ASR1, DM2
Un peu d'assembleur de votre processeur

Florent de Dinechin

10 décembre 2007

Intro technico-commerciale

Ce devoir est à faire par deux, et à rendre pour mercredi 19 décembre à 22h par mail. La section 1 est triviale, et les sections 2.1 et 2.2 sont totalement indépendante, sauf qu'elles requièrent exactement la même expertise. Vous êtes invités à les attaquer en parallèle.

Il faut commencer par récupérer le fichier `VLIW2007v1.0.tgz` de la page web d'ASR1[1], lire les README des répertoires asm, puis simu, et tester dans le simulateur les deux programmes objets fournis.

Donc l'écran se compose de 256 lignes de 320 caractères, commence à l'adresse 0, et se termine à l'adresse $256 \times 320/4$, car chaque pixel est codé par un octet, donc un quart de 32 bits.

Vous êtes invités à vous inspirer du programme `deco.asm` fourni.

Je vous suggère, au moins au début, de travailler en deux temps : d'abord écrire vos programmes sur une seule voie, et ensuite seulement chercher à utiliser les deux voies au mieux.

1 Le nez dans le programme d'assemblage

Modifiez le programme `asm.cc` pour y ajouter un warning pour chaque instruction indéterminée. Aucune connaissance de C++ n'est nécessaire, un peu d'imitation suffira.

Vous me rendrez un fichier `asm.cc` qui fonctionne toujours, et dans lequel vos changements sont précédés du commentaire `//DM2`.

Au passage, la structure de ce programme est un *recursive descent parser*.

Bonus : ce programme a été écrit sur un coin de table en période de surmenage. Tout bug original rapporté (on n'a pas laissé d'erreur dedans exprès, promis !) vaudra un nombre de point proportionnel à sa subtilité.

2 Écrivons le BIOS

Dans toute la suite on suppose que l'adresse de retour d'une sous-routine est sauvegardée dans `r15`, par exemple par l'instruction `sra r15 / jrf`

[1] http ://perso.ens-lyon.fr/florent.de.dinechin/enseignement/2007-2008/ASR1/

`toto`. On revient donc d'une sous-routine par `ja r15`. Cette spécification est le début d'une ABI *(application binary interface)*. Si une routine en appelle à son tour une autre, il est de sa responsabilité de sauvegarder r15 quelque part : soit dans un coin qu'elle a réservé pour cela, si elle n'est pas récursive. Soit sur une pile si elle est récursive. Dans ce DM on ne fait que des fonctions pas récursives.

2.1 Graphiques

Écrivez une routine `plot` qui affiche un point à l'écran. Elle prendra x, y et une couleur respectivement dans r1, r2 et r3. Elle se permet de modifier ces trois variables. Elle ne vérifie pas que le point est dans l'écran ($0 \leq x \leq 319$, $0 \leq y \leq 255$) ni que la couleur est bien un octet ($0 \leq$ `r3` ≤ 255), et a le droit de planter le système si ce n'est pas le cas.

Si vous êtes trop bête pour cela, désassemblez le programme `sinus.obj`. C'est du boulot aussi. Si vous écrivez un programme désassembleur je serai super content. Cela se fait vite en partant de simu.cc.

Vous pouvez aussi faire les malins en retrouvant la partie qui correspond à la routine `plot` dans `sinus.obj`, et en la recopiant sans la comprendre dans votre fichier asm, en précédent chaque ligne d'un underscore.

Sinon, ma fonction `plot` fait 19 instructions. Cette question vous rapportera donc un nombre de point proportionnel à $20 - n$ points, où n est le nombre d'instructions de votre routine à vous.

Réalisez une seconde routine `safeplot` qui précède la première des tests nécessaires. Un point hors écran n'est pas écrit, et sinon elle fait un `and r3 0xff`. Par ailleurs, `safeplot` renvoie dans r1 la couleur qui était à l'écran avant le plot.

Vous me rendrez un fichier `plot.asm` qui au minimum boucle sur l'affichage d'un point blanc au milieu de l'écran. Vous pouvez même faire varier la couleur, allez.

Bonus : Utilisez cette routine pour tracer ce que vous voulez. Quand on aura le clavier qui marche, on pourra faire un télécran.

2.2 Texte

Dans toute la suite les caractères sont des carrés de 8 pixels par 8 pixels, donc chaque ligne d'un caractère est codée par un octet. On vous fournit sur la page d'ASR1 le fichier `alt-8x8` (cherchez-le dans votre distribution linux préférée) qui contient des codages de caractères – vous pouvez en utiliser un autre si cela vous chante. Pour comprendre comment cela marche, ouvrez ce fichier en mode hexa dans Emacs (esc-X hexl-mode), et allez à la ligne 29. Elle contient visiblement 16 octets, donc deux caractères complets. Tracez sur une feuille quadrillée deux carrés 8x8, et replissez le premier avec les 8 premiers octets (bit à 1 : une croix, bit à 0 : on laisse blanc), et le second avec les 8 octets suivants. Qu'apparaît-il sous vos yeux éblouis ?

Écrivez une routine qui s'appelle `putcxy`, qui prend dans r1 un octet, dans r2 et r3 des coordonnées de caractère (respectivement entre 0 et $320/8 = 40$, et entre 0 et $256/8 = 32$), et dans r4 une couleur, et affiche le caractère ASCII correspondant aux coordonnées de caractère r2 et r3. Le fichier `alt-8x8` est dans l'ordre ASCII. Vous aurez à le recopier (au moins sa première moitié) dans votre fichier asm, après l'avoir converti au bon format.

Ecrivez une routine `putc` qui prend juste dans r1 un octet, et qui l'affiche à la position du cursor : `putc` appelle `putcxy`, et met à jour le curseur. Le curseur est initialement à x=0, y=0.

Ecrivez un programme qui affiche les caractères 33 à 126, saute deux lignes, puis affiche "Joyeux noel". Vous me rendrez un fichier `noel.asm`.

Bonus : au moyen de votre routine `plot`, faites tomber des flocons de neige sur l'écran.

3 Bonus pour les fayots et les oisifs

Réfléchissez à une ABI pour gérer une pile. On pourrait déclarer que r14 est le pointeur de pile, et que personne n'y touche. Réfléchissez à des séquences d'instructions qui sauvegardent des registres sur la pile, et qui les récupèrent.

Moi pendant ce temps je vais faire marcher le clavier dans le simulateur. Son interface occupera les adresses 0x7F00 à 0x7FFF. Je vous ferai passer le simulateur modifié à l'occasion. La pile sera donc une pile descendante qui part de 0x7EFF.

J'aimerais ensuite qu'on fasse au moins un programme qui affiche à la position du curseur un caractère tapé au clavier.

Ensuite, grâce à notre pile, on pourra porter OCaml.