Tiny Virtual Machine

Where you will build a tiny stack based virtual machine.
The machine has:

1. A stack of int32_t elements *(Hint: std::stack<int32_t>)*
2. Can execute opcodes that are encoded into int32_t words.
   (Hint: std::vector<int32_t>)
   The capacity of the stack is not less than 4096 words.
3. Current instruction pointer (the index of the next opcode.

Phase 1

OpCodes

The opcodes for this machines are (each encoded as 32 bits). You chose what value each opcode has. *(Hint: enum class)*

OpCode	Meaning
ADD	Add top 2 elements from the stack and push the result back push(pop() + pop())
SUB	Sub top 2 elements from the stack and push the result back push(pop() - pop())
MUL	Multiply top 2 elements from the stack and push the result back push(pop() * pop())
DIV	Divide top 2 integers from the stack and push the result back push(pop() / pop())
POP	Pop and throw away top of stack
PUSH *data*	Push next word from the code to the stack (2 WORDS) PUSH 42 will push(42)
DUP	Duplicate the top of stack: push(top())
SWAP	Swap the top 2 elements: a = pop(), b = pop(), push(b), push(a)
PRINT	Pop and print the number from top of stack, cout << pop()
PRINTC	Pop and print a char whose ascii code is at the top of stack

NOP	Do nothing
HALT	Stop the program
INC	**PUSH(POP()+1)**
DEC	**PUSH(POP()-1)**

Hint:
Execution loop:
loop:
opCode = Get next instruction();
If opCode == HALT then exit
execute(opCode)

Execute should use one of these methods:

1. Switch on the enum

Example code:

Code	Stack	Output
PUSH 12	12	
PUSH 8	12, 8	
DUP	12, 8, 8	
INC	12, 8, 9	
INC	12, 8, 10	
MUL	12, 80	
SUB	68	
DUP	68, 68	
PRINT	68	68
PRINTC		D
HALT		

Phase 2

Modify the execution function to to use an std::array of

a. Pointer to function
b. Std::function with lambda

Execution loop:

```
loop:
    opCode = Get next instruction();
    If opCode == HALT then exit
    arrat[opCode](...)      // just execute the function pointer at index opCode
```

And add more instructions for jump (2 WORDS one for opcode and one for address)

JMP *address*	Continue execution at given address
JZ *address*	Pop top of stack and if is zero continue execution at *address* otherwise continue to next opcode
JNZ *address*	Pop top of stack and if is not zero continue execution at *address* otherwise continue to next opcode

Demo Code

Address	**Code**	**Stack**
0	PUSH 64	64
2	INC	65
3	DUP	65, 65
4	PRINTC	65
5	DUP	65, 65
6	PUSH 70	65, 65, 70
8	SUB	65, 5
9	JNZ 2	65 and continue at address 2
11	HALT	

Will print: ABCDEF

Phase 3

For this phase you will implement a mechanism to call a procedure and return from it. In order to do that we will add a special call stack to hold the return address.

CALL *address*	Call procedure at *address* : Push to call stack the address of next opcode, continue execution at *address*
RET	Pop return address from call stack and continue execution at that address

Demo Code

Address	**Code**	**Stack**		**Call Stack**	
0	PUSH 64	64			
2	CALL ***9***	64		**4**	
4	NOP	65			
5	CALL ***9***	65		**7**	
7	NOP	66			
7	HALT	66			
8	NOP				
		Call from 2	**Call from 4**	**Call From 2**	**Call From 4**
9	INC	65	66	4	7
10	DUP	65, 65	66, 66	4	7
11	PRINTC	65 print ‘A’	66 print ‘B’	4	7
12	RET	65	66		