```
#pragma once

/*
Real types by Dog
*/

#ifndef _TYPE_MM
#define _TYPE_MM
#endif

#ifndef _MSC_VER
how ?
#endif

#ifndef _WIN64
need x64
#endif
/*
Если вы используете эти определения типов, то вы крутые. А вообще
    используйте эти типы, чтоб показать, что код написан вами
*/

//default types

typedef __int64 qindex;

typedef unsigned __int64 ptr;
typedef unsigned __int32 ptr32;

typedef char int8;
typedef unsigned char uint8;

typedef short int16;
typedef unsigned short uint16;

typedef __int32 int32;
typedef unsigned __int32 uint32;

typedef __int64 int64;
typedef unsigned __int64 uint64;

typedef float float32;
typedef double float64;

typedef const char* szstring;

//проверка типов
bool _isfloat(szstring);
bool _isstring(szstring);
bool _isuint(szstring);
bool _isint(szstring);
bool _isbool(szstring);

```

```
54  //это текст??
55  bool _isstring(szstring szValue) {
56      return ((*szValue) == '"' && szValue[strlen(szValue) - 1] == '"');
57  }
58
59  //это беззнаковое число??
60  bool _isuint(szstring szValue) {
61      if (_isstring(szValue) == true || _isfloat(szValue) == true)
62          return false;
63
64      while (*szValue)
65      {
66          uint8 wValue = *szValue - '0';
67          if (wValue > 9)
68              return false;
69          szValue++;
70      }
71      return true;
72  }
73
74  //это число??
75  bool _isint(szstring szValue) {
76      if (_isstring(szValue) == true || _isfloat(szValue) == true)
77          return false;
78
79      if (*szValue == '-')
80          szValue++;
81
82      while (*szValue)
83      {
84          uint8 wValue = *szValue - '0';
85          if (wValue > 9)
86              return false;
87          szValue++;
88      }
89      return true;
90  }
91
92  //это булиан?
93  bool _isbool(szstring szValue) {
94      return !strcmp(szValue, "true") || !strcmp(szValue, "false");
95  }
96
97  //это число с плавующей точкой??
98  bool _isfloat(szstring szValue) {
99      if (_isstring(szValue) == true)
100         return false;
101
102     if (*szValue == '-')
103         szValue++;
104
105     bool bIsFloat = false;
106     while (*szValue)
```

```
107     {
108         uint8 wValue = *szValue - '0';
109         if (wValue > 9 && !(*szValue == '.' || *szValue == 'f'))
110             return false;
111         else if (*szValue == '.' && !bIsFloat)
112             bIsFloat = true;
113         else if (bIsFloat && *szValue == '.')
114             return false;
115 
116         szValue++;
117     }
118     return bIsFloat;
119 }
120 
121 
122 #include <Windows.h>
123 #include <vector>
124 #include <map>
125 #include <iostream>
126 
127 /*
128 CHANGE LOG
129 ---11.12.2023 19:55
130     -добавлена функция call_to
131     -понял, что если оффсет функции будет равен индексу переменной,
          тогда поток не создастся
132 
133 
134 */
135 
136 enum BVType : char {
137     BVT_BYTE = 1,
138     BVT_WORD = 2,
139     BVT_DWORD = 4,
140     BVT_QWORD = 8
141 };
142 
143 enum JIF : unsigned char {
144     JIF_EQUAL = 0x85, //== je
145     JIF_NOT_EQUAL = 0x84, // != jne
146 
147     JIF_BIGGER_U = 0x86, //> ja 0x87
148     JIF_BIGGER_EQUAL_U = 0x82, //>= jae 0x83
149     JIF_LOWER_U = 0x83, //< jnae 0x82
150     JIF_LOWER_EQUAL_U = 0x87, //<= jna 0x86
151 
152     JIF_BIGGER = 0x8E, //> jnle 0x8F
153     JIF_BIGGER_EQUAL = 0x8C, //>= jnl 0x8D
154     JIF_LOWER = 0x8D, //< jnge 0x8C
155     JIF_LOWER_EQUAL = 0x8F, //<= jng 0x8E
156 };
157 
158 //ABOUT START FUNCTION!!!!!!
```

```
159 //ОНО ДАСТ ВАМ ВЗАИМОДЕЙСТВОВАТЬ С ФУНКЦИЯМИ (MessageBox типо)
160 /*
161     int _var_size = 6;
162     int _tmp_size = 5;
163 
164     start_func(32 + 16 * _var_size, 200 + 16 * _tmp_size);
165 
166 */
167 
168 //assembly code
169 class MAssembly {
170 private:
171     //assembly code
172     std::vector<std::vector<BYTE>> m_vCode;
173     std::vector<BYTE> m_vFullCode;
174 
175     //size
176     uint64 m_qOffsets;
177 
178     //if i ll using temp
179     uint64 m_qSubOffset;
180 
181     //clear all code
182     void __setFullCode() {
183         m_vFullCode.clear();
184 
185         for (auto& dCode : m_vCode)
186         {
187             for (uint64 i = 0; i < dCode.size(); i++)
188             {
189                 m_vFullCode.push_back(dCode.data()[i]);
190             }
191         }
192     }
193 public:
194     MAssembly() : m_qOffsets(8), m_qSubOffset(0) {
195     }
196     ~MAssembly() {
197         m_vCode.clear();
198         m_vFullCode.clear();
199         m_qOffsets = m_qSubOffset = 0;
200     }
201 
202     std::vector<BYTE>& __data() {
203         __setFullCode();
204         return m_vFullCode;
205     }
206 
207     std::vector < std::vector<BYTE> > & __notdata() {
208         return m_vCode;
209     }
210 
211     //set to var by index
```

```
212     void set_var(const int32 qIndex, const void* pData, const BVType
          bvtType) {  //set i var to value
213         switch (bvtType)
214         {
215         case BVT_BYTE:
216         {
217             m_vCode.resize(m_vCode.size() + 1);
218             BYTE aCode[] = { 0xC6, 0x85, 0x0, 0x0, 0x0, 0x0, 0x1 };
219             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
220             memcpy(&aCode[6], pData, BVT_BYTE);
221             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
222             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
223         }
224             break;
225         case BVT_WORD:
226         {
227             m_vCode.resize(m_vCode.size() + 1);
228             BYTE aCode[] = { 0x66, 0xC7, 0x85 ,0x77, 0x04, 0x00, 0x20,
                   0x66, 0x25 };
229             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
230             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
231             memcpy(&aCode[7], pData, bvtType);
232             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
233         }
234             break;
235         case BVT_DWORD:
236         {
237             m_vCode.resize(m_vCode.size() + 1);
238             BYTE aCode[] = { 0xC7, 0x85, 0x0, 0x0, 0x0, 0x0, 0x1, 0x1,
                   0x1, 0x1 };
239             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
240             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
241             memcpy(&aCode[6], pData, bvtType);
242             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
243         }
244             break;
245         case BVT_QWORD:
246         {
247             m_vCode.resize(m_vCode.size() + 1);
248             BYTE aCode[] = { 0x49, 0xBE, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1,
                   0x1, 0x1,
249             0x4C, 0x89, 0xB5, 0x0,0x0,0x0,0x0 };
250             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
251             memcpy(&aCode[13], &qIndex, sizeof(qIndex));
252             memcpy(&aCode[2], pData, bvtType);
253             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
254         }
255             break;
256         }
```

```
257
258     }
259     void cpy_var(const int32 qIndex, const int32 qIndex2, const BVType
           bvtType) {  //copy from 1 var to 2
260         switch (bvtType)
261         {
262         case BVT_BYTE:
263         {
264             m_vCode.resize(m_vCode.size() + 1);
265             BYTE aCode[] = { 0x44, 0x8A, 0xB5, 0x0, 0x0, 0x0, 0x0,
266                 0x44, 0x88, 0xB5, 0x0, 0x0, 0x0, 0x0
267             };
268             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
269             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
270             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
271             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
272         }
273         break;
274         case BVT_WORD:
275         {
276             m_vCode.resize(m_vCode.size() + 1);
277             BYTE aCode[] = { 0x66, 0x44, 0x8B, 0xB5, 0x0, 0x0, 0x0,
                0x0,
278                 0x44, 0x88, 0xB5, 0x0, 0x0, 0x0, 0x0
279             };
280             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
281             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
282             memcpy(&aCode[11], &qIndex2, sizeof(qIndex2));
283             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
284         }
285         break;
286         case BVT_DWORD:
287         {
288             m_vCode.resize(m_vCode.size() + 1);
289             BYTE aCode[] = { 0x44, 0x8B, 0xB5, 0x0, 0x0, 0x0, 0x0,
290                 0x44, 0x89, 0xB5, 0x0, 0x0, 0x0, 0x0
291             };
292             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
293             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
294             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
295             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
296         }
297         break;
298         case BVT_QWORD:
299         {
300             m_vCode.resize(m_vCode.size() + 1);
301             BYTE aCode[] = { 0x4C, 0x8B, 0xB5, 0x0, 0x0, 0x0, 0x0,
302                 0x4C, 0x89, 0xB5, 0x0, 0x0, 0x0, 0x0
303             };
304             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
```

```
305             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
306             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
307             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
308         }
309         break;
310         }
311     }
312     void cmp_var(const int32 qIndex, const int32 qIndex2, const BVType
          bvtType) {  //compare var, r10, r11
313         switch (bvtType)
314         {
315         case BVT_BYTE:
316         {
317             m_vCode.resize(m_vCode.size() + 1);
318             BYTE aCode[] = { 0x44, 0x8A, 0x95, 0x0, 0x0, 0x0, 0x0,
319                 0x44, 0x8A, 0x9D, 0x0, 0x0, 0x0, 0x0,
320                 0x45, 0x38, 0xDA
321             };
322             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
323             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
324             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
325             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
326         }
327         break;
328         case BVT_WORD:
329         {
330             m_vCode.resize(m_vCode.size() + 1);
331             BYTE aCode[] = { 0x66, 0x44, 0x8B, 0x95, 0x0, 0x0, 0x0,
                  0x0,
332             0x66, 0x44, 0x8B, 0x9D, 0x0, 0x0, 0x0, 0x0,
333             0x66, 0x45, 0x39, 0xDA
334             };
335             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
336             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
337             memcpy(&aCode[12], &qIndex2, sizeof(qIndex2));
