```
1  #pragma once
2  #define ENDOP1 0xDE
3  #define ENDOP2 0xAD
4  #define ENDOP3 0xBE
5  #define ENDOP4 0xEF
6  //terms: iv: immediate value
7  typedef enum {
8      NOP,            //no operation
9      MOV,            //value from r0 goes into r1    ex: MOV r0 r1
10     LOADI,          //loads an iv into r#           ex: LOADI r0 1337
11     LOADS,          //loads a str into sr#          ex: LOADS r0 HELLO
12
13     INC, DEC, ADD, SUB, MUL, DIV,
14     //increment r#, decrement r#,
15     //adds/subtracts/multiplies/divides r# r# into REG_RESULT
16     //ex: ADD r0 r1      (REG_RESULT is now r0+r1)
17
18     SWP,            //swaps two registers values
19
20     POP,            //pops the stack into r#        ex: POP r#
21     PUSH,           //push a register value;        ex: PUSH r#
22     PUSHI,          //push an immediate to r#       ex: PUSH 0
23
24     SYSCALL,
25     CALL,           //calls a label with name       ex: CALL lblname
26     RET,            //returns to the calling funk   ex: RET
27
28     INLINE_STR_START, INLINE_STR_END, INLINE_ID_REF,
29
30     END
31 } OPCODES;
32
33 typedef enum {
34     WRITEVAL, WRITEL
35 } SYSCALLS;
36
37 typedef enum {
38     REG_RETVAL = 50, REG_ACCUMULATE, REG_RESULT
39 } SPECIAL_REGISTERS;
40
41 typedef enum {
42     LBL_START=END, LBL_END
43 } KEYWORDS;
44
45 #include <stdint.h>
46
47 #define BASE_TYPE uint32_t
48 #define ID_REGI_START 0
49 #define ID_STRREGI_START 505
50
51 #include "packer.h"
52 #include <vector>
```

```
53  #include "trace.h"
54  #include "VMException.h"
55  #include <string>
56  #define nxti (++i < n ? str[i] : 0)
57  using namespace std;
58  class subroutine_assember;
59  class assemblr {
60  public:
61      std::vector<BASE_TYPE> m_prog;
62
63      assemblr* s(string str);
64      assemblr* call(char* _name);
65      assemblr* syscall(SYSCALLS sc);
66      assemblr* opcode(OPCODES op);
67      assemblr* kw(KEYWORDS kw);
68      assemblr* immediate(BASE_TYPE val);
69      assemblr* regi(BASE_TYPE num);
70      assemblr* strRegi(BASE_TYPE num);
71      assemblr* end();
72      virtual assemblr* finish() {
73          return this;
74      }
75      subroutine_assember* subroutine(char* lbl);
76      void _subroutine_finished(subroutine_assember *sa);
77      BASE_TYPE* assemble(int &n) {
78          n = m_prog.size();
79          BASE_TYPE *cc = new BASE_TYPE[n];
80          for (int i = 0; i < n; i++)
81              cc[i] = m_prog.at(i);
82          return cc;
83      }
84  };
85  class subroutine_assember : public assemblr {
86  public:
87      assemblr *m_ctx;
88      char *m_lbl;
89      subroutine_assember(assemblr *ctx, char* lbl) :m_ctx(ctx), m_lbl(lbl) {
90          kw(LBL_START)->immediate(_hash_sdbm((unsigned char*)lbl));
91      }
92      assemblr* finish() {
93          kw(LBL_END);
94          m_ctx->_subroutine_finished(this);
95          return m_ctx;
96      }
97  };
```