Programmation Générale

D A E M O N L A B

# REDSTEEL

*Une machine dans votre machine*

**- DaemonLab -**

*Ce projet consiste à réaliser une partie d'un émulateur. Un émulateur est un logiciel simulant le fonctionnement complet d'une machine. La dite machine est donc une « machine virtuelle ».*

*La machine dont vous allez compléter la simulation est un ordinateur 16 bits à l'architecture originale. La partie qui vous incombe est… son **micro-processeur**. Ce micro-processeur disposera de registres 16 bits, d'un bus de données de 16 bits, d'un bus d'adresse de 16bits et son byte ne sera pas un octet mais un seizet… soit 16bits et non 8.*



# INDEX

## Avant-propos :



## 01 – Détails administratifs

Votre travail doit être rendu le dossier **~/projets/redsteel/** dans l'espace personnel de l'administrateur de l'équipe.

---

Ce travail est a effectuer en équipe de 4. Durant ce projet, vous serez à même de travailler ensemble votre compréhension de la nature du micro-processeur autant que son implémentation. Lors de la soutenance finale, vous serez tous interrogé sur des aspects essentiels des ordinateurs afin de nous assurer de la compréhension de chacun

## 02 – Propreté de votre rendu

Votre rendu, c'est à dire le contenu de l'archive ou du dépôt que vous entrez sur l'interface du TechnoCentre, doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** ichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale.

**La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.**

## 03 – Fonctions autorisées

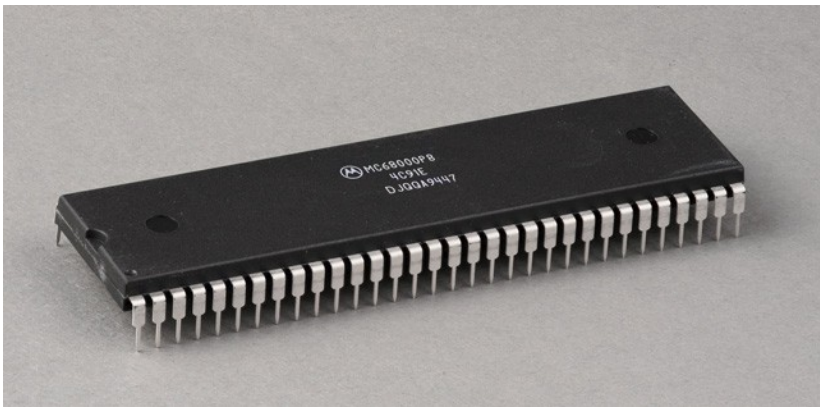
Concernant le travail sur le CPU, vous n'avez le droit à aucune fonction.

Concernant un éventuel travail bonus, vous avez le droit aux fonctions suivantes :

- open, close, read, write, malloc, free, alloca
- srand, rand, cos, sin, atan2, sqrt
- opendir, readdir, closedir, mkdir, stat, getgrgid, getpwuid
- strftime, localtime, time

# 04 – Micro-processeur ? Émulateur ? Machine virtuelle ?

Qu'est ce qu'un processeur ? Il s'agit d'un système dont le métier consiste à réaliser des taches, généralement simple, sur ordre. Qu'est ce qu'un micro-processeur ? Il s'agit d'un processeur fabriqué dans un **micro-circuit**, un « circuit intégré », dit « microchip » en anglais. C'est à dire une puce électronique.




Le Motorola 68000, un micro-processeur emblématique des années 80.

Les taches à effectuer sont décrites dans un format appelé **langage machine**. Ce format est propre au micro-processeur. Vous pouvez voir le micro-processeur comme l'interprète de son propre langage machine. Une tache porte le nom d'**instruction**.

Un émulateur est un logiciel réalisant la simulation d'un appareil. Votre terminal par exemple est en fait un **émulateur de terminal** car un terminal est un appareil électronique à l'origine :

Le VT100, un terminal produit par DEC, extrêmement répandu, permettant de se connecter à un « ordinateur central », c'est à dire un serveur, en mode texte.



Une « machine virtuelle » est donc l'effet produit par un émulateur lorsque celui-ci simule le fonctionnement d'un ordinateur.
Le terme « machine virtuelle » fait aussi souvent référence aux interprètes de langages binaires tel que la « Java virtual machine » ou JVM qui interprète le langage JAVA une fois celui-ci traduit en binaire.
Le terme est également utilisé par des logiciels effectuant une isolation d'une partie de l'environnement de l'ordinateur pour exécuter un environnement spécifique autre que celui d'exécution du logiciel isolant : VMWare, VirtualBox par exemple.