338             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
339         }
340         break;
341         case BVT_DWORD:
342         {
343             m_vCode.resize(m_vCode.size() + 1);
344             BYTE aCode[] = { 0x44, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0,
345                 0x44, 0x8B, 0x9D, 0x0, 0x0, 0x0, 0x0,
346                 0x45, 0x39, 0xDA
347             };
348             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
349             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
350             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
351             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
```

```
352         }
353         break;
354         case BVT_QWORD:
355         {
356             m_vCode.resize(m_vCode.size() + 1);
357             BYTE aCode[] = { 0x4C, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0,
358                 0x4C, 0x8B, 0x9D, 0x0, 0x0, 0x0, 0x0,
359                 0x4D, 0x39, 0xDA
360             };
361             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
362             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
363             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
364             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
365         }
366         break;
367         }
368     }
369     void ret_var(const int32 qIndex, const BVType bvtType) { //mov r9d
        to i
370         switch (bvtType)
371         {
372         case BVT_BYTE:
373         {
374             m_vCode.resize(m_vCode.size() + 1);
375             BYTE aCode[] = { 0x44, 0x8A, 0x8D, 0x0, 0x0, 0x0, 0x0};
376             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
377             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
378             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
379         }
380         break;
381         case BVT_WORD:
382         {
383             m_vCode.resize(m_vCode.size() + 1);
384             BYTE aCode[] = { 0x66, 0x44, 0x8B, 0x8D, 0x0, 0x0, 0x0,
                0x0 };
385             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
386             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
387             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
388         }
389         break;
390         case BVT_DWORD:
391         {
392             m_vCode.resize(m_vCode.size() + 1);
393             BYTE aCode[] = { 0x44, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
394             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
395             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
396             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
397         }
398         break;
```

```
399             case BVT_QWORD:
400             {
401                 m_vCode.resize(m_vCode.size() + 1);
402                 BYTE aCode[] = { 0x4C, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
403                 m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
404                 memcpy(&aCode[3], &qIndex, sizeof(qIndex));
405                 memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
406             }
407             break;
408             }
409         }
410         void ret_var_ex(const BVType bvtType) { //better use tmp_set
411             switch (bvtType)
412             {
413             case BVT_QWORD:
414             {
415                 m_vCode.resize(m_vCode.size() + 1);
416                 BYTE aCode[] = { 0x49, 0x8B, 0xC1};
417                 m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
418                 memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
419             }
420             break;
421             case BVT_DWORD:
422             {
423                 m_vCode.resize(m_vCode.size() + 1);
424                 BYTE aCode[] = { 0x41, 0x8B, 0xC1 };
425                 m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
426                 memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
427             }
428             break;
429             case BVT_WORD:
430             {
431                 m_vCode.resize(m_vCode.size() + 1);
432                 BYTE aCode[] = { 0x41, 0x0F, 0xB7, 0xC1};
433                 m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
434                 memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
435             }
436             break;
437             case BVT_BYTE:
438             {
439                 m_vCode.resize(m_vCode.size() + 1);
440                 BYTE aCode[] = { 0x41, 0x0F, 0xB6, 0xC1 };
441                 m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
442                 memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
443             }
444             break;
445             }
446         }
```

```
447     void get_result(const int32 qIndex, const BVType bvtType) { //r9d to dword ptr[i]
448         switch (bvtType)
449         {
450         case BVT_BYTE:
451         {
452             m_vCode.resize(m_vCode.size() + 1);
453             BYTE aCode[] = { 0x44, 0x88, 0x8D, 0x0, 0x0, 0x0, 0x0 };
454             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
455             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
456             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode[m_vCode.size() - 1].size());
457         }
458         break;
459         case BVT_WORD:
460         {
461             m_vCode.resize(m_vCode.size() + 1);
462             BYTE aCode[] = { 0x66, 0x44, 0x89, 0x8D, 0x0, 0x0, 0x0, 0x0 };
463             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
464             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
465             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof(aCode));
466         }
467         break;
468         case BVT_DWORD:
469         {
470             m_vCode.resize(m_vCode.size() + 1);
471             BYTE aCode[] = { 0x44, 0x89, 0x8D, 0x0, 0x0, 0x0, 0x0 };
472             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
473             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
474             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof(aCode));
475         }
476         break;
477         case BVT_QWORD:
478         {
479             m_vCode.resize(m_vCode.size() + 1);
480             BYTE aCode[] = { 0x4C, 0x89, 0x8D, 0x0, 0x0, 0x0, 0x0 };
481             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
482             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
483             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof(aCode));
484         }
485         break;
486         }
487     }
488     void get_result_ex(const int32 qIndex, const BVType bvtType) { // r9d to dword ptr[i]
489         switch (bvtType)
490         {
491         case BVT_QWORD:
492         {
```

```
493             m_vCode.resize(m_vCode.size() + 1);
494             BYTE aCode[] = { 0x48, 0x89, 0x85, 0x0, 0x0, 0x0, 0x0 };
495             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
496             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
497             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
498         }
499         break;
500         case BVT_DWORD:
501         {
502             m_vCode.resize(m_vCode.size() + 1);
503             BYTE aCode[] = { 0x89, 0x85, 0x0,0x0,0x0, 0x0 };
504             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
505             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
506             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
507         }
508         break;
509         };
510     }
511     void get_argument(const int32 qIndex, const int32 qIndex2, const
        int32 qFuncOffset, const BVType bvtType) { //get argument, i -
        arg original
512         static DWORD dRbpOffset = 8;
513         this->cpy_var((8 * 3) + qIndex + qFuncOffset, qIndex2,
            bvtType);
514     }
515     void lea_var(const int32 qIndex, const int32 qIndex2) { //set
        pointer i to i2
516         m_vCode.resize(m_vCode.size() + 1);
517         BYTE aCode[] = { 0x4C, 0x8D, 0xAD, 0x0, 0x0, 0x0, 0x0,
518         0x4C, 0x89, 0xAD, 0x0, 0x0, 0x0, 0x0,
519         };
520         memcpy(&aCode[3], &qIndex, sizeof(qIndex));
521         memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
522         m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
523         memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
524     }
525     void set_ptr_var(const int32 qIndex, int32 qIndex2, const BVType
        bvtType) { //set by i-is pointer, i2 - value
526         switch (bvtType)
527         {
528         case BVT_QWORD:
529         {
530             m_vCode.resize(m_vCode.size() + 1);
531             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
532                 0x4C, 0x8B, 0xB5, 0x0, 0x0, 0x0, 0x0,
533                 0x4D, 0x89, 0x75, 0x0,
534             };
535             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
536             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
537             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
```

```
538             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
539         }
540         break;
541         case BVT_DWORD:
542         {
543             m_vCode.resize(m_vCode.size() + 1);
544             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
545                 0x44, 0x8B, 0xB5, 0x0, 0x0, 0x0, 0x0,
546                 0x45, 0x89, 0x75, 0x0,
547             };
548             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
549             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
550             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
551             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
552         }
553         break;
554         case BVT_WORD:
555         {
556             m_vCode.resize(m_vCode.size() + 1);
557             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
558                 0x66, 0x44, 0x8B, 0xB5, 0x0, 0x0, 0x0, 0x0,
559                 0x66, 0x45, 0x89, 0x75, 0x0,
560             };
561             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
562             memcpy(&aCode[11], &qIndex2, sizeof(qIndex2));
563             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
564             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
565         }
566         break;
567         case BVT_BYTE:
568         {
569             m_vCode.resize(m_vCode.size() + 1);
