Travaux Pratiques Projet de compilation
Informatique 2ème année. ENSEIRB-MATMECA 2014/2015

Résumé assembleur Intel

Registres disponibles :

Les machines de l'école sont 64 bits, les registres pour manipuler les adresses devront être sur 8 octets donc.

- Général purpose : registres rax, rbx, rcx, rdx, rsi et rdi utilisables en mode, 64, 32 16 ou 8bits :

<table>
<tr><td>64</td><td>32</td><td>16</td><td>8</td></tr>
<tr><td colspan="4">RAX</td></tr>
<tr><td></td><td colspan="3">EAX</td></tr>
<tr><td colspan="2"></td><td colspan="2">AX</td></tr>
<tr><td colspan="2"></td><td>AH</td><td></td></tr>
<tr><td colspan="3"></td><td>AL</td></tr>
</table>

- 8 registres 64 bits : r8 à r15
- stack pointer : rsp (pointeur de pile)
- frame pointer : rbp
- 16 registres SSE (128bits) : xmm0 à xmm15

Quelques instructions arithmétiques de base sur entiers : (utiliser le suffixe l pour les entiers 32 bits) :
`https://docs.oracle.com/cd/E26502_01/html/E28388/ennbz.html#eoizh`

Instructions SSE :

Les instructions SSE sont des instructions dites SIMD (single instruction on multiple data), elles opèrent sur les registres SSE. Ces registres peuvent contenir plusieurs scalaires (par exemple 4 float ou 2 doubles), une instruction SSE permet donc d'exécuter plusieurs opérations arithmétique en une seule instruction. Quelques instructions SSE :
`https://docs.oracle.com/cd/E26502_01/html/E28388/eojde.html#epmoe`

Convention :

Lors d'un appel de fonction, le premier paramètre est placé dans rdi, le second est placé dans rsi, les suivants dans rdx, rcx, r8, r9 et les suivants sur la pile ((%rsp), 8(%rsp), etc...). La valeur de retour d'une fonction est placée dans %rax.

▸**Exercice 1.** *Modifier du langage assembleur On considère le code suivant :*

```
#include <stdio.h>
#include <stdlib.h>
#define N 16
int calculscal(int n,int m) {
        int z;
        z=n*n+20;
        return z;
}
void calculararray(float *a, float *b) {
  int i;
  for (i=0; i<N; i++) {
    a[i]+=b[i];
  }
}
int main() {
  int i;
  int a,b;
  float *c,*d;
  a=13;
  b=27;
  c= (float *)malloc(sizeof(float)*N);
```

```
  d= (float *)malloc(sizeof(float)*N);
  for (i=0; i<N; i++) { d[i]=i; c[i]=N-i; }
  printf("%d\n", calculscal(a,b));
  calcularray(c,d);
  for (i=0; i<N; i++)
    printf("%f ",c[i]);
  return 0;
}
```

1. *Compiler le code avec le flag* `-S` *pour générer le code assembleur. Modifier le code assembleur pour que le calcul de* `z` *soit* `18*n-m`
2. *Modifier le code assembleur pour que le calcul de la fonction* `calcularray` *soit fait en vectoriel, SSE, en utilisant 4 floats par vecteur. Pour cela, on utilisera les instructions* `addps` *au lieu de* `addss`, `movups` *au lieu de* `movss`, *...*

▸**Exercice 2.** *Génération de code assembleur : les expressions On propose de faire un générateur de code pour des expressions, sans variable. Le code généré sera placé dans une fonction "calcule" :*

$$\begin{array}{lcl} S & \rightarrow & E \\ E & \rightarrow & T+E \\ & & |T-E \\ & & |T \\ T & \rightarrow & F*T \\ & & |F \\ F & \rightarrow & n \\ & & |(E) \end{array}$$

Les fichiers lex et yacc sont disponibles en ligne.

1. *Ecrire des actions sémantiques pour générer un code assembleur calculant les expressions. Les entiers seront sur 32 bits, et les calculs pourront se faire sur la pile. Pour ce faire, on utilisera* `pushl` *et* `popl`.
2. *Modifier le lexeur et l'analyseur syntaxique pour que les calculs soient sur des flottants. On fera le calcul à l'aide de SSE (scalaire).*