&&* Assembler.h 파일

```
#ifndef _ASSEMBLER_H_
#define _ASSEMBLER_H_

////////////////////////////////////////////////////////////////////////
// Table 구조 선언부
typedef struct _inst_unit
{
	char		*strName;	// OPCODE의 이름
	char		xValue[2];	// OPCODE의 16진수 값
	int		nType;		// OPCODE의 타입
	struct _inst_unit *pNext;
} inst_unit;
typedef inst_unit	*OPTAB;

typedef struct _sym_unit
{
	char		*strName;	// SYMBOL의 이름
	int		nLocation;	// SYMBOL의 위치값
	int		nBlockNum;	// SYMBOL이 속한 블록
	struct _sym_unit  *pNext;
} sym_unit;
typedef sym_unit *SYMTAB;

typedef struct _lit_unit
{
	char		*strName;	// LITERAL의 이름
	char		*strValue;	// LITERAL의 Data
	char		cType;		// LITERAL의 타입 ex) 'C', 'X'
	int		nLocation;	// LITERAL들이 LTORG만났을때 배정되는 위치값
	struct _lit_unit  *pNext;
} lit_unit;
typedef lit_unit *LITTAB;

typedef struct _ext_unit
{
	char		*strExtSymName;	// EXTDEF, EXTREF, EXTREFED의 이름
	char		cOperator;	// EXTREFED의 덧셈인지 뺄셈인지
	int		nLength;	// EXTREFED의 수정해야할 길이
	int		nLocation;	// EXTDEF, EXTREF, EXTREFED의 위치값
	int		nBlock;		// EXTDEF, EXTREF, EXTREFED의 블록 위치
	int		nDefOrRef;	// Def, Ref, Refed인지 갖는 인자.
	struct _ext_unit  *pNext;
} ext_unit;
typedef ext_unit *EXTTAB;

////////////////////////////////////////////////////////////////////////
// Define 선언부
#define		MAX_ONELINE	70
#define		MAX_SECTION	10
#define		MAX_TOKEN_NUM	10
#define		MAX_OBJECT	10
#define		MAX_TEXTCODE	60
#define		SEPERATOR	",.WtWn "
#define		OPERATOR	"-+*/ "
#define		ALPHABET	"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
#define		NOBASE		-1
#define		INDIRECT	0
#define		IMMEDIATE	1
#define		INDEX		2
#define		BASE		3
#define		PC		4
#define		EXTENT		5
#define		LITERAL		6
#define		WHAT_OPTAB	100
#define		WHAT_SYMTAB	101
#define		WHAT_LITTAB	102
```

```
#define         WHAT_EXTTAB       103
#define         EXTDEF            -1
#define         EXTREF            -2
#define         EXTREFED          -3

////////////////////////////////////////////////////////////////////////
// 함수 선언부
// Table을 다루는 함수들
void InitOpcodeTable ( );
void AddToSYMTAB ( int nLoc, char *strLab );
void AddToLITTAB ( char *strLiteral );
void *SearchFromTable ( int nWhatTable, char *str );
void AddToEXTTAB ( char *strExt, int nDefOfRef );
void ClearTable ( int nWhatTable );
void AddRefAddrToEXTTAB ( char *strExt, int nLocation, int nLength, char cOperator );

// 스트링을 처리하고 변환하는 함수들
int MakeTokenFromInput ( char *pOut[], char *pInputString, char *strSep );
int CharToHexnum ( char  *strVal );
int RegisterToDecnum ( char *strReg );
int CalculatorExpression ( char *strExpression, int nLoc );
void Format4 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2, int nLoc );
void Format3 ( char *strObject,    char *strOpcode, char *strOperand1,
             char *strOperand2, int nLocation, int nBas );
void Format2 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2 );

// Intermediate파일과 Object파일을 만드는 함수들
void Pass1 ( char *strFile );
void Pass2 ( );

// 최종 Object를 만드는데 쓰이는 함수들
void PrintHead ( FILE *pObejctFile, char *strProgName, int nStartAddress, int nLength );
void PrintDefOrRef ( FILE *pObjectFile, int nDefOrRef );
void PrintText ( FILE *pObjectFile, char *strTextBuffer, int nStartAddress, char *strObject );
void PrintModify ( FILE *pObjectFile );
void PrintEnd ( FILE *pObejctFile, int nStartAddress );


#endif
```

&&* Assembler.cpp 파일

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Assembler.h"


/////////////////////////////////////////////////////////////////////////////
// 전역 변수 선언부
char            *pTokenOfOneLine[MAX_TOKEN_NUM];
int             nNumOfToken, nSTART = 0, nBlock = 0, nLOCCTR[MAX_SECTION] = { 0, }, nCheckError = 0;
OPTAB           pOPTAB_Header, pOPTAB_Tail;
SYMTAB          pSYMTAB_Header, pSYMTAB_Tail;
LITTAB          pLITTAB_Header, pLITTAB_Tail;
EXTTAB          pEXTTAB_Header, pEXTTAB_Tail;

/////////////////////////////////////////////////////////////////////////////
// 메인 함수 부분
int main ( int argc, char **argv )
{
        // 실행 인자 부족시 에러 처리
        if ( argc != 2 )
        {
                printf ( "Usage : hw2 <FileName>!Wn" );
                return 0;
        }

        // OPTAB을 초기화 시키는 부분
        InitOpcodeTable ( );
        Pass1 ( argv[1] );
        Pass2 ( );


        // OPTAB, SYMTAB을 지우는 부분
        ClearTable ( WHAT_OPTAB );
        ClearTable ( WHAT_SYMTAB );
        ClearTable ( WHAT_LITTAB );
        ClearTable ( WHAT_EXTTAB );

        return 0;
}


/////////////////////////////////////////////////////////////////////////////
// Pass1 함수
// 기능 : 소스를 중간파일로 만드는 함수
void Pass1 ( char *strFile )
{
        char            *pSourceOneLine, *pTempSourceOneLine;
        FILE            *pSourceFile, *pIntermediateFile;
        int             loop, nCheckExtent;
        OPTAB           pSearchReturnOPTAB;
        SYMTAB          pSearchReturnSYMTAB;
        LITTAB          pTemp_Header;



        // 실행 인자로 입력받는 파일을 연다
        pSourceFile = fopen ( strFile, "r+" );
        if ( pSourceFile == NULL )
        {
                // Source File Open 에러 발생 처리부분
                printf ( "%s File Not Founded!!!Wn", strFile );
                exit ( 1 );
        }

        // 쓸 중간파일을 연다.
```

```
	pIntermediateFile = fopen ( "Intermediate.txt", "w+" );
	if ( pIntermediateFile == NULL )
	{
		// 출력 File Open 에러 발생 처리부분
		printf ( "Intermediate File Open Error!!!₩n" );
	}

	// 한줄씩 입력받을 버퍼에 메모리 할당하는 부분
	pSourceOneLine = (char *) malloc ( MAX_ONELINE );

	// 입력받은 버퍼를 임시 보관하기 위한 버퍼에 메모리 할당하는 부분
	pTempSourceOneLine = (char *) malloc ( MAX_ONELINE );

	// 파일로 부터 한줄씩 읽어들여 온다. 파일의 끝까지
	while ( fgets ( pSourceOneLine, MAX_ONELINE, pSourceFile ) != 0 )
	{
		// pTokenOfOneLine를 초기화 해준다.
		for ( loop = 0 ; loop < MAX_TOKEN_NUM ; loop++ )
		{
			pTokenOfOneLine[loop] = NULL;
		}