Dans le cadre du projet REDSTEEL, vous allez devoir programmer un émulateur de micro-processeur. Ce micro-processeur comprendra un langage dont la description vous est donné parmi les ressources du projet.

Le fonctionnement du micro-processeur est de de récupérer en mémoire une information : une instruction à exécuter, ainsi que les paramètres de cette instruction, normalement situé immédiatement après. Les instructions sont toujours simples : copier une valeur, faire une addition, déplacer la tête de lecture du programme, etc.

La documentation du REDSTEEL vous est fournie au format texte en annexe du projet, parmi les ressources, au même endroit que rasm, remu et que la carte vidéo.



## 05 – Votre tâche

Contrairement à d'autres projets où vous avez du **tout** réaliser, dans le projet REDSTEEL, vous ne serez à l'origine que d'une partie du logiciel final.

Votre travail consistera à réaliser une bibliothèque dynamique implémentant certaines fonctions qui seront utilisé par remu, l'émulateur.

Le travail de remu étant de lier les différentes parties de la machine virtuelle, son micro-processeur, son clavier, sa carte vidéo, sa carte son, sa carte de gestion des disques et sa carte réseau. L'émulateur apporte la mémoire centrale ainsi que le mécanisme de chargement de deux fichiers .red contenant du code machine.

Les fichiers sont rom.red, contenant la ROM de la machine, et un autre fichier .red quelconque qui sera chargé comme « cartouche » à la manière d'une console de jeu.

Actuellement, l'émulation est hautement incomplète : il manque l'élément principal, le micro-processeur, dont le rôle est de lire et d'exécuter les instructions situés dans la mémoire de l'ordinateur. **Programmer cette partie là est votre rôle.**

Pour vous aider, vous avez bien sur remu, mais également rasm, qui permet de traduire en code machine (les fichiers .red) un langage de programmation basique appelé « assembleur ». L'assembleur est la représentation en texte du langage du micro-processeur, dans le cadre du projet, il s'agit des fichiers .rs. rasm comporte également un désassembler, c'est à dire une fonctionnalité capable de lire un fichier .red pour en sortir sous forme de texte les instructions qu'il contient.

Ce n'est pas tout, vous avez à disposition également plusieurs fichiers .rs que vous pourrez exécuter sur votre micro-processeur afin d'en tester le fonctionnement. Vous disposerez également de la TurboFX GeBansheeForce 2000RTX Voodoo, une carte vidéo virtuelle compatible avec remu et qui vous permettra d'aller encore plus loin dans vos tests.

Pour terminer, vous disposerez également d'un micro-processeur virtuel conçu par le DAEMONLAB comme référence de fonctionnement. Vous êtes bien sur libre d'étendre votre propre micro-processeur tant qu'il demeure compatible (par exemple, en ajoutant des instructions, des registres ou toute autre chose que vous pourrez imaginer)



# 06 – L'assembleur

Le logiciel rasm pour RedAssembler est un logiciel permettant d'effectuer une traduction assembleur vers langage machine (opération appelé assemblage) et également l'opération inverse (appelé désassemblage)

Le logiciel rasm vous servira à générer des programmes pour remu.

Notez bien que le désassemblage ne permet pas de résoudre tous les symboles du programme ! En effet, si vous avez crée des emplacements pour des données rangées derrière un nom, ce nom n'est jamais présent dans le programme final : l'endroit auquel il fait référence est directement utilisé. De même, les données seront désassemblé comme des instructions et non comme des données ! Bien entendu, les commentaires seront également perdus.

Par exemple, le programme suivant :

```
; BOUCLE INFINIE SUR UNE LIGNE
START:
    SET         #START, PC
```

Une fois assemblé, puis désassemblé, celui-ci donne le code suivant :

```
    SET         #0, PC
```

Le symbole **start** étant présent au début du programme (à l'adresse 0 de celui-ci), sa valeur est donc de 0. Lors de l'assemblage, les appels fait à lui sont donc remplacé par sa valeur. L'opération inverse n'est pas possible car rien ne permet de savoir que la valeur était un label dans le code original.

L'opération **SET** ici met donc la valeur 0 dans le registre « **PC** » (Program counter). Un registre est un emplacement mémoire situé dans le micro-processeur. Le **Program Counter** contient l'adresse de la prochaine instruction qui sera exécuté. Ici, c'est très clair : la prochaine instruction, c'est celle-là, donc, comme le dis le commentaire, c'est une boucle infinie.

