Distributed Stack Machine

Implement a simulator of a distributed stack machine in x86_64 assembly language. The machine consists of N cores, numbered from 0 to N - 1, where N is a constant set during the compilation of the simulator. The simulator will be used with the C language in the following way: N threads will be launched, and in each thread, the function will be called:

```
uint64_t core(uint64_t n, char const *p);
```

The parameter `n` contains the core number. The parameter `p` is a pointer to a null-terminated ASCII string defining the computation the core should perform. The computation consists of operations performed on a stack, which is initially empty. We interpret characters of the string as follows:

- `+` – pop two values from the stack, calculate their sum, and push the result onto the stack;
- `*` – pop two values from the stack, calculate their product, and push the result onto the stack;
- `-` – negate the value at the top of the stack;
- `0` to `9` – push the respective value onto the stack;
- `n` – push the core number onto the stack;
- `B` – pop a value from the stack, if the value at the top of the stack is nonzero, treat the popped value as a two's complement number and shift by that many operations;
- `C` – pop a value from the stack and discard it;
- `D` – duplicate the value at the top of the stack;
- `E` – swap the top two values on the stack;
- `G` – push onto the stack the value obtained from calling (implemented elsewhere in C) the function `uint64_t get_value(uint64_t n);`
- `P` – pop a value from the stack (let's denote it as w) and call (implemented elsewhere in C) the function `void put_value(uint64_t n, uint64_t w);`
- `S` – synchronize cores, pop a value from the stack, treat it as the core number m, wait for operation S of core m with the core number n popped from the stack, and swap the values at the top of the stacks of cores m and n.

After a core finishes the computation, its result, i.e., the result of the function core, is the value at the top of the stack. All operations are performed on 64-bit numbers modulo 2 to the power of 64.

It is allowed to assume that the given core number is valid. It is also allowed to assume that the computation is correct, i.e., it contains only the described characters, ends with a zero byte, does not attempt to access a value from an empty stack, and does not lead to deadlock. The behavior of the core for an incorrect computation is undefined.

As the stack used by the core for the described computations, the hardware stack of the processor should be used. It is not allowed to use any libraries. Core synchronization, i.e., operation S, should be implemented using some variant of a spin lock.

The result of the computation "1nS" is undefined and depends on the implementation, although treating the sequence of operations n and S as an operation C seems reasonable.

No upper limit should be assumed on the value of N other than what arises from the processor architecture and available memory.

In the specification of the operations performed by the core, the terms "pop from the stack" and "push onto the stack" should be understood as the definition of the operation to be performed, not as a requirement to use the pop or push instruction.

Compilation

The solution will be compiled with the command:

```
nasm -DN=XXX -f elf64 -w+all -w+error -o core.o core.asm
```

where XXX specifies the value of the constant N.

Usage example

An example of usage is provided in the attached file `example.c`. It can be compiled using the following commands:

```
nasm -DN=2 -f elf64 -w+all -w+error -o core.o core.asm
gcc -c -Wall -Wextra -std=c17 -O2 -o example.o example.c
gcc -z noexecstack -o example core.o example.o -lpthread
```