		// 토큰분류할때 변형되기 때문에... 읽어 들인 한줄을 임시 저장해놓음.
		strcpy ( pTempSourceOneLine, pSourceOneLine );

		// 읽어 들인 한줄을 토큰으로 나눠서 저장해 놓음
		nNumOfToken = MakeTokenFromInput ( pTokenOfOneLine,
						pTempSourceOneLine, SEPERATOR );

		// START 토큰이면 nSTART와 nLOCCTR[nBlock]을 그값으로 초기화하는 부분
		if ( nNumOfToken > 1 && ( strcmp ( pTokenOfOneLine[1], "START" ) == 0 ) )
		{
			nLOCCTR[nBlock] = nSTART = atoi ( pTokenOfOneLine[2] );
			//continue;
		}
		else if ( nNumOfToken > 0 && ( strcmp ( pTokenOfOneLine[0], "END" ) == 0 ) )
		{
			// END 문자열이 나오면 읽기를 중단하는 부분
			// END 뒷에 LTORG 못하고 남은 Literal들 처리 해줌
			pTemp_Header = pLITTAB_Header;
			while ( pTemp_Header != NULL )
			{
				// LTORG를 만나서 배정한 것이면 배정할 필요가 없으므로 스킵한다
				if ( pTemp_Header->nLocation != -1 )
				{
					pTemp_Header = pTemp_Header->pNext;
					continue;
				}

				pTemp_Header->nLocation = nLOCCTR[nBlock];
				fprintf ( pIntermediateFile, "%04X₩t*₩t%s₩n",
					nLOCCTR[nBlock], pTemp_Header->strName );
				if ( pTemp_Header->cType == 'C' )
				{
					nLOCCTR[nBlock] += (int) strlen ( pTemp_Header->strValue );
				}
				else if ( pTemp_Header->cType == 'X' )
				{
					nLOCCTR[nBlock] += (int) strlen ( pTemp_Header->strValue ) / 2;
				}
				pTemp_Header = pTemp_Header->pNext;
			}
		}
		// CSECT 부터는 새로운 LOCCTR로 시작해야 하기 때문에 블록 변경해준다.
		else if ( nNumOfToken > 1 && ( strcmp ( pTokenOfOneLine[1], "CSECT" ) == 0 ) )
		{
			nBlock++;
		}
```

```
		// Intermediate 파일에 LOCCTR과 원본 소스를 출력해 주는 부분
		fprintf ( pIntermediateFile, "%04XWt%s", nLOCCTR[nBlock], pSourceOneLine );

		// 한 줄의 토큰 수만 큼 루프를 돌면서 OPCODE의 위치를 찾고 LOCCTR을 증가시켜주는 부분
		for ( loop = 0 ; loop < nNumOfToken ; loop++ )
		{
			// '+' 4형식 기호가 있으면 4형식 체크 및 +제거
			if ( pTokenOfOneLine[loop][0] == '+' )
			{
				nCheckExtent = 1;
				pTokenOfOneLine[loop] = strtok ( pTokenOfOneLine[loop], "+" );
			}
			else
			{
				nCheckExtent = 0;
			}

			// pTokenOfOneLine[loop]값을 OPTAB에서 찾아본다.
			if ( (pSearchReturnOPTAB = (OPTAB) SearchFromTable ( WHAT_OPTAB,
											pTokenOfOneLine[loop] ) ) != NULL )
			{
				// OPCODE 타입에 따라 LOCCTR을 증가시켜 준다.
				nLOCCTR[nBlock] += pSearchReturnOPTAB->nType;
				nLOCCTR[nBlock] += nCheckExtent;

				// OPCODE 뒤에 OPERAND가 존재하는 지 확인하고
				// OPERAND앞에 '='이 있는 경우 Literal Table에 넣어줌
				if ( ( loop + 1 < nNumOfToken ) && ( pTokenOfOneLine[loop+1][0] == '=' ) )
				{
					// LITTAB에서 이미 존재하는지 확인하고 존재 않하면
					// Literal을 LITTAB에 넣어주는 함수 호출
					if ( SearchFromTable ( WHAT_LITTAB,
								pTokenOfOneLine[loop+1] ) == NULL )
					 {
						AddToLITTAB ( pTokenOfOneLine[loop+1] );

					}
				}
				break;
			}
			else if ( strcmp ( pTokenOfOneLine[loop], "BYTE" ) == 0 )
			{
				// 변수 선언중 BYTE 선언문인 경우 뒤에 온 문자열의 갯수만큼 LOCCTR 증가
				if ( pTokenOfOneLine[loop+1][0] == 'C' )
				{
					nLOCCTR[nBlock] += ( (int) strlen ( pTokenOfOneLine[loop+1] ) - 3 );
				}
				else if ( pTokenOfOneLine[loop+1][0] == 'X' )
				{
					nLOCCTR[nBlock] += ( (int) strlen ( pTokenOfOneLine[loop+1] ) - 3 ) / 2;
				}

				break;
			}
			else if ( strcmp ( pTokenOfOneLine[loop], "WORD" ) == 0 )
			{
				// WORD로 선언한 경우에는 3바이트만 증가
				nLOCCTR[nBlock] += 3;
				break;
			}
			else if ( strcmp ( pTokenOfOneLine[loop], "RESW" ) == 0 )
			{
				// RESW인 경우 뒤에 갯수만큼 3의 배수로 LOCCTR 증가
				nLOCCTR[nBlock] += 3 * atoi ( pTokenOfOneLine[loop+1] );
				break;
			}
			else if ( strcmp ( pTokenOfOneLine[loop], "RESB" ) == 0 )
			{
				// RESB인 경위 뒤에 갯수만큼 LOCCTR 증가
```

```
                                nLOCCTR[nBlock] += atoi ( pTokenOfOneLine[loop+1] );
                                break;
                        }
                        else if ( strcmp ( pTokenOfOneLine[loop], "LTORG" ) == 0 )
                        {
                                // LTORG를 만났을 경우에는 LITTAB 안에 Literal들 각각 Location Counter들을 반영한다
                                pTemp_Header = pLITTAB_Header;
                                while ( pTemp_Header != NULL )
                                {
                                        pTemp_Header->nLocation = nLOCCTR[nBlock];
                                        fprintf ( pIntermediateFile, "%04XWt*Wt%sWn",
                                                        nLOCCTR[nBlock], pTemp_Header->strName );
                                        if ( pTemp_Header->cType == 'C' )
                                        {
                                                nLOCCTR[nBlock] += (int) strlen ( pTemp_Header->strValue );
                                        }
                                        else if ( pTemp_Header->cType == 'X' )
                                        {
                                                nLOCCTR[nBlock] += (int) strlen ( pTemp_Header->strValue ) / 2;
                                        }
                                        pTemp_Header = pTemp_Header->pNext;
                                }
                                break;
                        }
                        else if ( strcmp ( pTokenOfOneLine[loop], "EQU" ) == 0 )
                        {
                                // strOperand가 '*' 인경우는 별 처리 필요 없이 현제 Location 값과
                                // EQU Label값으로 SYMTAB에 넣으면 된다.
                                if ( strcmp ( pTokenOfOneLine[loop+1], "*" ) == 0 )
                                {
                                        if ( ( pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB,
                                                                pTokenOfOneLine[loop-1] ) ) == NULL )
                                        {
                                                AddToSYMTAB ( nLOCCTR[nBlock], pTokenOfOneLine[loop-1] );
                                        }
                                }
                                break;
                        }
                        else if ( strcmp ( pTokenOfOneLine[loop], "CSECT" ) == 0 )
                        {
                                // Symbol을 넣을때 CSECT앞에 Symbol이 전 블록 번호로 등록되어있으므로
                                // 그 Symbol을 찾아서 현재 블록으로 수정함.
                                if ( ( pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB,
                                                                        pTokenOfOneLine[loop-1] ) ) != NULL )
                                {
                                        AddToEXTTAB ( pTokenOfOneLine[loop-1], EXTDEF );
                                        pSearchReturnSYMTAB->nLocation = 0;
                                }
                                break;
                        }
                        else {
                                // 심볼을 잡아서 넣어주는 부분 코딩필요
                                if ( SearchFromTable ( WHAT_SYMTAB, pTokenOfOneLine[0] ) == NULL )
                                {
                                        AddToSYMTAB ( nLOCCTR[nBlock], pTokenOfOneLine[0] );
                                }
                        }

