NOP (Not Only a Parser)

Our goal is to implement a compiler that is capable of translating the following ANSI C language constructs to machine code that is runnable on the DCPU-16 architecture described in the appendix.

- Assignments of variables
- Arithmetic operations (+, -, *, /, %)
- Logical operations (AND, OR, NOT, XOR)
- Function calls
- For and while loops
- If / else statements

Optional implementations include the following

- Elseif statements
- Bitwise logical operations

However, due to limitations of the DCPU-16 architecture, handling of floating point numbers will not be implemented.

The DCPU-16 hereby is a CPU that will be featured inside the newest game *0x10c* of Minecraft creator Notch. The computer in the game is a fully functioning emulated 16 bit CPU that can be used to control your players ship, or just to play games on while waiting for a large mining operation to finish.

References

Website of 0x10c: http://0x10c.com
DCPU16 assembler and simulator: http://fingswotidun.com/dcpu16

DCPU-16 Specification

Version 1.1 (Check 0x10c.com for updated versions)

- 16 bit unsigned words
- 0x10000 words of ram
- 8 registers (A, B, C, X, Y, Z, I, J)
- program counter (PC)
- stack pointer (SP)
- overflow (O)

In this document, anything within [brackets] is shorthand for "the value of the RAM at the location of the value inside the brackets.
For example, SP means stack pointer, but [SP] means the value of the RAM at the location the stack pointer is pointing at.

Whenever the CPU needs to read a word, it reads [PC], then increases PC by one. Shorthand for this is [PC++].
In some cases, the CPU will modify a value before reading it, in this case the shorthand is [++PC].

Instructions are 1-3 words long and are fully defined by the first word.
In a basic instruction, the lower four bits of the first word of the instruction are the opcode, and the remaining twelve bits are split into two six bit values, called a and b.
a is always handled by the processor before b, and is the lower six bits.
In bits (with the least significant being last), a basic instruction has the format: bbbbbbaaaaaaoooo

Values: (6 bits)

0x00-0x07:	register (A, B, C, X, Y, Z, I or J, in that order)
0x08-0x0f:	[register]
0x10-0x17:	[next word + register]
0x18:	POP / [SP++]
0x19:	PEEK / [SP]
0x1a:	PUSH / [--SP]
0x1b:	SP
0x1c:	PC
0x1d:	O
0x1e:	[next word]
0x1f:	next word (literal)
0x20-0x3f:	literal value 0x00-0x1f (literal)

- "next word" really means "[PC++]". These increase the word length of the instruction by 1.
- If any instruction tries to assign a literal value, the assignment fails silently. Other than that, the instruction behaves as normal.

- All values that read a word (0x10-0x17, 0x1e, and 0x1f) take 1 cycle to look up. The rest take 0 cycles.
- By using 0x18, 0x19, 0x1a as POP, PEEK and PUSH, there's a reverse stack starting at memory location 0xffff. Example: "SET PUSH, 10", "SET X, POP"

Basic opcodes: (4 bits)

0x0: non-basic instruction - see below	
0x1: SET a, b	- sets a to b
0x2: ADD a, b	- sets a to a+b, sets O to 0x0001 if there's an overflow, 0x0 otherwise
0x3: SUB a, b	- sets a to a-b, sets O to 0xffff if there's an underflow, 0x0 otherwise
0x4: MUL a, b	- sets a to a*b, sets O to ((a*b)>>16)&0xffff
0x5: DIV a, b	- sets a to a/b, sets O to ((a<<16)/b)&0xffff. if b==0, sets a and O to 0 instead.
0x6: MOD a, b	- sets a to a%b. if b==0, sets a to 0 instead.
0x7: SHL a, b	- sets a to a<<b, sets O to ((a<<b)>>16)&0xffff
0x8: SHR a, b	- sets a to a>>b, sets O to ((a<<16)>>b)&0xffff
0x9: AND a, b	- sets a to a&b
0xa: BOR a, b	- sets a to a|b
0xb: XOR a, b	- sets a to a^b
0xc: IFE a, b	- performs next instruction only if a==b
0xd: IFN a, b	- performs next instruction only if a!=b
0xe: IFG a, b	- performs next instruction only if a>b
0xf: IFB a, b	- performs next instruction only if (a&b)!=0

Non-basic opcodes always have their lower four bits unset, have one value and a six bit opcode.In binary, they have the format: aaaaaaoooooo0000
The value (a) is in the same six bit format as defined earlier.

Non-basic opcodes: (6 bits)

0x00:	reserved for future expansion
0x01:	JSR a - pushes the address of the next instruction to the stack, then sets PC to a
0x02-0x3f:	reserved