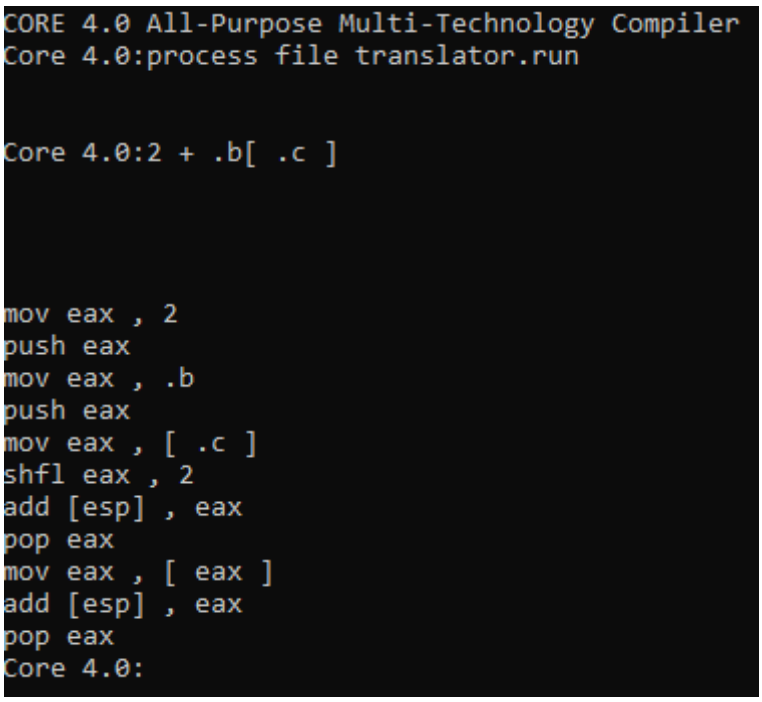
Core 4.0: Function Call Example

We have seen the assembly translation of indexed expressions. Let's start from there, with an image:



We have seen that expressions with index can be nested according to a recursive logic as desired. Now let's deal with function calls. A function call is a statement that must respect the following syntactic definition of the parser:

```
exec&: exec|.FUNCTION|(frw) (|.%PARAM|(**) ,|(**)|(opt) )
```

Since it is a statement we can create a statement function for its translation. Function that will translate the expressions for each parameter by pushing eax, after the translation of each individual expression. At the end there is the actual call to the function with the call instruction, the parameters are on the stack. Stack which must be cleared of parameters when the function returns. All this translates into the following function:

```
public function exec.translate ( :EX ) {{

        set .code = "" ;

        for ( set .p = 0 ; .p < size ::PARAM ; .p = .p + 1 ) {{

                set .i = 0 ;
                .code = .code + exec translate ( :EX , ::PARAM[ .p ].expression , .i ) + "\npush eax"

        }} ;

        .code = .code + "\ncall " + ( text ::FUNCTION ) + "\nadd esp , " + ( text 4 * size ::PARAM ) ;
        return ( .code )

}} ;
```

Which is launched by translate as follows:

```
if ( .ty == "e" ) {{ set .ex stm = resolve .so get 1 ; .code = .code + exec .ex stm.translate ( :EX ) ; continue }} ;
```

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

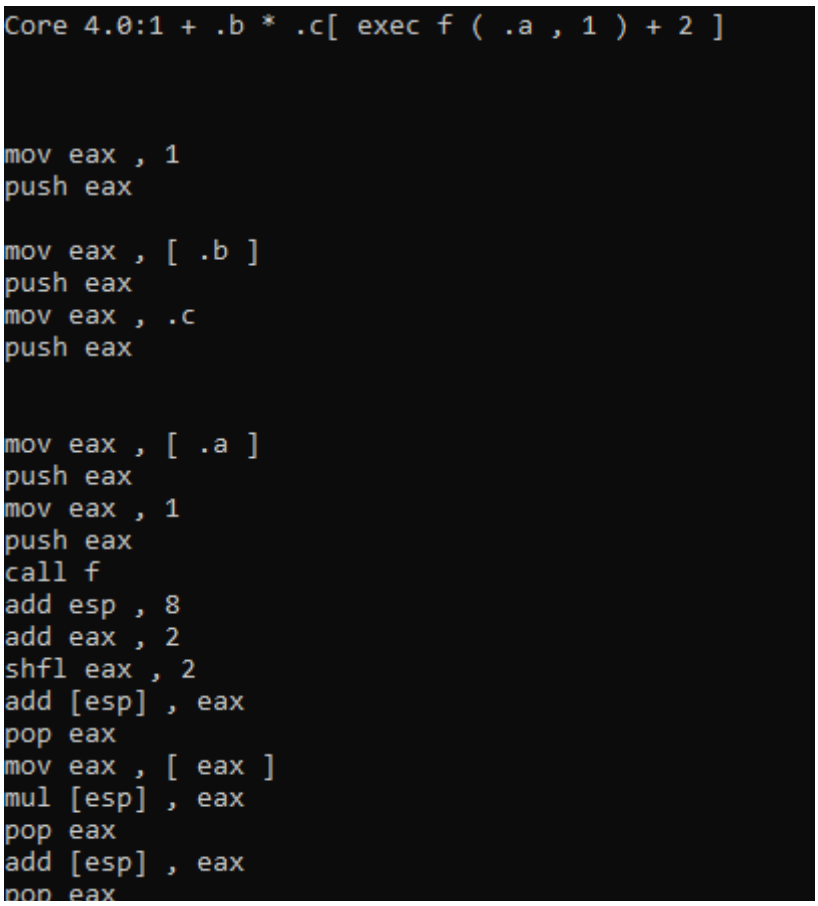
```
CORE 4.0 All-Purpose Multi-Technology Compiler
Core 4.0:process file translator.run


Core 4.0:exec f ( .a + 1 , 2 )




mov eax , [ .a ]
add eax , 1
push eax
mov eax , 2
push eax
call f
add esp , 8
Core 4.0:
```

Believe me we can complicate it any way we want, but the document page is short, so let's just give an example of medium complexity:



Now we can take various paths to continue our examples and exercises. What matters is this image, it is the compiler that is very simple to use and also for this reason, "all-purpose":


Core 4.0
Multi-technology Compiler All-Purpo