                }

        }

        // pSourceOneLine 메모리 해제
        free ( pSourceOneLine );

        // pTempSourceOneLine 메모리 해제
        free ( pTempSourceOneLine );

        // 열었던 파일 닫는 부분
```

```
        fclose ( pIntermediateFile );
        fclose ( pSourceFile );

        if ( nCheckError != 0 )
        {
                // Pass1상에서 Error가 발생되었다고 표시
                printf ( "Pass1 : There are %d errors!!!₩n", nCheckError );
                exit ( 1 );
        }
        else
        {
                // 프로그램작동완료 되었다고 화면에 표시!!!
                printf ( "Pass1 : Intermediate File Created!!! Thank you!!!₩n₩n" );
        }



}



////////////////////////////////////////////////////////////////////////
// Pass2 함수
// 기능 : 중간파일로 부터 오브젝트 파일을 만드는 함수
void Pass2 ( )
{
        OPTAB           pSearchReturnOPTAB;
        SYMTAB          pSearchReturnSYMTAB;
        FILE            *pIntermediateFile, *pObjectFile;
        char            *pInterOneLine, *pTempInterOneLine, strTextBuffer[70] = { '₩0', };
        int             loop, loop1, nCheckExtent, nInstructionType, nBase_Loc = NOBASE, nLocation;
        int             nOperandNum, nFlagOfRES = 0, nDefOrRef;
        char            *strObject, strAscii[3];

        nBlock = 0;
        // Path2에서 사용할 Intermediate 파일을 읽기 모드로 연다.
        pIntermediateFile = fopen ( "Intermediate.txt", "r" );
        if ( pIntermediateFile == NULL )
        {
                printf ( "Intermediate File Not Founded!!!₩n" );
                return;
        }

        // Path2에서 생성할 Object 파일을 쓰기 모드로 연다.
        pObjectFile = fopen ( "Object.txt", "w+" );
        if ( pObjectFile == NULL )
        {
                printf ( "Object File Open Error!!!₩n" );
                return;
        }

        // 한줄씩 입력받을 버퍼에 메모리 할당하는 부분
        pInterOneLine = (char *) malloc ( MAX_ONELINE );

        // Object 코드를 가지고 있을 버퍼에 메모리 할당
        strObject = (char *) malloc ( MAX_OBJECT );

        // 파일로 부터 한줄씩 읽어들여 온다. 파일의 끝까지
        while ( fgets ( pInterOneLine, MAX_ONELINE, pIntermediateFile ) != 0 )
        {
                // strObject를 초기화 해준다
                memset ( strObject, 0, MAX_OBJECT );

                // pTokenOfOneLine를 초기화 해준다.
                for ( loop = 0 ; loop < MAX_TOKEN_NUM ; loop++ )
                {
                        pTokenOfOneLine[loop] = NULL;
                }
```

```
		// 읽어 들인 한줄을 토큰으로 나눠서 저장해 놓음
		nNumOfToken = MakeTokenFromInput ( pTokenOfOneLine, pInterOneLine, SEPERATOR );

		// 첫번재 토큰은 무조건 LOCCTR이므로 nLocation에 정장해 논다.
		nLocation = nLOCCTR[nBlock] = CharToHexnum ( pTokenOfOneLine[0] );

		// 한 줄의 토큰 수만 큼 루프를 돌면서 OPCODE의 위치를 찾음.
		for ( loop = 1 ; loop < nNumOfToken ; loop++ )
		{
			// '+' 4형식 기호가 있으면 4형식 체크 및 +제거
			if ( pTokenOfOneLine[loop][0] == '+' )
			{
				nCheckExtent = 1;
				pTokenOfOneLine[loop] = strtok ( pTokenOfOneLine[loop], "+" );
			}
			else
			{
				nCheckExtent = 0;
			}

			// OPCODE인지 확인하고 OPCODE이면 처리하는 부분
			if ( (pSearchReturnOPTAB = (OPTAB) SearchFromTable ( WHAT_OPTAB, pTokenOfOneLine[loop] ) ) != NULL )
			{
				// OPCODE 타입을 가지고 그타입에 맞는 Instruction을 구성하는 함수 호출한다.
				nInstructionType = pSearchReturnOPTAB->nType;
				nOperandNum = nNumOfToken - loop - 1;

				if ( nInstructionType == 1 )
				{
					// 1형식 함수 호출
				}
				else if ( nInstructionType == 2 )
				{
					// 2형식 함수 호출
					// Operand의 갯수에 따라 Format2 함수를 적절히 호출한다
					if ( nOperandNum == 1 )
					{
						// Operand가 하나 있는 경우
						Format2 ( strObject, pSearchReturnOPTAB->strName,
							pTokenOfOneLine[loop+1], NULL );
					}
					else if ( nOperandNum == 2 )
					{
						// Operand가 두개 있는 경우
						Format2 ( strObject, pSearchReturnOPTAB->strName,
							pTokenOfOneLine[loop+1], pTokenOfOneLine[loop+2] );
					}
					else {
						// 그 밖에 Operand가 많은 경우 에러처리
						nCheckError++;
						printf ( "error : %dWt%sWt%s %sWt<= Too Many Operand!!!Wn",
							nLocation, pSearchReturnOPTAB->strName,
							pTokenOfOneLine[loop+1], pTokenOfOneLine[loop+2] );
						break;
					}

				}
				else if ( nInstructionType == 3 && nCheckExtent == 0 )
				{
					// 3형식 함수 호출
					// Operand의 갯수를 계산해서 Format3 함수에 적절히 호출한다
					if ( nOperandNum == 0 )
					{
						// Operand가 없는 경우
						Format3 ( strObject, pSearchReturnOPTAB->strName,
							NULL, NULL, nLocation, nBase_Loc );
					}
					else if ( nOperandNum == 1 )
					{
```

