Simulator.h Source :

```
#ifndef _SIMULATOR_H_
#define _SIMULATOR_H_
#include "MemoryManageUnit.h"
#include "Control.h"
#include "Loader.h"
#include "TabProcessing.h"
#include "ExecutionUnit.h"
#include "resource.h"
#include "Richedit.h"
#include <windows.h>
#include <stdio.h>
// 프로시저 선언
LRESULT CALLBACK WndProc(HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM lParam);
void Execute ( ObjectCode *pObject );
void GetMemoryInfo ( char *strBuffer, int nProgramLen );
void Macro ( char *pStrFileName );
// Define선언
#define ID_INDIRECT_CHECKBOX                    101
#define ID_IMMEDIATE_CHECKBOX                   102
#define ID_INDEX_CHECKBOX                       103
#define ID_BASE_CHECKBOX                        104
#define ID_PC_CHECKBOX                          105
#define ID_EXTENT_CHECKBOX                      106
#define ID_A_REGISTER_EDITBOX                   201
#define ID_X_REGISTER_EDITBOX                   202
#define ID_L_REGISTER_EDITBOX                   203
#define ID_PC_REGISTER_EDITBOX                  204
#define ID_SW_REGISTER_EDITBOX                  205
#define ID_B_REGISTER_EDITBOX                   206
#define ID_S_REGISTER_EDITBOX                   207
#define ID_T_REGISTER_EDITBOX                   208
#define ID_OPCODE_EDITBOX                       209
#define ID_TARGETADDR_EDITBOX                   210
#define ID_MEMORY_EDITBOX                       211
#define ID_OBJECTCODE_LISTBOX                   301
#define ID_TREE_CONTROLSECTION_LABEL            401
#define ID_EDIT_MEMORY_LABEL                    402
#define ID_LIST_OBJECTCODE_LABEL                403
#define ID_EDIT_A_REGISTER_LABEL                404
#define ID_EDIT_X_REGISTER_LABEL                405
#define ID_EDIT_L_REGISTER_LABEL                406
#define ID_EDIT_PC_REGISTER_LABEL               407
#define ID_EDIT_SW_REGISTER_LABEL               408
#define ID_EDIT_B_REGISTER_LABEL                409
#define ID_EDIT_S_REGISTER_LABEL                410
#define ID_EDIT_T_REGISTER_LABEL                411
#define ID_EDIT_OPCODE_LABEL                    412
#define ID_EDIT_TARGETADDR_LABEL                413
#define ID_NIXBPE_LABEL                         414
#define ID_EXECUTE_BUTTON                       501
#define ID_ONESTEP_BUTTON                       502
#define MEMORY_SIZE                             1024*1024
#define START_ADDRESS                           0
#endif
```

Simulator.cpp Source :

```
#include "Simulator.h"

////////////////////////////////////////////////////////////////////////
// 전역 변수
ESTAB    *pESTAB_Header, *pESTAB_Tail;
OPTAB    *pOPTAB_Header, *pOPTAB_Tail;
char     *MemoryLocation, *strBuffer;
int      nProgramLen, L_Register, A_Register, T_Register, X_Register, SW_Register;
int      B_Register, PC_Register, S_Register, F_Register, CSADDR, nCheckLoad = 0, nObjectNum;
FILE              *fp;
ObjectCode        *AllObject[100];
LPCTSTR  lpszClass = TEXT ( "Simulator" );
HINSTANCE         g_hInst;

////////////////////////////////////////////////////////////////////////
// WinMain
// 기능 : 윈도우 속성 및 크기를 결정하고 생성해준다.
int APIENTRY WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstace, LPSTR lpszCmdParam, int nCmdShow )
{
        HWND                        hWnd;
        WNDCLASS            WndClass;
        MSG                         Message;
        HMODULE             hMod;

        // 리치 에디트가 정의되어 있는 DLL 로드한다.
        hMod = LoadLibrary ( "Riched20.dll" );

        g_hInst = hInstance;

        // 윈도우 속성을 설정한다.
        WndClass.cbClsExtra = 0;
        WndClass.cbWndExtra = 0;
        WndClass.hbrBackground = (HBRUSH) GetStockObject ( WHITE_BRUSH );
        WndClass.hCursor = LoadCursor ( NULL,IDC_ARROW );
        WndClass.hIcon = LoadIcon ( NULL, IDI_APPLICATION );
        WndClass.hInstance = hInstance;
        WndClass.lpfnWndProc = (WNDPROC) WndProc;
        WndClass.lpszClassName = lpszClass;
        WndClass.lpszMenuName=MAKEINTRESOURCE(IDR_MENU1);
        WndClass.style = CS_HREDRAW | CS_VREDRAW;
        RegisterClass ( &WndClass );

        // 메인 윈도우 생성
        hWnd = CreateWindow ( lpszClass, lpszClass, WS_SYSMENU,
                  200, 80, 800, 600,
                  NULL, ( HMENU ) NULL, hInstance, NULL );
        ShowWindow ( hWnd, nCmdShow );

        while ( GetMessage ( &Message, 0, 0, 0 ) ) {
                  TranslateMessage ( &Message );
                  DispatchMessage ( &Message );
        }

        // 로드 했던 DLL을 해제 한다.
        FreeLibrary ( hMod );

        return Message.wParam;
}

////////////////////////////////////////////////////////////////////////
// WndProc 콜백 함수
// 기능 : 메인 윈도우의 메시지를 처리해준다.
LRESULT CALLBACK WndProc(HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM lParam)
{
        char                strControlSectionInfo[30], strObject[30], lpstrFile[MAX_PATH] ="";
        ESTAB               *pESTAB_Temp;
        OPTAB               *pSearchReturnOPTAB;
        static Tree         *treeControlSection;
        static List         *listObjectCode;
        static CheckBox     *checkNIXBPE[6];
        static Edit         *editRegister[6], *editOPCODE, *editTargetAddress, *editMemory;
        GroupBox  *groupboxNIXBPE;
```

```
	Button		*buttonExecute, *buttonOneStep;
	Label		*labelTreeControl, *labelListObjectCode, *labelEditRegister[6];
	Label		*labelOPCODE, *labelTargetAddress, *labelMemory;
	int		loop, loop1;
	FILE		*pObjectFile;
	HTREEITEM hFather;
	OPENFILENAME	OFN;
	CHARFORMAT2	cf;

	switch(iMessage)
	{
	case WM_CREATE :
		// 컨트롤 섹션 정보를 타나낼 Tree Control을 생성한다.
		labelTreeControl = new Label ( hWnd, ID_TREE_CONTROLSECTION_LABEL,
					10, 10, 100, 20, "ControlSection" );
		labelTreeControl->Create ( );
		treeControlSection = new Tree ( hWnd, 5, 30, 500, 150 );
		treeControlSection->Create ( );

		// 메모리 정보를 나타낼 Edit 박스 생성
		labelMemory = new Label ( hWnd, ID_EDIT_MEMORY_LABEL,
				10, 180, 70, 20, "MEMORY" );
		labelMemory->Create ( );
		editMemory = new Edit ( hWnd, ID_MEMORY_EDITBOX, 5, 200, 500, 130,
		    WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL | ES_READONLY | ES_MULTILINE );
		editMemory->CreateRichedit ( );

		// Richedit 문자 속성 정하는 부분
		memset ( &cf, 0, sizeof ( CHARFORMAT2 ) );
		cf.cbSize = sizeof ( CHARFORMAT2 );
		cf.dwMask = CFM_COLOR | CFM_FACE | CFM_SIZE;
		cf.crTextColor = RGB ( 0, 0, 0 );
		cf.yHeight = 250;
		strcpy ( cf.szFaceName, "바탕체" );
		editMemory->SetCharFormat ( cf );

		// Object 코드들을 나타낼 리스트 박스 생성
		labelListObjectCode = new Label ( hWnd, ID_LIST_OBJECTCODE_LABEL,
					520, 10, 80, 20, "ObjectCode" );
		labelListObjectCode->Create ( );
		listObjectCode = new List ( hWnd, ID_OBJECTCODE_LISTBOX, 515, 30, 270, 400 );
		listObjectCode->Create ( );

		// NIXBPE 모드를 나타낼 체크 박스 생성
		groupboxNIXBPE = new GroupBox ( hWnd, 0, 30, 340, 400, 90, "NIXBPE Flag" );
		groupboxNIXBPE->Create ( );
		checkNIXBPE[0] = new CheckBox ( hWnd, ID_INDIRECT_CHECKBOX,
				50, 370, 90, 20, "INDIRECT" );
		checkNIXBPE[0]->Create ( );
		checkNIXBPE[1] = new CheckBox ( hWnd, ID_IMMEDIATE_CHECKBOX,
				180, 370, 110, 20, "IMMEDDIATE" );
		checkNIXBPE[1]->Create ( );
		checkNIXBPE[2] = new CheckBox ( hWnd, ID_INDEX_CHECKBOX, 335, 370, 65, 20, "INDEX" );
		checkNIXBPE[2]->Create ( );
		checkNIXBPE[3] = new CheckBox ( hWnd, ID_BASE_CHECKBOX, 50, 390, 60, 20, "BASE" );
		checkNIXBPE[3]->Create ( );
		checkNIXBPE[4] = new CheckBox ( hWnd, ID_PC_CHECKBOX, 180, 390, 40, 20, "PC" );
		checkNIXBPE[4]->Create ( );
		checkNIXBPE[5] = new CheckBox ( hWnd, ID_EXTENT_CHECKBOX, 335, 390, 75, 20, "EXTENT" );
		checkNIXBPE[5]->Create ( );


		// 레지스터 값들을 나타내줄 Edit 박스 생성
		labelEditRegister[0] = new Label ( hWnd, ID_EDIT_A_REGISTER_LABEL,
					40, 440, 80, 20, "A Register : " );
		labelEditRegister[0]->Create ( );
		editRegister[0] = new Edit ( hWnd, ID_A_REGISTER_EDITBOX, 130, 440, 100, 20 );
		editRegister[0]->Create ( );

		labelEditRegister[1] = new Label ( hWnd, ID_EDIT_B_REGISTER_LABEL,
					290, 440, 80, 20, "B Register : " );
		labelEditRegister[1]->Create ( );
		editRegister[1] = new Edit ( hWnd, ID_B_REGISTER_EDITBOX, 380, 440, 100, 20 );
		editRegister[1]->Create ( );
```

```
                labelEditRegister[2] = new Label ( hWnd, ID_EDIT_X_REGISTER_LABEL,
                                                   40, 480, 80, 20, "X Register : " );
                labelEditRegister[2]->Create ( );
                editRegister[2] = new Edit ( hWnd, ID_X_REGISTER_EDITBOX, 130, 480, 100, 20 );
                editRegister[2]->Create ( );

                labelEditRegister[3] = new Label ( hWnd, ID_EDIT_PC_REGISTER_LABEL,
                                                   282, 480, 90, 20, "PC Register : " );
                labelEditRegister[3]->Create ( );
                editRegister[3] = new Edit ( hWnd, ID_PC_REGISTER_EDITBOX, 380, 480, 100, 20 );
                editRegister[3]->Create ( );

                labelEditRegister[4] = new Label ( hWnd, ID_EDIT_L_REGISTER_LABEL,
                                                   40, 520, 80, 20, "L Register : " );
                labelEditRegister[4]->Create ( );
                editRegister[4] = new Edit ( hWnd, ID_L_REGISTER_EDITBOX, 130, 520, 100, 20 );
                editRegister[4]->Create ( );

                labelEditRegister[5] = new Label ( hWnd, ID_EDIT_SW_REGISTER_LABEL,
                                                   278, 520, 90, 20, "SW Register : " );
                labelEditRegister[5]->Create ( );
                editRegister[5] = new Edit ( hWnd, ID_SW_REGISTER_EDITBOX, 380, 520, 100, 20 );
                editRegister[5]->Create ( );

                labelEditRegister[6] = new Label ( hWnd, ID_EDIT_S_REGISTER_LABEL,
                                                   40, 530, 80, 20, "S Register : " );
                labelEditRegister[6]->Create ( );
                editRegister[6] = new Edit ( hWnd, ID_S_REGISTER_EDITBOX, 130, 530, 100, 20 );
                editRegister[6]->Create ( );

                labelEditRegister[7] = new Label ( hWnd, ID_EDIT_T_REGISTER_LABEL,
                                                   293, 530, 80, 20, "T Register : " );
                labelEditRegister[7]->Create ( );
                editRegister[7] = new Edit ( hWnd, ID_T_REGISTER_EDITBOX, 380, 530, 100, 20 );
                editRegister[7]->Create ( );


                // OPCDOE값을 나타내줄 Edit 박스 생성
                labelOPCODE = new Label ( hWnd, ID_EDIT_OPCODE_LABEL, 570, 430, 70, 20, "OPCODE : " );
                labelOPCODE->Create ( );
                editOPCODE = new Edit ( hWnd, ID_OPCODE_EDITBOX, 645, 430, 140, 20 );
                editOPCODE->Create ( );

                // Target Address를 나타내 줄 Edit 박스 생성
                labelTargetAddress = new Label ( hWnd, ID_EDIT_TARGETADDR_LABEL,
                                                 528, 460, 110, 20, "Target Address : " );
                labelTargetAddress->Create ( );
                editTargetAddress = new Edit ( hWnd, ID_TARGETADDR_EDITBOX, 645, 460, 140, 20 );
                editTargetAddress->Create ( );

                // 샐행 Button 생성
                buttonExecute = new Button ( hWnd, ID_EXECUTE_BUTTON, 515, 500, 130, 40, "Execution" );
                buttonExecute->Create ( );
                buttonOneStep = new Button ( hWnd, ID_ONESTEP_BUTTON, 655, 500, 130, 40, "One Step" );
                buttonOneStep->Create ( );
                return 0;
        case WM_COMMAND:
                switch ( LOWORD ( wParam ) )
                {
                case ID_MACRO :
                        // 매크로 메뉴의 실행 메뉴를 눌렀을때 처리 하는 부분
                        memset ( &OFN, 0, sizeof ( OPENFILENAME ) );
                        OFN.lStructSize = sizeof ( OPENFILENAME );
                        OFN.hwndOwner = hWnd;
                        OFN.lpstrFilter = "SIC/XE Obj파일₩0*.obj₩0'";
                        OFN.lpstrFile = lpstrFile;
                        OFN.Flags=OFN_NOCHANGEDIR;//여기
                        OFN.nMaxFile = MAX_PATH;
                        if ( GetOpenFileName ( &OFN ) != 0 )
                        {
                                // 매크로 처리 할 부분
                                // [Load Time : *]
                                Macro ( (char *)OFN.lpstrFile );
                        }
                        return 0;
```

```
		case ID_OPEN :
			// 메뉴에서 OPEN메뉴 눌렀을때 처리
			memset ( &OFN, 0, sizeof ( OPENFILENAME ) );
			OFN.lStructSize = sizeof ( OPENFILENAME );
			OFN.hwndOwner = hWnd;
			OFN.lpstrFilter = "SIC/XE Obj파일WO*.objWO'";
			OFN.lpstrFile = lpstrFile;
			OFN.Flags=OFN_NOCHANGEDIR;//여기
			OFN.nMaxFile = MAX_PATH;
			if ( GetOpenFileName ( &OFN ) != 0 )
			{
				// Object 파일이 로드되었다는 것을 체크
				nCheckLoad = 1;
				// 윈도우창에 현재 연 파일이름을 출력 해준다.
