Analyse lexicale

Jump to bottom

VBrazhnik a modifié cette page le 3 janvier 2019 · 1 révision

Avant de commencer à traduire du code écrit en langage d'assemblage en bytecode, le programme de traduction effectue ce que l'on appelle **l'analyse lexicale du** code source.

> En informatique, l'analyse lexicale («tokenizing») est le processus d'analyse analytique de la séquence d'entrée de caractères en groupes reconnus - lexèmes, afin d'obtenir à la sortie des séquences identifiées appelées «jetons» (comme le regroupement de lettres dans des mots).

Source: Analyse lexicale - Wikipédia

Ce processus consiste à analyser le code du champion en composants séparés, dont chacun sera affecté à l'un des types.

Par exemple, après l'analyse lexicale, la ligne suivante se transforme en ...:

```
loop: sti r1, %:live, %1
```

... liste des jetons:

pièce	Contenu	Un type
une	`loop:`	`LABEL`
2	`sti`	`INSTRUCTION`
3	`r1`	`REGISTER`
4	`,`	`SEPARATOR`
5	`%:live`	`DIRECT_LABEL`
6	`,`	`SEPARATOR`
sept	`%1`	`DIRECT`

Vous pouvez étudier en détail comment le programme de traduction fourni identifie chaque type de jeton à l'aide des messages d'erreur qu'il affiche.

Caractéristiques du travail

C'est grâce à ces messages et à la "méthode de la force brute" que vous pouvez apprendre ce qui suit:

Ordre de commande

Les commandes de définition du nom et du commentaire peuvent être permutées:

```
.comment    "This city needs me"
.name       "Batman"
```

Ligne

Le jeton de type `STRING` comprend tout, du devis d'ouverture au devis de clôture. Et ni les sauts de ligne ni les symboles de commentaire ne peuvent l'arrêter en cours de route.

Par conséquent, l'exemple ci-dessous est absolument correct:

```
.name "one
#two
three"
```

Espaces et onglets facultatifs

Dans les cas où le traducteur peut séparer sans ambiguïté des composants les uns des autres, les caractères d'espacement entre eux peuvent être omis:

```
live%42
```

```
loop:sti r1, %:live, %1
```

```
.name"Batman"
.comment"This city needs me"
```

Mais il y a des cas où vous ne pouvez pas vous passer d'au moins un espace ou une tabulation.

± 0

Les arguments d'opération peuvent contenir des nombres moins, mais l'utilisation du signe plus provoquera une erreur. Il ne peut pas être inclus dans le numéro.

Mais des zéros non significatifs peuvent être présents:

```
live %0000042
```

Registres

Pour le programme fourni, le `asm` registre est une chaîne composée d'un caractère `r` et d'un ou deux chiffres.

Par conséquent, tous les exemples suivants seront traduits avec succès en bytecode:

```
r2
```

```
r01
```

```
r99
```

Une attention particulière doit être portée au dernier d'entre eux.

Le fait que sa traduction ne génère aucune erreur indique qu'il `asm` ne sait rien de la configuration de la machine virtuelle et de la signification de la constante `REG_NUMBER`.

Ce comportement du programme est absolument logique. Après tout, le traducteur et la machine virtuelle sont des entités distinctes qui ne connaissent pas la configuration (ni même l'existence) de l'autre.

Et si la valeur de la constante change `REG_NUMBER`, le même code d'octet peut devenir invalide pour la machine virtuelle de invalide, et vice versa.

> Si la valeur est `REG_NUMBER` égale, l' `16` opération avec l'argument register `r42` sera invalide. Et si la valeur est définie, `49` elle sera exécutée sans problème.

Deux cas particuliers

Mais deux cas particuliers du fonctionnement du programme `asm` soulèvent des questions - `r0` et `r00`.

Le traducteur d'origine gérera également ces deux exemples. Bien que cela soit contraire à la logique de la mission, qui dit:

> Registre: (`r1` <–> `rx` avec `x` = `REG_NUMBER`)

Et si l'ignorance par le programme de la valeur de la borne supérieure (`REG_NUMBER`) est logique et facile à expliquer. En effet, lors du changement de configuration d'une machine virtuelle, la même section de bytecode peut devenir invalide d'être valide. Les résultats de la traduction `r0`, et `r00` ne seront pas corrects à tout moment.

Par conséquent, lors de la mise en œuvre du vôtre, `asm` ce problème mérite d'être résolu.

Taille du code exécutable

La taille du code exécutable n'est en aucun cas limitée par le programme de traduction. Autrement dit, cela `asm` fonctionnera aussi bien avec le code source composé d'une instruction, qu'avec le code contenant des centaines de milliers de ces instructions.

Avec une machine virtuelle, les choses sont un peu différentes. Il a une limite sur la quantité maximale de code exécutable en octets en utilisant une constante `CHAMP_MAX_SIZE`.

Dans le fichier de sortie, `op.h` cette constante est initialisée avec les lignes suivantes:

```
# define MEM_SIZE           (4 * 1024)
// ...
# define CHAMP_MAX_SIZE     (MEM_SIZE / 6)
```

Autrement dit, dans ce cas, la taille du code exécutable champion ne doit pas dépasser 682 octets.

Avec une bordure minimale, les choses sont un peu plus intéressantes.

La machine virtuelle acceptera facilement un `.cor` fichier dans lequel la taille du code exécutable sera égale à zéro.

Cependant, il n'est pas si facile de produire un tel fichier à l'aide du traducteur fourni.

Si, en plus de définir le nom et le commentaire du champion, rien d'autre n'est écrit dans le `.s`-file, alors il ne sera pas possible de le convertir en bytecode.

Mais si vous ajoutez une seule étiquette à la fin, vous pouvez vous retrouver avec le fichier souhaité sans code exécutable:

```
.comment    "This city needs me"
.name       "Batman"

loop:
# Конец файла
```

Peut-être que les créateurs de l'original ont `asm` voulu ainsi interdire la diffusion du champion sans code exécutable. Pour qu'au moins une opération soit toujours présente dans le fichier.

Mais dans ce cas, ils n'ont pas pris en compte certains points et il est toujours possible de créer un tel fichier.

Comment vivre avec?

Dans votre travail, vous pouvez, ainsi qu'améliorer cette protection pour garantir la présence d'au moins une opération.

Alors supprimez-le complètement, ce qui ressemble à une solution plus logique. Après tout, la machine virtuelle d'origine fonctionne de manière transparente avec des `.cor` fichiers sans code exécutable.

Cloner ce wiki localement