L'écriture d'une valeur n'étant pas un multiple de 3 dans le **Program Counter** provoque sa ré-écriture immédiate par le micro-processeur au multiple de trois qui suis immédiatement.



Outre les instructions du micro-processeur, le programme rasm vous permet de manipuler le binaire de sortie lorsqu'il assemble :

La directive **data** vous permet de placer des valeurs arbitraires à l'emplacement actuel dans le programme.

```
DATA    0, 1, 2 ; 3 ENTIERS
DATA    3.0, 3.1, 3.2 ; 3 FLOAT
DATA    "ABC" ; 3 CARACTÈRES
; 1 BYTE A 0B0001101100011011
DATA    '0123 =0X'
DATA    [32] ; 32 BYTES A 0
```

Les trois entiers sont chacun sur 1 byte.

Les flottants prennent chacun 2 bytes (partie entière, partie décimale) : la partie entière exploite la capacité du byte tandis que la partie décimale varie uniquement jusqu'à 10000.

La chaîne de caractère n'incorpore pas de terminateur nul et chaque caractère occupe un octet et non un byte.

Le symbole apostrophe sert à indiquer une plage de couples de bits. Les symboles 0123 valent respectivement… 0 1 2 et 3. Les symboles espace = 0 et X valent également 0 1 2 et 3. Chaque valeur occupe 2 bits seulement. Ce format est disponible afin de faciliter l'utilisation de la carte vidéo fournie par le laboratoire de programmation générale qui exploite des couleurs sur 2 bits.

Le symbole crochet gauche, suivi d'un entier et d'un crochet droite permet d'indiquer une plage de byte mis à zéro.

La transition depuis une plage de données vers une instruction provoque un réalignement de l'adresse d'écriture sur un multiple de 3.

La directive **add_offset** permet de sauter une quantité données de bytes passée comme unique paramètre. Les bytes sautés sont initialisés à zéro.

La directive **set_position** permet de sauter une quantité de données jusqu'à la position passée en paramètre, qui ne peut être inférieure à la position actuelle.

La directive **set_label_start** permet d'ajouter une valeur à tous les labels situés après la directive. Par défaut, cette valeur est 0. Un paramètre est requis. Cette directive est indispensable à la programmation de la cartouche.



# 07 – RedEmulator

Le logiciel remu pour RedEmulator est un logiciel permettant d'effectuer la jonction entre différentes bibliothèques logicielles dynamiques dont votre micro-processeur virtuel. Il apporte un panneau de débogage permettant d'afficher des valeurs de la mémoire de l'ordinateur, qu'il s'agisse d'instruction ou de données ainsi que bien sûr l'état du micro-processeur lui-même.

Les autres bibliothèques chargées par remu sont toutes optionnelles, seule la votre est obligatoire. Ces autres bibliothèques sont les émulateurs des périphériques de saisie (clavier, souris, manette…), de carte vidéo, de carte son, de gestionnaire de disques et de carte réseau. Le DAEMONLAB vous fournit pour rappel un émulateur de carte vidéo. Cet émulateur permet un affichage à résolution variable en multi-couche. Plus la résolution est élevée, moins de couches sont disponibles. Sa documentation est avec les ressources du projet. Vous pouvez bien entendu réaliser vos propres périphériques.

Pour fonctionner, votre bibliothèque dynamique de micro-processeur devra implémenter les fonctions suivantes. Pour information, le type byte est un int16_t.

```
BYTE    GET_REGISTER(CONST CHAR  *NAME,
                     INT         DEPTH);
```

Cette fonction doit renvoyer la valeur du registre nommé **name**, cela au niveau de profondeur d'appel **depth**. Pour savoir ce que signifie cette profondeur, rendez-vous dans le fichier architecture situé en ressource.

Votre bibliothèque de micro-processeur devra également comporter :

```
VOID        RUN_DEVICE(VOID);
VOID        CLOCK(BOOL                UP);
```

La fonction clock permet d'indiquer au micro-processeur qu'il y a progression dans l'horloge de la carte mère. Vrai est envoyé si l'horloge vient de passer d'un état bas à un état haut.

L'horloge de l'ordinateur est un système qui passe son temps à changer d'état : lorsqu'on dit qu'un micro-processeur est cadencé à 3Ghz, cela signifie que son horloge fait « bas-haut » 3 milliards de fois par seconde.

Votre micro-processeur est censé agir lorsque l'horloge passe d'un état bas à un état haut.