				SetWindowText ( hWnd, OFN.lpstrFile );
				// SIC/XE Machine 메모리를 할당하는 부분
				MemoryLocation = new char [MEMORY_SIZE];
				strBuffer = new char [2*MEMORY_SIZE];

				// 변수 초기화
				CSADDR = START_ADDRESS;
				PC_Register = START_ADDRESS;
				// OPTAB을 초기화 한다
				pOPTAB_Header->InitOpcodeTable ( );

				// 오브젝트 파일을 읽어서 메모리에 올리는 부분
				pObjectFile = fopen ( OFN.lpstrFile, "r" );
				if ( pObjectFile == NULL )
				{
					PostQuitMessage(0);
					return 0;
				}
				nProgramLen = Loader1 ( pObjectFile );
				Loader2 ( pObjectFile );
				fclose ( pObjectFile );

				// 메모리에서 Object를 가져온다
				GetMemoryInfo ( strBuffer, nProgramLen );
				editMemory->SetText ( strBuffer );
				nObjectNum = LoadAllObject ( AllObject, nProgramLen );

				// 컨트롤 섹션 정보를 타나내는 Tree 초기화
				pESTAB_Temp = pESTAB_Header;
				while ( pESTAB_Temp != NULL )
				{
					if ( pESTAB_Temp->GetLength ( )		!= 0 )
					{
						hFather = treeControlSection->Insert ( 0,
							pESTAB_Temp->GetName ( ) );
						sprintf ( strControlSectionInfo,
							"Start Address : %d",
							pESTAB_Temp->GetAddr ( ) );
						treeControlSection->Insert ( hFather,
								strControlSectionInfo );
						sprintf ( strControlSectionInfo,
							"ControlSection Length : %d",
							pESTAB_Temp->GetLength ( ) );
						treeControlSection->Insert ( hFather,
								strControlSectionInfo );
						treeControlSection->Expand ( hFather );
					}
					pESTAB_Temp = pESTAB_Temp->GetNext ( );
				}

				// Object 코드들을 나타낼 List 초기화
				for ( loop = 0 ; loop < nObjectNum ; loop++ )
				{
					sprintf ( strObject, " %04X : %s",
						AllObject[loop]->GetPC ( ),
					  AllObject[loop]->GetStrObjectTwoBytePointer ( ) );
					listObjectCode->Insert ( strObject );
				}

			}
```

```
                return 0;
        case ID_EXIT :
                // 메뉴에서 EXIT메뉴 눌렀을때 처리
                PostQuitMessage(0);
                return 0;
        case ID_EXECUTE_BUTTON :
                // Execution 버튼을 눌렀을때 한번에 끝까지 실행되도록 처리

                if ( nCheckLoad == 0 )
                {
                        return 0;
                }
                else
                {
                        while ( 1 )
                        {
                                // 메모리에서 PC_Register가 가리키는 Instruction을 가져온다.
                                nResult = OneObject->GetObjectCode
                                        ( MemoryLocation + PC_Register, nProgramLen );
                                if ( nResult == -1 )
                                {
                                        MessageBox ( hWnd, "PC_Register Wrong Value!!",
                                                "SIC/XE", MB_OK );
                                        return 0;
                                }
                                else
                                {
                                        // Exectuion한다.
                                        Execute ( OneObject );

                                        // 지금 로드하는 오브젝트들도 서브루틴이기 때문에...
                                             종료 지점을 서브루틴의 끝으로 강제정의
                                        if ( PC_Register == 39 )
                                        {
                                                // Memory Edit 창을 갱신한다
                                                GetMemoryInfo ( strBuffer, nProgramLen );
                                                editMemory->SetText ( strBuffer );
                                                MessageBox ( hWnd, "프로그램이 끝났습니다",
                                                        "SIC/XE", MB_OK );
                                                nOffsetOfRD = nOffsetOfWD = 0;
                                                PC_Register = START_ADDRESS;
                                                return 0;
                                        }

                                }
                        }
                }
                return 0;
        case ID_ONESTEP_BUTTON :
                // 실행 버튼을 눌렀을때 처리
                if ( nCheckLoad == 0 )
                {
                        return 0;
                }
                else
                {
                        // 메모리에서 PC_Register가 가리키는 Instruction을 가져온다.
                        nResult = OneObject->GetObjectCode ( MemoryLocation + PC_Register, nProgramLen );
                        if ( nResult == -1 )
                        {
                                MessageBox ( hWnd, "PC_Register Wrong Value!!",  "SIC/XE", MB_OK );
                                return 0;
                        }
                        else
                        {
                                // 실행한 OPCODE문자열을 리스트에서 선택해주도록 한다
                                loop = 0;
                                while ( loop < nObjectNum )
                                {
                                        if ( AllObject[loop]->GetPC ( )  == PC_Register )
                                        {
                                                listObjectCode->SetSelect ( loop );
                                                break;
                                        }
                                        loop++;
```

```
                                        }

                                        // Exectuion한다.
                                        Execute ( OneObject );

                                        // 변화된 Register 값을 화면에 출력 해 준다.
                                        editRegister[0]->SetText ( A_Register );
                                        editRegister[1]->SetText ( B_Register );
                                        editRegister[2]->SetText ( X_Register );
                                        editRegister[3]->SetText ( PC_Register );
                                        editRegister[4]->SetText ( L_Register );
                                        editRegister[5]->SetText ( SW_Register );
                                        editRegister[6]->SetText ( S_Register );
                                        editRegister[7]->SetText ( T_Register );
                                        editTargetAddress->SetText ( OneObject->GetTargetAddr ( ) );

                                        // 실행한 OPCODE문자열을 구해서 Edit창에 뿌려준다.
                                        pSearchReturnOPTAB = pOPTAB_Header->SearchByValue
                                                                ( OneObject->GetOPCODE ( ) );
                                        editOPCODE->SetText ( pSearchReturnOPTAB->GetName ( ) );

                                        // NIXBPE 체크 박스를 갱신한다.
                                        for ( loop1 = 0 ; loop1 < 6 ; loop1++ )
                                        {
                                                checkNIXBPE[loop1]->SetCheck
                                                        ( OneObject->GetNIXBPE ( loop1 ) );
                                        }

                                        // Memory Edit 창을 갱신한다
                                        GetMemoryInfo ( strBuffer, nProgramLen );
                                        editMemory->SetText ( strBuffer );

                                        // 지금 로드하는 오브젝트들도 서브루틴이기 때문에...
                                              종료 지점을 서브루틴의 끝으로 강제정의
                                        if ( PC_Register == 39 )
                                        {
                                                MessageBox ( hWnd, "프로그램이 끝났습니다",
                                                        "SIC/XE", MB_OK );
                                                nOffsetOfRD = nOffsetOfWD = 0;
                                                PC_Register = START_ADDRESS;
                                                return 0;
                                        }
                                }
                        }
                        break;
                }
                return 0;
        case WM_DESTROY:
                PostQuitMessage(0);
                return 0;
        }
        return ( DefWindowProc ( hWnd, iMessage, wParam, lParam ) );
}

////////////////////////////////////////////////////////////////////////
// Execute 함수
// pObject : ObjectCode 객체를 입력 받는 인자
// 기능 : ObjectCode 객체를 받아서 객체 속에 OPOCDE를 보고 그에 따른 실행 함수 호출한다.
void Execute ( ObjectCode *pObject )
{
        char                *pOPCODE;
        OPTAB               *pSearchReturnOPTAB;

        // OPTAB에서 OPCODE 값으로 찾는다.
        pOPCODE = pObject->GetOPCODE ( );
        pSearchReturnOPTAB = pOPTAB_Header->SearchByValue ( pOPCODE );
        pOPCODE = pSearchReturnOPTAB->GetName ( );

        if ( strcmp ( pOPCODE, "STL" ) == 0 )
        {
                STL ( pObject );
        }
        else if ( strcmp ( pOPCODE, "STA" ) == 0 )
        {
                STA ( pObject );
```

```
        }
        else if ( strcmp ( pOPCODE, "STX" ) == 0 )
        {
                STX ( pObject );
        }
        else if ( strcmp ( pOPCODE, "LDA" ) == 0 )
        {
                LDA ( pObject );
        }
        else if ( strcmp ( pOPCODE, "LDT" ) == 0 )
        {
                LDT ( pObject );
        }
        else if ( strcmp ( pOPCODE, "J" ) == 0 )
        {
                J ( pObject );
        }
        else if ( strcmp ( pOPCODE, "JSUB" ) == 0 )
        {
                JSUB ( pObject );
        }
        else if ( strcmp ( pOPCODE, "JEQ" ) == 0 )
        {
                JEQ ( pObject );
        }
        else if ( strcmp ( pOPCODE, "JLT" ) == 0 )
        {
                JLT ( pObject );
        }
        else if ( strcmp ( pOPCODE, "COMP" ) == 0 )
        {
                COMP ( pObject );
        }
        else if ( strcmp ( pOPCODE, "COMPR" ) == 0 )
        {
                COMPR ( pObject );
        }
        else if ( strcmp ( pOPCODE, "CLEAR" ) == 0 )
        {
                CLEAR ( pObject );
        }
        else if ( strcmp ( pOPCODE, "TD" ) == 0 )
        {
                TD ( pObject );
        }
        else if ( strcmp ( pOPCODE, "WD" ) == 0 )
        {
                WD ( pObject );
        }
        else if ( strcmp ( pOPCODE, "RD" ) == 0 )
        {
                RD ( pObject );
        }
        else if ( strcmp ( pOPCODE, "LDCH" ) == 0 )
        {
                LDCH ( pObject );
        }
        else if ( strcmp ( pOPCODE, "STCH" ) == 0 )
        {
                STCH ( pObject );
        }
        else if ( strcmp ( pOPCODE, "TIXR" ) == 0 )
        {
                TIXR ( pObject );
        }
        else if ( strcmp ( pOPCODE, "RSUB" ) == 0 )
        {
                RSUB ( pObject );
        }
        else
        {
                // 구현 안된 OPCODE 처리...
