

Rapport de projet : Architecture des ordinateurs

Benjamin Saint-Sever, Steven Ratton

21 avril 2014

Table des matières

1	Introduction	2
2	Exercice 1	3
2.1	implémentation de leal	3
2.2	Architecture séquentielle	5
2.3	Architecture pipelines	7

Chapitre 1

Introduction

Le but de ce projet est de montrer la possibilité d'extension de l'architecture y86, en ajoutant des instructions et en manipulant les prédictions de branchement. Ce projet a pour vocation de nous enseigner la programmation de langage machine tout en tirant parti au mieux des capacités des ordinateurs.

Chapitre 2

Exercice 1

2.1 implémentation de leal

Description

Le but de ce premier exercice est de créer une nouvelle instruction "leal" pour "load effective address", cette dernière est existante sous l'architecture x86. Cette instruction charge l'adresse de la source dans dest (leal(%regS),%regD). On souhaite que cette instruction permette un déplacement mémoire, leal depl(%regS),regD. L'avantage de cette solution est que l'on peut effectuer cette opération avec une seule instruction. Son équivalent est : "rrmovl %regS,%regD ; iaddl depl,%regD"

Assembleur

Il est possible d'ajouter cette instruction sans consommer un nouvel opcode, il suffit de reprendre le formalisme de irmovl est de donner une valeur de ifun différente afin de faire la distinction (ifun =1).

Première étapes, on insère dans le code assembleur la déclaration de cette nouvelle instruction :

1. isa.c

```
instr_t instruction_set [] =
{
/*Ajout de leal avec iFun a 1*/
{"leal", HPACK(I_IRMOVL, 1), 6, M_ARG, 1, 0, R_ARG, 1, 1 },
```

2. yas-grammar.lex

```
Instr          leal | rrmovl | rmmovl | mrmovl | irmovl |
```

Fichier test_leal.y :

On déplace de 8 à partir de l'adresse du registre %eax et on attribue l'adresse dans le registre %ebx :

```
irmovl 3, %eax
leal 8(%eax), %ebx
halt
```

Code HCL :

On ajoute le code d'instruction irmovl pour identifier leal :

```
intsig LEAL      'I_IRMOVL'
```

2.2 Architecture séquentielle

Etape **FETCH** :

On ne modifie rien puisque l'on interprète le même fonctionnement que l'instruction `irmovl`.

Etape **Decode** :

le contenu du registre 'B' (Source) est attribué à la source A.

```
int srcA = [  
    #icode vaut irmovl ou leal, pour obtenir leal -> ifun=1  
    (icode == LEAL) && (ifun == 1):rB;  
];
```

le contenu du registre 'A' est attribué à la E destination afin d'écrire la nouvelle adresse de destination.

```
int dstE = [  
    (icode == IRMOVL) && (ifun == 0) : rB;  
    (icode == LEAL) && (ifun == 1) : rA;  
];
```

Etape **Execute** :

On place `valC` dans l'`aluA` et `valA` dans l'`aluB`, cela permet de effectuer le calcul de déplacement :

```
int aluA = [  
    (icode == IRMOVL) && (ifun == 0) : valC;  
    (icode == LEAL) && (ifun == 1): valC;  
];  
int aluB = [  
    (icode == IRMOVL) && (ifun == 0) : 0;  
    (icode == LEAL) && (ifun == 1): valA;  
];
```

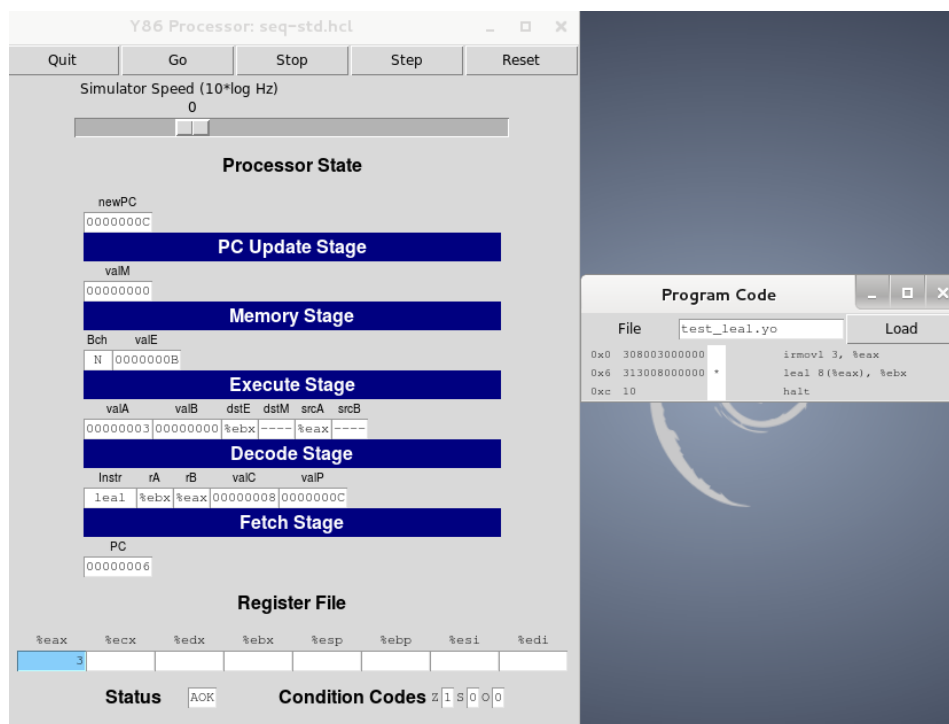


FIGURE 2.1 – Fetch

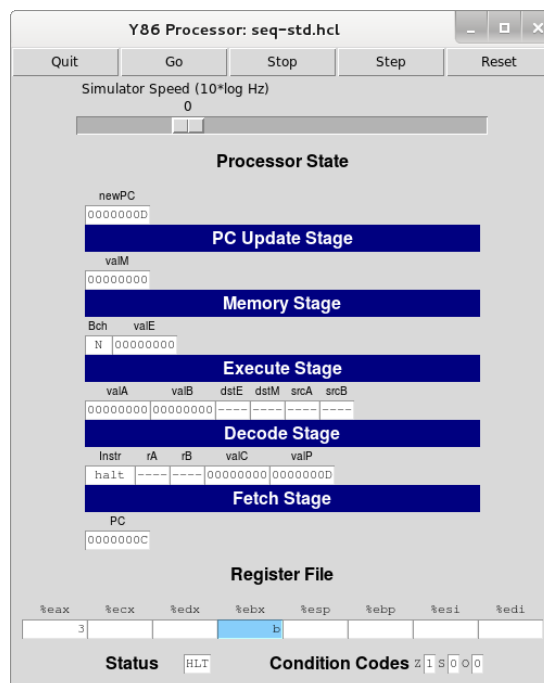


FIGURE 2.2 – Decode

2.3 Architecture pipelines