La fonction **run_device** est appelée lorsque remu est lancé. Un thread est alors associé au périphérique.

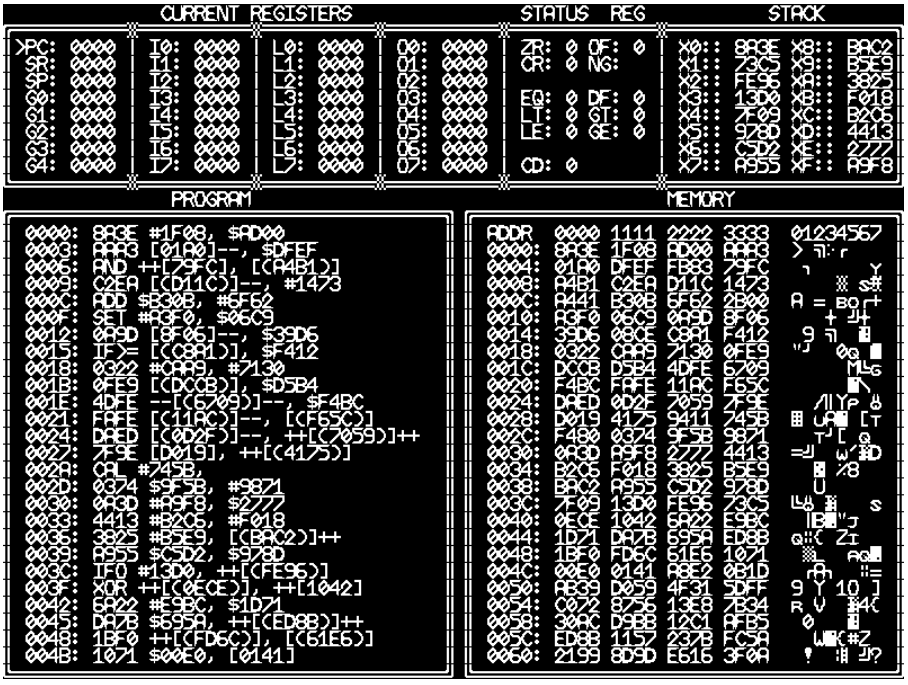
Si vous souhaitez réaliser une carte graphique, votre bibliothèque devra comporter la fonction suivante :

```
BOOL TRANSFERT_PIXEL(T_BUNNY_PIXELARRAY*PX);
```

Cette fonction permettant de remplir la fenêtre du logiciel remu. Le logiciel appellera votre fonction en vous donnant une image dans lequel dessiner ce que la carte vidéo est censé afficher. Si votre fonction renvoi faux, remu n'affichera pas px.

Le programme remu exploite la ligne de commande et un fichier de configuration afin de régler les liaisons qui existent entre lui et les bibliothèques extérieures. Un exemple vous est fourni.

Concernant le débogueur, voici comment il fonctionne :




La partie supérieure gauche « **Current Register** » montre l'état des registres tels qu'ils sont accessibles maintenant.

Les registres I/L/O dépendant de la profondeur dans la pile d'appel de fonctions. Le bloc « **Status Reg** » est un affichage alternatif pour le registre de statut.

Le bloc « **Stack** » permet d'afficher les éléments présents depuis le haut de la pile (X0) jusqu'à 15 bytes après le sommet.

Les blocs « **Program** » et « **Memory** » permettent d'afficher des emplacements mémoires arbitraires mais de manière différente : le premier bloc désassemble les données comme si il s'agissait d'instructions (Ce qui n'est pas forcément le cas) tandis que le second affiche les données de manière brute comme si il ne s'agissait que de simples données (Ce qui n'est pas forcément le cas)

La souris ainsi que la touche *tabulation* permettent de se déplacer entre les différents blocs. Les touches fléchées permettent de se déplacer parmi les éléments d'un bloc. La touche *entrer* permet d'entrer en mode édition. Actuellement, le bloc **Program** ne permet pas de modification.



# 08 – L'architecture du REDSTEEL

Vous trouverez parmi les ressources du projet ARCHITECTURE, décrivant celle-ci dans un format « rétro » adapté à l'ambiance générale du projet.

Parmi les éléments remarquables et généraux qui peuvent être notés ici se trouvent le découpage de la mémoire du RedSteel.