        }
}
```

```
////////////////////////////////////////////////////////////////////////
// Macro 함수
// pStrFileName : Macro를 할 파일의 이름을 입력 받는 인자
// 기능 : Object파일을 읽어서 Macro 표시가 되어 있는 곳에 Macro를 적용함
void Macro ( char *pStrFileName )
{
        FILE                *pObjectFile;
        char                strTime[20];
        char                cChar;
        SYSTEMTIME          tm;
        GetLocalTime ( &tm );
        pObjectFile = fopen ( pStrFileName, "r+" );
        if ( pObjectFile == NULL )
        {
                // 파일 오픈이 실패했을때 에러 처리
        }

        while ( fread ( &cChar, sizeof ( char ), 1, pObjectFile ) != 0 )
        {
                // Macro 시작 마크인지 비교해서 Macro 표시 줄인지 확인한다.
                if ( cChar == '[' )
                {
                        // Macro를 덮어 쓸 부분까지 스트링 포인터를 이동한다.
                        fread ( &cChar, sizeof ( char ), 1, pObjectFile );
                        while (  cChar != '*' )
                        {
                                fread ( &cChar, sizeof ( char ), 1, pObjectFile );
                        }
                        // Macro 내용부분을 strTime에 써 넣는다.
                        sprintf ( strTime, "%d/%d/%d/%d:%d]₩n",
                                tm.wYear, tm.wMonth, tm.wDay, tm.wHour, tm.wMinute );
                        // Macro 내용부분을 파일에 쓴다.
                        fseek ( pObjectFile, -1, SEEK_CUR );
                        fwrite ( strTime, sizeof ( char ), strlen ( strTime ), pObjectFile );
                        break;
                }
        }
        fclose ( pObjectFile );
}

////////////////////////////////////////////////////////////////////////
// GetMemoryInfo 함수
// strBuffer : 메모리 정보를 담을 버퍼 주소를 받아들이는 인자
// nProgramLen : 프로그램이 사용되는 메모리 길이를 받아들이는 인자
// 기능 : 메모리를 읽어와서 정렬된 모습으로 16진수 2바이트 형식으로 버퍼에 저장한다.
void GetMemoryInfo ( char *strBuffer, int nProgramLen )
{
        char      *pMemory, strOneByte[20], strTwoByte[10];
        int                 loop, loop1, offset = 0;

        pMemory = MemoryLocation;

        for ( loop = 0 ; loop < nProgramLen ; loop += 16 )
        {
                // 첫줄의 시작 부분에 시작 주소를 출력 해주는 부분
                sprintf ( strOneByte, "  %06X  :  ", loop );
                memcpy ( strBuffer + offset, strOneByte, strlen ( strOneByte ) );
                offset += strlen ( strOneByte );

                for ( loop1 = 0 ; loop1 < 4 ; loop1++ )
                {
                        // 16진수 8글자 씩 4번 출력 해준다.
                        memcpy ( strOneByte, pMemory + loop + 4 * loop1, 4 );
                        OneByteToTwoByte ( strOneByte, strTwoByte, 4 );
                        memcpy ( strBuffer + offset, strTwoByte, 8 );
                        offset += 8;
                        strncpy ( strBuffer + offset, "  ", 2 );
                        offset += 2;
                }
                strncpy ( strBuffer + offset, "₩r₩n", 2 );
                offset += 2;
        }
        strBuffer[offset] = '₩0';
}
```

MemoryManageUnit.h Source :

```
#ifndef _MEMORYMANAGEUNIT_H_
#define _MEMORYMANAGEUNIT_H_

#include "TabProcessing.h"
#include <iostream>
#include <math.h>
using namespace std;

class ObjectCode
{
private :
        int                                     nNIXBPE[6];
        char                            strOP[3];
        int                                     nType;
        int                                     nDisp;
        int                                     nPC;
        unsigned char       *strObjectTwoByte;
        unsigned char       *strObjectOneByte;

public :
        ObjectCode ( );
        ~ObjectCode ( );
        void TwoByteToOneByte ( );
        void OneByteToTwoByte ( );
        int GetType ( );
        int GetDisp ( );
        int GetPC ( );
        int GetTargetAddr ( );
        int GetNIXBPE ( int nWhat );
        int GetObject ( char *strObj, int nLen );
        char *GetOPCODE ( );
        unsigned char *GetStrObjectTwoBytePointer ( );

};

extern char                 *MemoryLocation;
extern int                  CSADDR;

int HexToInt ( char *strHex, int nLength );
void IntToHex ( char *strHex, int nInt, int nLength );
void OneByteToTwoByte ( char *strOneByte, char *strTwoByte, int nLen );
int LoadAllObject ( ObjectCode *AllObject[], int nLen );

#define MAX_ONELINE                     70
#define INDIRECT            0
#define IMMEDIATE           1
#define INDEX                           2
#define BASE                            3
#define PC                                      4
#define EXTENT                          5

#endif
```

MemoryManageUnit.cpp Source :

```
#include "MemoryManageUnit.h"
////////////////////////////////////////////////////////////////////////////
// ObjectCode 생성자
// 기능 : Type 3에 맞게 메모리 할당 및 변수 생성
ObjectCode::ObjectCode ( )
{
        nType = 3;
        strObjectOneByte = new unsigned char[5];
        strObjectTwoByte = new unsigned char[9];
}


////////////////////////////////////////////////////////////////////////////
// ObjectCode 소멸자
ObjectCode::~ObjectCode ( )
{
        delete ( strObjectOneByte );
        delete ( strObjectTwoByte );
}

////////////////////////////////////////////////////////////////////////////
// GetObject 멤버 함수
// strObj : Object의 시작 주소를 받아들이는 인자
// 기능 : Object의 앞을 보고 타입을 판단하고 그에 맞게 멤버변수에 넣어준다.
int ObjectCode::GetObject ( char *strObj, int nLen )
{
        int             nTemp, nTemp1, loop, nOffset = 0;
        char    cTemp, strOpcode[3];
        OPTAB   *pSearchReturnOPTAB;

        while ( 1 )
        {
                if ( nOffset > nLen )
                {
                        return -1;
                }
                memset ( strObjectTwoByte, '₩0', 9 );
                memset ( strObjectOneByte, '₩0', 5 );

                // 메모리에서 일정 바이트를 얻어다가 two byte형식으로 바꾼다.
                strncpy ( (char *)strObjectOneByte, strObj + nOffset, 2 );
                OneByteToTwoByte ( );

                // OPCODE 부분만 임시 저장해 놓는다.
                strncpy ( strOpcode, (char *)strObjectTwoByte, 2 );
                strOpcode[2] = '₩0';

                // Object의 두번째 16진수 값으로 N, I를 판별한다.
                cTemp = strObjectTwoByte[1];
                if ( '0' <= cTemp && cTemp <= '9' )
                {
                        nTemp = cTemp - '0';
                }
                else
                {
                        nTemp = cTemp - 'A' + 10;
                }
                nTemp1 = nTemp;
                nNIXBPE[INDIRECT] = nNIXBPE[IMMEDIATE] = 0;
                for ( loop = 2 ; loop > 0 ; loop-- )
                {
                        nNIXBPE[loop-1] = nTemp % 2;
                        nTemp /= 2;
                }

                // Opcode 두번째 부분에서 N, I를 빼서 오리지널 Opcode만 구한다.
                nTemp1 -= ( 2 * nNIXBPE[INDIRECT] + nNIXBPE[IMMEDIATE] );
                if ( 0 <= nTemp1 && nTemp1 <= 9 )
                {
                        strOpcode[1] = nTemp1 + '0';
                }
                else
```

```
		{
			strOpcode[1] = nTemp1 - 10 + 'A';
		}

		// 위에서 OPCODE만 구한것으로 OPTABLE에서 찾아본다.
		if ( ( pSearchReturnOPTAB = pOPTAB_Header->SearchByValue ( strOpcode ) ) == NULL )
		{
			nOffset += 1;
			continue;
		}
		else
		{
			// Object의 세번째 16진수 값으로 X, B, P, E를 판별한다.
			cTemp = strObjectTwoByte[2];
			if ( '0' <= cTemp && cTemp <= '9' )
			{
				nTemp = cTemp - '0';
			}
			else
			{
				nTemp = cTemp - 'A' + 10;
			}
			nNIXBPE[INDEX] = nNIXBPE[BASE] = nNIXBPE[PC] = nNIXBPE[EXTENT] = 0;
			for ( loop = 4 ; loop > 0 ; loop-- )
			{
				nNIXBPE[loop+1] = nTemp % 2;
				nTemp /= 2;
			}

			switch ( pSearchReturnOPTAB->GetType ( ) )
			{
			case 1:
				if ( nNIXBPE[INDIRECT] == 0 && nNIXBPE[IMMEDIATE] == 0 )
				{
					nType = 1;
					strncpy ( strOP, strOpcode, 3 );
				}
				else
				{
					nOffset += 1;
					continue;
				}
				break;
			case 2:
				nType = 2;
				for ( loop = 0 ; loop < 6 ; loop++ )
				{
					nNIXBPE[loop] = 0;
				}
				strncpy ( strOP, strOpcode, 3 );
				break;
			case 3:
				if ( nNIXBPE[EXTENT] == 1 && ( nNIXBPE[BASE] == 1 || nNIXBPE[PC] == 1 ) )
				{
					nOffset += 1;
					continue;
				}
				else if ( nNIXBPE[EXTENT] == 1 &&
					( nNIXBPE[INDIRECT] == 1 || nNIXBPE[IMMEDIATE] == 1 ) )
				{
					nType = 4;
					strncpy ( strOP, strOpcode, 3 );
				}
				else if ( nNIXBPE[INDIRECT] == 0 && nNIXBPE[IMMEDIATE] == 0 )
				{
					nOffset += 1;
					continue;
				}
				else if ( nNIXBPE[BASE] == 1 && nNIXBPE[IMMEDIATE] == 1 )
				{
					nOffset += 1;
					continue;
				}
				else if ( nNIXBPE[INDIRECT] == 1 && nNIXBPE[INDEX] == 1 )
				{
```

```
                                        nOffset += 1;
                                        continue;
                                }
                                else
                                {
                                        // 나머지는 다 3형식으로 16진수 6개 멤버 변수에 복사
                                        nType = 3;
                                        strncpy ( strOP, strOpcode, 3 );
                                }
                                break;
                        }

                        // 위에서 정한 길이만큼
                        for ( loop = 0 ; loop < nType ; loop++ )
                        {
                                strObjectOneByte[loop] = (unsigned char) strObj[loop+nOffset];
                        }
                        OneByteToTwoByte ( );
                        nDisp = HexToInt ( (char *) strObjectTwoByte + 3, 2 * nType - 3 );
                        nPC = (int) ( strObj + nOffset ) - (int) MemoryLocation;
                        return nOffset + nType;
                }
        }
}

////////////////////////////////////////////////////////////////////////
// TwoByteToOneByte 멤버함수
// 기능 : strObjectTwo변수에 있는 값을 strObjectOne변수에 한바이트씩 넣는 기능
void ObjectCode::TwoByteToOneByte ( )
{
        int                             loop;

        // strObjectTwoByte를 두개씩 읽어서 strObjectOneByte에 넣어준다.
        for ( loop = 0 ; loop < nType ; loop++ )
        {
                strObjectOneByte[loop] = (unsigned char)HexToInt ( (char *)strObjectTwoByte + (loop*2), 2 );
        }
}

////////////////////////////////////////////////////////////////////////
// OneByteToTwoByte 멤버 함수
// 한 바이트로 팩킹한 strObjectOneByte를 strObjectTwoByte변수로 언팩킹하는 기능
void ObjectCode::OneByteToTwoByte ( )
{
        int                             loop;

        for ( loop = 0 ; loop < nType * 2 ; loop += 2 )
        {
                strObjectTwoByte[loop] = strObjectOneByte[loop/2] / 16;
                strObjectTwoByte[loop+1] = strObjectOneByte[loop/2] % 16;

                // strObjectTwoByte에서 첫번째 문자를 읽어 처리
                if ( 0 <= strObjectTwoByte[loop] && strObjectTwoByte[loop] <= 9 )
                {
                        strObjectTwoByte[loop] += '0';
                }
                else if ( 10 <= strObjectTwoByte[loop] && strObjectTwoByte[loop] <= 15 )
                {
                        strObjectTwoByte[loop] +=  ( 'A' - 10 );
                }

                // strObjectTwoByte에서 두번째 문자를 읽어 처리
                if ( 0 <= strObjectTwoByte[loop+1] && strObjectTwoByte[loop+1] <= 9 )
                {
                        strObjectTwoByte[loop+1] += '0';
                }
                else if ( 10 <= strObjectTwoByte[loop+1] && strObjectTwoByte[loop+1] <= 15 )
                {
                        strObjectTwoByte[loop+1] +=  ( 'A' - 10 );
                }
        }
        strObjectTwoByte[loop] = '₩0';
}

////////////////////////////////////////////////////////////////////////
```

```
// GetDisp 멤버 함수
// 기능 : 멤버 변수 nDisp값을 반환해준다.