```
                                        // Operand가 하나 있는 경우
                                        Format3 ( strObject, pSearchReturnOPTAB->strName,
                                                pTokenOfOneLine[loop+1], NULL, nLocation, nBase_Loc );
                                }
                                else if ( nOperandNum == 2 )
                                {
                                        // Operand가 두개 있는 경우
                                        Format3 ( strObject, pSearchReturnOPTAB->strName,
                                                  pTokenOfOneLine[loop+1], pTokenOfOneLine[loop+2],
                                                        nLocation, nBase_Loc );
                                }
                                else {
                                        // 그 밖에 Operand가 많은 경우 에러처리
                                        nCheckError++;
                                        printf ( "error : %dWt%sWt%s %sWt<= Too Many Operand!!!Wn",
                                                  nLocation, pSearchReturnOPTAB->strName,
                                                  pTokenOfOneLine[loop+1], pTokenOfOneLine[loop+2] );
                                        break;
                                }
                        }
                        else if ( nInstructionType == 3 && nCheckExtent == 1 )
                        {
                                // Operand의 갯수를 계산해서 Format4를 적절히 호출한다.
                                // 4형식 함수 호출
                                if ( nOperandNum == 1 )
                                {
                                        Format4 ( strObject, pSearchReturnOPTAB->strName,
                                                  pTokenOfOneLine[loop+1], NULL, nLocation );
                                }
                                else if ( nOperandNum == 2 )
                                {
                                        Format4 ( strObject, pSearchReturnOPTAB->strName,
                                                 pTokenOfOneLine[loop+1], pTokenOfOneLine[loop+2], nLocation );
                                }
                                else
                                {
                                        // 그 밖에 Operand가 많은 경우 에러처리
                                        nCheckError++;
                                        printf ( "error : %dWt%sWt%s %sWt<= Too Many Operand!!!Wn",
                                                  nLocation, pSearchReturnOPTAB->strName,
                                                  pTokenOfOneLine[loop+1], pTokenOfOneLine[loop+2] );
                                        break;
                                }
                        }
                        // strObject가지고 Object파일에 TextRecord를 쓰는 부분
                        PrintText ( pObjectFile, strTextBuffer, nLocation, strObject );

                        // RESB나 RESW의 연속을 해제함
                        nFlagOfRES = 0;
                        break;

                }
                else if ( strcmp ( pTokenOfOneLine[loop], "BYTE" ) == 0
                        || ( strcmp ( pTokenOfOneLine[loop], "*" ) == 0 && loop < nNumOfToken - 1 ) )
                {
                        // BYTE선언의 오브젝트 코드를 생성하는 부분
                        // 캐릭터 타입은 아스키 코드값으로 출력해야하고
                        // 16진수 타입은 그냥 출력 하면 된다.
                        pTokenOfOneLine[loop+1] = strtok ( pTokenOfOneLine[loop+1], "=" );
                        if ( pTokenOfOneLine[loop+1][0] == 'C' )
                        {
                                // 문자열 하나 하나 아스키 값 출력을 처리해야함 아직 처리 못함... 쿠쿠
                                pTokenOfOneLine[loop+1] = strtok ( pTokenOfOneLine[loop+1], "C" );
                                pTokenOfOneLine[loop+1] = strtok ( pTokenOfOneLine[loop+1], "'" );
                                // 첫번째 캐릭터 문자를 strObject에 아스키값으로 넣음
                                sprintf ( strObject, "%X", pTokenOfOneLine[loop+1][0] );
                                for ( loop1 = 1 ; loop1 < strlen ( pTokenOfOneLine[loop+1] ) ; loop1++ )
                                {
                                        // 두번째 문자부터 끝문자까지 루프를 돌면서
```

```
                                        // strAscci에 아스키값으로 넣었다가 strObject뒤에 붙인다.
                                        sprintf ( strAscii, "%X", pTokenOfOneLine[loop+1][loop1] );
                                        strcat ( strObject, strAscii );
                                }
                        }
                        else
                        {
                                pTokenOfOneLine[loop+1] = strtok ( pTokenOfOneLine[loop+1], "X" );
                                pTokenOfOneLine[loop+1] = strtok ( pTokenOfOneLine[loop+1], "'" );
                                sprintf ( strObject, "%s", pTokenOfOneLine[loop+1] );
                        }
                        // strObject가지고 Object파일에 TextRecord를 쓰는 부분
                        PrintText ( pObjectFile, strTextBuffer, nLocation, strObject );
                }
                else if ( strcmp ( pTokenOfOneLine[loop], "WORD" ) == 0 )
                {
                        // Expression을 계산해서 주소값이나 결과 값을 Object코드로 출력한다
                        sprintf ( strObject, "%06X", CalculatorExpression ( pTokenOfOneLine[loop+1], nLocation ) );
                        // strObject가지고 Object파일에 TextRecord를 쓰는 부분
                        PrintText ( pObjectFile, strTextBuffer, 0, strObject );
                }
                else if ( strcmp ( pTokenOfOneLine[loop], "RESW" ) == 0 ||
                                strcmp ( pTokenOfOneLine[loop], "RESB" ) == 0 )
                {
                        // 메모리 예약 키워드인 RESW나 RESB가 나오면 뒤에 공간이 남았어도
                        // 기존에 들어있던것 Object파일에 쓰고 새로운 줄로 시작한게 한다
                        // 대신 연속된 RESW나 RESB는 한번만 개행해야 한다
                        if ( nFlagOfRES == 0 )
                        {
                                PrintText ( pObjectFile, strTextBuffer, 0, NULL );
                                nFlagOfRES = 1;
                        }
                        break;
                }
                else if ( strcmp ( pTokenOfOneLine[loop], "BASE" ) == 0 )
                {
                        // BASE 문자열이면 Base를 지정해주는 부분
                        pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, pTokenOfOneLine[loop+1] );
                        if ( pSearchReturnSYMTAB ==  NULL )
                        {
                                // Base 지정 되어 있는 심볼이 존재 하지 않는 경우 에러 처리
                                nBase_Loc = NOBASE;
                                break;

                        }
                        nBase_Loc =  pSearchReturnSYMTAB->nLocation;
                }
                else if ( strcmp ( pTokenOfOneLine[loop], "START" ) == 0 )
                {
                        // START 문장을 만나면 메인 프로그램의 헤더 정보와 EXTDEF, EXTREF를 출력함
                        PrintHead ( pObjectFile, pTokenOfOneLine[loop-1], nLocation, nLOCCTR[nBlock] );
                }
                else if ( strcmp ( pTokenOfOneLine[loop], "END" ) == 0 )
                {

                }
                else if ( strcmp ( pTokenOfOneLine[loop], "CSECT" ) == 0 )
                {
                        // 컨트롤 섹션이 변경시에는 그 컨트롤 섹션의 헤더정보와 EXTDEF와 EXTREF를 출력 해야함....
                        // 새로운 컨트롤 섹션 오브젝트 출력하기
                        PrintText ( pObjectFile, strTextBuffer, nLocation, NULL );
                        PrintModify ( pObjectFile );
                        PrintEnd ( pObjectFile, nSTART );
                        nBlock++;
                        PrintHead ( pObjectFile, pTokenOfOneLine[loop-1], nLocation, nLOCCTR[nBlock] );
                }
                else if ( ( strcmp ( pTokenOfOneLine[loop], "EXTDEF" ) == 0 ) ||
                        ( strcmp ( pTokenOfOneLine[loop], "EXTREF" ) == 0 ) )
                {
```

```
                                // EXTDEF를 만나면 뒤에 정의된 심볼들 아무 곳에서나 사용하겠다고 처리
                                // EXTREF를 만나면 뒤에 정의된 심볼들을 지금 섹션에서 사용하겠다고 처리
                                if ( strcmp ( pTokenOfOneLine[loop], "EXTDEF" ) == 0 )
                                {
                                        nDefOrRef = EXTDEF;
                                }
                                else
                                {
                                        nDefOrRef = EXTREF;
                                }

                                // EXTDEF나 EXTREF의 뒤에 선언된 것들 EXTTAB에 넣음
                                while ( pTokenOfOneLine[loop+1] != NULL )
                                {
                                        AddToEXTTAB ( pTokenOfOneLine[loop+1], nDefOrRef );
                                        loop++;
                                }

                                // Object파일에 Define Record와 Reference Record를 쓴다
                                PrintDefOrRef ( pObjectFile, nDefOrRef );
                        }
                        else
                        {
                                continue;
                        }
                        break;
                }

