Un émulateur PROCSI
Projet de programmation en C
Polytech Sophia - Département Si3

Sylvain Lippi

26 janvier 2009

1 Introduction

L'émulation consiste à substituer un élément de matériel informatique tel un terminal informatique, un ordinateur ou une console de jeux par un logiciel appelé *émulateur*. On souhaite réaliser un émulateur permettant d'exécuter un sous-ensemble significatif des instructions de l'assembleur PROCSI. Toute ressemblance avec un autre cours n'est ni fortuite ni indépendante de la volonté de l'auteur.

2 Piqûre de rappel : l'assembleur PROCSI

On donne ici la description d'un langage assembleur simplifié.

2.1 Architecture générale

Il s'agit d'une architecture 16 bits ; un mot est donc codé sur 16 bits. De plus, une adresse mémoire correspond à un mot tout entier.[1] Le processeur contient 8 registres généraux $R_0, ..., R_7$. On considère qu'une étape d'exécution élémentaire est l'exécution d'une instruction assembleur. Autrement dit, on ne détaillera pas l'exécution en micro-instructions et l'on se contente donc des trois registres spécialisés suivants :

`PC`	Program Counter	adresse de l'instruction en cours d'exécution
`SP`	Stack Pointer	adresse du sommet de la pile d'exécution
`SR`	Status Register	résultat de la dernière opération de calcul

2.2 Jeu d'instructions

2.2.1 Modes d'adressage

Chaque opération fait intervenir, au plus, deux opérandes. On distingue l'opérande *source* qui est laissé inchangé de l'opérande destination qui est modifié. Le mode d'adressage détermine le mode d'accès aux opérandes *source* et *destination* d'une opération. On considère les modes d'adressage suivants :

- *Registre* : on donne le numéro du registre contenant la valeur de l'opérande.
- *Direct* : on donne l'adresse mémoire où se trouve l'opérande.
- *Indirect* : on donne le numéro du registre contenant l'adresse mémoire où se trouve l'opérande.
- *Immédiat* : on donne la valeur de l'opérande.

[1] et non pas à un octet comme dans la plupart des machines actuelles.

2.2.2 Liste des instructions

Codeop	Modes d'adressage Destination	Source	Exécution
load	registre	direct	destination $\leftarrow$ source
	registre	immédiat	
	registre	indirect	
store	direct	immédiat	destination $\leftarrow$ source
	direct	registre	
	indirect	immédiat	
	indirect	registre	
add	registre	registre	destination $\leftarrow$ source + destination
	registre	immédiat	
	registre	direct	
	registre	indirect	
sub	registre	registre	destination $\leftarrow$ source $-$ destination
	registre	immédiat	
	registre	direct	
	registre	indirect	
jmp	-	immédiat	PC $\leftarrow$ source
jeq	-	immédiat	PC $\leftarrow$ source si SR $= 0$
call	-	immédiat	[SP] $\leftarrow$ PC SP $\leftarrow$ SP $- 1$ PC $\leftarrow$ source
ret	-	-	SP $\leftarrow$ SP $+ 1$ PC $\leftarrow$ [SP]
push	-	registre	[SP] $\leftarrow$ source SP $\leftarrow$ SP $- 1$
	-	immédiat	
	-	direct	
	-	indirect	
pop	registre	-	SP $\leftarrow$ SP $+ 1$ destination $\leftarrow$ [SP]
	direct	-	
	indirect	-	
halt	-	-	Arrêt de l'exécution

Remarque 2.1 *La pile croît dans le sens des adresses décroissantes.*

2.2.3 Codage des instructions

Chaque instruction est codée sur un, deux ou trois mots.

- Le premier mot est codé de la façon suivante :

codeop	mode d'adressage	n. registre source	n. registre destination
6 bits	4 bits	3 bits	3 bits

- Un deuxième (voire un troisième) mot est nécessaire dans le cas où l'on utilise l'adressage immédiat ou direct. Dans ce cas, ce mot supplémentaire contient la donnée (mode immédiat) ou l'adresse (mode direct).

Le codage du mode d'adressage est le suivant :

Mode d'adressage		Code
Destination	Source	4 bits
registre	registre	0000
registre	immédiat	0100
registre	direct	1000
registre	indirect	1100
direct	immédiat	0101
direct	registre	0001
indirect	immédiat	0110
indirect	registre	0010

Remarque 2.2 *Selon le mode d'adressage, les deux derniers champs du premier mot sont éventuellement inutilisés et leur contenu indéfini.*

3 Travail à effectuer

3.1 Chargement d'un programme et initialisation

La mémoire centrale centrale d'une machine PROCSI correspond simplement à un tableau d'entiers de 16 bits. Un programme commence à l'adresse 0. Le registre `PC` est donc initialisé à zéro. Il n'est pas obligatoire d'implémenter l'assemblage du code assembleur ; vous pouvez donc directement initialiser la mémoire dans le code de l'émulateur. Comme montré dans l'exemple suivant, on utilisera les types unions et les champs de bits afin de charger facilement le codage d'un programme dans la mémoire.[2]

```
 1  typedef union
 2  {
 3      short brut;
 4      struct
 5      {
 6          unsigned codeop : 6;
 7          unsigned mode : 4;
 8          unsigned source : 3;
 9          unsigned dest : 3;
10      } codage;
11  } mot;
12
13  mot mem[TAILLE_MEM] = {
14      {.codage ={ ADD,  REGREG,3,4    }},
15      {.codage ={ LOAD, REGDIR,0,2    }},
16      {.brut =    1000                },
17      {.codage ={ JMP,  REGIMM        }},
18      {.brut =    1500                }
19  };
```

[2]Les identificateurs en majuscules correspondent à des constantes préalablement déclarées.

Remarque 3.1 *Conformément à la remarque 2.1, on initialisera le registre* `SP` *à la plus grande adresse valide.*

3.2 Exécution

L'exécution doit normalement s'arrêter sur l'instruction `halt`. Cependant, n'oubliez pas de tester les cas d'arrêt intempestif et d'afficher un message d'erreur correspondant.

3.3 Déverminage

Il doit être possible de suivre l'exécution pas à pas en affichant les instructions exécutées sous forme désassemblée c'est-à-dire lisibles par un être humain ! On doit pouvoir aussi afficher la valeur des registres et de la mémoire.

4 Consignes

Ce travail doit être effectué **obligatoirement en binôme** et doit être envoyé au plus tard le **vendredi 30 janvier** à `lippi@polytech.unice.fr` . Il est constitué d'une archive au nom du binôme contenant :

- l'ensemble des fichiers sources C (.h et .c)
- le fichier Makefile
- un ensemble d'exemples commentés permettant de tester votre émulateur
- Une documentation du code et de l'émulateur.

Remarque 4.1 *Vous pourrez tirer profit d'un outil de génération automatique de documentation comme celui présenté sur* `www.doxygen.org`.

Vous serez évalué en priorité sur la clarté du code et de la documentation. Terminons par une citation utile : «*Améliorer ce n'est pas ajouter mais retrancher*», Antoine de Saint-Exupéry.