570             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
571                 0x44, 0x8A, 0xB5, 0x0, 0x0, 0x0, 0x0,
572                 0x45, 0x88, 0x75, 0x0,
573             };
574             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
575             memcpy(&aCode[10], &qIndex2, sizeof(qIndex2));
576             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
577             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
578         }
579         break;
580         }
581     }
582     void get_ptr_var(const int32 qIndex, const int32 qIndex2, const
          BVType bvtType) { //get ptr from i1 to i2
583         switch (bvtType)
584         {
585         case BVT_BYTE:
```

```
586         {
587             m_vCode.resize(m_vCode.size() + 1);
588             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
589                 0x45, 0x8A, 0x75, 0x0, //r14b
590                 0x44, 0x88, 0xB6, 0x0, 0x0, 0x0, 0x0
591             };
592             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
593             memcpy(&aCode[14], &qIndex2, sizeof(qIndex2));
594             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
595             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
596         }
597         break;
598         case BVT_WORD:
599         {
600             m_vCode.resize(m_vCode.size() + 1);
601             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
602                 0x66, 0x45, 0x8B, 0x75, 0x0, //r14b
603                 0x66, 0x44, 0x89, 0xB5, 0x0, 0x0, 0x0, 0x0
604             };
605             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
606             memcpy(&aCode[16], &qIndex2, sizeof(qIndex2));
607             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
608             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
609         }
610         break;
611         case BVT_DWORD:
612         {
613             m_vCode.resize(m_vCode.size() + 1);
614             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
615                 0x45, 0x8B, 0x75, 0x0, //r14b
616                 0x44, 0x89, 0xB5, 0x0, 0x0, 0x0, 0x0
617             };
618             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
619             memcpy(&aCode[14], &qIndex2, sizeof(qIndex2));
620             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
621             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
622         }
623         break;
624         case BVT_QWORD:
625         {
626             m_vCode.resize(m_vCode.size() + 1);
627             BYTE aCode[] = { 0x4C, 0x8B, 0xAD, 0x0, 0x0, 0x0, 0x0,
628                 0x4D, 0x8B, 0x75, 0x0, //r14b
629                 0x4C, 0x89, 0xB5, 0x0, 0x0, 0x0, 0x0
630             };
631             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
632             memcpy(&aCode[14], &qIndex2, sizeof(qIndex2));
633             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
634             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
```

```
635         }
636         break;
637         }
638
639     }
640     void set_math_var_float(const int32 qIndex, const BVType
          bvtType) //set to xmm1
641     {
642         switch (bvtType)
643         {
644         case BVT_DWORD:
645         {
646             m_vCode.resize(m_vCode.size() + 1);
647             BYTE aCode[] = { 0xF3, 0x0F, 0x10, 0x8D, 0x0, 0x0, 0x0,
                0x0 };
648             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
649             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
650             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
651         }
652         break;
653         case BVT_QWORD:
654         {
655             m_vCode.resize(m_vCode.size() + 1);
656             BYTE aCode[] = { 0xF2, 0x0F, 0x10, 0x8D, 0x0, 0x0, 0x0,
                0x0 };
657             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
658             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
659             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
660         }
661         break;
662         }
663     }
664     void set_var_to_arg(const int32 qIndex, const int32 qIndex2, const
        BVType bvtType) { //i1 - og var, i2 - to arg
665         this->set_tmp(qIndex, bvtType);
666
667         switch (bvtType)
668         {
669         case BVT_BYTE:
670         {
671             m_vCode.resize(m_vCode.size() + 1);
672             BYTE aCode[] = { 0x88, 0x84, 0x24, 0x0, 0x0, 0x0, 0x0, //
                parser.main - 88 84 24 ABAAAAEB
673             };
674             memcpy(&aCode[3], &qIndex2, sizeof(qIndex2));
675             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
676             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
677         }
678         break;
679         case BVT_WORD:
```

```
680         {
681             m_vCode.resize(m_vCode.size() + 1);
682             BYTE aCode[] = { 0x66, 0x89, 0x84, 0x24, 0x0, 0x0, 0x0,
                  0x0, //parser.main+7 - 66 89 84 24 ABAAAAEB
683             };
684             memcpy(&aCode[4], &qIndex2, sizeof(qIndex2));
685             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
686             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
687         }
688         break;
689         case BVT_DWORD:
690         {
691             m_vCode.resize(m_vCode.size() + 1);
692             BYTE aCode[] = { 0x89, 0x84, 0x24, 0x0, 0x0, 0x0, 0x0, //
                  parser.main+F - 89 84 24 ABAAAAEB
693             };
694             memcpy(&aCode[3], &qIndex2, sizeof(qIndex2));
695             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
696             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
697         }
698         break;
699         case BVT_QWORD:
700         {
701             m_vCode.resize(m_vCode.size() + 1);
702             BYTE aCode[] = { 0x48, 0x89, 0x84, 0x24, 0x0, 0x0, 0x0,
                  0x0, //parser.main+16 - 48 89 84 24 ABAAAAEB
703             };
704             memcpy(&aCode[4], &qIndex2, sizeof(qIndex2));
705             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
706             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
707         }
708         break;
709         }
710     }
711     void set_arg_to_var(const int32 qIndex, const int32 qIndex2, const
        BVType bvtType) { //i1 - arg, i2 - to var
712
713         switch (bvtType)
714         {
715         case BVT_BYTE:
716         {
717             m_vCode.resize(m_vCode.size() + 1);
718             BYTE aCode[] = { 0x8A, 0x84, 0x24, 0x0, 0x0, 0x0, 0x0, //
                  parser.main - 8A 84 24 BCBBFEFF
719             };
720             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
721             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
722             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
723         }
```

```
724         break;
725         case BVT_WORD:
726         {
727             m_vCode.resize(m_vCode.size() + 1);
728             BYTE aCode[] = { 0x66, 0x8B, 0x84, 0x24, 0x0, 0x0, 0x0,
                0x0, //parser.main+7 - 66 8B 84 24 BCBBFEFF
729             };
730             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
731             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
732             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
733         }
734         break;
735         case BVT_DWORD:
736         {
737             m_vCode.resize(m_vCode.size() + 1);
738             BYTE aCode[] = { 0x8B, 0x84, 0x24, 0x0, 0x0, 0x0, 0x0, //
                parser.main+F - 8B 84 24 BCBBFEFF
739             };
740             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
741             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
742             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
743         }
744         break;
745         case BVT_QWORD:
746         {
747             m_vCode.resize(m_vCode.size() + 1);
748             BYTE aCode[] = { 0x48, 0x8B, 0x84, 0x24, 0x0, 0x0, 0x0,
                0x0, //parser.main+16 - 48 8B 84 24 BCBBFEFF
749             };
750             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
751             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
752             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
753         }
754         break;
755         }
756
757         this->get_tmp(qIndex2, bvtType);
758     }
759     void set_math_var(const int32 qIndex, const BVType bvtType) { //
        set to r12
760         switch (bvtType)
761         {
762         case BVT_BYTE:
763         {
764             m_vCode.resize(m_vCode.size() + 1);
765             BYTE aCode[] = { 0x44, 0x8A, 0xA5, 0x0, 0x0, 0x0, 0x0 };
766             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
767             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
768             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
```

```
769         }
770         break;
771         case BVT_WORD:
772         {
773             m_vCode.resize(m_vCode.size() + 1);
774             BYTE aCode[] = { 0x66, 0x44, 0x8B, 0xA5, 0x0, 0x0, 0x0,
                0x0 };
775             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
776             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
777             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
778         }
779         break;
780         case BVT_DWORD:
781         {
782             m_vCode.resize(m_vCode.size() + 1);
783             BYTE aCode[] = { 0x44, 0x8B, 0xA5, 0x0, 0x0, 0x0, 0x0 };
784             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
785             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
786             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
787         }
788         break;
789         case BVT_QWORD:
790         {
791             m_vCode.resize(m_vCode.size() + 1);
792             BYTE aCode[] = { 0x4C, 0x8B, 0xA5, 0x0, 0x0, 0x0, 0x0 };
793             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
794             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
795             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
796         }
797         break;
798         }
799     }
800     void push_ex_0_float(const int32 qIndex, const BVType bvtType)
        { //parser.main - F2 0F10 85 44040000
801
802         switch (bvtType)
803         {
804         case BVT_DWORD:
805         {
806             m_vCode.resize(m_vCode.size() + 1);
807             BYTE aCode[] = { 0xF3, 0x0F, 0x10, 0x85, 0x0, 0x0, 0x0,