        }

        // 마지막 부분에 남은 Object들 출력하고 Object파일 끝부분 마무리
        PrintText ( pObjectFile, strTextBuffer, 0, NULL );
        PrintModify ( pObjectFile );
        PrintEnd ( pObjectFile, nSTART );

        // pSourceOneLine 메모리 해제
        free ( pInterOneLine );

        // strObject 메모리 해제
        free ( strObject );

        // 열었던 파일 닫는 부분
        fclose ( pIntermediateFile );
        fclose ( pObjectFile );

        if ( nCheckError != 0 )
        {
                // Pass1상에서 Error가 발생되었다고 표시
                printf ( "Pass2 : There are %d errors!!!₩n", nCheckError );
                exit ( 1 );
        }
        else
        {
                // 프로그램작동완료 되었다고 화면에 표시!!!
                printf ( "Pass2 : Object File Created!!! Thank you!!!₩n₩n" );
        }

}
```

&&* PrintingProcessing.cpp 파일

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Assembler.h"

extern EXTTAB            pEXTTAB_Header;
extern int                       nBlock;

////////////////////////////////////////////////////////////////////////
// PrintHead 함수
// pObjectFile : Object들을 쓸 파일 포인터를 받아들이는 인자
// strProgName : Object파일에 넣을 프로그램 이름을 받아들이는 인자
// nStartAddress : 헤드를 출력하는 섹션의 시작 주소를 받아들이는 인자
// nLength : 헤드를 출력하는 섹션의 프로그램의 길이를 받아들이는 인자
void PrintHead ( FILE *pObejctFile, char *strProgName, int nStartAddress, int nLength )
{
        fprintf ( pObejctFile, "H%-7s%06X%06XWn", strProgName, nStartAddress, nLength );
        fprintf ( pObejctFile, "  ^          ^        ^Wn" );

}

////////////////////////////////////////////////////////////////////////
// PrintDefAndRef 함수
// pObjectFile : Object들을 쓸 파일 포인터를 받아들이는 인자
void PrintDefOrRef ( FILE *pObjectFile, int nDefOrRef )
{

        EXTTAB                   pTempEXTTAB;
        char                     strDefBuffer[MAX_ONELINE] = { 'WO', }, strRefBuffer[MAX_ONELINE] = { 'WO', };
        char                     strTemp[MAX_ONELINE] = { 'WO', };
        static char              strSeperatorCode[MAX_ONELINE] = { 'WO', };
        int                      loop, nLenOfExt;

        // EXDEF출력 해 주는 부분
        pTempEXTTAB = pEXTTAB_Header;
        strcpy ( strDefBuffer, "D" );
        strcpy ( strRefBuffer, "R" );
        strcpy ( strSeperatorCode, " " );
        while ( pTempEXTTAB != NULL )
        {
                if ( nDefOrRef == EXTDEF && pTempEXTTAB->nDefOrRef == EXTDEF && pTempEXTTAB->nBlock == nBlock )
                {
                        // EXTDEF선언 부분이면 심볼의 이름과 주소를 붙여서 strDefBuffer에 붙여준다.
                        sprintf ( strTemp, "%s%06X", pTempEXTTAB->strExtSymName, pTempEXTTAB->nLocation );
                        strcat ( strDefBuffer, strTemp );
                        // '^' 넣고 strObject 길이만큼 공백을 뛰고...
                        nLenOfExt = strlen ( pTempEXTTAB->strExtSymName );
                        strcat ( strSeperatorCode, "^" );
                        for ( loop = 0 ; loop < nLenOfExt * 1.7 ; loop++ )
                        {
                                strcat ( strSeperatorCode, " " );
                        }
                        strcat ( strSeperatorCode, "^        " );
                }

                if ( nDefOrRef == EXTREF && pTempEXTTAB->nDefOrRef == EXTREF && pTempEXTTAB->nBlock == nBlock )
                {
                        // EXTREF선언 부분이면 심볼의 이름만 strRefBuffer에 붙여준다.
                        strcat ( strRefBuffer, pTempEXTTAB->strExtSymName );
                        // '^' 넣고 strObject 길이만큼 공백을 뛰고...
                        nLenOfExt = strlen ( pTempEXTTAB->strExtSymName );
                        strcat ( strSeperatorCode, "^" );
                        for ( loop = 0 ; loop < nLenOfExt * 1.7 ; loop++ )
                        {
                                strcat ( strSeperatorCode, " " );
                        }
```

```
                }
                pTempEXTTAB = pTempEXTTAB->pNext;
        }

        if ( nDefOrRef == EXTDEF )
        {
                // EXTDEF 출력해준다.
                fprintf ( pObjectFile, "%sWn", strDefBuffer );
        }
        else
        {
                // EXTREF 출력해준다.
                fprintf ( pObjectFile, "%sWn", strRefBuffer );
        }
        fprintf ( pObjectFile, "%sWn", strSeperatorCode );
}

////////////////////////////////////////////////////////////////////////
// PrintText 함수
// pObjectFile : Object들을 쓸 파일 포인터를 받아들이는 인자
// strTextBuffer : TextRecord를 한줄 가지고 있을 버퍼주소를 받아들이는 인자
// nStartAddress : TextRecord한줄마다 처음에 넣을 Object코드 시작부분 받아들이는 인자
// strObject : TextRecord에 넣을 Object Code를 받아들이는 인자
void PrintText ( FILE *pObjectFile, char *strTextBuffer, int nStartAddress, char *strObject )
{
        int                     nLenOfTextBuffer, nLenOfObject, loop;
        static char             strTextHeadBuffer[MAX_ONELINE] = { 'W0', };
        static char             strSeperatorCode[MAX_ONELINE] = { 'W0', };

        // strTextBuffer의 길이를 구해논다.
        nLenOfTextBuffer = (int) strlen ( strTextBuffer );

        // 빈 strTextBuffer에 strObject를 처음 붙이는 것이면 시작 주소를 strTextBuffer에 먼저 써준다.
        if ( strcmp ( strTextHeadBuffer, "" ) == 0 )
        {
                sprintf ( strTextHeadBuffer, "T%06X", nStartAddress );
                sprintf ( strSeperatorCode, " ^         ^ " );
        }

        // 섹션이 변경되었다고 신호를 주면 지금까지 저장했던 TextBuffer를 Object파일에 그냥 출력해준다.
        // 섹션 변경 신호는 strObject가 NULL이라는 것으로 주어질때라고 정한다.
        if ( strObject == NULL )
        {
                fprintf ( pObjectFile, "%7s%02X%sWn", strTextHeadBuffer, nLenOfTextBuffer / 2, strTextBuffer );
                // 밑에 '^'로 구분해주는것 출력
                fprintf ( pObjectFile, "%sWn", strSeperatorCode );
                memset ( strSeperatorCode, 'W0', MAX_ONELINE );
                memset ( strTextBuffer, 'W0', MAX_ONELINE );
                memset ( strTextHeadBuffer, 'W0', MAX_ONELINE );
        }
        else
        {
                // strObject의 길이를 구해논다.
                nLenOfObject = (int ) strlen ( strObject );

                // 기존의 TextCode에다가 새로운 strObject를 붙였을 경우 MAX_TEXTCODE를 넘는지 확인하고
                // 넘으면 기존꺼 출력하고 새로운 곳에 strObject를 붙인다.
                // 넘지 않으면 계속 기존 TextCode에다가 붙인다.
                if ( nLenOfTextBuffer + nLenOfObject <= MAX_TEXTCODE )
                {
                        strcat ( strTextBuffer, strObject );
                        strcat ( strSeperatorCode, "^" );
                        // '^' 넣고 strObject 길이만큼 공백을 뛰고...
                        for ( loop = 0 ; loop < nLenOfObject * 1.47 ; loop++ )
                        {
                                strcat ( strSeperatorCode, " " );
                        }
                }
                else
```

```
                {
                        // 새로운 strObject를 붙일경우 넘기때문에 기존에 들어있던 것을 Object파일에 출력하고
                        // 새로운 줄에다가 쓰게 만든다.
                        fprintf ( pObjectFile, "%7s%02X%sWn", strTextHeadBuffer, nLenOfTextBuffer / 2, strTextBuffer );
                        // 밑에 '^'로 구분해주는것 출력
                        fprintf ( pObjectFile, "%sWn", strSeperatorCode );
                        memset ( strSeperatorCode, 'WO', MAX_ONELINE );
                        memset ( strTextBuffer, 'WO', MAX_ONELINE );
                        memset ( strTextHeadBuffer, 'WO', MAX_ONELINE );
                        PrintText ( pObjectFile, strTextBuffer, nStartAddress, strObject );
                }
        }

}


////////////////////////////////////////////////////////////////////////
// PrintModify 함수
// pObjectFile : Object들을 쓸 파일 포인터를 받아들이는 인자
void PrintModify ( FILE *pObjectFile )
{
        EXTTAB          pTempEXTTAB;

        pTempEXTTAB = pEXTTAB_Header;
        while ( pTempEXTTAB != NULL )
        {
                if ( pTempEXTTAB->nDefOrRef == EXTREFED && pTempEXTTAB->nBlock == nBlock )
                {
                        fprintf ( pObjectFile, "M%06X%02d%c%sWn",
                                pTempEXTTAB->nLocation, pTempEXTTAB->nLength, pTempEXTTAB->cOperator,
                                  pTempEXTTAB->strExtSymName );
                        fprintf ( pObjectFile, "  ^         ^ ^Wn" );
                }
                pTempEXTTAB = pTempEXTTAB->pNext;

        }
}


////////////////////////////////////////////////////////////////////////
// PrintEnd 함수
// pObjectFile : Object들을 쓸 파일 포인터를 받아들이는 인자
// nStartAddress : 메인 섹션의 EndRecord에는 프로그램의 시작주소가 들어가는데
//                                      이 값을 받아들이는 인자
void PrintEnd ( FILE *pObejctFile, int nStartAddress )
{
        if ( nBlock > 0 )
        {
                // nStartAddress가 음수일때는 'E'표시 뒤에 리턴주소를 적지않고 Object파일에 쓴다.
                fprintf ( pObejctFile, "EWnWnWn" );
        }
        else
        {
                // nStartAddress를 넣어서 'E'+'nStartAddress'를 Object파일에 써준다.
                fprintf ( pObejctFile, "E%06XWn", nStartAddress );
                fprintf ( pObejctFile, " ^WnWn" );
        }

}
```