int ObjectCode::GetDisp ( )
{
        return nDisp;
}

////////////////////////////////////////////////////////////////////////
// GetPC 멤버 함수
// 기능 : 멤버 변수 nPC값을 반환해준다.
int ObjectCode::GetPC ( )
{
        return nPC;
}

////////////////////////////////////////////////////////////////////////
// GetNIXBPE 멤버 함수
// nWhat : 원하는 N, I, X, B, P, E를 고를 값을 받아들이는 인자
// 기능 : 원하는 N, I, X, B, P, E 값을 리턴해준다.
int ObjectCode::GetNIXBPE ( int nWhat )
{
        return nNIXBPE[nWhat];
}

////////////////////////////////////////////////////////////////////////
// GetType 멤버 함수
// 기능 :  멤버 변수 nType 값을 리턴해준다.
int ObjectCode::GetType ( )
{
        return nType;
}

////////////////////////////////////////////////////////////////////////
// GetStrObjectTwoBytePointer 멤버 함수
// 기능 : 멤버 변수 strObjectTwoByte의 포인터를 리턴해준다.
unsigned char *ObjectCode::GetStrObjectTwoBytePointer ( )
{
        return strObjectTwoByte;
}

////////////////////////////////////////////////////////////////////////
// GetTargetAddr 멤버 함수
// 기능 : 멤버 변수 nDisp를 리턴해준다
int ObjectCode::GetTargetAddr ( )
{
        int                nAddr;

        if ( nNIXBPE[PC] == 1 )
        {
                nAddr = nDisp + nPC + 3;
        }
        else if ( nNIXBPE[EXTENT] == 1 || nNIXBPE[IMMEDIATE        ] )
        {
                nAddr = nDisp;
        }
        else
        {
                nAddr = nDisp;
        }
        return nAddr;
}

////////////////////////////////////////////////////////////////////////
// GetOPCODE 멤버 함수
// 기능 : 멤버변수 strOP의 포인터를 리턴해준다.
char *ObjectCode::GetOPCODE ( )
{
        return strOP;
}

////////////////////////////////////////////////////////////////////////
// HexToInt 함수
// strHex : 문자열 Hex값을 받아들이는 인자
// nLength : 문자열의 길이를 받아들이는 인자
// 기능 : 문자열 Hex값을 받아들여서 int형으로 반환해주는 기능
```

```
int HexToInt ( char *strHex, int nLength )
{
	int		nSum = 0, loop = 0;
	double	x = 16;

	if ( strHex[0] == 'F' )
	{
		nSum -= (int) ( pow ( x, ( nLength - 1 ) ) );
		loop = 1;
	}

	// nLength 만큼 루프를 돈다
	for (  ; loop < nLength ; loop++ )
	{
		if ( ( '0' <= strHex[loop] ) && ( strHex[loop] <= '9' ) )
		{
			// 만약 16진수가 0~9 사이의 숫자이면
			nSum += (int) ( pow ( x, (nLength - loop - 1) ) * ( strHex[loop] - '0' ) );
		}
		else if ( ( 'A' <= strHex[loop] ) && ( strHex[loop] <= 'F' ) )
		{
			// 만약 A~F사이의 문자이면
			nSum += (int) ( pow ( x, (nLength - loop - 1) ) * ( strHex[loop] - 'A' + 10 ) );
		}
	}

	// 10진수 결과값 반환
	return nSum;
}

////////////////////////////////////////////////////////////////////////
// IntToHex 함수
// strHex : 주어진 Int형 수를 16진수로 저장할 포인터를 받아들이는 인자
// nInt : 16진수로 변환할 int형 값을 받아들이는 인자
// nLength : 16진수로 변환할 수 있는 최대 길이를 받아들이는 인자
void IntToHex ( char *strHex, int nInt, int nLength )
{
	int		nQuotient, nRemain, nNeededCharNum;

	// 출력 버퍼를 초기화 시킨다.
	memset ( strHex, '₩0', nLength );

	// 임시로 16으로 소인수 분해 보아서 필요한 버퍼의 길이를 알아본다.
	nQuotient = nInt;
	for ( nNeededCharNum = 1 ; ( nQuotient /= 16 ) > 16 ; nNeededCharNum++ )
	{
	}

	if ( nLength - 1 < nNeededCharNum )
	{
		// 제한 길이보다 16진수변환 길이가 더 길면 에러 처리
	}

	// 16으로 소인수 분해 하면서 버퍼에 끝부터 나눈 나머지를 나눈다.
	nNeededCharNum = nLength - 1;
	nQuotient = nInt;
	while ( nQuotient >= 16 )
	{
		// 나눈 나머지를 버퍼에 넣는다.
		strHex[nNeededCharNum] = (char) ( nQuotient % 16 );
		nQuotient /= 16;

		if ( 0 <= strHex[nNeededCharNum] && strHex[nNeededCharNum] <= 9 )
		{
			strHex[nNeededCharNum] += '0';
		}
		else if ( 10 <= strHex[nNeededCharNum] && strHex[nNeededCharNum] <= 15 )
		{
			strHex[nNeededCharNum] +=  ( 'A' - 10 );
		}
		nNeededCharNum--;
	}

	// 마지막 나머지를 넣는다.
	strHex[nNeededCharNum] = (char) nQuotient;
```

```
        if ( 0 <= strHex[nNeededCharNum] && strHex[nNeededCharNum] <= 9 )
        {
                strHex[nNeededCharNum] += '0';
        }
        else if ( 10 <= strHex[nNeededCharNum] && strHex[nNeededCharNum] <= 15 )
        {
                strHex[nNeededCharNum] +=  ( 'A' - 10 );
        }

        nNeededCharNum--;
        while ( nNeededCharNum > 0 )
        {
                strHex[nNeededCharNum--] = '0';
        }
}
////////////////////////////////////////////////////////////////////////
// OneByteToTwoByte 전역 오버로딩 함수
// strOneByte : 팩킹된 문자열을 받아들이는 인자
// strTwoByte : 패킹한 것을 언팩해서 저장할 인자
// nLen : 길이 값을 받아들이는 인자
// 기능 : 1바이트로 팩킹된 문자열을 2바으트로 바꾸어서 저장해준다.
void OneByteToTwoByte ( char *strOneByte, char *strTwoByte, int nLen )
{
        int                             loop;

        for ( loop = 0 ; loop < nLen * 2 ; loop += 2 )
        {
                strTwoByte[loop] = (unsigned char) strOneByte[loop/2] / 16;
                strTwoByte[loop+1] = (unsigned char) strOneByte[loop/2] % 16;

                // strTwoByte에서 첫번째 문자를 읽어 처리
                if ( 0 <= strTwoByte[loop] && strTwoByte[loop] <= 9 )
                {
                        strTwoByte[loop] += '0';
                }
                else if ( 10 <= strTwoByte[loop] && strTwoByte[loop] <= 15 )
                {
                        strTwoByte[loop] +=  ( 'A' - 10 );
                }

                // strTwoByte에서 두번째 문자를 읽어 처리
                if ( 0 <= strTwoByte[loop+1] && strTwoByte[loop+1] <= 9 )
                {
                        strTwoByte[loop+1] += '0';
                }
                else if ( 10 <= strTwoByte[loop+1] && strTwoByte[loop+1] <= 15 )
                {
                        strTwoByte[loop+1] +=  ( 'A' - 10 );
                }
        }
        strTwoByte[loop] = '₩0';
}

////////////////////////////////////////////////////////////////////////
// LoadAllObject 함수
// AllObject : 메모리에서 얻어온 Object를 저장할 배열 주소를 받아들이는 인자
// nLen : Program의 길이를 받아들이는 인자
// 기능 : Object배열에 Object를 하나씩 메모리에서 읽어서 넣어준다
int LoadAllObject ( ObjectCode *AllObject[], int nLen )
{
        int                             offset = 0, nObjectNum = 0, n;

        while ( 1 )
        {
                AllObject[nObjectNum] = new ObjectCode;
                n = AllObject[nObjectNum]->GetObject ( MemoryLocation + offset, nLen );
                if ( n == -1 )
                {
                        break;
                }
                offset += n;
                nObjectNum++;
        }
        return nObjectNum;
}
```

Loader.h Source :

```
#ifndef _LOADER_H_
#define _LOADER_H_
#include "TabProcessing.h"
#include "MemoryManageUnit.h"
#include "Simulator.h"

int Loader1 ( FILE *pObjectFile );
void Loader2 ( FILE *pObjectFile );
extern char                 *MemoryLocation;
extern int                  CSADDR;
#endif
```

Loader.cpp Source :

```
#include "Loader.h"

////////////////////////////////////////////////////////////////////////
// Loader1 함수
// pObjectFile : Object파일의 포인터를 받아들이는 인자
int Loader1 ( FILE *pObjectFile )
{
        char                ObjectOneLine[MAX_ONELINE];
        char                *strSym, strTemp[7];
        int                         nControlSection = 0, loop, CSLTH = 0;

        // 프로그램을 올릴 메모리 시작주소를 CSADDR에 배정한다.
        CSADDR = 0;

        while ( fgets ( ObjectOneLine, MAX_ONELINE, pObjectFile ) != 0 )
        {
                if ( ObjectOneLine[0] == ' ' || ObjectOneLine[0] == 10 || ObjectOneLine[0] == '[' )
                {
                        // 개행이나 '^' 표시줄은 지나가는 것 처리
                        continue;
                }
                else if ( ObjectOneLine[0] == 'H' )
                {
                        // 첫번째 문자가 Header의 'H'인 경우처리
                        ESTAB               *pESTABNew;
                        strSym = strtok ( (char *) ObjectOneLine+1, " " );
                        CSLTH = HexToInt ( ObjectOneLine+14, 6 );

                        // ESTAB에 같은 심볼이 있는지 검사하고 넣어줌
                        if ( pESTAB_Header->Search ( strSym ) == NULL )
                        {
                                pESTABNew = new ESTAB ( strSym, (int) CSADDR, CSLTH, nControlSection );
                                pESTABNew->AddToESTAB ( );
                        }
                        else
                        {
                                // ESTAB에 같은 심볼이 존재하므로 에러 처리
                        }
                }

                // 현재 컨트롤 섹션의 End레코드가 나올때까지 읽어서 처리
                while ( fgets ( (char *)ObjectOneLine, MAX_ONELINE, pObjectFile ) != 0 )
                {
                        // End 레코드이면 그만 읽는다.
                        if ( ObjectOneLine[0] == 'E' )
                        {
                                break;
                        }
                        else if ( ObjectOneLine[0] == 'D' )
                        {
                                for ( loop = 1 ; loop < strlen ( ObjectOneLine ) - 1 ; loop += 12 )
                                {
                                        // Define 된것을 ESTAB에 주소와 같이 넣는다.
                                        ESTAB               *pESTABNew;
                                        strncpy ( strTemp, ObjectOneLine + loop, 6 );
                                        strTemp[6] = '₩0';
                                        strSym = strtok ( strTemp, " " );
```

```
                                        // ESTAB에 같은 심볼이 있는지 검사하고 넣어준다.
