC avec l'adresse passée en paramètre, si Cmp1<=Cmp2 (non signé)

4.9.61 LED_rr (0xa7)

Modifie une led. Le premier registre contient le numéro de la led, le second le temps de transition (compté en pas). La nouvelle couleur de la led est dans les registres R0, R1, R2.

4.9.62 PALETTE_rr (0xa8)

Règle les registre R0, R1 et R2 avec une couleur. Le premier registre définit la couleur (0 : noir, 1 : rouge, 2 : vert, 3 : jaune, 4 : bleu, 5 : violet, 6 : cyan, 7 : blanc, …) et le second définit l'intensité (0 : noir, …, 255 : très clair).

4.9.63 PUSH_ii (0xaa)

Sauvegarde des registres dans la pile système. Les deux entiers définissent 16 bits. Chaque bit correspond à un registre à sauvegarder (R0, …, R14, CC).
Pour le premier opérande : CC, R14, R13, …, R8 (dans l'ordre bit7 vers bit0).
Pour le second opérande : R7, R6, …, R0 (dans l'ordre bit7 vers bit0).

4.9.64 PULL_ii (0xab)

Récupère des registres depuis la pile système. . Les deux entiers définissent 16 bits. Chaque bit correspond à un registre à récupérer (R0, …, R14, CC).
Pour le premier opérande : CC, R14, R13, …, R8 (dans l'ordre bit7 vers bit0).
Pour le second opérande : R7, R6, …, R0 (dans l'ordre bit7 vers bit0).

4.9.65 BSR_w (0xac)

Effectue un saut non conditionnel vers l'adresse passée en paramètre, et sauvegarde le PC courant (appel à un sous-programme) dans la pile système.

4.9.66 RTS_o (0xad)

Fin de sous-programme : dépile la valeur PC de la pile système

4.9.67 MOTOR_rr (0xae)

Commande les moteurs : le premier registre indique le numéro du moteur (0 ou 1), le second indique la direction (0, 1 ou 2 ; 0 signifie l'arrêt)

4.9.68 MIDIPLAY_r (0xaf)

Lance le player midi sur un fichier audio. Le registre contient le numéro du fichier audio dans la tables des fichiers audio.

4.9.69 MIDISTOP_o (0xb0)

Arrête le player midi.

4.9.70 WAVPLAY_r (0xb1)

Lance le player adpcm sur un fichier audio. Le registre contient le numéro du fichier audio dans la tables des fichiers audio.

4.9.71 WAVSTOP_o (0xb2)

Arrête le player adpcm.

4.9.72 MSEC_rr (0xb3)

Retourne dans une valeur 16 bits correspondant à un temps en millisecondes. Le premier registre va contenir l'octet de poids fort, le second l'octet de poids faible.

4.9.73 SEC_rr (0xb4)

Retourne dans une valeur 16 bits correspondant à un temps en secondes. Le premier registre va contenir l'octet de poids fort, le second l'octet de poids faible.

4.9.74 BUT3_r (0xb5)

Charge le registre avec la position du bouton 3 états (0, 1 ou 2)

4.9.75 VOL_r (0xb6)

Règle le volume applicatif avec la valeur du registre.

4.9.76 MVOL_r (0xb7)

Règle le master volume avec la valeur du registre. Le volume réel est le produit du volume applicatif et du master volume.

4.9.77 PUSHBUTTON_r (0xb8)

Charge le registre avec la position du bouton poussoir (0 ou 1)

4.9.78 SRC_rr (0xb9)

Charge le premier registre avec la valeur de la i-ème source (où i est la valeur du second registre).
Modifie CMP1, CMP2.

4.9.79 BRAT_rw (0xba)

Effectue un saut inconditionnel. La nouvelle valeur du PC est recherchée dans la table d'adresses. Le registre contient l'index dans cette table.

4.9.80 BSRT_rw (0xbb)

Effectue un saut inconditionnel. La nouvelle valeur du PC est recherchée dans la table d'adresses. Le registre contient l'index dans cette table. Le PC courant est sauvegardé dans la pile système (appel à un sous-programme)

4.9.81 OSC_rr (0xbc)

Charge le premier registre avec la valeur 128*(1-cos i*pi/128), ou i est la valeur du second registre.
Modifie CMP1, CMP2.

4.9.82 INV_rr (0xbd)

Effectue le calcul 65536/r (où r est le second registre), et écrit le poids fort du résultat dans le premier registre, et le poids faible dans le second.

4.9.83 DIV_rr (0xbe)

Effectue la division 256*r1/r2, et écrit le poids fort du résultat dans le premier registre, et le poids faible dans le second.

4.9.84 HSV_o (0xbf)

Effectue une conversion HSV -> RGB sur le registres R0, R1, R2.

4.9.85 MOTORGET_rr (0xc0)

Charge le premier registre avec le compteur associé au moteur dont le numéro est passé dans le second registre.

4.9.86 MUSIC_r (0xc1)

Charge le registre avec l'état des players (0 : pas de lecture audio en cours, 1 : lecture midi en cours, 2 : lecture adpcm en cours).

4.9.87 DOWNLOAD_r (0xc2)

Charge le registre avec l'état de l'automate réseau (bit 1 : téléchargement en cours, bit 2 : requête réseau en cours).
Modifie CMP1, CMP2.

4.9.88 SEND_rr (0xc4)

Demande l'envoi d'une valeur 16 bits au serveur ; le premier registre contient le poids fort, le second contient le poids faible.

4.9.89 SENDREADY_r (0xc5)

Charge le registre avec l'état de l'automate d'envoi (1 : prêt à émettre).
Modifie CMP1, CMP2.

4.9.90 LASTPING_rr (0xc6)

Retourne la valeur 16 bits du temps écoulé, en secondes, depuis la dernière connexion réussie vers le serveur. Le premier registre contient le poids fort, le second contient le poids faible.
Modifie CMP1, CMP2.

5 Format des trames

5.1 Structure de la réponse du serveur

La réponse est constituée d'un nombre variable d'opérations.
Chaque opération a la structure suivante :

- un code opération
- la taille des arguments de l'opération, sur 24 bits
- puis les arguments en question

Le code $ff signifie la fin de la réponse, il n'est pas suivi de la taille.

Les codes principaux sont :

$04	modification des sources d'une trame de la machine virtuelle
$05	nouvelle trame de bytecode pour la machine virtuelle
$ff	fin de la réponse

5.2 Structure des modifications des sources

Les opérations sur les sources permettent au serveur de modifier les valeurs de la RAM « sources » de la machine virtuelle, accessible via l'instruction SRC de la machine virtuelle.

La structure de l'opération sur les sources est la suivante :

- identifiant de la trame : 4 octets
- puis de 1 à 63 octets contenant les valeurs à placer dans la RAM « sources », à partir de l'adresse 0.

5.3 Structure des trames de bytecode

Offset	**Taille**	**Label**	**Description**
000	(5)	magic	'amber'
005	(4)	id	identifiant de la trame
009	(1)	transition	flag de transition (1 : immédiate)
00a	(4)	size_prog	taille du programme
00e	(size_prog)	program	code du programme (le premier octet de ce bloc correspond à l'adresse 17, ou 0x11)
(size_prog+14)	(4)	nb_mus	nombre de musiques
(size_prog+18)	(4)	music1	offset bloc musique 1 (=taille musique 0)
…			
(size_prog+4.nb_mus+18)		(..)	music0 données de la musique 0 (offset pour le calcul des blocs de musique)
…	(1)	checksum	
…	(4)	magic	'mind'

Le checksum est calculé de manière à ce que la somme de tous les octets de la trame fasse 255.