&&* StringProcessing.cpp 파일

```
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "Assembler.h"

////////////////////////////////////////////////////////////////////////
// 다른 파일에서 선언된 전역변수 extern으로 선언하는 부분
extern int              nNumOfToken, nBlock, nCheckError, nLOCCTR[MAX_SECTION];
extern OPTAB            pOPTAB_Header;
extern SYMTAB           pSYMTAB_Header;
extern LITTAB           pLITTAB_Header;

////////////////////////////////////////////////////////////////////////
// MakeTokenOfOneLine 함수
// pOut[] : 토큰을 나눠서 토클별 주소를 넣어 가져갈 인자
// pInputString : 문자열 한줄의 주소를 받아들이는 인자
// strSep : 어떤 구분자로 나눌지 받아들이는 인자
// 기능 : 문자열 한줄에서 토큰별로 나눠서 pTokenOfOneLine 전역 변수에 저장하는 기능
int MakeTokenFromInput ( char *pOut[], char *pInputString, char *strSep )
{

        char            *pStr;
        int             nNum;

        // 새로운 토큰 생성시 토큰넘 초기화
        nNum = 0;

        // 첫번째 토큰을 구해서 저장하는 부분
        pStr = strtok ( pInputString, strSep );
        if ( pStr != NULL )
        {
                pOut[nNum++] = pStr;
        }

        // 마지막 토큰이 나올때까지 루프돌면서 토큰을 저장하는 부분
        while ( pStr != NULL )
        {
                pStr = strtok ( NULL, strSep );
                if ( pStr == NULL )
                {
                        break;
                }
                pOut[nNum++] = pStr;
        }

        return nNum;

}

////////////////////////////////////////////////////////////////////////
// Format2 함수
// strOpcode : 문자열 명령어를 받아들이는 인자
// strOperand1 : 첫번째 Operand를 받아들이는 인자
// strOperand2 : 두번째 Operand를 받아들이는 인자
// 기능 : 2형식 Instruction을 만들어주는 기능
void Format2 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2 )
{
        OPTAB           pSearchReturnOPTAB;

        // OPTAB에서 strOpcode를 찾아서 그 값을 리턴받는다
        pSearchReturnOPTAB = (OPTAB) SearchFromTable ( WHAT_OPTAB, strOpcode );
        if ( pSearchReturnOPTAB == NULL )
        {
                // 정확하지 않은 OPCODE이므로 에러처리
                nCheckError++;
```

```
                printf ( "error : %dWt%sWt%s %s <= Not found OPCODE!!!Wn", nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2 );
                return;
        }

        // Opcode와 Operand를 조합해서 문자열을 만든다.
        sprintf ( strObject, "%s%X%X",
                pSearchReturnOPTAB->xValue, RegisterToDecnum ( strOperand1 ), RegisterToDecnum ( strOperand2 ) );

}


////////////////////////////////////////////////////////////////////////
// Format3 함수
// strObject : Format3 함수에서 생성한 코드를 받아갈 인자
// strOpcode : 문자열 명령어를 받아들이는 인자
// strOperand1 : 첫번째 Operand를 받아들이는 인자
// strOperand2 : 두번째 Operand를 받아들이는 인자
// nLocation : 현재의 LOCATION을 받아들이는 인자 For PC Relative
// nBas : BASE 레지스터 값을 받아들이는 인자 For BASE Relative
// 기능 : 주어진 인자가지고 PC Relative나 Base Relative 형식으로 Instruction을 구성 하는 기능
void Format3 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2, int nLocation, int nBas )
{
        OPTAB           pSearchReturnOPTAB;
        SYMTAB          pSearchReturnSYMTAB;
        LITTAB          pSearchReturnLITTAB;
        int             nDisp, nPC, nDestination;
        int             blnstruction[7] = { 0, };
        char            *strDisp;

        //strObject 메모리 초기화
        memset ( strObject, 'WO', sizeof ( strObject ) );

        // OPTAB에서 strOpcode를 찾아서 그 값을 리턴받는다.
        pSearchReturnOPTAB = (OPTAB) SearchFromTable ( WHAT_OPTAB, strOpcode );
        if ( pSearchReturnOPTAB == NULL )
        {
                // 정확하지 않은 OPCODE이므로 에러처리
                nCheckError++;
                printf ( "error : %dWt%sWt%s %s <= Not found OPCODE!!!Wn", nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2 );
                return;
        }

        // Operand가 하나도 없는 경우 처리하는 부분
        if ( strOperand1 == NULL && strOperand2 == NULL )
        {
                blnstruction[INDIRECT] = 1;
                blnstruction[IMMEDIATE] = 1;
                blnstruction[INDEX] = 0;
                blnstruction[PC] = 0;
                blnstruction[BASE] = 0;
                nDisp = 0;
        }
        else
        {
                // Operand1의 첫번째 #, @ 유무의 따라 Immediate, Indirect, SIC/XE 처리
                switch ( strOperand1[0] )
                {
                case '#' :
                        blnstruction[INDIRECT] = 0;
                        blnstruction[IMMEDIATE] = 1;
                        strOperand1 = strtok ( strOperand1, "#" );
                        break;
                case '@' :
                        blnstruction[INDIRECT] = 1;
                        blnstruction[IMMEDIATE] = 0;
                        strOperand1 = strtok ( strOperand1, "@" );
                        break;
                case '=' :
                        blnstruction[INDIRECT] = 1;
```

```
                blnstruction[IMMEDIATE] = 1;
                blnstruction[LITERAL] = 1;
                break;
        default :
                blnstruction[INDIRECT] = 1;
                blnstruction[IMMEDIATE] = 1;
        }

        // Operand2의 유무에 따라 Index addressing 처리
        if ( strOperand2 !=NULL && strOperand2[0] == 'X' )
        {
                // 두번째 Operand가 있으므로 Index addressing 체크
                blnstruction[INDEX] = 1;
        }
        else
        {
                blnstruction[INDEX] = 0;

        }
        blnstruction[EXTENT] = 0;

        // SYMTAB에서 strOperand1을 찾아서 그 값을 리턴받는다.
        pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, strOperand1 );
        pSearchReturnLITTAB = (LITTAB) SearchFromTable ( WHAT_LITTAB, strOperand1 );
        if ( pSearchReturnSYMTAB == NULL && pSearchReturnLITTAB == NULL )
        {
                // SYMTAB에 없는 것이므로 상수값 직접 사용
                if ( blnstruction[IMMEDIATE] == 1 && blnstruction[INDIRECT] == 0 )
                {
                        // Immediate Addressing으로 직접 값을 사용 하는 경우