Les adresses 0 à 8191 sont associés à l'accès de la ROM du REDSTEEL. Une ROM est une mémoire en lecture seule. Lorsque remu démarre, c'est la ROM qui est exécute en première car le **Program Counter** démarre à zéro. La ROM est fournie par le DAEMONLAB mais vous êtes libres d'écrire la votre. Le rôle de la ROM est d'approter quelques outils utiles, quel que soit le programme qui sera ensuite chargé. La ROM fournie contient principalement une table de caractère graphique (allant de ' ' à '~') exploitable à l'adresse 0x0001. Chaque caractère mesure 7 caractères de haut et 5 de large. Chaque pixel est sur 2 bits, soit le format binaire géré par rasm et par la carte vidéo qui vous est fournie.
Sur remu, la ROM n'est pas en lecture seule pour des raisons d'implémentation.

Les 8KB (8-16) suivants sont associés à la « **cartouche** », c'est à dire le programme passé par la ligne de commande.

Les 8KB (16-24) suivants sont associés à la **mémoire** centrale, la RAM, d'intérêt général.

Les périphériques de **saisie** sont situés de 24 à 32.

La carte **vidéo** est située de 32 à 40.

La carte **son** de 40 à 48.

Le contrôleur de **disque** de 48 à 56.

La carte **réseau** de 56 à 64.



# 09 – Étendre REDSTEEL

Outre l'écriture de périphériques, vous pouvez également apporter de nouvelles instructions au micro-processeur, ainsi que de nouveaux registres. Le panneau de débogage de remu ne sera pas en mesure de s'adapter à vos ajouts mais le système lui-même fonctionnera.

La grande question étant : quelle absence vous chagrine lorsque vous programmez en assembleur ? Une instruction en plus vous arrangerait ? Un registre vous manque ? Des interruptions extérieures ?

Bien sur, l'adjonction d'instruction n'est pas la seule façon d'améliorer le langage ! Il est également possible de créer des constructions par dessus le langage actuel, des macros par exemple, de la même manière que vous le faites en C.

Un jeu de macro peut apporter énormément de confort à la programmation. Tellement d'ailleurs que certains ont eu l'idée il y a des dizaines d'années de programmer exclusivement avec des langages apportant ces « sucres syntaxiques » qui facilitent la vie… Vous aurez compris qu'il s'agit des langages de programmation en général.

Ne vous lancez cependant pas dans une amélioration de REDSTEEL avant d'avoir d'abord accompli l'essentiel : la réalisation d'un micro-processeur compatible avec les programmes imposés.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Une proposition d'ajout : la possibilité de définir :

- La possibilité d'enregistrer une identité d'exécution.
- Le lancement d'une fonction avec une certaine identité, pour un nombre de tour d'horloge donné, avec sauvegarde automatique de la position ou l'exécution s'arrête une fois le nombre de tour d'horloge épuisé.
- La possibilité de relancer une exécution de fonction qui s'est arrêté.
- L'enregistrement de droits de lecture et d'écriture en mémoire pour une identité d'exécution donné et pour chaque bloc de 512 et l'arrêt de l'exécution du fil actuel en cas d'infraction.
- La mise en place d'une table de correspondance entre un espace mémoire virtuel associé à une identité d'exécution et une adresse réelle en mémoire.
- La possibilité pour une identité d'exécution mineure d'appeler une fonction de l'identité d'exécution principale.

Cet ensemble d'ajouts permettrait la mise en place d'un système d'exploitation multi-tache sur le REDSTEEL.



# 10 – Votre démo

Une fois réalisé votre micro-processeur virtuel, écrivez une démo pour le REDSTEEL : des flammes, un plasma, un dégradé dynamique, vous êtes libre.

Le code source de votre programme devra être fourni en assembleur .rs dans votre rendu, avec son propre dossier, compilable avec un Makefile de la même manière que votre programme, sauf qu'en lieu des règles types .c.o, vous utiliserez .rs.red.



# 11 – Projets annexes

REDSTEEL est membre d'une série de projets. Ci-dessous la liste des projets faisant partie de la même série.

REDASM, consistant à programmer l'équivalent de rasm.

REDTONG, consistant à programmer un compilateur pour le langage du projet BABL vers REDASM. BABL étant un projet de programmation d'interprète.

REDSPICE, consistant à programmer un simulateur de circuits électroniques à composants numériques.

REDSTEEL2, consistant à implémenter le micro-processeur et sa carte mère dans l'environnement de REDSPICE.

REDSTEEL3, consistant à implémenter le micro-processeur en VHDL.

Il ne s'agit pas d'un projet scolaire, mais d'une proposition aux passionnés de matériel informatique, d'électronique et de programmation bas niveau : la réalisation matérielle d'un REDSTEEL.