                                        if ( pESTAB_Header->Search ( strSym ) == NULL )
                                        {
                                                pESTABNew = new ESTAB ( strSym,
                                               (int) CSADDR + HexToInt ( ObjectOneLine+loop+6, 6 ),
                                                        0, nControlSection );
                                                pESTABNew->AddToESTAB ( );
                                        }
                                        else
                                        {
                                                // ESTAB에 같은 심볼이 있기때문에 에러 처리
                                        }
                                }
                        }
                }
                // 이전 컨트롤섹션의 길이를 컨트롤섹션 시작값에 더한다
                CSADDR += CSLTH;
                nControlSection++;
        }
        return CSADDR;
}
////////////////////////////////////////////////////////////////////////
// Loader1 함수
// pObjectFile : Object파일의 포인터를 받아들이는 인자
void Loader2 ( FILE *pObjectFile )
{
        char                ObjectOneLine[MAX_ONELINE];
        char                *strSym, cOperator;
        char                strValTwoByte1[7] = { '₩0', }, strValOneByte[7] = { '₩0', }, strValTwoByte[7] = { '₩0', };
        int                 nControlSection = 0, loop, CSLTH = 0;
        int                 nObjectStartAddr, nObjectLen;
        ESTAB               *pSearchReturnESTAB;
        int                 nModifyWhere, nModifyValue, nVal;

        fseek ( pObjectFile, 0, SEEK_SET );

        // 프로그램을 올릴 메모리 시작 주소를 CSADDR에 배정한다
        CSADDR = 0;

        while ( fgets ( ObjectOneLine, MAX_ONELINE, pObjectFile ) != 0 )
        {
                if ( ObjectOneLine[0] == ' ' || ObjectOneLine[0] == 10 || ObjectOneLine[0] == '[' )
                {
                        // 개행이나 '^' 표시줄은 지나가는 것 처리
                        continue;
                }
                else if ( ObjectOneLine[0] == 'H' )
                {
                        // 첫번째 문자가 Header의 'H'인 경우처리
                        CSLTH = HexToInt ( ObjectOneLine+14, 6 );
                }

                // 현재 컨트롤 섹션의 End레코드가 나올때까지 읽어서 처리
                while ( fgets ( ObjectOneLine, MAX_ONELINE, pObjectFile ) != 0 )
                {
                        if ( ObjectOneLine[0] == 'E' )
                        {
                                // End 레코드이면 그만 읽는다.
                                // 이전 컨트롤섹션의 길이를 컨트롤섹션 시작값에 더한다
                                CSADDR += CSLTH;
                                nControlSection++;
                                break;
                        }
                        else if ( ObjectOneLine[0] == 'T' )
                        {
                                // Text 레코드이면 Object들을 가져와서 메모리에 올려준다.
                                nObjectStartAddr = HexToInt ( ObjectOneLine + 1, 6 );
                                nObjectLen = HexToInt ( ObjectOneLine + 7, 2 );

                                for ( loop = 0 ; loop < nObjectLen ; loop++ )
                                {
                                        // Text Record 한줄을 두개의 바이트를 한개의 바이트로 팩킹해서 메모리에 올려준다.
                                        *( MemoryLocation + CSADDR + nObjectStartAddr + loop )
                                    = (unsigned char) HexToInt ( ObjectOneLine + (2 * loop) + 9, 2 );
```

```
                                        }
                                }
                                else if ( ObjectOneLine[0] == 'M' )
                                {
                                        // 변수 초기화
                                        memset ( strValOneByte, 'WO', 7 );
                                        memset ( strValTwoByte, 'WO', 7 );
                                        memset ( strValTwoByte1, 'WO', 7 );

                                        // Modify 레코드이면 지정된 메모리주소에 값을 빼거나 더한다.
                                        nModifyWhere = (int) CSADDR + HexToInt ( ObjectOneLine + 1, 6 );
                                        nObjectLen = HexToInt ( ObjectOneLine + 7, 2 );
                                        cOperator = ObjectOneLine[9];
                                        strSym = strtok ( ObjectOneLine + 10, " WtWn" );

                                        // 수정하고자하는 레코드의 길이가 홀수이면 반바이트 이동해서 수정한다.
                                        if ( ( pSearchReturnESTAB = pESTAB_Header->Search ( strSym ) ) != NULL )
                                        {
                                                // 수정할 장소의 팩킹된 3byte 값을 가져온다.
                                                for ( loop = 0 ; loop < 3 ; loop++ )
                                                {
                                                        strValOneByte[loop] = *(MemoryLocation+nModifyWhere+loop);
                                                }

                                                // 가져온 3byte 패킹값을 언팩킹한다.
                                                OneByteToTwoByte ( strValOneByte, strValTwoByte, 3 );

                                                // 수정할 레코드의 길이가 홀수이면 한바이트 더 가서 값을가져온다.
                                                if ( nObjectLen % 2 != 0 )
                                                {
                                                        // 수정하고자 하는 심볼을 ESTAB에서 찾은 경우
                                                        // 수정할 값을 얻어온다.
                                                        nModifyValue = pSearchReturnESTAB->GetAddr ( );
                                                        nVal = HexToInt ( strValTwoByte + 1, 5 );
                                                        loop = 1;

                                                }
                                                else {
                                                        nModifyValue = pSearchReturnESTAB->GetAddr ( );
                                                        nVal = HexToInt ( strValTwoByte, 6 );
                                                        loop = 0;
                                                }
                                                // cOperator에 있는 연산자에 따라 계산해준다.
                                                if ( cOperator == '+' )
                                                {
                                                        nVal += nModifyValue;
                                                }
                                                else if ( cOperator == '-' )
                                                {
                                                        nVal -= nModifyValue;
                                                }

                                                // 계산 정수 값을 다시 16진수 2byte형식으로 만든다.
                                                IntToHex ( strValOneByte, nVal, 6 );
                                                // 수정할 값을 strValTwoByte에 쓴다.
                                                for ( ; loop < 6 ; loop++ )
                                                {
                                                        strValTwoByte[loop] = strValOneByte[loop];
                                                }
                                                // 수정된 값이 들어있는 strValTwoByte를 팩킹한다.
                                                strValOneByte[0] = HexToInt ( strValTwoByte, 2 );
                                                strValOneByte[1] = HexToInt ( strValTwoByte + 2, 2 );
                                                strValOneByte[2] = HexToInt ( strValTwoByte + 4, 2 );

                                                // 수정한 값을 메모리에 다시 써준다.
                                                for ( loop = 0 ; loop < 3 ; loop++ )
                                                {
                                                        *(MemoryLocation+nModifyWhere+loop)
                                                                                = strValOneByte[loop];
                                                }
                                        }
                                }
                        }
                }
        }
```

ExecutionUnit.h Source :

```
#ifndef _EXECUTIONUNIT_H_
#define _EXECUTIONUNIT_H_
#include "MemoryManageUnit.h"
#include <iostream>

int GetValueFromReg ( int nReg );
void InputValueToReg ( int nReg, int nVal );
void STL ( ObjectCode *Instruction );
void STA ( ObjectCode *Instruction );
void STX ( ObjectCode *Instruction );
void LDA ( ObjectCode *Instruction );
void LDT ( ObjectCode *Instruction );
void J ( ObjectCode *Instruction );
void JSUB ( ObjectCode *Instruction );
void JEQ ( ObjectCode *Instruction );
void JLT ( ObjectCode *Instruction );
void COMP ( ObjectCode *Instruction );
void COMPR ( ObjectCode *Instruction );
void CLEAR ( ObjectCode *Instruction );
void TD ( ObjectCode *Instruction );
void WD ( ObjectCode *Instruction );
void RD ( ObjectCode *Instruction );
void LDCH ( ObjectCode *Instruction );
void STCH ( ObjectCode *Instruction );
void TIXR ( ObjectCode *Instruction );
void RSUB ( ObjectCode *Instruction );

extern int                  L_Register, A_Register, T_Register, X_Register;
extern int                  B_Register, PC_Register, SW_Register, S_Register, F_Register;
extern FILE                 *fp;
#endif
```

ExecutionUnit.cpp Source :

```
#include "ExecutionUnit.h"

////////////////////////////////////////////////////////////////////////
// GetValueFromReg 함수
// nReg : 값을 넣을 Register 번호를 받아들이는 인자
// 기능 : 원하는 Register에 원하는 값을 리턴해준다.
int GetValueFromReg ( int nReg )
{
        switch ( nReg )
        {
        case 0 :
                return A_Register;
        case 1 :
                return X_Register;
        case 2 :
                return L_Register;
        case 3 :
                return B_Register;
        case 4 :
                return S_Register;
        case 5 :
                return T_Register;
        case 6 :
                return F_Register;
        case 8 :
                return PC_Register;
        case 9 :
                return SW_Register;
        }
        return -1;
}

////////////////////////////////////////////////////////////////////////
// InputValueToReg 함수
// nRegNum : 값을 넣을 Register 번호를 받아들이는 인자
```

```
// nVal : 원하는 Register에 넣을 값을 받아들이는 인자
// 기능 : 원하는 Register에 원하는 값을 넣어 준다.
void InputValueToReg ( int nReg, int nVal )
{
	switch ( nReg )
	{
	case 0 :
		A_Register = nVal;
		break;
	case 1 :
		X_Register = nVal;
		break;
	case 2 :
		L_Register = nVal;
		break;
	case 3 :
		B_Register = nVal;
		break;
	case 4 :
		S_Register = nVal;
		break;
	case 5 :
		T_Register = nVal;
		break;
	case 6 :
		F_Register = nVal;
		break;
	case 8 :
		PC_Register = nVal;
		break;
	case 9 :
		SW_Register = nVal;
		break;
	}
}
////////////////////////////////////////////////////////////////////////
// LDT 함수
// Instruction : LDT명령을 포함하는 Instruction 전체를 받아들이는 인자
// 기능 : T Register에 Instruction에 포함된 데이터를 저장한다.
void LDT ( ObjectCode *Instruction )
{
	int			nAddr = 0;
	char		strVal[7];

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	if ( Instruction->GetNIXBPE ( IMMEDIATE ) == 1 && Instruction->GetNIXBPE ( INDIRECT ) == 0 )
	{
		// Immediate Addressing 방법이면 그냥 값을 넣는다.
		A_Register = Instruction->GetDisp ( );
	}
	else if ( Instruction->GetNIXBPE ( INDIRECT ) == 1 && Instruction->GetNIXBPE ( IMMEDIATE ) == 0 )
	{
		// Indirect Addressing 방법이면 메모리를 두번참조해서 값을 넣는다.
		OneByteToTwoByte ( MemoryLocation + nAddr, strVal, 3 );
		A_Register = HexToInt ( MemoryLocation + HexToInt ( strVal, 6 ), 6 );
	}
	else
	{
		// 위에 방법 외에는 Direct Addressing 방법으로 값을 넣는다.
		OneByteToTwoByte ( MemoryLocation + nAddr, strVal, 3 );
		T_Register = HexToInt ( strVal, 6 );
	}

	PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}
```

```
////////////////////////////////////////////////////////////////////////
// LDA 함수
// Instruction : LDA명령을 포함하는 Instruction 전체를 받아들이는 인자
// 기능 : A Register에 Instruction에 포함된 데이터를 저장한다.
void LDA ( ObjectCode *Instruction )
{
	int				nAddr = 0;
	char		strVal[7];

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	if ( Instruction->GetNIXBPE ( IMMEDIATE ) == 1 && Instruction->GetNIXBPE ( INDIRECT ) == 0 )
	{
		// Immediate Addressing 방법이면 그냥 값을 넣는다.
		A_Register = Instruction->GetDisp ( );
	}
	else if ( Instruction->GetNIXBPE ( INDIRECT ) == 1 && Instruction->GetNIXBPE ( IMMEDIATE ) == 0 )
	{
		// Indirect Addressing 방법이면 메모리를 두번참조해서 값을 넣는다.
		OneByteToTwoByte ( MemoryLocation + nAddr, strVal, 3 );
		A_Register = HexToInt ( MemoryLocation + HexToInt ( strVal, 6 ), 6 );
	}
	else
	{
		// 위에 방법 외에는 Direct Addressing 방법으로 값을 넣는다.