                        blnstruction[PC] = 0;
                        blnstruction[BASE] = 0;
                        nDisp = atoi ( strOperand1 );
                }
                else {
                        blnstruction[PC] = 0;
                        blnstruction[BASE] = 0;
                        nDisp = 0;
                        nCheckError++;
                        printf ( "error : %dWt%sWt%s %s <= %s Symbol Not Found!!!Wn",
                                nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2, strOperand1 );
                        return;
                }
        }
        else
        {
                // SYMTAB에 존재하는 Operand이고 Immediate Addressing 방식도 아니고 Indirect 방식도 아니므로
                // PC Relative 방식이나 BASE Relative 방식을 사용한다
                // SYMTAB에서 존재하지 않는 경우가 Literal일 경우도 있으므로 이것도 처리해줘야함

                // PC값을 값을 받아서 3을 증가시켜준다.
                nPC = nLocation + pSearchReturnOPTAB->nType;

                // Symbol인지 Literal인지 구분해서 한다.
                // 우선 PC Relative 방식으로 Disp를 구해 본다.
                if ( pSearchReturnSYMTAB == NULL )
                {
                        // Operand가 Literal인 경우
                        nDestination = pSearchReturnLITTAB->nLocation;
                }
                else
                {
                        // Operand가 Symbol인 경우
                        nDestination = pSearchReturnSYMTAB->nLocation;
                }
                nDisp = nDestination - nPC;

                // PC Relative 방식이 사용 가능한 범위인지 따져보는 부분
                if ( ( -2048 < nDisp ) && ( nDisp < 2047 ) )
```

```
                        {
                                // PC Relative 방식으로 접근
                                bInstruction[BASE] = 0;
                                bInstruction[PC] = 1;

                                // Instruction을 문자열로 구성한다
                                if ( nDisp < 0 )
                                {
                                        // nDisp가 음수일경우... nDisp 16진수 값을 문자열로 바꾼뒤
                                        // 앞에 5자리가 불필요하게 'F'로 채워져 나오므로 제거하고
                                        // 앞에 제거하고 남은 3자리만 출력하게 해줌
                                        strDisp = (char *) malloc ( sizeof ( char ) * 9 );
                                        sprintf ( strDisp, "%X", nDisp );
                                        strcpy ( strDisp, &strDisp[5] );
                                        nDisp = CharToHexnum ( strDisp );
                                        sprintf ( strObject, "%02X%01X%s", CharToHexnum ( pSearchReturnOPTAB->xValue ) +
                                                        ( 2 * bInstruction[INDIRECT] ) + bInstruction[IMMEDIATE],
                                                         ( 8 * bInstruction[INDEX] ) + 2, strDisp );
                                        free ( strDisp );
                                        return;
                                }
                        }
                        else if ( nBas != NOBASE )          // BASE가 지정되어 있는지 검사
                        {
                                nDisp = nDestination - nBas;
                                if ( ( 0 <= nDisp ) && ( nDisp < 2096 ) ) {
                                        // BASE Relative 방식으로 Disp를 구해 본다.
                                        bInstruction[BASE] = 1;
                                        bInstruction[PC] = 0;
                                }
                                else
                                {
                                        // BASE가 지원하는 범위 넘어감. 처리할 방식이 없는 경우 에러발생
                                        nCheckError++;
                                        printf ( "error : %dWt%sWt%s %s <= Addressing extend Over!!!Wn",
                                                        nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2 );
                                        return;
                                }
                        }
                        else
                        {
                                // PC도 안되고 BASE도 안되므로 에러 처리
                                nCheckError++;
                                printf ( "error : %dWt%sWt%s %s <= Not Exit Proper Addressing!!!Wn",
                                                nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2 );
                                return;
                        }
                }
        }

        // Instruction을 문자열로 구성한다
        sprintf ( strObject, "%02X%01X%03X",
             CharToHexnum ( pSearchReturnOPTAB->xValue ) + ( 2 * bInstruction[INDIRECT] ) + bInstruction[IMMEDIATE],
                ( 8 * bInstruction[INDEX] ) + ( 4 * bInstruction[BASE] ) + ( 2 * bInstruction[PC] ), nDisp );
}

////////////////////////////////////////////////////////////////////////
// Format4 함수
// strObject : Format4함수에서 구성한 ObjectCode를 가져갈 인자
// strOpcode : 문자열 명령어를 받아들이는 인자
// strOperand1 : 첫번째 Operand를 받아들이는 인자
// strOperand2 : 두번째 Operand를 받아들이는 인자
// nLoc : 현재 Format4가 사용되는 Location을 받아들이는 인자
// 기능 : 주어진 인자가지고 4형식 Instruction을 만들어서 리턴해주는 기능
void Format4 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2, int nLoc )
{
        OPTAB           pSearchReturnOPTAB;
        SYMTAB          pSearchReturnSYMTAB;
        int             nDisp, nIndexAddressing = 0, nCheckImmediate = 0;
```

```
	// OPTAB에서 strOpcode를 찾아서 그 값을 리턴받는다.
	pSearchReturnOPTAB = (OPTAB) SearchFromTable ( WHAT_OPTAB, strOpcode );
	if ( pSearchReturnOPTAB == NULL )
	{
		// 정확한 OPCODE가 아니므로 오류 출력
		nCheckError++;
		printf ( "error : %dWt%sWt%s %s <= Not found OPCODE!!!Wn", nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2 );
		return;

	}