                0x0 }; //parser.main - F3 0F10 85 44440400
808             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
809             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
810             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
811         }
812         break;
813         case BVT_QWORD:
814         {
```

```
815             m_vCode.resize(m_vCode.size() + 1);
816             BYTE aCode[] = { 0xF2, 0x0F, 0x10, 0x85, 0x0, 0x0, 0x0,
                  0x0 };
817             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
818             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
819             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
820         }
821         break;
822         }
823
824     }
825     void push_ex_1_float(const int32 qIndex, const BVType bvtType) {
826
827         switch (bvtType)
828         {
829         case BVT_DWORD:
830         {
831             m_vCode.resize(m_vCode.size() + 1);
832             BYTE aCode[] = { 0xF3, 0x0F, 0x10, 0x8D, 0x0, 0x0, 0x0,
                  0x0 };
833             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
834             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
835             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
836         }
837         break;
838         case BVT_QWORD:
839         {
840             m_vCode.resize(m_vCode.size() + 1);
841             BYTE aCode[] = { 0xF2, 0x0F, 0x10, 0x8D, 0x0, 0x0, 0x0,
                  0x0 };
842             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
843             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
844             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
845         }
846         break;
847         }
848     }
849     void push_ex_2_float(const int32 qIndex, const BVType bvtType) {
850
851         switch (bvtType)
852         {
853         case BVT_DWORD:
854         {
855             m_vCode.resize(m_vCode.size() + 1);
856             BYTE aCode[] = { 0xF3, 0x0F, 0x10, 0x95, 0x0, 0x0, 0x0,
                  0x0 };
857             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
858             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
859             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
```

```
860         }
861         break;
862         case BVT_QWORD:
863         {
864             m_vCode.resize(m_vCode.size() + 1);
865             BYTE aCode[] = { 0xF2, 0x0F, 0x10, 0x95, 0x0, 0x0, 0x0,
                0x0 };
866             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
867             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
868             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
869         }
870         break;
871         }
872
873     }
874     void push_ex_3_float(const int32 qIndex, const BVType bvtType) {
875
876         switch (bvtType)
877         {
878         case BVT_DWORD:
879         {
880             m_vCode.resize(m_vCode.size() + 1);
881             BYTE aCode[] = { 0xF3, 0x0F, 0x10, 0x9D, 0x0, 0x0, 0x0,
                0x0 };
882             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
883             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
884             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
885         }
886         break;
887         case BVT_QWORD:
888         {
889             m_vCode.resize(m_vCode.size() + 1);
890             BYTE aCode[] = { 0xF2, 0x0F, 0x10, 0x9D, 0x0, 0x0, 0x0,
                0x0 };
891             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
892             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
893             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
894         }
895         break;
896         }
897     }
898     void push_ex_0(const int32 qIndex, const BVType bvtType) {
899         switch (bvtType)
900         {
901         case BVT_BYTE:
902         {
903             m_vCode.resize(m_vCode.size() + 1);
904             BYTE aCode[] = { 0x8A, 0x8D, 0x0, 0x0, 0x0, 0x0 };
905             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
906             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
```

```
907             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
908         }
909         break;
910         case BVT_WORD:
911         {
912             m_vCode.resize(m_vCode.size() + 1);
913             BYTE aCode[] = { 0x66, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
914             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
915             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
916             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
917         }
918         break;
919         case BVT_DWORD:
920         {
921             m_vCode.resize(m_vCode.size() + 1);
922             BYTE aCode[] = { 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
923             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
924             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
925             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
926         }
927         break;
928         case BVT_QWORD:
929         {
930             m_vCode.resize(m_vCode.size() + 1);
931             BYTE aCode[] = { 0x48, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
932             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
933             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
934             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
935         }
936         break;
937         }
938     }
939     void push_ex_1(const int32 qIndex, const BVType bvtType) {
940         switch (bvtType)
941         {
942         case BVT_BYTE:
943         {
944             m_vCode.resize(m_vCode.size() + 1);
945             BYTE aCode[] = { 0x8A, 0x95, 0x0, 0x0, 0x0, 0x0 };
946             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
947             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
948             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
949         }
950         break;
951         case BVT_WORD:
952         {
953             m_vCode.resize(m_vCode.size() + 1);
954             BYTE aCode[] = { 0x66, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0 };
```

```
955             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
956             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
957             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
958         }
959         break;
960         case BVT_DWORD:
961         {
962             m_vCode.resize(m_vCode.size() + 1);
963             BYTE aCode[] = { 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0 };
964             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
965             memcpy(&aCode[2], &qIndex, sizeof(qIndex));
966             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
967         }
968         break;
969         case BVT_QWORD:
970         {
971             m_vCode.resize(m_vCode.size() + 1);
972             BYTE aCode[] = { 0x48, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0 };
973             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
974             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
975             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
976         }
977         break;
978         }
979     }
980     void push_ex_2(const int32 qIndex, const BVType bvtType) {
981         switch (bvtType)
982         {
983         case BVT_BYTE:
984         {
985             m_vCode.resize(m_vCode.size() + 1);
986             BYTE aCode[] = { 0x44, 0x8A, 0x85, 0x0, 0x0, 0x0, 0x0 };
987             memcpy(&aCode[3], &qIndex, sizeof(qIndex));
988             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
989             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
990         }
991         break;
992         case BVT_WORD:
993         {
994             m_vCode.resize(m_vCode.size() + 1);
995             BYTE aCode[] = { 0x66, 0x44, 0x8B, 0x85, 0x0, 0x0, 0x0,
                  0x0 };
996             m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
997             memcpy(&aCode[4], &qIndex, sizeof(qIndex));
998             memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
999         }
1000        break;
1001        case BVT_DWORD:
```

```
1002            {
1003                m_vCode.resize(m_vCode.size() + 1);
1004                BYTE aCode[] = { 0x44, 0x8B, 0x85, 0x0, 0x0, 0x0, 0x0 };
1005                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1006                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1007                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1008            }
1009            break;
1010            case BVT_QWORD:
1011            {
1012                m_vCode.resize(m_vCode.size() + 1);
1013                BYTE aCode[] = { 0x4C, 0x8B, 0x85, 0x0, 0x0, 0x0, 0x0 };
1014                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1015                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1016                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1017            }
1018            break;
1019            }
1020        }
1021        void push_ex_3(const int32 qIndex, const BVType bvtType) {
1022            switch (bvtType)
1023            {
1024            case BVT_BYTE:
1025            {
1026                m_vCode.resize(m_vCode.size() + 1);
1027                BYTE aCode[] = { 0x44, 0x8A, 0x8D, 0x0, 0x0, 0x0, 0x0 };
1028                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1029                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1030                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1031            }
1032            break;
1033            case BVT_WORD:
1034            {
1035                m_vCode.resize(m_vCode.size() + 1);
1036                BYTE aCode[] = { 0x66, 0x44, 0x8B, 0x8D, 0x0, 0x0, 0x0,
                      0x0 };
1037                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1038                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1039                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1040            }
1041            break;
1042            case BVT_DWORD:
1043            {
1044                m_vCode.resize(m_vCode.size() + 1);
1045                BYTE aCode[] = { 0x44, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
