*Année 2015-2016*

*Devoir de programmation de L'UE Compilation Avancée (4I504)*

# *La machine Universelle UM*

réalisée par :

- CHELBI Zahir

Enseignants:

Emmanuel Chailloux,

Jérémie Salvucci

# 1. Introduction:

L'objectif de ce devoir de programmation , est de réaliser une machine virtuelle capable d'interpréter « tous les programmes de la machine universelle UM ,y compris les fichiers non compressés .um et fichiers auto-extractibles .umz »[1] ; selon la spécification donnée sur le site du concours icfp2006 : *http://www.boundvariable.org/task.shtml#materials*

# 2. Définition de La machine UM :

La machine qu'on va construire dans ce projet comporte :

- 8 registre a usage général

- une collection de tableaux de plateaux ; dont le tableau a l'indice '0' sera réservé pour contenir le programme de la machine .

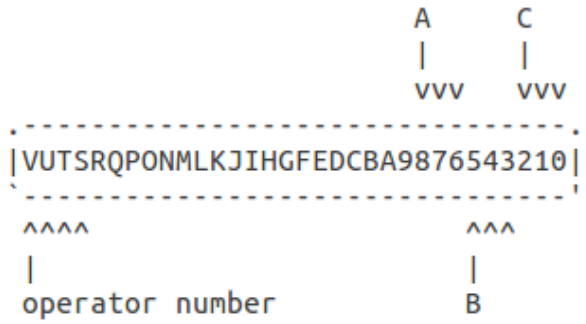
- une console pour les entré/sortie de la machine(lecture/écriture) ; nous en se servira du terminal du système .

Le code interprété par la machine serra une suit de mots binaires de 32 bits , chaque mot définie un opérateur ainsi que les registres utilisés tel que :

- les 4 bits de poids forts donne l'opérateur .

- les 3 bits de poids faible donnent le contenue du registre C , les 3 suivant donnent le contenue du registre B , et Les 3 autres qui suivent donnent la valeur du registre A .

la figure suivante explique le contenue d'un mots et les bits qui donnent l'opérateur et ces registres .




## 2.1. Initialisation de la machine :

Au démarrage on initialise l'ensemble des registres a zéro , puis en commence a lire les instructions et les stocker dans un tableau qui servira par la suite du tableau 0 qui contient le programme de la machine , puis en commence a décoder les instructions l'une après l'autre ;

# 3. Implantation de la machine UM :

Le Langage utilisés est Le **« C » ,** la machine comporte essentiellement 3 partie :

initialisation du tableau initial et des registres , lecture du fichier et stocker les instruction dans le tableau initial , et enfin interprétation de instruction .

## 3.1. Les Macro utilisés

pour facilité la lecture du code on définie un ensemble de **macro** :

**# define C mot & 7 :** le contenue du registre C .

**# define B (mot >> 3) & 7 :** le contenue du registre B .

**# define A (mot >> 6) & 7 :** le contenue du registre A .

**# define SP (mot >> 25) & 7 :** le contenue du registre spécial SP .

**# define VAL mot & 0x1ffffff :** la valeur du registre spécial .

## 3.2. La taille du fichier

La fonction **get_file_size** donne la taille du fichier lu , cette fonction va être utile pour définir la taille du tableau initiale grâce a deux fonction **fseek et ftell** qui gère les déplacement dans un fichier

## 3.3. Initialisation du tableau '0'

On commence par allouer un espace mémoire pour stocker le programme, pour cela on récupère la taille du fichiers , et on utilise la fonction **Calloc** qui alloue l'espace mémoire et initialise ces case a zéro :

```
initTab = calloc( (unsigned int) get_file_size(argv[1])/4,4) ;
```

## 3.4. Lecture du fichier

Pour initialiser le tableau '0' on fait une boucle qui lit du fichier 4 caractères , qu'on décale a chaque fois de 8 pour constitué un **entier non signé de 32bits** puis en t'ajoute au tableau '0' :

```
while(EOF != (a = fgetc(file))) {
        a=a << 24|fgetc(file)<< 16 | fgetc(file) << 8 |fgetc(file) ;
        initTab[i] =  a;  i++;
}
fclose(file);
```

## 3.5. La boucle principale :

C'est la partie la plus importante de la machine virtuelle, elle lit notamment les instructions une par une du tableau initial grâce au compteur du code **pc** , puis elle stocke l'instruction dans un entier 32 bits non signé **unsigned int mot = initTab[pc++];** et en fonction de l'opérateur obtenu après décalage on traite l'instruction .

