```c
#include "VFM_Macros.h"
/* Macro Assembler */
//
#define NAME_LENGTH_MAX 31
extern MemoryImage *M;
//
int32_t IMEDD = 0x80;
int32_t COMPO = 0x40;
int32_t BRAN = 0, QBRAN = 0, DONXT = 0, DOTQP = 0, STRQP = 0, TOR = 0, ABORQP = 0;
//
#define ALIGN_IP M->IP = M->P >> 2;
#define DEPOSIT_WORD_INC M->data[M->IP++]
#define ALIGN_P M->P = M->IP << 2;
#define DEPOSIT_BYTE_INC M->cdata[M->P++]
//
void HEADER(int32_t lex, const char seq[]) {
  int32_t len = lex & 0x1F;        // seperate name length
  int32_t nfa = M->thread;         // thread == name field of prior word
  //
  ALIGN_IP                         // align Integer Pointer from Character Pointer
  DEPOSIT_WORD_INC = nfa;          // place the nfa
  ALIGN_P                          // update Character Pointer to match IP
  M->thread = M->P;                // thread == name field of current word
  DEPOSIT_BYTE_INC = (int8_t) lex; // copy compile & immediate bits with length of name into dicitonary
  //
  for (int i = 0; i < len; i++){ DEPOSIT_BYTE_INC = seq[i]; }  // copy name chars into dictionary
  while (M->P & 3){ DEPOSIT_BYTE_INC = (char) 0; }             // pad name with zeros
}
//
int32_t CODE(int32_t len, int8_t c0, int8_t c1, int8_t c2, int8_t c3, int8_t c4, int8_t c5, int8_t c6,
int8_t c7){
  int32_t addr = M->P;
  switch (len){ // variable number of bytes are sent to the dictionary
    case 8:
      M->cdata[(M->P)+4] = c4;
      M->cdata[(M->P)+5] = c5;
      M->cdata[(M->P)+6] = c6;
      M->cdata[(M->P)+7] = c7;
    case 4:
      M->cdata[(M->P)+0] = c0;
      M->cdata[(M->P)+1] = c1;
      M->cdata[(M->P)+2] = c2;
      M->cdata[(M->P)+3] = c3;
  }
  switch (len){ // update the character P Pointer appropriately
    case 8:
      M->P += 8;
      break;
    case 4:
      M->P += 4;
      break;
  }
  return(addr); // return nfa of this CODE
}
//
int32_t COLON(int32_t len, ...) {
    int32_t addr = M->P;
    M->IP = M->P >> 2;
    M->data[(M->IP)++] = 6; /* dolist */
    va_list argList;
    va_start(argList, len);
    /*print32_tf(" %X ",6);*/
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
    return addr;
}
```

```
int32_t LABEL(int32_t len, ...) {
    int32_t addr = M->P;
    M->IP = M->P >> 2;
    va_list argList;
    va_start(argList, len);
    /*print32_tf("\n%X ",addr);*/
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
    return addr;
}
void BEGIN(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X BEGIN ",M->P);*/
    pushR = M->IP;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void AGAIN(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X AGAIN ",M->P);*/
    M->data[(M->IP)++] = BRAN;
    M->data[(M->IP)++] = popR << 2;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void UNTIL(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X UNTIL ",M->P);*/
    M->data[(M->IP)++] = QBRAN;
    M->data[(M->IP)++] = popR << 2;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void WHILE(int32_t len, ...) {
    M->IP = M->P >> 2;
    int32_t k;
    /*print32_tf("\n%X WHILE ",M->P);*/
    M->data[(M->IP)++] = QBRAN;
    M->data[(M->IP)++] = 0;
    k = popR;
    pushR = (M->IP - 1);
    pushR = k;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void REPEAT(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X REPEAT ",M->P);*/
    M->data[(M->IP)++] = BRAN;
    M->data[(M->IP)++] = popR << 2;
    M->data[popR] = M->IP << 2;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void IF(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X IF ",M->P);*/
    M->data[(M->IP)++] = QBRAN;
    pushR = M->IP;
    M->data[(M->IP)++] = 0;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void ELSE(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X ELSE ",M->P);*/
    M->data[(M->IP)++] = BRAN;
    M->data[(M->IP)++] = 0;
    M->data[popR] = M->IP << 2;
    pushR = M->IP - 1;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void THEN(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X THEN ",M->P);*/
    M->data[popR] = M->IP << 2;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void FOR(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X FOR ",M->P);*/
    M->data[(M->IP)++] = TOR;
    pushR = M->IP;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void NEXT(int32_t len, ...) {
    M->IP = M->P >> 2;
    /*print32_tf("\n%X NEXT ",M->P);*/
    M->data[(M->IP)++] = DONXT;
    M->data[(M->IP)++] = popR << 2;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void AFT(int32_t len, ...) {
    M->IP = M->P >> 2;
    int32_t k;
    /*print32_tf("\n%X AFT ",M->P);*/
    M->data[(M->IP)++] = BRAN;
    M->data[(M->IP)++] = 0;
    k = popR;
    pushR = M->IP;
    pushR = M->IP - 1;
    va_list argList;
    va_start(argList, len);
    for (; len; len--) {
        int32_t j = va_arg(argList, int32_t);
        M->data[(M->IP)++] = j;
        /*print32_tf(" %X",j);*/
    }
    M->P = M->IP << 2;
    va_end(argList);
}
void DOTQ(const char seq[]) {
    M->IP = M->P >> 2;
    int32_t i;
    int32_t len = strlen(seq);
    M->data[(M->IP)++] = DOTQP;
    M->P = M->IP << 2;
    M->cdata[(M->P)++] = len;
    for (i = 0; i < len; i++)
    {
        M->cdata[(M->P)++] = seq[i];
    }
    while (M->P & 3) { M->cdata[(M->P)++] = 0; }
    /*print32_tf("\n%X ",M->P);*/
    /*print32_tf(seq);*/
}
void STRQ(const char seq[]) {
    M->IP = M->P >> 2;
    int32_t i;
    int32_t len = strlen(seq);
    M->data[(M->IP)++] = STRQP;
    M->P = M->IP << 2;
    M->cdata[(M->P)++] = len;
    for (i = 0; i < len; i++)
    {
        M->cdata[(M->P)++] = seq[i];
    }
    while (M->P & 3) { M->cdata[(M->P)++] = 0; }
    /*print32_tf("\n%X ",M->P);*/
    /*print32_tf(seq);*/
}
void ABORQ(const char seq[]) {
    M->IP = M->P >> 2;
    int32_t i;
    int32_t len = strlen(seq);
    M->data[(M->IP)++] = ABORQP;
    M->P = M->IP << 2;
    M->cdata[(M->P)++] = len;
    for (i = 0; i < len; i++)
    {
        M->cdata[(M->P)++] = seq[i];
    }
    while (M->P & 3) { M->cdata[(M->P)++] = 0; }
    /*print32_tf("\n%X ",M->P);*/
    /*print32_tf(seq);*/
}
```