		OneByteToTwoByte ( MemoryLocation + nAddr, strVal, 3 );
		A_Register = HexToInt ( strVal, 6 );
	}

	PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// STA 함수
// Instruction : STA 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : A Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STA ( ObjectCode *Instruction )
{
	int				nAddr = 0;
	char		strVal[7];

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	// Instruction이 가리키는 메모리에 A Register 값을 쓴다
	sprintf ( strVal, "%06X", A_Register );
	strVal[0] = HexToInt ( strVal, 2 );
	strVal[1] = HexToInt ( strVal + 2, 2 );
	strVal[2] = HexToInt ( strVal + 4, 2 );

	*(MemoryLocation + nAddr) = strVal[0];
	*(MemoryLocation + nAddr+1) = strVal[1];
	*(MemoryLocation + nAddr+2) = strVal[2];

	PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}
```

```
////////////////////////////////////////////////////////////////////////
// STL 함수
// Instruction : STL 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : L Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STL ( ObjectCode *Instruction )
{
        int                             nAddr = 0;
        char                strVal[7];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        // Instruction이 가리키는 메모리에 A Register 값을 쓴다
        sprintf ( strVal, "%06X", L_Register );
        strVal[0] = HexToInt ( strVal, 2 );
        strVal[1] = HexToInt ( strVal + 2, 2 );
        strVal[2] = HexToInt ( strVal + 4, 2 );

        *(MemoryLocation + nAddr) = strVal[0];
        *(MemoryLocation + nAddr+1) = strVal[1];
        *(MemoryLocation + nAddr+2) = strVal[2];

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// STX 함수
// Instruction : STX 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : X Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STX ( ObjectCode *Instruction )
{
        int                             nAddr = 0;
        char                strVal[7];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        // Instruction이 가리키는 메모리에 A Register 값을 쓴다
        sprintf ( strVal, "%06X", X_Register );
        strVal[0] = HexToInt ( strVal, 2 );
        strVal[1] = HexToInt ( strVal + 2, 2 );
        strVal[2] = HexToInt ( strVal + 4, 2 );


        *(MemoryLocation + nAddr) = strVal[0];
        *(MemoryLocation + nAddr+1) = strVal[1];
        *(MemoryLocation + nAddr+2) = strVal[2];

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// J 함수
// Instruction : J 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : PC Register에 Instruction에 포함된 값을 쓴다.
void J ( ObjectCode *Instruction )
{
        int                             nAddr;
        char                strVal[7];
```

```
	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	// Indirect Addressing 방법일 경우 처리
	if ( Instruction->GetNIXBPE ( INDIRECT ) == 1 && Instruction->GetNIXBPE ( IMMEDIATE ) == 0 )
	{
		OneByteToTwoByte ( MemoryLocation + nAddr, strVal, 3 );
		nAddr = HexToInt ( strVal, 6 );
	}

	PC_Register = nAddr;
}

////////////////////////////////////////////////////////////////////////
// JSUB 함수
// Instruction : JSUB 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : 현재 PC값을 L Register에 저장하고 PC Register에 Instruction에 포함된 값을 넣는다.
void JSUB ( ObjectCode *Instruction )
{
	int				nAddr;

	// 현재 PC값을 L Register에 저장
	L_Register = PC_Register + Instruction->GetType ( );

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	// 점플 할 주소를 PC Register에 넣는다.
	PC_Register = nAddr;
}

////////////////////////////////////////////////////////////////////////
// JEQ 함수
// Instruction : JEQ 명령을 포함하는 전체 Instructoin을 받아들이는 인자
// 기능 : Condition Code를 보고 0이면 Instruction에 포함한 값을 PC Register에 넣는다
void JEQ ( ObjectCode *Instruction )
{
	int				nAddr;

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	if ( SW_Register == 0 )
	{
		PC_Register = nAddr;
	}
	else
	{
		PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
	}
```

```
}

////////////////////////////////////////////////////////////////////////
// JLT 함수
// Instruction : JLT 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Condition Code를 보고 음수이면 Instruction에 포함된 값을 PC Register에 넣는다.
void JLT ( ObjectCode *Instruction )
{
	int				nAddr;

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	if ( SW_Register < 0 )
	{
		PC_Register = nAddr;
	}
	else
	{
		PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
	}
}

////////////////////////////////////////////////////////////////////////
// COMP 함수
// Instruction : COMP 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : A Register값과 Instruction에 포함된 값을 비교하여 Condition Code 값을 변경한다
void COMP ( ObjectCode *Instruction )
{
	int				nAddr;
	int				nValue;

	// PC Relative일때 메모리주소 계산과 또는
	// 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
	if ( Instruction->GetNIXBPE ( PC ) == 1 )
	{
		nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
	}
	else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
	{
		nAddr = Instruction->GetDisp ( );
	}

	if ( Instruction->GetNIXBPE ( IMMEDIATE ) == 1 )
	{
		nValue = Instruction->GetDisp ( );
	}
	else
	{
		nValue = HexToInt ( MemoryLocation + nAddr, 6 );
	}

	// Instruction에서 주어진 값과 A Register 값을 비교해서
	// Condition Code값을 변화 시킨다.
	SW_Register = nValue - A_Register;

	PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// COMPR 함수
// Instruction : COMPR 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 두개의 레지스터 값을 비교해서 Condition Code값을 변화시킨다
void COMPR ( ObjectCode *Instruction )
{
	int					nVal1, nVal2;
	unsigned char		*pStrObjectTwoByte;
```

```
        pStrObjectTwoByte = Instruction->GetStrObjectTwoBytePointer ( );

        // Instruction의 3번째 4번째에 있는 레지스터 번호를 보고
        // 그에 맞는 레지스터에서 값을 받아온다.
        nVal1 = GetValueFromReg ( pStrObjectTwoByte[2] - '0' );
        nVal2 = GetValueFromReg ( pStrObjectTwoByte[3] - '0' );

        // 두개의 레지스터의 값으로 SW Register의 Condition Code를 변화시킨다.
        SW_Register = nVal1 - nVal2;

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// CLEAR 함수
// Instruction : CLEAR 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 한개의 레지스터 값을 0으로 초기화시켜준다.
void CLEAR ( ObjectCode *Instruction )
{
        unsigned char       *pStrObjectTwoByte;

        pStrObjectTwoByte = Instruction->GetStrObjectTwoBytePointer ( );

        InputValueToReg ( pStrObjectTwoByte[2] - '0', 0 );

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// TD 함수
// Instruction : TD 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 디바이스가 사용가능 한지 테스트한다
void TD ( ObjectCode *Instruction )
{
        int                         nAddr;
        char                InputDeviceOneByte[2], InputDeviceTwoByte[3];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        InputDeviceOneByte[0] = *(MemoryLocation + nAddr);
        InputDeviceOneByte[1] = '\0';
        OneByteToTwoByte ( InputDeviceOneByte, InputDeviceTwoByte, 1 );

        fp = fopen ( InputDeviceTwoByte, "a+" );

        if ( fp == NULL )
        {
                SW_Register = 0;
        }
        else
        {
                SW_Register = -1;
        }

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// WD 함수
// Instruction : WD 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Device로 A Register의 가장 오른쪽 데이터부터 쓴다.
void WD ( ObjectCode *Instruction )
{
        int                         nAddr, nLen;
        static int          nOffset = 0;
```

```
        char                strVal[7];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        sprintf ( strVal, "%06X", A_Register );
        strVal[0] = HexToInt ( strVal, 2 );
        strVal[1] = HexToInt ( strVal + 2, 2 );
        strVal[2] = HexToInt ( strVal + 4, 2 );

        fwrite ( &strVal[2], 1, 1, fp );

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );

        fclose ( fp );
}

////////////////////////////////////////////////////////////////////////
// RD 함수
// Instruction : RD 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Device로 부터 한바이트씩 값을 읽어 A Register의 가장 오른쪽부터 쓴다.
void RD ( ObjectCode *Instruction )
{
        int                         nAddr, nLen;
        static int          nOffset = 0;
        char                strVal[7], strVal1[7];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        // A Register에 들어있는 값을 strVal에 16진수 2바이트형식으로 저장한다
        sprintf ( strVal, "%06X", A_Register );
        strVal[0] = HexToInt ( strVal, 2 );
        strVal[1] = HexToInt ( strVal + 2, 2 );
        strVal[2] = HexToInt ( strVal + 4, 2 );

        // 디바이스에서 1바이트 문자열을 받아와서 strVal의 가장 오른쪽에 저장
        fseek ( fp, 0, SEEK_END );
        nLen = ftell ( fp );

        if ( nOffset < nLen )
        {
                fseek ( fp, nOffset, SEEK_SET );
                nOffset += fscanf ( fp, "%c", strVal + 2 );
        }
        else
        {
                strVal[2] = '₩0';
        }

        // strVal를 다시 A Register에 Int형으로 변환해서 저장해준다.
        OneByteToTwoByte ( strVal, strVal1, 3 );
        A_Register = HexToInt ( strVal1, 6 );

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );

        fclose ( fp );
}
```

```
////////////////////////////////////////////////////////////////////////
// LDCH 함수
// Instruction : LDCH 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 버퍼로 부터 한바이트를 읽어와서 A Register의
//                          가장 오른쪽에 저장한다.
void LDCH (ObjectCode *Instruction )
{
        int                             nAddr;
        char                strVal[7], strVal1[7];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        // A Register 값을 16진수 2바이트 형식으로 만들고
        // 다시 1바이트로 팩킹한다.
        sprintf ( strVal, "%06X", A_Register );
        strVal[0] = HexToInt ( strVal, 2 );
        strVal[1] = HexToInt ( strVal + 2, 2 );
        strVal[2] = HexToInt ( strVal + 4, 2 );

        // Instruction에 주어진 버퍼와 Index로 버퍼에서 한바이트 읽어옴
        strVal[2] = *(MemoryLocation + nAddr + X_Register);

        // strVal를 다시 A Register에 Int형으로 변환해서 저장해준다.
        OneByteToTwoByte ( strVal, strVal1, 3 );
        A_Register = HexToInt ( strVal1, 6 );

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// STCH 함수
// Instruction : STCH 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : A Register의 가장 오른쪽 한바이트를 Instruction에 주어진 버퍼에 저장한다.
void STCH (ObjectCode *Instruction )
{
        int                             nAddr;
        char                strVal[7];

        // PC Relative일때 메모리주소 계산과 또는
        // 4형식일때는 Immediate나 아니냐에 따라 레지스터에 값을 넣는다
        if ( Instruction->GetNIXBPE ( PC ) == 1 )
        {
                nAddr = Instruction->GetDisp ( ) + Instruction->GetPC ( ) + 3;
        }
        else if ( Instruction->GetNIXBPE ( EXTENT ) == 1 )
        {
                nAddr = Instruction->GetDisp ( );
        }

        // A Register를 16진수 2바이트 형식으로 만들고
        // 1바이트로 팩킹한다.
        sprintf ( strVal, "%06X", A_Register );
        strVal[0] = HexToInt ( strVal, 2 );
        strVal[1] = HexToInt ( strVal + 2, 2 );
        strVal[2] = HexToInt ( strVal + 4, 2 );

        // A Register의 패킹한 값의 가장 오른쪽 부분을 Instruction에 주어진 버퍼에 저장한다.
        *(MemoryLocation + nAddr + X_Register) = strVal[2];

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );
}

////////////////////////////////////////////////////////////////////////
// TIXR 함수
// Instruction : TIXR 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : X Register 값을 하나 증가 시키고 Instruction에 주어진 Register와 비교해서
```

```
//                          Condition Code를 변화 시켜준다.
void TIXR ( ObjectCode *Instruction )
{
        int                             nVal;
        unsigned char       *pStrObjectTwoByte;

        // TIXR뒤에 있는 레지스터로 부터 값을 가져온다.
        pStrObjectTwoByte = Instruction->GetStrObjectTwoBytePointer ( );
        nVal = GetValueFromReg ( pStrObjectTwoByte[2] - '0' );

        // X Register값을 하나 증가시킨다.
