[Architecture et documentation du projet]

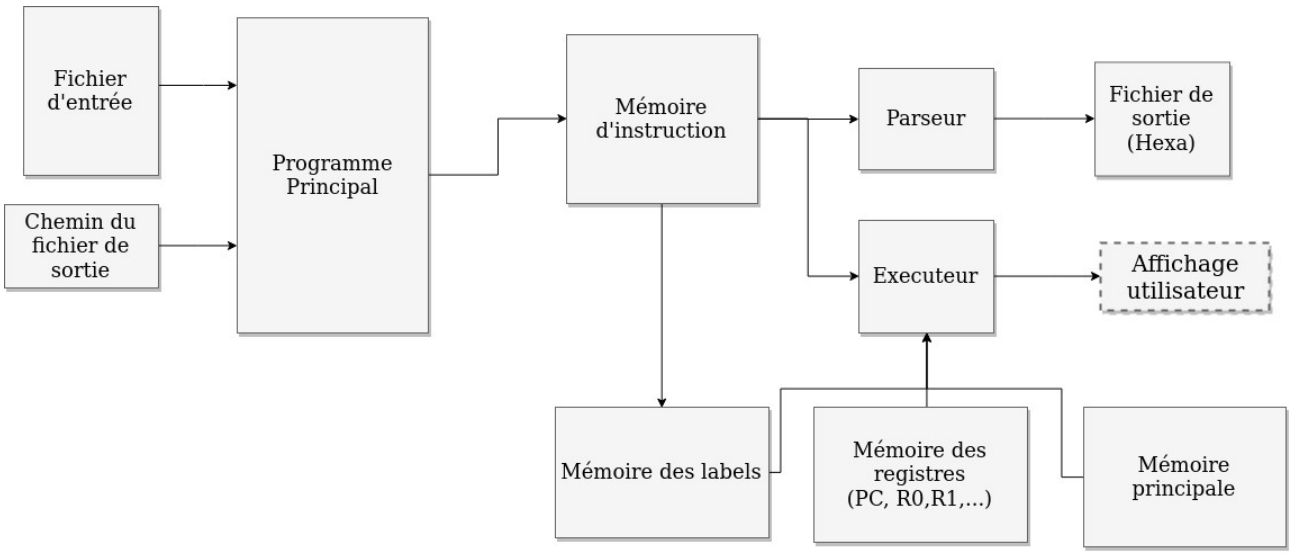
Dautey Marin
Ribiere Matthieu

Projet MIPS

Année 2019-2020

Architecture de l'émulateur MIPS

Pour concevoir notre émulateur, nous avons fait le choix de l'architecture suivante :



Cette architecture peut être décomposée en quatre blocs : un bloc de traitement d'entrée, un bloc de mémoires relié à un bloc de traitement principal, et un bloc de traitement de sortie.
Nous allons maintenant aborder plus en détails chacune de ces parties, puis résumer notre avancée ainsi que ce qu'il nous reste à faire.

I) Le Programme principal

Le programme principal est responsable du démarrage de tout le reste de l'application. Il doit notamment vérifier que tous les arguments nécessaires au bon fonctionnement du programme ont été fournis. Il va ensuite se charger de stocker toutes les instructions dans la mémoire d'instruction.
Cette dernière remplit ce rôle en s'appuyant sur un module que nous avons conçu dans l'optique d'effectuer le moins d'opérations possible sur le fichier (lesquelles sont coûteuses en temps), et permet également de faire remonter les cas d'erreurs (fichier manquant, problèmes de permission,...). Ce module est également conçu pour l'opération inverse, soit l'écriture de chaînes de caractères dans un fichier. Les même cas d'erreur que précédemment seront traités.
Nous donnerons un tableau récapitulatif des codes d'erreur et leurs

significations en annexe I de ce document.
Cette portion du code est également responsable de la déclaration, de l'initialisation, et de la transmission des différentes mémoires de l'émulateur aux autres modules.

II) Les mémoires d'Instruction, de données, de labels et les registres

Ces différentes mémoires constituent notre bloc dédié à la mémoire virtuelle, bien que possédant des propriétés très différentes

La mémoire d'instruction utilise une structure simple composée d'un tableau de chaînes de caractères, pour retenir toutes les instructions. On remarquera cependant que les labels sont complètement absents de cette mémoire. Ils ont été mis dans un espace mémoire réservé pour un usage à postériori.

Cette partie réservée nommée mémoire des labels, repose sur une mécanique extrêmement simple : elle fait correspondre un nom de label à un index dans la mémoire de programme. Cela permet de revenir à une étape antérieur du programme sans encombrer la mémoire de labels, ou de convertir ces labels en décimal signé.

La mémoire principale ainsi que celle des registres sont basées sur la même structure. Celle-ci sera donnée en annexe II et III de ce document.
Ce choix d'avoir la même structure mais deux objets différents, est un choix qui certe allonge le code de quelques lignes, mais en contrepartie, on gagne énormément en lisibilité dans le code. De plus on évite les confusions entre les registres, et la mémoire principale qui même si ayant la même structure, ont des rôles très différents.

En effet la mémoire des registres a été conçue dans le but de retenir les opérandes utilisés lors des instructions. Certains registres auront toutefois un rôle un peu plus particulier (par exemple le compteur ordinal ou PC sera chargé d'indiquer la prochaine instruction à exécuter, le registre R0 contiendra toujours la valeur 0x00,...).
La mémoire principale, aussi appelée mémoire de données, est comme son nom l'indique, une mémoire de très grande taille qui servira à stocker de grande quantités de données.

III) Le bloc d’exécution et le parseur

A l’aide des mémoires d’instruction, de données, de labels et des registres, nous serons alors en mesure d’exécuter soit tout le programme d’un seule traite, soit une instruction après l’autre (saisie par l’utilisateur). Néanmoins, quel que soit le mode choisi, le traitement restera sensiblement le même.
Le bloc exécuteur, comme son nom l’indique, exécutera l’instruction qui doit l'être, mais devra également afficher le résultat de cette exécution dans le terminal.
Le bloc parseur sera quant à lui chargé d’écrire ses instructions en hexadécimal dans un fichier.

IV) Conclusion et avancée actuelle

L’architecture précédente nous permettra ainsi de proposer les deux modes demandés (interactif avec l’utilisateur ou non). Chaque mémoire ou fichier pourra être accédée en lecture ou en écriture, ce qui offrira une grande souplesse pour la suite.
Les blocs principaux, à savoir le programme principal, les différentes fonctions de lecture/écriture, ainsi que le parseur sont dans un état quasi-final. Les mémoires de registres, de label, et principal, sont en cours de développement. Ils produisent à l’heure actuelle, des résultats prometteurs et qui permettront de développer l'exécuteur sur des bases aussi flexibles que robustes.

- DAUTREY Marin
- RIBIERE Matthieu

ANNEXE I :

Code d'erreurs	Signification
1	Pas assez d'arguments en appelant le programme
2	Erreur lors de l'ouverture du fichier d'entrée ou de sortie
3	Le registre n'est pas accessible en lecture ou en écriture
4	L'emplacement mémoire n'est pas accessible en lecture ou en écriture
50	L'instruction n'est pas reconnue

ANNEXE II :

```
typedef struct procRegister {
        int memorySize;
        int *mem;
}ProcRegister;
```

ANNEXE III :

```
typedef struct mainMemory {
        int memorySize;
        int *mem;
} MainMemory;
```