1046                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1047                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1048                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
```

```
1049            }
1050            break;
1051            case BVT_QWORD:
1052            {
1053                m_vCode.resize(m_vCode.size() + 1);
1054                BYTE aCode[] = { 0x4C, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0 };
1055                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1056                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1057                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1058            }
1059            break;
1060            }
1061        }
1062        void push_var_ex(const int32 qIndex, const int32 qIndex2, const
              BVType bvtType) { //i - rbp, i2 - rsp
1063            switch (bvtType)
1064            {
1065            case BVT_BYTE:
1066            {
1067                m_vCode.resize(m_vCode.size() + 1);
1068                BYTE aCode[] = { 0x44, 0x8A, 0x8D, 0x0, 0x0, 0x0, 0x0,
1069                    0x44, 0x88, 0x8C, 0x24, 0x0, 0x0, 0x0, 0x0
1070                };
1071                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1072                memcpy(&aCode[11], &qIndex2, sizeof(qIndex2));
1073                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1074                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1075            }
1076            break;
1077            case BVT_WORD:
1078            {
1079                m_vCode.resize(m_vCode.size() + 1);
1080                BYTE aCode[] = { 0x66, 0x44, 0x8B, 0x8D, 0x0, 0x0, 0x0,
                      0x0,
1081                    0x66, 0x44, 0x89, 0x8C, 0x24, 0x0, 0x0, 0x0, 0x0
1082                };
1083                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1084                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1085                memcpy(&aCode[13], &qIndex2, sizeof(qIndex2));
1086                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1087            }
1088            break;
1089            case BVT_DWORD:
1090            {
1091                m_vCode.resize(m_vCode.size() + 1);
1092                BYTE aCode[] = { 0x44, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0,
1093                    0x44, 0x89, 0x8C, 0x24, 0x0, 0x0, 0x0, 0x0
1094                };
1095                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1096                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
```

```
1097            memcpy(&aCode[11], &qIndex2, sizeof(qIndex2));
1098            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1099        }
1100        break;
1101        case BVT_QWORD:
1102        {
1103            m_vCode.resize(m_vCode.size() + 1);
1104            BYTE aCode[] = { 0x4C, 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0,
1105                0x4C, 0x89, 0x8C, 0x24, 0x0, 0x0, 0x0, 0x0
1106            };
1107            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1108            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1109            memcpy(&aCode[11], &qIndex2, sizeof(qIndex2));
1110            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1111        }
1112        break;
1113        }
1114    }
1115    void add_var_float(const int32 qIndex, const BVType bvtType) { //
          xmm1 + dword ptr
1116
1117        switch (bvtType)
1118        {
1119        case BVT_DWORD:
1120        {
1121            m_vCode.resize(m_vCode.size() + 1);
1122            BYTE aCode[] = { 0xF3, 0x0F, 0x58, 0x8D, 0x0, 0x0, 0x0,
                  0x0 };
1123            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1124            memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1125            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1126        }
1127        break;
1128        case BVT_QWORD:
1129        {
1130            m_vCode.resize(m_vCode.size() + 1);
1131            BYTE aCode[] = { 0xF2, 0x0F, 0x58, 0x8D, 0x0, 0x0, 0x0,
                  0x0 };
1132            memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1133            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1134            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1135        }
1136        break;
1137        }
1138    }
1139    void sub_var_float(const int32 qIndex, const BVType bvtType) { //
          xmm1 - dword ptr
1140
1141        switch (bvtType)
```

```
1142            {
1143            case BVT_DWORD:
1144            {
1145                m_vCode.resize(m_vCode.size() + 1);
1146                BYTE aCode[] = { 0xF3, 0x0F, 0x5C, 0x8D, 0x0, 0x0, 0x0,    ↵
                    0x0 };
1147                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1148                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1149                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof   ↵
                    (aCode));
1150            }
1151            break;
1152            case BVT_QWORD:
1153            {
1154                m_vCode.resize(m_vCode.size() + 1);
1155                BYTE aCode[] = { 0xF2, 0x0F, 0x5C, 0x8D, 0x0, 0x0, 0x0,    ↵
                    0x0 };
1156                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1157                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1158                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode  ↵
                    [m_vCode.size() - 1].size());
1159            }
1160            break;
1161            }
1162        }
1163        void mul_var_float(const int32 qIndex, const BVType bvtType) { //  ↵
              xmm1 + dword ptr
1164
1165            switch (bvtType)
1166            {
1167            case BVT_DWORD:
1168            {
1169                m_vCode.resize(m_vCode.size() + 1);
1170                BYTE aCode[] = { 0xF3, 0x0F, 0x59, 0x8D, 0x0, 0x0, 0x0,    ↵
                    0x0 };
1171                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1172                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1173                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof   ↵
                    (aCode));
1174            }
1175            break;
1176            case BVT_QWORD:
1177            {
1178                m_vCode.resize(m_vCode.size() + 1);
1179                BYTE aCode[] = { 0xF2, 0x0F, 0x59, 0x8D, 0x0, 0x0, 0x0,    ↵
                    0x0 };
1180                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1181                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1182                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof   ↵
                    (aCode));
1183            }
1184            break;
1185            }
```

```
1186
1187      }
1188      void div_var_float(const int32 qIndex, const BVType bvtType) { //  ↪
            xmm1 + dword ptr
1189
1190          switch (bvtType)
1191          {
1192          case BVT_DWORD:
1193          {
1194              m_vCode.resize(m_vCode.size() + 1);
1195              BYTE aCode[] = { 0xF3, 0x0F, 0x5E, 0x8D, 0x0, 0x0, 0x0,     ↪
                    0x0 };
1196              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1197              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1198              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof    ↪
                    (aCode));
1199          }
1200          break;
1201          case BVT_QWORD:
1202          {
1203              m_vCode.resize(m_vCode.size() + 1);
1204              BYTE aCode[] = { 0xF2, 0x0F, 0x5E, 0x8D, 0x0, 0x0, 0x0,     ↪
                    0x0 };
1205              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1206              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1207              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof    ↪
                    (aCode));
1208          }
1209          break;
1210          }
1211      }
1212      void add_var(const int32 qIndex, const BVType bvtType) { //r12 +    ↪
            ptr = i
1213          switch (bvtType)
1214          {
1215          case BVT_BYTE:
1216          {
1217              m_vCode.resize(m_vCode.size() + 1);
1218              BYTE aCode[] = { 0x44, 0x02, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1219              memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1220              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1221              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode   ↪
                    [m_vCode.size() - 1].size());
1222          }
1223          break;
1224          case BVT_WORD:
1225          {
1226              m_vCode.resize(m_vCode.size() + 1);
1227              BYTE aCode[] = { 0x66, 0x44, 0x03, 0xA5, 0x0, 0x0, 0x0,     ↪
                    0x0 };
1228              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1229              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1230              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof    ↪
```

```
                (aCode));
1231        }
1232        break;
1233        case BVT_DWORD:
1234        {
1235            m_vCode.resize(m_vCode.size() + 1);
1236            BYTE aCode[] = { 0x44, 0x03, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1237            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1238            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1239            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
1240        }
1241        break;
1242        case BVT_QWORD:
1243        {
1244            m_vCode.resize(m_vCode.size() + 1);
1245            BYTE aCode[] = { 0x4C, 0x03, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1246            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1247            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1248            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
1249        }
1250        break;
1251        }
1252    }
1253    void sub_var(const int32 qIndex, const BVType bvtType) { //r12 -
        r13=i
1254        switch (bvtType)
1255        {
1256        case BVT_BYTE:
1257        {
1258            m_vCode.resize(m_vCode.size() + 1);
1259            BYTE aCode[] = { 0x44, 0x2A, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1260            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1261            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1262            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                [m_vCode.size() - 1].size());
1263        }
1264        break;
1265        case BVT_WORD:
1266        {
