Core 4.0 : Assignments

We deal with assignments, i.e. expressions of the type: a = 2 or a = .b + 2 but also a[.b] = 10 for example and also a = b = c + 1. To do this it is necessary to introduce a further variable in the translation function, which we will call :var

```
public function translate ( :EX , :e , :i , :var ) {{
```

We must immediately take care of updating all function calls by passing an additional variable defined as an empty string.

Once this is done we will apply the following logic. If we deal with a variable and the operation is "mov" we will only store the variable in :var

```
if ( .opr == "mov" ) {{ :var = .so get 1 ; continue }} ;
```

We define the operator "=" and implement it:

```
binary operator "=" to register as "=" with priority 0
```

```
if ( .so == "o|=" ) {{
        if ( :var == "" ) {{ rule error "error on assignment (ex. .a + 1 = ..." }} else {{
                set .var = "" ; :i = :i + 1 ;
                .code = .code + exec translate ( :EX , :e , :i , .var ) + "\nmov [ " + :var + " ] , eax"  ;
                :var = "" ; :i = :i - 1 ; continue
        }}
}} ;
```

Now in the case of an operator, whose previous operator was "mov", we will insert the mov using :var

```
if ( .opr == "mov" ) {{ .code = .code + "\nmov eax , [ " + :var + " ]" }} ; :var = "" ;
```

A similar line should also be used before returns (.code). For example :

```
if ( .so == "r|)" ) {{ if ( .opr == "mov" && :var != "" ) {{ .code = .code + "\nmov eax , [ " + :var + " ]" }} ; return ( .code ) }}
```

In the vector index resolution operation, we make the following change:

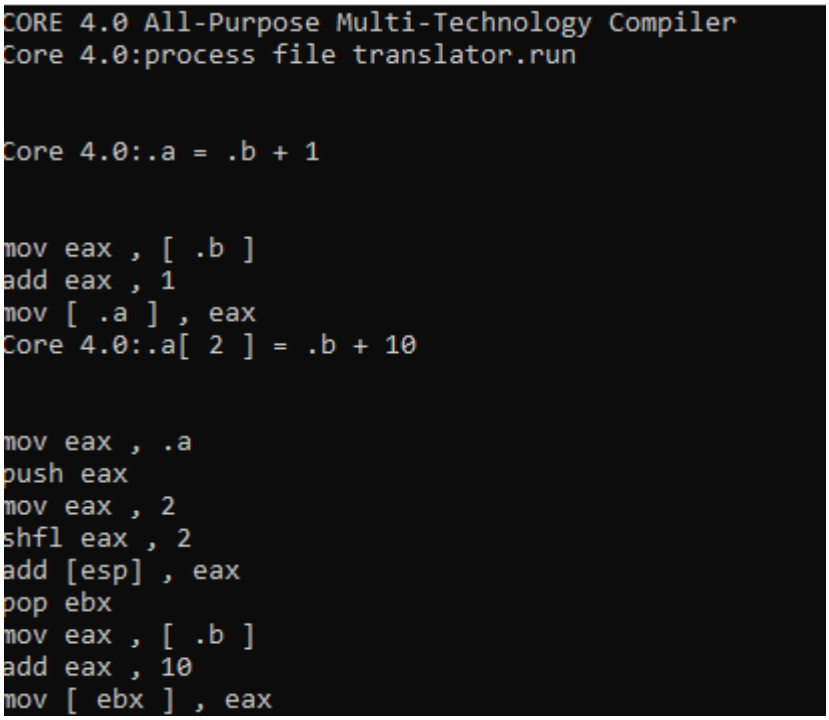
```
.code = .code + "\nshfl eax , 2\nadd [esp] , eax\npop ebx" : :var = "ebx" ;
```

The :var variable must be set to null in all other cases, when we have a number, or a function call. What remains however is what will be with regards

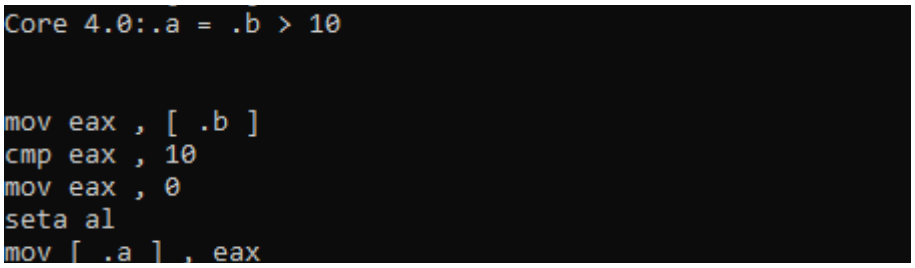
to operators such as parentheses:

```
if ( .ty == "n" ) {{ :var = "" ; .code = .code + "\n" + .opr + " eax , " + ( .so get 1 ) + .cmp ; continue }} ;
if ( .ty == "e" ) {{ :var = "" ; set .ex_stm = resolve .so get 1 ; .code = .code + exec .ex_stm.translate ( :EX ) ; continue }} ;
```

And we're done. Let's launch the application:



In the next exercise the if construct. I forgot it also works with conditional expressions for example:



Good evening