**MONSTER8**

La monster8 (« *Monster8 »*) est une console de jeu virtuelle, programmée en C++ avec Gtk3 sous Ubuntu Linux.

**Propriétés de la console :**

Elle dispose d’un processeur 8/32 bits vaguement inspiré du z80, avec une mémoire adressable 24 bits. Elle dispose de 8 registre 32 bits, nommés A0 à A7, et de 8 registres 8 bits, nommés D0 à D7. Le registre D7 contient les drapeaux.

La console dispose d’une mémoire de 16 Mo de base, d’une mémoire de 16 Mo pour les sprites, d’une mémoire de 16 Mo pour les tilesets, d’une mémoire de 16 Mo pour les sons OGG, et d’une mémoire de 16Mo pour les musiques OGG.

Elle dispose d’un éditeur de code source assembler (« *Monster8Editor »*), qui permet aussi d’importer les différents médias.

**Les drapeaux :**

Il y a cinq drapeaux. Ce sont S, Z, H, PO et C. Respectivement, ce sont le drapeau de Signe, le drapeau Zéro, la Half-Carry (demi-retenue), Parité/Overflow et Carry (retenue).

Après certaines opérations, les drapeaux sont mis à 1 ou à 0.

Voici les conditions que l’on peut tester suite à cela :

Drapeau S : N ou P (négatif ou positif)

Drapeau Z : Z ou NZ (zéro ou non-zéro)

Drapeau H : H ou NH (demi retenue ou non-demi retenue)

Drapeau PO : OV ou NO (overflow ou non-overflow)

Drapeau C : C ou NC (retenue ou non-retenue)

**Processeur 32bits et mémoire 24 bits**

Toute tentative d’accès à la mémoire inexistante provoque un halt.

**Instructions du processeur virtuel :**

NOP : Ne rien faire

HALT : stopper le microprocesseur, obligeant l’utilisateur à éteindre la console pour de nouveau s’en servir.

JP flag,adresse : saute à l’adresse en fonction de l’état du drapeau. (voir plus haut, « les drapeaux »)

JP Ax : saute à l’adresse dans le registre Ax.

JP adresse : saute à l’adresse.

CALL adresse : appelle un sous-programme à l’adresse.

RET flag : retour d’un sous-progra mme en fonction du drapeau.

RET flag : retour d’un sous-progra mme.

SEZ : set Z, met le flag Z à 1.

CLZ : reset Z, met le flag Z à zéro.

SEC : set C, met le flag C à 1.

CLC : reset C, met le flag C à zéro.

CP [A0]: compare D0 avec [A0]. (avec l’octet pointé par A0)

CP D1 : compare D0 avec D1.

CP A0,adresse : compare A0 avec l’adresse. (comparaison 24 bits)

LD Dx,[Ax] : charge en Dx l’octet pointé par l’adresse Ax. (x va de 0 à 6)

LD [Ax],Dx : charge Dx dans l’octet pointé par l’adresse Ax. (x va de 0 à 6)

LD Dx,Dy : charge Dy dans Dx. (x et y vont de 0 à 6)

LD Ax,adresse : charge l’adresse (ou valeur 24 bits) dans Ax. (x va de 0 à 7)

LD Dx,adresse : charge l’adresse (ou valeur 8 bits) dans Dx. (x va de 0 à 6)

LDI : charge D1 dans D0, incrément D0 et D1, décrémente D2.

LDIR : charge D1 dans D0, incrément D0 et D1, décrémente D2, répète tant que D2 > 0.

LDD : charge D1 dans D0, décrémente D0 et D1, décrémente D2.

LDDR : charge D1 dans D0, décrément D0 et D1, décrémente D2, répète tant que D2 > 0.

ADD Dx : ajoute Dx à D0, résultat dans D0. (x va de 1 à 6)

SUB Dx : soustrait Dx à D0, résultat dans D0. (x va de 1 à 6)

MULT D0,Dx :multiplie D0 par Dx, résultat dans D0. (x va de 1 à 6)

MULT A0,A1 : multiplie A0 par A1, résultat dans A0.