1267            m_vCode.resize(m_vCode.size() + 1);
1268            BYTE aCode[] = { 0x66, 0x44, 0x2B, 0xA5, 0x0, 0x0, 0x0,
                0x0 };
1269            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1270            memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1271            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                (aCode));
1272        }
1273        break;
1274        case BVT_DWORD:
1275        {
1276            m_vCode.resize(m_vCode.size() + 1);
```

```
1277                BYTE aCode[] = { 0x44, 0x2B, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1278                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1279                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1280                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1281            }
1282            break;
1283            case BVT_QWORD:
1284            {
1285                m_vCode.resize(m_vCode.size() + 1);
1286                BYTE aCode[] = { 0x4C, 0x2B, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1287                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1288                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1289                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1290            }
1291            break;
1292            }
1293        }
1294        void mul_var(const int32 qIndex, const BVType bvtType) { //r12 *
              r13=i
1295            switch (bvtType)
1296            {
1297            case BVT_BYTE:
1298            {
1299                m_vCode.resize(m_vCode.size() + 1);
1300                BYTE aCode[] = { 0x44, 0x88, 0xE0,
1301                    0x8A, 0x9D, 0x0, 0x0, 0x0, 0x0,
1302                    0xF6, 0xE3,
1303                    0x41, 0x88, 0xC4
1304                };
1305                memcpy(&aCode[5], &qIndex, sizeof(qIndex));
1306                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1307                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1308            }
1309            break;
1310            case BVT_WORD:
1311            {
1312                m_vCode.resize(m_vCode.size() + 1);
1313                BYTE aCode[] = { 0x66, 0x44, 0x89, 0xE0,
1314                    0x66, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0,
1315                    0x66, 0xF7, 0xE2,
1316                    0x66, 0x41, 0x89, 0xC4
1317                };
1318                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1319                memcpy(&aCode[7], &qIndex, sizeof(qIndex));
1320                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1321            }
1322            break;
1323            case BVT_DWORD:
1324            {
```

```
1325                m_vCode.resize(m_vCode.size() + 1);
1326                BYTE aCode[] = { 0x44, 0x0F, 0xAF, 0xA5, 0x0, 0x0, 0x0,
                      0x0,
1327                };
1328                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1329                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1330                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1331            }
1332            break;
1333            case BVT_QWORD:
1334            {
1335                m_vCode.resize(m_vCode.size() + 1);
1336                BYTE aCode[] = { 0x4C, 0x0F, 0xAF, 0xA5, 0x0, 0x0, 0x0,
                      0x0,
1337                };
1338                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1339                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1340                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1341            }
1342            break;
1343            }
1344        }
1345        void div_var(const int32 qIndex, const BVType bvtType) { //r12
1346            switch (bvtType)
1347            {
1348            case BVT_BYTE:
1349            {
1350                m_vCode.resize(m_vCode.size() + 1);
1351                BYTE aCode[] = { 0x41, 0x0F, 0xB6, 0xC4,
1352                    0x0F, 0xB6, 0x8D, 0x0, 0x0, 0x0, 0x0,
1353                    0x99,
1354                    0xF7, 0xF9,
1355                    0x41, 0x88, 0xC4
1356                };
1357                memcpy(&aCode[7], &qIndex, sizeof(qIndex));
1358                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1359                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1360            }
1361            break;
1362            case BVT_WORD:
1363            {
1364                m_vCode.resize(m_vCode.size() + 1);
1365                BYTE aCode[] = { 0x41, 0x0F, 0xB7, 0xC4,
1366                    0x0F, 0xB7, 0x8D, 0x0, 0x0, 0x0, 0x0,
1367                    0x99,
1368                    0xF7, 0xF9,
1369                    0x66, 0x41, 0x89, 0xC4
1370                };
1371                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1372                memcpy(&aCode[7], &qIndex, sizeof(qIndex));
```

```
1373            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1374        }
1375        break;
1376        case BVT_DWORD:
1377        {
1378            m_vCode.resize(m_vCode.size() + 1);
1379            BYTE aCode[] = { 0x8B, 0x8D, 0x0, 0x0, 0x0, 0x0,
1380                0x41, 0x8B, 0xC4,
1381                0x99,
1382                0xF7, 0xF9,
1383                0x44, 0x8B, 0xE0
1384            };
1385            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1386            memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1387            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1388        }
1389        break;
1390        case BVT_QWORD:
1391        {
1392            m_vCode.resize(m_vCode.size() + 1);
1393            BYTE aCode[] = {
1394                0x49, 0x8B, 0xC4,
1395                0x48, 0x99,
1396                0x48, 0xF7, 0xBD, 0x0, 0x0, 0x0, 0x0,
1397                0x4C, 0x8B, 0xE0
1398            };
1399            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1400            memcpy(&aCode[8], &qIndex, sizeof(qIndex));
1401            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1402        }
1403        break;
1404        }
1405    }
1406    /*
1407        void div_var(const int32 qIndex, const BVType bvtType) { //
              r12 / r13=i
1408        switch (bvtType)
1409        {
1410        case BVT_BYTE:
1411        {
1412            m_vCode.resize(m_vCode.size() + 1);
1413            BYTE aCode[] = { 0x44, 0x88, 0xE0,
1414                0x8A, 0x9D, 0x0, 0x0, 0x0, 0x0,
1415                0xF6, 0xF0,
1416                0x41, 0x88, 0xC4
1417            };
1418            memcpy(&aCode[5], &qIndex, sizeof(qIndex));
1419            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1420            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
```

```
1421            }
1422            break;
1423            case BVT_WORD:
1424            {
1425                m_vCode.resize(m_vCode.size() + 1);
1426                BYTE aCode[] = { 0x66, 0x44, 0x89, 0xE0,
1427                    0x66, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0,
1428                    0x66, 0xF7, 0xF2,
1429                    0x66, 0x41, 0x89, 0xC4
1430                };
1431                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1432                memcpy(&aCode[7], &qIndex, sizeof(qIndex));
1433                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1434            }
1435            break;
1436            case BVT_DWORD:
1437            {
1438                m_vCode.resize(m_vCode.size() + 1);
1439                BYTE aCode[] = { 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0,
1440                    0x41, 0x8B, 0xC4,
1441                    0xF7, 0xFA,
1442                    0x44, 0x8B, 0xE0
1443                };
1444                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1445                memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1446                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1447            }
1448            break;
1449            case BVT_QWORD:
1450            {
1451                m_vCode.resize(m_vCode.size() + 1);
1452                BYTE aCode[] = { 0x48, 0x8B, 0x95, 0x0, 0x0, 0x0, 0x0,
1453                    0x49, 0x8B, 0xC4,
1454                    0x48, 0xF7, 0xFA,
1455                    0x4C, 0x8B, 0xE0
1456                };
1457                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1458                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1459                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1460            }
1461            break;
1462            }
1463        }
1464        */
1465        void get_math_float(const int32 qIndex, const BVType bvtType) {
1466
1467            switch (bvtType)
1468            {
1469            case BVT_DWORD:
1470            {
```

```
1471                m_vCode.resize(m_vCode.size() + 1);
1472                BYTE aCode[] = { 0xF3, 0x0F, 0x11, 0x8D, 0x0, 0x0, 0x0,
                      0x0 };
1473                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1474                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1475                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1476            }
1477            break;
1478            case BVT_QWORD:
1479            {
1480                m_vCode.resize(m_vCode.size() + 1);
1481                BYTE aCode[] = { 0xF2, 0x0F, 0x11, 0x8D, 0x0, 0x0, 0x0,
                      0x0 };
1482                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1483                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1484                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1485            }
1486            break;
1487            }
1488        }
1489        void get_math(const int32 qIndex, const BVType bvtType) { //set
              math result to i
1490            switch (bvtType)
1491            {
1492            case BVT_BYTE:
1493            {
1494                m_vCode.resize(m_vCode.size() + 1);
1495                BYTE aCode[] = { 0x44, 0x88, 0xA5, 0x0, 0x0, 0x0, 0x0};
1496                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1497                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1498                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1499            }
1500            break;
1501            case BVT_WORD:
1502            {
1503                m_vCode.resize(m_vCode.size() + 1);
1504                BYTE aCode[] = { 0x66, 0x44, 0x89, 0xA5, 0x0, 0x0, 0x0,
                      0x0 };
