home ⇐ course03 course05 ⇒

Implementing a FORTH virtual machine - 4

course04.c (304 lines) - conditionals

Now we implement conditions like if and while and unconditional loops

Application

Well, now as we have conditionals in our programs it looks a bit alien to braces used developers. As one gets more comfortable with this language, the syntax become natural.

```
$ ./course04
: is-true? if "yes, it is true" else "no, it is not" then type cr ;
ok> 0 is-true?
no, it is not
ok> 1 is-true?
yes, it is true
ok>
```

Display the square of the numbers 10 to 1. Counter always is on top of Stack. Don't put in the comments, since the compiler doesn't know how to handle them.

(begins a comment in Forth
) ends a comment in Forth
(a b--c) a stack comment which means a on next of stack (NOS), b as top of stack (TOS) and c is the result
while is a conditional jump (jump on true on top of stack) which discards the top of stack. To keep the top of stack (which is our counter) we have to duplicate it.

```
$ ./course04
: square ( n--n*n) dup * ;
ok> 4 square
16 ok> drop
ok> : .square ( n--n) dup . dup square . cr ;
ok> 4 .square
4 16
4 ok> drop
ok> : squares ( n--) begin .square -1 + dup while drop ;
ok> 10 squares
10 100
9 81
8 64
7 49
6 36
5 25
4 16
3 9
2 4
1 1
ok>
```

again is an uncoditional jump to begin

```
$ ./course04
: endless-loop ( --) begin again ;
ok> endless-loop
... never comes back
```

Implementation

The compiling words (if, else, then, begin, again, while)) are all macros which pushes some code address on stack to be resolved later as a jump or conditional jump address.

f_0branch, f_1_branch are conditional jumps on zero or not zero which disacrds the top of stack. f_branch is an uncoditional jump.

```
static void register_primitives(void) {
   ...
        xt_0branch=add_word("0branch", f_0branch); // jump if zero
        xt_1branch=add_word("1branch", f_1branch); // jump if not zero
        xt_branch =add_word("branch",  f_branch); // unconditional jump

        definitions=&macros;
        add_word("if",    f_if);    // compiles an if condition
        add_word("then",  f_then);  // this is the endif
        add_word("else",  f_else);
        add_word("begin", f_begin); // begin of while loop
        add_word("while", f_while); // while loop (condition at end of loop)
        add_word("again", f_again); // unconditional loop to begin
        ...
}

static void f_if(void) { // macro, execute at compiletime
        *code++=xt_0branch;
        sp_push((cell_t)code++); // push forward reference on stack
}
static void f_else(void) { // macro, execute at compiletime
        xt_t ***dest=(void*)sp_pop(); // pop address (from f_if)
        *code++=xt_branch; // compile a jump
        sp_push((cell_t)code++); push forward reference on stack
        *dest=code; resolve forward reference given by f_if
}
static void f_then(void) { // macro, execute at compiletime
        xt_t ***dest=(void*)sp_pop();
        *dest=code; resolve forward reference given by f_if or f_else
}
static void f_begin(void) { // macro, execute at compiletime
        sp_push((cell_t)code); // push current compilation address for loop
```

```
}
static void f_while(void) { // macro, execute at compiletime
        *code++=xt_1branch; // compile a jump if not zero
        *code++=(void*)sp_pop(); // jump back to f_begin address
}
static void f_again(void) { // macro, execute at compiletime, unconditional loop
        *code++=xt_branch; // compile a jump
        *code++=(void*)sp_pop();// jump back to f_begin address
}
```

Now go to the last part of out course, how to implement an disassembler course05 ⇒