## 3.6. Les opérateurs de base

**A . Conditional Move ( 0 )**

Une simple affectation du registre A par la valeur du registre B , sauf si le registre C a la valeur 0 :

**if (reg[C]) reg[A] = reg[B];**

**B . Array Index ( 1 )**

Le registre A devient égale a la valeur du tableau identifié par B dans l'offset C ; et si le registre B est vide , le registre A reçois la valeur du tableau 0 a la place de B ;

**C. Array Amendment ( 2 )**

modification du tableau identifié par le registre A a l'offset B par la valeur du registre C , et si le registre A vaut zéro alors on modifie le tableau 0 ;

**D. Les opérateurs de base**

Les opérateurs 3 , 4 , 5 et 6 sont de simples opérations sur les registres.

## 3.7. Les Autres opérateurs

**E. Halt ( 7 )**

cet opérateur n'utilise aucun registre , il sert uniquement a mettre fin au programme .

**F. Allocation( 8 )**

cette opération permet de créer un nouveau tableau de capacité égale a la valeur du registre C, pour facilité la récupération de ça taille on alloue un tableau de taille **reg[C]+1** , la case supplémentaire sert a stocker la taille du tableau ; on utilise calloc pour l'initialiser a zéro en même temps , l'adresse de ce tableau serra mis dans le registre B :

```
unsigned int *r = (unsigned int*)calloc((reg[C]+1),4);
r[0]=reg[C];
reg[B]=(unsigned int)(r+1);  break;
```

**G. Abandonment ( 9 )**

cette opération permet de libérer un tableau déja allouer , et comme lors de l'allocation on a ajouter une case pour stocker sa taille la léberation commence de 'indice **reg[C]-1 :**

```
free( (unsigned int*)reg[C]-1 ); break;
```

**H. Output (10 )**

on écrit sur la console la valeur du registre C .

**I. Input ( 11 )**

récupère l'entrée de la console et la stocke dans le registre C .

**J. Load Program( 12 )**

dans cette opération on duplique le tableau identifier par le registre B, on commence par libérer le tableau '0'

```
free(initTab );
```

puis en alloue un nouveau tableau de la même taille que B :

```
int size = ((unsigned int*)reg[B])[-1];
initTab = (unsigned int*)malloc(size*4);
```

puis grâce a la fonction **memcopy** on copie le tableau identifier par B dans le tableau '0'

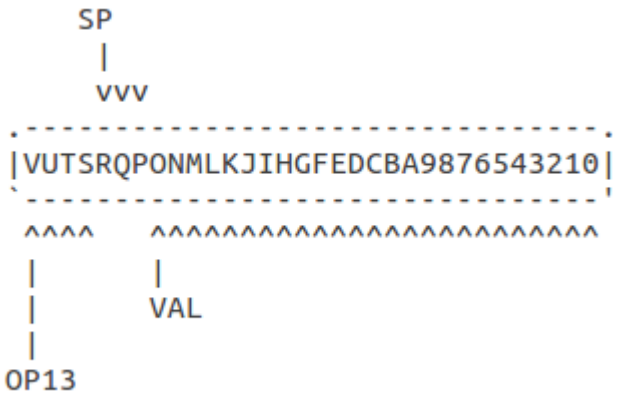
```
memcpy(initTab, (unsigned int*)reg[B], size * 4);
```

et enfin en met la valeur du registre C qui correspond au compteur du code dans **pc** :

```
pc = reg[C]; break;
```

**J. Opérateur spécial ( 13 )**

le registre spécial **SP** reçois la valeur **Val** des 25 bits restants .




# 4. Compilation et Tests de la machine UM :

## 4.1. Les Autres opérateurs

Pour éviter les problème de représentation des entier 32 bits sur des processeurs 64 bits en utilise l'option **m32** du compilateur **gcc ,** donc au final la compilation se fait avec la commande suivante :

**gcc -m32 -o machine machine.c**

pour facilité la tache notre projet contient un **Makefile ,** qui réalise ces opération .

### 4.1. Les tests :

Afin de verifier que la machine donne bien le comportement attendu en la tester sur des fichier .um suivant :

**1. les fichier recupérer sur le site icfp2006 [1]**

codex.umz sandmark.umz

**2. des fichier d'extention .um recupéré sur le site COMP40 [2]**

advent.umz hello.um cat.um midmark.um

**2. Autres fichier**

cette machine est capable d'executer n'import quel fichier de type um ou umz ; pour executer il suffit de concaténer le programme avec le fichier um.um[] :

**cat um.um c.um > cmu.um**

## 5. Conclusion :

Une machine virtuelle est une programme capable d'executer du code compilé , elle permet de sémuler le fonctionement d'un processeur .

La vm réalisé dans ce devoir est une machine virtuelle qui execute des programme UM

Ce devoir de programmation est tres interressant , il nous a permet de comprendre le fonctionnnement général des machines virtuelles , plus précisement la machine Universelle

*Réferences :*

*[1] : http://www.boundvariable.org/task.shtml#materials*

*[2] : http://www.cs.tufts.edu/comp/40/*

*[3] : https://www-apr.lip6.fr/~chaillou/Public/enseignement/2015-2016/ca/*