1505                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1506                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1507                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                      (aCode));
1508            }
1509            break;
1510            case BVT_DWORD:
1511            {
1512                m_vCode.resize(m_vCode.size() + 1);
1513                BYTE aCode[] = { 0x44, 0x89, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1514                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1515                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
```

```
1516            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1517        }
1518        break;
1519        case BVT_QWORD:
1520        {
1521            m_vCode.resize(m_vCode.size() + 1);
1522            BYTE aCode[] = { 0x4C, 0x89, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1523            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1524            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1525            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof
                  (aCode));
1526        }
1527        break;
1528        }
1529    }
1530    void rax_to_var(const int32 qIndex) {
1531        m_vCode.resize(m_vCode.size() + 1);
1532        BYTE aCode[] = { 0x48, 0x89, 0x85, 0x0, 0x0, 0x0, 0x0 };
1533        memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1534        m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1535        memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
1536    }
1537    void call_var(const int32 qIndex) { //called function by i-adr,
          index - index of function
1538        m_vCode.resize(m_vCode.size() + 1);
1539        BYTE aCode[] = { 0xFF, 0x95, 0x0, 0x0, 0x0, 0x0};
1540        memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1541        m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1542        memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
1543    }
1544    void call_to(const uint64 qAddress) { //called function by i-adr,
          index - index of function
1545        m_vCode.resize(m_vCode.size() + 1);
1546        BYTE aCode[] = { 0xFF, 0x15, 0x02, 0x00, 0x00, 0x00, 0xEB,
              0x08, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
1547        };
1548        memcpy(&aCode[8], &qAddress, sizeof(qAddress));
1549        m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1550        memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
1551    }
1552    void call_offset(const uint32 qAddress) { //called function by i-
          adr, index - index of function
1553        m_vCode.resize(m_vCode.size() + 1);
1554        BYTE aCode[] = { 0xE8, 0x15, 0x02, 0x00, 0x00
1555        };
1556        memcpy(&aCode[1], &qAddress, sizeof(qAddress));
1557        m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1558        memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
```

```
1559      }
1560      void jmp_var(const int32 qIndex) { //jmp index for jmphere
1561          m_vCode.resize(m_vCode.size() + 1);
1562          BYTE aCode[] = { 0xFF, 0xA5, 0x0, 0x0, 0x0, 0x0 };
1563          memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1564          m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1565          memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
            [m_vCode.size() - 1].size());
1566      }
1567      void jmp_to(const int64 qAddress) { //jmp index for jmphere
1568          m_vCode.resize(m_vCode.size() + 1);
1569          BYTE aCode[] = { 0xFF, 0x25, 0x0, 0x0, 0x0, 0x0, 0x1, 0x1,
            0x1, 0x1, 0x1, 0x1, 0x1, 0x1};
1570          memcpy(&aCode[6], &qAddress, sizeof(qAddress));
1571          m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1572          memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
            [m_vCode.size() - 1].size());
1573      }
1574      void jmp_offset(const int32 qOffset) { //jmp index for jmphere
1575          m_vCode.resize(m_vCode.size() + 1);
1576          BYTE aCode[] = { 0xE9, 0x0, 0x0, 0x0, 0x0};
1577          memcpy(&aCode[1], &qOffset, sizeof(qOffset));
1578          m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1579          memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
            [m_vCode.size() - 1].size());
1580      }
1581      void set_if(const int32 qSize, const JIF jmpType) { //jump if to
          index for jmphere
1582          //TODO: улучшить
1583          m_vCode.resize(m_vCode.size() + 1);
1584          BYTE aCode[] = {
1585              0x0F, 0x1, 0x0, 0x0, 0x0, 0x0
1586          };
1587          memcpy(&aCode[1], &jmpType, sizeof(jmpType));
1588          memcpy(&aCode[2], &qSize, sizeof(qSize));
1589          m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1590          memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
            [m_vCode.size() - 1].size());
1591      }
1592      /*
1593          48 0FBE 85 60020000   - movsx rax,byte ptr [rbp+00000260]
1594          48 0FBF 85 60020000   - movsx rax,word ptr [rbp+00000260]
1595          48 63 85 60020000     - movsxd  rax,dword ptr [rbp+00000260]
1596          48 8B 85 60020000     - mov rax,[rbp+00000260]
1597
1598
1599      48 0FBE 14 25 22220200  - movsx rdx,byte ptr [00022222]
1600      48 0FBF 14 25 22220200  - movsx rdx,word ptr [00022222]
1601      48 63 14 25 22220200  - movsxd  rdx,dword ptr [00022222]
1602      48 8B 14 25 22220200  - mov rdx,[00022222]
1603
1604      */
1605      void cmp_rax_rdx() {
```

```
1606            m_vCode.resize(m_vCode.size() + 1);
1607            BYTE aCode[] = { 0x48, 0x39, 0xD0 };
1608            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1609            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1610        }
1611        void set_rax(const int32 qIndex, const BVType bvtType) {
1612            switch (bvtType)
1613            {
1614            case BVT_BYTE:
1615            {
1616                m_vCode.resize(m_vCode.size() + 1);
1617                BYTE aCode[] = { 0x48, 0x0F, 0xBE, 0x85,
1618                    0x0, 0x0, 0x0, 0x0 };
1619                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1620                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1621                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                    [m_vCode.size() - 1].size());
1622            }
1623            break;
1624            case BVT_WORD:
1625            {
1626                m_vCode.resize(m_vCode.size() + 1);
1627                BYTE aCode[] = { 0x48, 0x0F, 0xBF, 0x85,
1628                    0x0, 0x0, 0x0, 0x0 };
1629                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1630                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1631                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                    [m_vCode.size() - 1].size());
1632            }
1633            break;
1634            case BVT_DWORD:
1635            {
1636                //48 63 04 25 22220200
1637                m_vCode.resize(m_vCode.size() + 1);
1638                BYTE aCode[] = { 0x48, 0x63, 0x85,
1639                    0x0, 0x0, 0x0, 0x0 };
1640                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1641                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1642                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                    [m_vCode.size() - 1].size());
1643            }
1644            break;
1645            case BVT_QWORD:
1646            {
1647                m_vCode.resize(m_vCode.size() + 1);
1648                BYTE aCode[] = { 0x48, 0x8B, 0x85,
1649                    0x0, 0x0, 0x0, 0x0 };
1650                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1651                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1652                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                    [m_vCode.size() - 1].size());
1653            }
```

```
1654            break;
1655            }
1656        }
1657        void set_rdx(const int32 qIndex, const BVType bvtType) {
1658            switch (bvtType)
1659            {
1660            case BVT_BYTE:
1661            {
1662                m_vCode.resize(m_vCode.size() + 1);
1663                BYTE aCode[] = { 0x48, 0x0F, 0xBE, 0x95,
1664                    0x0, 0x0, 0x0, 0x0 };
1665                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1666                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1667                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1668            }
1669            break;
1670            case BVT_WORD:
1671            {
1672                m_vCode.resize(m_vCode.size() + 1);
1673                BYTE aCode[] = { 0x48, 0x0F, 0xBF, 0x95,
1674                    0x0, 0x0, 0x0, 0x0 };
1675                memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1676                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1677                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1678            }
1679            break;
1680            case BVT_DWORD:
1681            {
1682                //48 63 04 25 22220200
1683                m_vCode.resize(m_vCode.size() + 1);
1684                BYTE aCode[] = { 0x48, 0x63, 0x95,
1685                    0x0, 0x0, 0x0, 0x0 };
1686                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1687                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1688                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1689            }
1690            break;
1691            case BVT_QWORD:
1692            {
1693                m_vCode.resize(m_vCode.size() + 1);
1694                BYTE aCode[] = { 0x48, 0x8B, 0x95,
1695                    0x0, 0x0, 0x0, 0x0 };
1696                memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1697                m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1698                memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                      [m_vCode.size() - 1].size());
1699            }
1700            break;
1701            }
1702        }
```

```
1703      void set_tmp_float(const int32 qIndex, const BVType bvtType) {
1704
1705          switch (bvtType)
1706          {
1707          case BVT_DWORD:
1708          {
1709              m_vCode.resize(m_vCode.size() + 1);
1710              BYTE aCode[] = { 0xF3, 0x0F, 0x10, 0x85, //parser.main+D - 8B 85 ABAAFAFF
1711                  0x0, 0x0, 0x0, 0x0 };
