Краткая схема процессорного модуля:

Легенда – таким цветом выделены модули, которые создает пользователь

Ins.cpp

```
#include <ida.hpp>
#include <idp.hpp>
#include “ins.hpp”

instruc_t Instructions[] = {
//объявляем массив инструкций:
//мнемоника и флажки
        {“имя”, CF_xxx}
        …
}
```

ida.hpp(<IDASDK>\include)
содержит структуру idainfo inf (общая для всех IDA проектов)

idp.hpp (<IDASDK>\include)
Объявление структур:
instruct_t (описывает инструкции: надо указать имя и флажки CF_xxx каждой инструкции)
asm_t (описывает один из текущих ассемблеров)
processor_t (описывает процессорный модуль; пользователь работает только с одной такой структурой через переменную **LPH**, а ядро копирует ее в структуру **ph**)
Объявление структур для IDP модулей:
- текущий ассемблер (глобальное имя **ash** тип asm_t)
- текущий процессор (глобальное имя **ph** тип processor_t)

ins.hpp
Опкоды инструкций проц. модуля

```
enum opcodes {
    имя(мнемоника)=опкод
}
extern instruc_t Instructions[];
```

Reg.cpp

```
#include “имяпроц.hpp”
#include <diskio.hpp>
#include <ieee.h>

static char *RegNames[] = {
//объявляем массив регистров
        “имярег1”,
        “имярег2”,
        …
}
//основная структура процессорного модуля
processor_t LPH = {
  IDP_INTERFACE_VERSION,    //версия ядра
  PLFM_MYID, //идентификатор Вашего модуля (должен быть более 0x8000)
  PR_xxx, //флажки возможностей процессора
  cnbits,  // число бит в байте для сегмента кода
  dnbits,  // число бит в байте для сегмента данных
  shnames, //массив структур коротких имен процессора
  lnames,    //массив структур полных имен процессора
  asms,      //массив структур ассемблеров
  notify,    //функция обработки кодов уведомления
  …
  my_ana, //функция анализатора
  my_emu, // функция эмулятора
  my_out,  // функция вывода на экран
  my_outop, // функция вывода операнда
  my_data,   // функция вывода форматированных данных
  …
  qnumber(RegNames),  //число регистров
  RegNames, // массив регистров
 …
  int instruc_start;//целочисленный код 1-й инструкции
  int instruc_end;    //целочисленный код последней инструкции + 1
  Instructions, //массив инструкций
  …
}
static asm_t имяпроц_asm = {…}

static asm_t *asms[] = { &имяпроц_asm, NULL}
static char *shnames[] = { “короткое имя”,  NULL}
static char *lnames[] = { “полное имя”,  NULL}

//Работа с кодами уведомления процессора
static int idaapi notify(processor_t::idp_notify msgid, ...) {
…
}
```

diskio.hpp (<IDASDK>\include)
Основное предназначение:
1) Описаны функции работы с файлами. В пользовательских модулях нельзя использовать стандартные С файловые функции.
2) Функция вызова команд операционной системы call_system.

ieee.h(<IDASDK>\include)
IEEE-представление чисел с плавающей точкой

имяпроц.hpp (имяпроц – произвольное краткое имя, связанное с Вашим проц. модулем; например, x86.hpp или java.hpp)

```
#define PLFM_MYID 0x31337
#include <pro.h>
#include <kernwin.hpp>
#include "../idaidp.hpp"
#include <fpro.h>
#include "ins.hpp"

enum имяпроц_registers {
     r_имярег1, r_имярег2…
}

int   idaapi my_ana(void);
int   idaapi my_emu(void);
void  idaapi my_out(void);
bool  idaapi my_outop(op_t &op);
void  idaapi my_data(ea_t ea);
…
```

Переменная **cmd** является глобальным экземпляром объекта **insn_t**. Класс **insn_t** определен в ***ua.hpp (<IDASDK>\include)***.

Ana.cpp (1-й этап: Анализатор)

Поле **itype** должно принимать одно из значений enum opcodes (*ins.hpp*)

```
#include "имяпроц.hpp"

int idaapi my_ana(void) {
  //анализ одной инструкции, заполняем поля глобальной переменной cmd:
  //cmd.itype, cmd.Operands[6]
  return cmd.size;
}
```

Операнды инструкций представлены с использованием экземпляров класса **op_t**, который определен в ***ua.hpp***

frame.hpp(<IDASDK>\include) Функции, отслеживающие поведение указателя стека и стекового фрейма функции (например, **add_auto_stkpnt2**)

Emu.cpp (2-й этап: Эмулятор)

```
#include <ida.hpp>
#include <frame.hpp>
#include "имяпроц.hpp"

//простой шаблон эмулятора
void TouchArg(op_t &op, int isRead) {
 // Реализуется автором процессорного модуля
 // Проверяет операнд и создает перекрестные ссылки
 // Отслеживаем поведение указателя стека
}
int idaapi my_emu() {
 ulong feature = cmd.get_canon_feature(); //получить флажки команды CF_xxx
 if (feature & CF_USE1) TouchArg(cmd.Op1, 1);
 if (feature & CF_USE2) TouchArg(cmd.Op2, 1);

 if (feature & CF_CHG1) TouchArg(cmd.Op1, 0);
 if (feature & CF_CHG1) TouchArg(cmd.Op2, 0);
 if ((feature & CF_STOP) == 0) {
 // добавить “ссылку на следующую команду”(ordinary flow по
 // терминологии IDA)
  ua_add_cref(0, cmd.ea + cmd.size, fl_F);
 }
 return 1; //как ни странно, большинство примеров модулей возвращает 1
}
```

Функция объявлена в ***idp.hpp***

Функция объявлена в ***ua.hpp***

Флажок объявлен в ***xref.hpp(<IDASDK>\include)***

Out.cpp (3-й этап: Генератор печати)

```
#include “имяпроц.hpp”

//Вывод на экран одной дизассемблированной инструкции
void idaapi my_out() {
 char str[MAXSTR]; // MAXSTR определен в pro.h (<IDASDK>\include)
 init_output_buffer(str, sizeof(str)); //инициализация нашего буфера
 Outxxx и out_xxx
 term_output_buffer();
 gl_comm = 1; //разрешаем добавлять комментарии к инструкции
 MakeLine(str); //вывод строки на экран с отступом по умолчанию
}
// схема вывода операндов: в my_out() происходит вызов (один раз или
// несколько – зависит от количества операндов конкретной инструкции)
// out_one_operand(n – индекс массива cmd.Operands[] от 0..5), которая
// неявно вызывает my_outop(x) для вывода конкретного операнда
bool idaapi my_outop(op_t& x) {
}
void idaapi my_data(ea_t ea) {
//Вывод на экран форматированных данных (например, отобразить данные с
//типом dword) по указанному адресу ea
}
```

Функции вывода мнемоники команды, регистров команды, операндов (объявлены в ***ua.hpp***)

Функции проверки типов данных (**isWord**, **isDwrd** и т.д.) объявлены в ***bytes.hpp(<IDASDK>\include)***