        X_Register += 1;

        // Condition Code를 변화 시킨다.
        if ( X_Register < nVal )
        {
                SW_Register = -1;
        }
        else
        {
                SW_Register = 1;
        }

        PC_Register = Instruction->GetPC ( ) + Instruction->GetType ( );

}

////////////////////////////////////////////////////////////////////////
// RSUB 함수
// Instruction : RSUB 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : L Register에 들어가있는 값을 PC Register에 넣어준다.
void RSUB ( ObjectCode *Instruction )
{
        PC_Register = L_Register;
}
```

Control.h Source :

```
#ifndef _TREE_H_
#define _TREE_H_
#include <windows.h>
#include <CommCtrl.h>
#include "Richedit.h"
class Tree
{
private :
        HWND                        hTree;
        HWND                        hWnd;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         nSize_Y;
        DWORD                       dwStyle;
        TVINSERTSTRUCT              TI;

public :
        Tree ( );
        Tree ( HWND hWindow, int nPosX, int nPosY, int nSizeX, int nSizeY );
        void Create ( );
        HTREEITEM Insert ( HTREEITEM hFather, char *pStr );
        void Expand ( HTREEITEM );
        ~Tree ( );
};

class List
{
private :
        HWND                        hList;
        HWND                        hWnd;
        HMENU                       hListID;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         Size_Y;
        DWORD                       dwStyle;

public :
        List ( );
        List ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY );
        void Create ( );
        void Insert ( char *pStr );
        void SetSelect ( int nCur );
};

class Edit
{
private :
        HWND                        hEdit;
        HWND                        hWnd;
        HMENU                       hEditID;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         nSize_Y;
        DWORD                       dwStyle;

public :
        Edit ( );
        Edit ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY );
        Edit ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, DWORD style );
        void Create ( );
        void CreateRichedit ( );
        void SetCharFormat ( CHARFORMAT2 cf );
        void SetText ( int nValue );
        void SetText ( char *strValue );
};

class Button
{
private :
```

```
        HWND                        hWnd;
        HMENU                       hButtonID;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         nSize_Y;
        DWORD                       dwStyle;
        LPSTR                       pszName;

public :
        Button ( );
        Button ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr );
        void Create ( );
};

class GroupBox
{
private :
        HWND                        hWnd;
        HMENU                       hGroupBoxID;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         nSize_Y;
        DWORD                       dwStyle;
        LPSTR                       pszName;

public :
        GroupBox ( );
        GroupBox ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr );
        void Create ( );
};

class CheckBox
{
private :
        HWND                        hWnd;
        HWND                        hCheckBox;
        HMENU                       hCheckID;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         nSize_Y;
        DWORD                       dwStyle;
        LPSTR                       pszName;

public :
        CheckBox ( );
        CheckBox ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr );
        void Create ( );
        void SetCheck ( int nCheckOrUncheck );
};

class Label
{
private :
        HWND                        hWnd;
        HMENU                       hLabelID;
        int                         nPos_X;
        int                         nPos_Y;
        int                         nSize_X;
        int                         nSize_Y;
        DWORD                       dwStyle;
        LPSTR                       pszName;

public :
        Label ( );
        Label ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr );
        void Create ( );
};

extern HINSTANCE g_hInst;

#endif
```

Control.cpp Source :

```
#include "Control.h"
#include "MemoryManageUnit.h"
#include "Richedit.h"
#include <windows.h>
#include <CommCtrl.h>
#include <stdio.h>
////////////////////////////////////////////////////////////////////////////
// Tree 기본 생성자
Tree::Tree ( )
{
}

////////////////////////////////////////////////////////////////////////////
// Tree 오버로딩 생성자
// hWnd : Tree를 생성할 윈도우 핸들을 받아들이는 인자
// nPos_X : 윈도우의 X좌표 어디에 생성할지 받아들이는 인자
// nPos_Y : 윈도우의 Y좌표 어디에 생성할지 받아들이는 인자
// nSize_X : 생성 할 Tree의 X축 크기를 받아들이는 인자
// nSize_Y : 생성 할 Tree의 Y축 크기를 받아들이는 인자
// 기능 : 주어진 위치 좌표에 주어진 크기 값을 멤버 변수에 할당
Tree::Tree ( HWND hWindow, int nPosX, int nPosY, int nSizeX, int nSizeY )
{
        hWnd = hWindow;
        nPos_X = nPosX;
        nPos_Y = nPosY;
        nSize_X = nSizeX;
        nSize_Y = nSizeY;
        dwStyle = WS_CHILD | WS_VISIBLE | WS_BORDER | TVS_LINESATROOT | TVS_HASLINES | TVS_SHOWSELALWAYS;
}

////////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 정해진 멤버변수 값으로 Tree 컨트롤러 생성
void Tree::Create ( )
{
        hTree = CreateWindow ( WC_TREEVIEW, "", dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, NULL, g_hInst, NULL );
}

////////////////////////////////////////////////////////////////////////////
// Expand 멤버 함수
// 기능 : 트리를 확장해준다.
void Tree::Expand ( HTREEITEM h )
{
        TreeView_Expand ( hTree, h, TVM_EXPAND );
}

////////////////////////////////////////////////////////////////////////////
// Tree 소멸자
// 기능 :
Tree::~Tree ( )
{
}

////////////////////////////////////////////////////////////////////////////
// Insert 멤버 함수
// hFather : 어느 부모에 Item을 넣을지 결정하는 인자
// Str : 어느 값을 넣을지 결정하는 인자
// 기능 : 넣고자 하는 부모 밑에 원하는 값을 Itemd을 넣어 준다.
HTREEITEM Tree::Insert ( HTREEITEM hFather, char *Str )
{
        HTREEITEM hReturn;
        TI.hParent = hFather;
        TI.hInsertAfter = TVI_LAST;
        TI.item.pszText = Str;
        // 지금 넣은 트리값의 핸들을 리턴해주기위해 저장
        hReturn = (HTREEITEM) SendMessage ( hTree, TVM_INSERTITEM, 0, (LPARAM) (LPTV_INSERTSTRUCT) &TI );
        return hReturn;
}

////////////////////////////////////////////////////////////////////////////
// List 기본 생성자
List::List ( )
```

```
{
}

////////////////////////////////////////////////////////////////////////
// List 오버로딩 생성자
// hWindow : 리스트가 속할 윈도우의 핸들을 받아들이는 인자
// hID : 리스트 고유 아이디를 받아들이는 인자
// nPosX : 리스트를 위치시킬 X좌표 값을 받아들이는 인자
// nPosY : 리스트를 위치시킬 Y좌표 값을 받아들이는 인자
// SizeX : 리스트의 크기 X좌표 값을 받아들이는 인자
// SizeY : 리스트의 크기 Y좌표 값을 받아들이는 인자
List::List ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY )
{
        hWnd = hWindow;
        hListID = (HMENU) hID;
        nPos_X = nPosX;
        nPos_Y = nPosY;
        nSize_X = nSizeX;
        nSize_Y = nSizeY;
        dwStyle = WS_CHILD | WS_VISIBLE | WS_BORDER | LBS_NOTIFY | WS_VSCROLL;
}


////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 멤버 변수에 들어간 속성을 바탕으로 화면 리스트 박스를 만들어줌
void List::Create ( )
{
        hList = CreateWindow ( "listbox", NULL, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hListID, g_hInst, NULL );
}

////////////////////////////////////////////////////////////////////////
// Insert 멤버 함수
// Str : 리스트에 들어갈 값을 받아들이는 인자
void List::Insert ( char *Str )
{
        SendMessage( hList, LB_ADDSTRING, 0, (LPARAM) Str );
}

////////////////////////////////////////////////////////////////////////
// SetSelect 멤버 함수
// nCur : 선택할 리스트 번호를 입력 받는 인자
void List::SetSelect ( int nCur )
{
        SendMessage ( hList, LB_SETCURSEL, nCur, 0 );
}

////////////////////////////////////////////////////////////////////////
// Edit 기본 생성자
Edit::Edit ( )
{
}

////////////////////////////////////////////////////////////////////////
// Edit 오버로딩 생성자
// hWindow : Edit 박스가 속할 윈도우의 핸들을 받아들이는 인자
// hID : Edit 박스 고유 ID를 받아들이는 인자
// nPosX : Edit 박스를 위치시킬 X좌표를 받아들이는 인자
// nPosY : Edit 박스를 위치시킬 Y좌표를 받아들이는 인자
// nSizeX : Edit 박스의 크기 X값을 받아들이는 인자
// nSizeY : Edit 박스의 크기 Y값을 받아들이는 인자
Edit::Edit ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY )
{
        hWnd = hWindow;
        hEditID = (HMENU) hID;
        nPos_X = nPosX;
        nPos_Y = nPosY;
        nSize_X = nSizeX;
        nSize_Y = nSizeY;
        dwStyle = WS_CHILD | WS_VISIBLE | WS_BORDER | ES_READONLY | ES_CENTER;
}

////////////////////////////////////////////////////////////////////////
// Edit 오버로딩 생성자
// hWindow : Edit 박스가 속할 윈도우의 핸들을 받아들이는 인자
```

```
// hID : Edit 박스 고유 ID를 받아들이는 인자
// nPosX : Edit 박스를 위치시킬 X좌표를 받아들이는 인자
// nPosY : Edit 박스를 위치시킬 Y좌표를 받아들이는 인자
// nSizeX : Edit 박스의 크기 X값을 받아들이는 인자
// nSizeY : Edit 박스의 크기 Y값을 받아들이는 인자
// dwSty : Edit 박스의 스타일 값을 받아들이는 인자
Edit::Edit ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, DWORD dwSty )
{
	hWnd = hWindow;
	hEditID = (HMENU) hID;
	nPos_X = nPosX;
	nPos_Y = nPosY;
	nSize_X = nSizeX;
	nSize_Y = nSizeY;
	dwStyle = dwSty;
}

////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 멤버 변수에 들어간 Edit박스 속성을 기반으로 Edit박스 생성
void Edit::Create ( )
{
	hEdit = CreateWindow ( "edit", NULL, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hEditID, g_hInst, NULL );
	SetWindowText ( hEdit, "000000" );
}

////////////////////////////////////////////////////////////////////////
// CreateRichedit 멤버 함수
// 기능 : 멤버 변수에 들어간 Edit 박스 속성을 기반으로 RichEdit 박스 생성
void Edit::CreateRichedit ( )
{
	hEdit = CreateWindow ( RICHEDIT_CLASS, NULL, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hEditID, g_hInst, NULL );
}

////////////////////////////////////////////////////////////////////////
// SetCharFormat 멤버 함수
void Edit::SetCharFormat ( CHARFORMAT2 cf )
{
	SendMessage ( hEdit, EM_SETCHARFORMAT, SCF_ALL, (LPARAM) &cf );
}

////////////////////////////////////////////////////////////////////////
// SetText 멤버 함수
// nValue : Edit창에 띄울 값을 받아들이는 인자
// 기능 : nValue값을 Edit 윈도우에 띄운다.
void Edit::SetText ( int nValue )
{
	char		strValue[100];
	int			nLen;

	sprintf ( strValue, "%06X", nValue );
	nLen = (int) strlen ( strValue );

	if ( nValue < 0 )
	{
		SetWindowText ( hEdit, strValue + nLen - 6 );
	}
	else
	{
		SetWindowText ( hEdit, strValue );
	}
}

////////////////////////////////////////////////////////////////////////
// SetText 오버로딩 멤버 함수
// strValue : Edit창에 띄울 문자열 값을 받아들이는 인자
// 기능 : strValue값을 Edit 윈도웨 띄운다.