MULT A0,D0 : multiplie A0 par D0, résultat dans A0.

DIV A0,A1 : divise A0 par A1, résultat dans A0 ou 0xffffff si division par zéro.

DIV A0,D0 : divise A0 par D0, résultat dans A0 ou 0xffffff si division par zéro.

AND Dx : ET logique, D0 and Dx, résultat dans D0. (x va de 1 à 6)

OR Dx : OU logique, D0 or Dx, résultat dans D0. (x va de 1 à 6)

XOR Dx : OU exclusif logique, D0 xor Dx, résultat dans D0. (x va de 0 à 6)

SET D0,x : met à 1 le bit x de D0. (x va de 0 à 7)

RESET D0,x : met à 0 le bit x de D0. (x va de 0 à 7)

BIT D0,x : teste le bit x de D0, résultat dans Z (x va de 0 à 7)

PUSH Ax : empile le registre Ax.

PUSH D0,D1,D2 : empile la valeur 24 bits formée par les trois registres 8 bits.

PUSH D1,D2,D3 : empile la valeur 24 bits formée par les trois registres 8 bits.

PUSH D2,D3,D4 : empile la valeur 24 bits formée par les trois registres 8 bits.

PUSH D3,D4,D5 : empile la valeur 24 bits formée par les trois registres 8 bits.

POP Ax : désempile le registre Ax.

POP D0,D1,D2 : désempile la valeur 24 bits formée par les trois registres 8 bits.

POP D1,D2,D3 : désempile la valeur 24 bits formée par les trois registres 8 bits.

POP D2,D3,D4 : désempile la valeur 24 bits formée par les trois registres 8 bits.

POP D3,D4,D5 : désempile la valeur 24 bits formée par les trois registres 8 bits.

SHL D0 : décale les bits de D0 à gauche d’un cran.

SHR D0 : décale les bits de D0 à droite d’un cran.

RL D0 : rotation des bits de D0 à gauche d’un cran.

RR D0 : rotation des bits de D0 à droite d’un cran.

INC Dx : incrémente le régistre Dx. (x va de 0 à 6)

INC Ax : incrémente le régistre Ax. (x va de 0 à 7)

DEC Dx : incrémente le régistre Dx. (x va de 0 à 6)

DEC Ax : incrémente le régistre Ax. (x va de 0 à 7)

**Instructions GPU et audio :**

CLS : efface l’écran

SPRITE : dessine le sprite, coordonnées A0,A1, D0 et D1 contiennent la taille, et A2 contient l’adresse de l’image

BLOCK : dessine le block, coordonnées A0,A1, D0 et D1 contiennent la taille, et A2 contient l’adresse de l’image

GRAB : récupère le block à l’écran puis le stocke en mémoire, coordonnées A0,A1, D0 et D1 contiennent la taille, et A2 contient l’adresse de l’image

GETJOYSTICK : récupère dans D0 les valeurs du joystick

LOCATE x,y : positionne le curseur texte

PRINTCHAR : affiche le caractère en D0.

PRINT adresse : affiche le texte à l’adresse, termine par l’octet 0.

PEN : couleur du stylo d’écriture, de 0 à 255.

PAPER : couleur du papier, de 0 à 255.

BORDER : couleur du bord, de 0 à 255.

PLOT A0,A1,D0 : affiche un point aux coordonnées A0,A1 de couleur D0.

POINT : récupère dans D0 la couleur du point aux coordonnées A0,A1.

SCROLL : effectue un scrolling horizontal et/ou vertical.

FLIP : échange le double buffer, affiche tout. (à ne pas oublier !)

SOUND adresse, longueur : joue un son OGG.

MUSIC adresse, longueur : joue une musique OGG en boucle.

STOP : stoppe la musique courante.

**Mnémoniques :**

DB a,b,c,... : stocke des valeur s 8 bits en mémoire.

DB « Phrase »,0, stocke une phrase en mémoire.

DS x,y : prépare un bloc de mémoire de taille x et l’initialise à la valeur y.