1712              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1713              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1714              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode[m_vCode.size() - 1].size());
1715          }
1716          break;
1717          case BVT_QWORD:
1718          {
1719              m_vCode.resize(m_vCode.size() + 1);
1720              BYTE aCode[] = { 0xF2, 0x0F, 0x10, 0x85, //parser.main+D - 8B 85 ABAAFAFF
1721                  0x0, 0x0, 0x0, 0x0 };
1722              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1723              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1724              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, sizeof(aCode));
1725          }
1726          break;
1727          }
1728      }
1729      void get_tmp_float(const int32 qIndex, const BVType bvtType) {
1730
1731          switch (bvtType)
1732          {
1733          case BVT_DWORD:
1734          {
1735              m_vCode.resize(m_vCode.size() + 1);
1736              BYTE aCode[] = { 0xF3, 0x0F, 0x11, 0x85,
1737                  0x0, 0x0, 0x0, 0x0 };
1738              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1739              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1740              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode[m_vCode.size() - 1].size());
1741          }
1742          break;
1743          case BVT_QWORD:
1744          {
1745              m_vCode.resize(m_vCode.size() + 1);
1746              BYTE aCode[] = { 0xF2, 0x0F, 0x11, 0x85,
1747                  0x0, 0x0, 0x0, 0x0 };
1748              memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1749              m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1750              memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
```

```
                [m_vCode.size() - 1].size());
1751        }
1752        break;
1753        }
1754    }
1755    void float_to_int(const int32 qIndex, const int32 qIndex2, const
          BVType bvtType, const BVType floatBvtType) { //i1 - float, i2 -
          int
1756        //00007FF7FC0E4252 F3 0F 2C 85 24 02 00 00 cvttss2si
              eax,dword ptr [x]
1757
1758        switch (floatBvtType)
1759        {
1760        case BVT_DWORD:
1761        {
1762            m_vCode.resize(m_vCode.size() + 1);
1763            BYTE aCode[] = { 0xF3, 0x0F, 0x2C, 0x85, //parser.main -
              8A 85 ABAAFAFF
1764                0x0, 0x0, 0x0, 0x0 };
1765            memcpy(&aCode[4], &qIndex, sizeof(qIndex));
1766            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1767            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
1768        }
1769        break;
1770        case BVT_QWORD:
1771        {
1772            m_vCode.resize(m_vCode.size() + 1);
1773            BYTE aCode[] = { 0xF2, 0x48, 0x0F, 0x2C, 0x85, //
              parser.main - 8A 85 ABAAFAFF
1774                0x0, 0x0, 0x0, 0x0 };
1775            memcpy(&aCode[5], &qIndex, sizeof(qIndex));
1776            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1777            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
1778        }
1779        break;
1780        }
1781
1782        this->get_tmp(qIndex2, bvtType);
1783    }
1784    void int_to_float(const int32 qIndex, const int32 qIndex2, const
          BVType bvtType, const BVType floatBvtType) { //i1 - int, i2 -
          float
1785        //F3 0F 2A C0          cvtsi2ss    xmm0,eax
1786
1787        this->set_tmp(qIndex, bvtType);
1788
1789        switch (floatBvtType)
1790        {
1791        case BVT_DWORD:
1792        {
1793            m_vCode.resize(m_vCode.size() + 1);
```

```
1794            BYTE aCode[] = { 0xF3, 0x0F, 0x2A, 0xC0 //parser.main - 8A
                   85 ABAAFAFF
1795            };
1796            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1797            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1798        }
1799        break;
1800        case BVT_QWORD:
1801        {
1802            m_vCode.resize(m_vCode.size() + 1);
1803            BYTE aCode[] = { 0xF2, 0x48, 0x0F, 0x2A, 0xC0
1804            };
1805            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1806            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1807        }
1808        break;
1809        }
1810
1811        this->get_tmp_float(qIndex2, floatBvtType);
1812    }
1813    void clear_rax() {
1814        //parser.main - 48 B8 0000000000000000
1815        m_vCode.resize(m_vCode.size() + 1);
1816        BYTE aCode[] = { 0x48, 0xB8, 0,0,0,0,0,0,0,0
1817        };
1818        m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1819        memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
              [m_vCode.size() - 1].size());
1820    }
1821    void set_tmp(const int32 qIndex, const BVType bvtType) {
1822        this->clear_rax();
1823        switch (bvtType)
1824        {
1825        case BVT_BYTE:
1826        {
1827            m_vCode.resize(m_vCode.size() + 1);
1828            BYTE aCode[] = { 0x8A, 0x85, //parser.main - 8A 85
                  ABAAFAFF
1829                0x0, 0x0, 0x0, 0x0 };
1830            memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1831            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1832            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1833        }
1834        break;
1835        case BVT_WORD:
1836        {
1837            m_vCode.resize(m_vCode.size() + 1);
1838            BYTE aCode[] = { 0x66, 0x8B, 0x85, //parser.main+6 - 66 8B
                   85 ABAAFAFF
1839                0x0, 0x0, 0x0, 0x0 };
```

```
1840            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1841            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1842            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1843        }
1844        break;
1845        case BVT_DWORD:
1846        {
1847            //48 63 04 25 22220200
1848            m_vCode.resize(m_vCode.size() + 1);
1849            BYTE aCode[] = { 0x8B, 0x85, //parser.main+D - 8B 85
                  ABAAFAFF
1850                0x0, 0x0, 0x0, 0x0 };
1851            memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1852            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1853            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1854        }
1855        break;
1856        case BVT_QWORD:
1857        {
1858            m_vCode.resize(m_vCode.size() + 1);
1859            BYTE aCode[] = { 0x48, 0x8B, 0x85, //parser.main+13 - 48
                  8B 85 ABAAFAFF
1860                0x0, 0x0, 0x0, 0x0 };
1861            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1862            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1863            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1864        }
1865        break;
1866        }
1867    }
1868    void get_tmp(const int32 qIndex, const BVType bvtType) {
1869        switch (bvtType)
1870        {
1871        case BVT_BYTE:
1872        {
1873            m_vCode.resize(m_vCode.size() + 1);
1874            BYTE aCode[] = { 0x88, 0x85, //parser.main - 8A 85
                  ABAAFAFF
1875                0x0, 0x0, 0x0, 0x0 };
1876            memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1877            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1878            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1879        }
1880        break;
1881        case BVT_WORD:
1882        {
1883            m_vCode.resize(m_vCode.size() + 1);
1884            BYTE aCode[] = { 0x66, 0x89, 0x85, //parser.main+6 - 66 8B
                   85 ABAAFAFF
```

```
1885                0x0, 0x0, 0x0, 0x0 };
1886            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1887            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1888            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1889        }
1890        break;
1891        case BVT_DWORD:
1892        {
1893            //48 63 04 25 22220200
1894            m_vCode.resize(m_vCode.size() + 1);
1895            BYTE aCode[] = { 0x89, 0x85, //parser.main+D - 8B 85
                  ABAAFAFF
1896                0x0, 0x0, 0x0, 0x0 };
1897            memcpy(&aCode[2], &qIndex, sizeof(qIndex));
1898            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1899            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1900        }
1901        break;
1902        case BVT_QWORD:
1903        {
1904            m_vCode.resize(m_vCode.size() + 1);
1905            BYTE aCode[] = { 0x48, 0x89, 0x85, //parser.main+13 - 48
                  8B 85 ABAAFAFF
1906                0x0, 0x0, 0x0, 0x0 };
1907            memcpy(&aCode[3], &qIndex, sizeof(qIndex));
1908            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1909            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1910        }
1911        break;
1912        }
1913    }
1914    void start_func(const int32 dVarsOffset, const int32 dTmpOffset)
        { //start function
1915        m_vCode.resize(m_vCode.size() + 1);
1916        BYTE aCode[] = {
1917            0x40, 0x55,
1918            0x57,
1919            0x48, 0x81, 0xEC, 0x0, 0x0, 0x0, 0x00,
1920            0x48, 0x8D, 0xAC, 0x24, 0x0, 0x0, 0x0, 0x0,
1921            0x48, 0x8D, 0xBC, 0x24, 0x0, 0x0, 0x0, 0x0,
1922            //0x48, 0x89, 0xE5,
1923            //0x41, 0x51,//
1924        };
1925
1926        DWORD dRspOffset = dVarsOffset + dTmpOffset;
1927
1928        //int32 dFixed = dFuncOffset - 32;
1929        memcpy(&aCode[6], &dRspOffset, sizeof(dRspOffset));
1930        memcpy(&aCode[14], &dVarsOffset, sizeof(dVarsOffset));
1931        memcpy(&aCode[22], &dVarsOffset, sizeof(dVarsOffset));
```

```
1932            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1933            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1934
1935        }
1936        void end_func(const int32 dVarsOffset, const int32 dTmpOffset)
              { //end function
1937            m_vCode.resize(m_vCode.size() + 1);
1938            BYTE aCode[] = {
1939                0x48, 0x8D, 0xA5, 0x0, 0x0, 0x0, 0x0,
1940                0x5F,
1941                0x5D,
1942                //0x41, 0x59, //
1943                0xC3 };
1944
1945
1946            int32 dFixed = (dVarsOffset + dTmpOffset) - dVarsOffset;
1947            memcpy(&aCode[3], &dFixed, sizeof(dFixed));
1948            m_vCode[m_vCode.size() - 1].resize(sizeof(aCode));
1949            memcpy(m_vCode[m_vCode.size() - 1].data(), aCode, m_vCode
                  [m_vCode.size() - 1].size());
1950        }
1951    };
```