void Edit::SetText ( char *strValue )
{
	SetWindowText ( hEdit, strValue );
}

////////////////////////////////////////////////////////////////////////
// Button 기본 생성자
Button::Button ( )
```

```
{
}

////////////////////////////////////////////////////////////////////////
// Button 오버로딩 생성자
// hWindow : Button이 속할 윈도우의 핸들을 받아들이는 인자
// hID : Button의 고유 ID를 받아들이는 인자
// nPosX : Button을 위치시킬 X좌표 값을 받아들이는 인자
// nPosY : Button을 위치시킬 Y좌표 값을 받아들이는 인자
// nSizeX : Button의 X크기 값을 받아들이는 인자
// nSizeY : Button의 Y크기 값을 받아들이는 인자
// pszStr : Button 위에 나타낼 문자열을 받아들이는 인자
Button::Button ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr )
{
        hWnd = hWindow;
        hButtonID = (HMENU) hID;
        nPos_X = nPosX;
        nPos_Y = nPosY;
        nSize_X = nSizeX;
        nSize_Y = nSizeY;
        dwStyle = WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON;
        pszName = pszStr;
}

////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 멤버변수에 설정된 속성값을 기반으로 Button을 생성
void Button::Create ( )
{
        CreateWindow ( "button", pszName, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hButtonID, g_hInst, NULL );
}

////////////////////////////////////////////////////////////////////////
// GroupBox 기본 생성자
GroupBox::GroupBox ( )
{
}

////////////////////////////////////////////////////////////////////////
// GroupBox 오버로딩 생성자
// hWindow : GroupBox이 속할 윈도우의 핸들을 받아들이는 인자
// hID : GroupBox의 고유 ID를 받아들이는 인자
// nPosX : GroupBox을 위치시킬 X좌표 값을 받아들이는 인자
// nPosY : GroupBox을 위치시킬 Y좌표 값을 받아들이는 인자
// nSizeX : GroupBox의 X크기 값을 받아들이는 인자
// nSizeY : GroupBox의 Y크기 값을 받아들이는 인자
// pszStr : GroupBox 위에 나타낼 문자열을 받아들이는 인자
GroupBox::GroupBox ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr )
{
        hWnd = hWindow;
        hGroupBoxID = (HMENU) hID;
        nPos_X = nPosX;
        nPos_Y = nPosY;
        nSize_X = nSizeX;
        nSize_Y = nSizeY;
        dwStyle = WS_CHILD | WS_VISIBLE | BS_GROUPBOX;
        pszName = pszStr;
}

////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 멤버변수에 설정된 속성값을 기반으로 GroupBox을 생성
void GroupBox::Create ( )
{
        CreateWindow ( "button", pszName, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hGroupBoxID, g_hInst, NULL );
}

////////////////////////////////////////////////////////////////////////
// CheckBox 기본 생성자
CheckBox::CheckBox ( )
{
}
////////////////////////////////////////////////////////////////////////
// CheckBox 오버로딩 생성자
// hWindow : CheckBox가 속할 윈도우의 핸들을 받아들이는 인자
```

```
// hID : CheckBox의 고유 ID를 받아들이는 인자
// nPosX : CheckBox를 위치시킬 X값을 받아들이는 인자
// nPosY : CheckBox를 위치시킬 Y값을 받아들이는 인자
// nSizeX : CheckBox의 X크기 값을 받아들이는 인자
// nSizeY : CheckBox의 Y크기 값을 받아들이는 인자
// pszStr : CheckBox의 옆에 나타낼 문자열 값을 받아들이는 인자
CheckBox::CheckBox ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr )
{
	hWnd = hWindow;
	hCheckID = (HMENU) hID;
	nPos_X = nPosX;
	nPos_Y = nPosY;
	nSize_X = nSizeX;
	nSize_Y = nSizeY;
	dwStyle = WS_CHILD | WS_VISIBLE | BS_3STATE;
	pszName = pszStr;
}

////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 멤버변수에 설정된 속성값을 기반으로 CheckBox을 생성
void CheckBox::Create ( )
{
	hCheckBox = CreateWindow ( "button", pszName, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hCheckID, g_hInst, NULL );
}

////////////////////////////////////////////////////////////////////////
// SetCheck 멤버 함수
// 기능 : 체크박스를 체크상태로 만든다
void CheckBox::SetCheck ( int nCheckOrUncheck )
{
	if ( nCheckOrUncheck == 1 )
	{
		SendMessage ( hCheckBox, BM_SETCHECK, BST_CHECKED, 0 );
	}
	else
	{
		SendMessage ( hCheckBox, BM_SETCHECK, BST_UNCHECKED, 0 );
	}
}

////////////////////////////////////////////////////////////////////////
// Label 기본 생성자
Label::Label ( )
{
}

////////////////////////////////////////////////////////////////////////
// Label 오버로딩 생성자
// hWindow : Label가 속할 윈도우의 핸들을 받아들이는 인자
// hID : Label의 고유 ID를 받아들이는 인자
// nPosX : Label를 위치시킬 X값을 받아들이는 인자
// nPosY : Label를 위치시킬 Y값을 받아들이는 인자
// nSizeX : Label의 X크기 값을 받아들이는 인자
// nSizeY : Label의 Y크기 값을 받아들이는 인자
// pszStr : Label의 옆에 나타낼 문자열 값을 받아들이는 인자
Label::Label ( HWND hWindow, int hID, int nPosX, int nPosY, int nSizeX, int nSizeY, LPSTR pszStr )
{
	hWnd = hWindow;
	hLabelID = (HMENU) hID;
	nPos_X = nPosX;
	nPos_Y = nPosY;
	nSize_X = nSizeX;
	nSize_Y = nSizeY;
	dwStyle = WS_CHILD | WS_VISIBLE;
	pszName = pszStr;
}

////////////////////////////////////////////////////////////////////////
// Create 멤버 함수
// 기능 : 멤버변수에 정의된 속성을 기반으로 Label 생성
void Label::Create ( )
{
	CreateWindow ( "static", pszName, dwStyle, nPos_X, nPos_Y, nSize_X, nSize_Y, hWnd, hLabelID, g_hInst, NULL );
}
```

TabProcessing.h Source :

```
#ifndef _TABPROCESSING_H_
#define _TABPROCESSING_H_

class ESTAB
{
private :
        char      *strName;          // External Symbol, Control Section 의 이름을 저장할 멤버변수
        int       nAddr;             // External Symbol의 주소를 저장할 멤버변수
        int       nLength;           // Control Section의 길이를 저장할 멤버변수
        int       nControlSec;       // External Symbol이 속해 있는 Control Section을 저장할 멤버변수
        ESTAB     *pNext;            // 다음 링크를 가리키는 멤버변수

public :
        ESTAB ( );
        ESTAB ( char *strName, int nAddr, int nLenght, int nControlSec );
        ~ESTAB ( );
        void AddToESTAB ( );
        ESTAB *Search ( char *strName );
        int GetLength ( );
        char *GetName ( );
        int       GetAddr ( );
        ESTAB *GetNext ( );
};

class OPTAB
{
private :
        char      *strName;          // OPCODE의 이름
        char      xValue[2];         // OPCODE의 16진수 값
        int       nType;             // OPCODE의 타입
        OPTAB     *pNext;

public :
        OPTAB ( );
        OPTAB ( char *pInst, char *pValue, int nType );
        ~OPTAB ( );
        int GetType ( );
        char *GetName ( );
        void InitOpcodeTable ( );
        OPTAB *SearchByName ( char *strName );
        OPTAB *SearchByValue ( char *strValue );
};

#define  SEPERATOR           ",.WtWn "
#define MAX_ONELINE                  70

extern ESTAB       *pESTAB_Header, *pESTAB_Tail;
extern OPTAB       *pOPTAB_Header, *pOPTAB_Tail;

#endif
```

TabProcessing.cpp Source :

```
#include "TabProcessing.h"
#include <iostream>
using namespace std;

////////////////////////////////////////////////////////////////////////
// GetAddr 멤버 함수
// 기능 : 멤버변수 nAddr의 값을 리턴해준다.
int ESTAB::GetAddr ( )
{
        return nAddr;
}

////////////////////////////////////////////////////////////////////////
// Search 멤버 함수
// strNa : 찾고자 하는 External Symbol 이름을 받아들이는 인자
ESTAB* ESTAB::Search ( char *strNa )
{
        ESTAB              *pTempESTAB;
```

```
	pTempESTAB = pESTAB_Header;
	// 리스트의 끝까지 루프돌면서 원하는 값 찾음
	while ( pTempESTAB != NULL )
	{
		if ( strcmp ( pTempESTAB->strName, strNa ) == 0 )
		{
			return pTempESTAB;
		}
		else
		{
			pTempESTAB = pTempESTAB->pNext;
		}
	}
	return NULL;
}

////////////////////////////////////////////////////////////////////////
// AddToESTAB 멤버 함수
// pHeader : ESTAB 링크의 Header 포인터를 받아들이는 인자
// pTail : ESTAB 링크의 Tail 포인터를 받아들이는 인자
// 기능 : ESTAB에 현 객체를 연결해주는 기능
void ESTAB::AddToESTAB ( )
{
	if ( pESTAB_Header == NULL )
	{
		pESTAB_Header = this;
		pESTAB_Tail = this;
	}
	else
	{
		pESTAB_Tail->pNext = this;
		pESTAB_Tail = this;
	}
}

////////////////////////////////////////////////////////////////////////
// ESTAB 생성자
// strName : External Symbol의 이름을 받아들이는 인자
// nAddr : External Symbol의 주소를 받아들이는 인자
// nControlSec : External Symbol의 컨트롤 섹션값을 받아들이는 인자
ESTAB::ESTAB ( char *strNa, int nAdd, int nLen, int nControl )
{
	int			nLenOfstrName;

	// strName을 메모리 할당하고 배정하는 부분
	nLenOfstrName = (int) strlen ( strNa );
	strName = new char[nLenOfstrName+1];
	strcpy ( strName, strNa );
	// 나머지 배정하는 부분
	nAddr = nAdd;
	nLength = nLen;
	nControlSec = nControl;
	pNext = NULL;
}

////////////////////////////////////////////////////////////////////////
// ESTAB 소멸자
// 기능 : 생성할때 할당한 메모리 해제
ESTAB::~ESTAB ( )
{
	delete ( strName );
}

////////////////////////////////////////////////////////////////////////
// GetLength 멤버 함수
// 기능 : 멤버변수 nLength 값을 반환해준다.
int ESTAB::GetLength ( )
{
	return nLength;
}

////////////////////////////////////////////////////////////////////////
// GetName 멤버 함수
// 기능 : 멤버변수 strName의 포인터를 반환해 준다.
char *ESTAB::GetName ( )
```

```
{
        return strName;
}

////////////////////////////////////////////////////////////////////////
// GetNext 멤버 함수
// 기능 : 멤버변수 pNext의 포인터를 반환해 준다.
ESTAB *ESTAB::GetNext ( )
{
        return pNext;
}

////////////////////////////////////////////////////////////////////////
// InitOpcodeTable 멤버함수
// 기능 : 테이블을 OPTAB 파일에서 자료를 읽어들여와 초기화 하다.
void OPTAB::InitOpcodeTable ( )
{
        FILE                *pOPTAB_File;
        char                *pInst, *pValue, *pOneLine;
        int                         nType;

        // OPTAB파일을 연다
        pOPTAB_File = fopen ( "OPTAB", "r" );
        if ( pOPTAB_File == NULL )
        {
                printf ( "OPTAB File Not Founded!!!Wn" );
                return;
        }

        // pStr에 메모리 잡아준다
        pOneLine = (char *) malloc ( MAX_ONELINE );

        // 한줄씩 파일의 끝까지 읽어들여 리스트 완성시킴
        while ( fgets ( pOneLine, MAX_ONELINE, pOPTAB_File ) != 0 )
        {
                pInst = strtok ( pOneLine, SEPERATOR );
                pValue = strtok ( NULL, SEPERATOR );
                nType = atoi ( strtok ( NULL, SEPERATOR ) );

                OPTAB               *pNew;

                // pNew에 메모리 할당
                pNew = new OPTAB ( pInst, pValue, nType );

                // 테이블 리스트에 연결하는 부분
                if ( pOPTAB_Header == NULL )
                {
                        // 헤더가 NULL일경우. 아직 리스트에 아무것도 없는 경우처리
                        pOPTAB_Header = pNew;
                        pOPTAB_Tail = pNew;
                }
                else
                {
                        // 기존 리스트 끝에 연결하는 부분
                        pOPTAB_Tail->pNext = pNew;
                        pOPTAB_Tail = pNew;
                }
        }

        // pOneLine 메모리 해제
        free ( pOneLine );
}

////////////////////////////////////////////////////////////////////////
// OPTAB 생성자 오버로딩 멤버 함수
// pInst : Instruction의 포인터를 받아들이는 인자
// pValue : Instruction의 값의 포인터를 받아들이는 인자
// nType : Instruction의 타입을 받아들이는 인자
// 기능 : 주어진 인자가지고 객체를 초기화해서 생성해 준다.
OPTAB::OPTAB ( char *pInst, char *pValue, int nTyp )
{
        strName = (char *) malloc ( strlen ( pInst ) + 1 );

        // 데이터 입력 하는 부분
        strcpy ( xValue, pValue );
```

```
	strcpy ( strName, pInst );
	nType = nTyp;
	pNext = NULL;
}

////////////////////////////////////////////////////////////////////////
// OPTAB 소멸자
// 기능 : 생성할때 할당한 메모리를 해제한다.
OPTAB::~OPTAB ( )
{
	delete ( strName );
}

////////////////////////////////////////////////////////////////////////
// SearchByName 멤버 함수
// strNa : 찾고자 하는 OPCODE이름을 받아들이는 인자
OPTAB* OPTAB::SearchByName ( char *strNa )
{
	OPTAB		*pTempOPTAB;

	pTempOPTAB = pOPTAB_Header;
	// 리스트의 끝까지 루프돌면서 원하는 값 찾음
	while ( pTempOPTAB != NULL )
	{
		if ( strcmp ( pTempOPTAB->strName, strNa ) == 0 )
		{
			return pTempOPTAB;
		}
		else
		{
			pTempOPTAB = pTempOPTAB->pNext;
		}
	}
	return NULL;
}

////////////////////////////////////////////////////////////////////////
// SearchByValue 멤버 함수
// strVal : 찾고자 하는 OPCODE의 값을 받아들이는 인자
OPTAB* OPTAB::SearchByValue ( char *strVal )
{
	OPTAB		*pTempOPTAB;

	pTempOPTAB = pOPTAB_Header;
	// 리스트의 끝까지 루프돌면서 원하는 값 찾음
	while ( pTempOPTAB != NULL )
	{
		if ( strcmp ( pTempOPTAB->xValue, strVal ) == 0 )
		{
			return pTempOPTAB;
		}
		else
		{
			pTempOPTAB = pTempOPTAB->pNext;
		}
	}
	return NULL;
}

////////////////////////////////////////////////////////////////////////
// GetName 멤버 함수
// 기능 : 멤버 변수 strName 포인터를 리턴해준다.
char *OPTAB::GetName ( )
{
	return strName;
}

////////////////////////////////////////////////////////////////////////
// GetType ( ) 멤버 함수
// 기능 : 멤버변수 nType 값을 리턴해준다.
int OPTAB::GetType ( )
{
	return nType;
}
```