	if ( strOperand1[0] == '#' )
	{
		nCheckImmediate = 1;
		strOperand1 = strtok ( strOperand1, "#" );
	}

	if ( strOperand2 != NULL && strOperand2[0] == 'X' )
	{
		nIndexAddressing = 8;
	}

	// SYMTAB에서 strOperand1을 찾아서 그 값을 리턴받는다.
	pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, strOperand1 );
	if ( pSearchReturnSYMTAB == NULL )
	{
		if ( nCheckImmediate == 1 )
		{
			// 존재 하지 않는 심볼일경우 직접 값을 가지고 Instruction을 구성해야함
			nDisp = atoi ( strOperand1 );
		}
		else
		{
			nDisp = 0;
			if ( SearchFromTable ( WHAT_EXTTAB, strOperand1 ) == NULL )
			{
				// 찾고자 하는 심볼이 심볼테이블에서는 없었고
				// Extdef에서도 찾아서 없는 경우는 에러출력
				nCheckError++;
				printf ( "error : %dWt%sWt%s %s <= %s Symbol Not Found!!!Wn",
					nLOCCTR[nBlock], strOpcode, strOperand1, strOperand2, strOperand1 );
				return;
			}
			else
			{
				// 찾고자 하는 심볼이 심볼테이블에서는 없었지만
				// Extdef에서는 찾을 수 있기때문에 Extref에 참조되는 Addr을 연결해주는 함수 호출
				AddRefAddrToEXTTAB ( strOperand1, nLoc + 1, 5, '+' );
			}


		}
	}
	else
	{
		// 존재 하는 심볼이므로 SYMTAB에서 찾은 값가지고 Instruction을 구성해야함
		nDisp = pSearchReturnSYMTAB->nLocation;
	}

	// Instruction을 문자열로 구성해서 리턴해준다
	sprintf ( strObject, "%02X%01X%05X", CharToHexnum ( pSearchReturnOPTAB->xValue ) + 3, nIndexAddressing + 1, nDisp );

}


//////////////////////////////////////////////////////////////////////
// CharToHexnum 함수
// strVal : 16진수 문자열을 받아들이는 인자
```

```
// 기능 : 16진수를 10진수로 변환해 리턴해주는 기능
int CharToHexnum ( char   *strVal )
{

        int      nSum = 0, loop, len;
        double   x = 16;

        len = (int)strlen ( strVal ) - 1;

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

        // 10진수 결과값 반환
        return nSum;

}

//////////////////////////////////////////////////////////////////////////
// RegisterToDecnum 함수
// cReg : Register 캐릭터 문자를 받아오는 부분
// 기능 : Register 캐릭터 문자를 받아와서 대응하는 숫자로 리턴해주는 함수
int     RegisterToDecnum ( char *strReg )
{
        if ( strReg == NULL )                       return 0;
        else if ( strcmp ( strReg, "A" ) == 0 )     return 0;
        else if ( strcmp ( strReg, "X" ) == 0 )     return 1;
        else if ( strcmp ( strReg, "L" ) == 0 )     return 2;
        else if ( strcmp ( strReg, "B" ) == 0 )     return 3;
        else if ( strcmp ( strReg, "S" ) == 0 )     return 4;
        else if ( strcmp ( strReg, "T" ) == 0 )     return 5;
        else if ( strcmp ( strReg, "F" ) == 0 )     return 6;
        else if ( strcmp ( strReg, "PC" ) == 0 )    return 8;
        else if ( strcmp ( strReg, "SW" ) == 0 )    return 9;
        else
        {
                // 여기에 적절치 못한 Register가 올경우 에러 처리해줘야함
                nCheckError++;
                printf ( "error : %dWt%s Register Not found!!!Wn", nLOCCTR[nBlock], strReg );
                return -1;
        }
}

//////////////////////////////////////////////////////////////////////////
// CalculatorExpression 함수
// strExpr : EQU 뒷부분에 Operand 전체를 받아들이는 인자
// 기능 : Expression을 계산해서 주소값이나 계산 결과값을 리턴해준다.
int CalculatorExpression ( char *strOperand, int nLoc )
{
        char            *strTemp, *strTemp1;
        char            *pTokenOfOperand[MAX_TOKEN_NUM], *pTokenOfOperator[MAX_TOKEN_NUM];
        int             nNumOfOperand = 0, nNumOfOperator= 0, loop, result = 0, value;
        SYMTAB          pSearchReturnSYMTAB;

        // strOperand를 유지하기위해 Temp변수에 copy해서 사용한다.
        strTemp = (char *) malloc ( strlen ( strOperand ) + 1 );

        // strTemp에서 Operand만 구해서 pTokenOfOperand에 넣는다.
```

```
	strcpy ( strTemp, strOperand );
	nNumOfOperand = MakeTokenFromInput ( pTokenOfOperand, strTemp, OPERATOR );

	// strTemp에서 Operator만 구해서 pTokenOfOperator에 넣는다.
	strTemp1 = (char *) malloc ( strlen ( strOperand ) + 1 );
	strcpy ( strTemp1, strOperand );
	nNumOfOperator = MakeTokenFromInput ( pTokenOfOperator, strTemp1, ALPHABET);

	if ( nNumOfOperand == 1 )
	{
		// Operand가 하나 인 경우는 그냥 그 Operand의 주소값을 사용함
		if ( ( pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, pTokenOfOperand[0] ) ) != NULL )
		{
			// Temp 메모리 해제
			free ( strTemp );
			free ( strTemp1 );
			return pSearchReturnSYMTAB->nLocation;
		}
		else
		{
			// EXTTAB에 Extdef되어있는 심볼일 경우 Extref 처리해줌
			// Operand에 맞는 Symbol이 없는 경우 0 리턴 Temp 메모리 해제
			if ( SearchFromTable ( WHAT_EXTTAB, pTokenOfOperand[0] ) == NULL )
			{
				// 찾는 기본 심볼에도 없고 Extdef도 없으면 에러처리
				nCheckError++;
				printf ( "error : %dWt%s <= %s Not found Extref!!!Wn",
						nLOCCTR[nBlock], strOperand, pTokenOfOperand[0] );
				return 0;

			}
			AddRefAddrToEXTTAB ( pTokenOfOperand[0], nLoc, 6, '+' );
			free ( strTemp );
			free ( strTemp1 );
			return 0;
		}
	}
	else
	{
		nNumOfOperand--;
		nNumOfOperator--;
		// Operand가 여러개 인 경우 처리하는 부분 계산이 조금 필요함
		// Operator수 만큼 사칙 연산해야 하므로 Operator 수 만큼 루프 돈다
		if ( ( pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, pTokenOfOperand[nNumOfOperand] ) ) != NULL )
		{
			value = pSearchReturnSYMTAB->nLocation;
		}
		else
		{
			// EXTTAB에 Extdef되어있는 심볼일 경우 Extref 처리해줌
			// Operand에 맞는 Symbol이 없는 경우 0 리턴 Temp 메모리 해제
			if ( SearchFromTable ( WHAT_EXTTAB, pTokenOfOperand[nNumOfOperand] ) == NULL )
			{
				// 찾는 기본 심볼에도 없고 Extdef도 없으면 에러처리
				nCheckError++;
				printf ( "error : %dWt%s <= %s Not found Extref!!!Wn",
					nLOCCTR[nBlock], strOperand, pTokenOfOperand[nNumOfOperand] );
				return 0;

			}
			AddRefAddrToEXTTAB ( pTokenOfOperand[nNumOfOperand--], nLoc, 6, *pTokenOfOperator[nNumOfOperator] );

		}

		for ( loop = nNumOfOperator ; loop >= 0 ; loop-- )
		{
			if ( ( pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, pTokenOfOperand[nNumOfOperand] ) ) == NULL )
			{
				// EXTTAB에 Extdef되어있는 심볼일 경우 Extref 처리해줌
```

```
                                // Operand에 맞는 Symbol이 없는 경우 0 리턴 Temp 메모리 해제
                                if ( SearchFromTable ( WHAT_EXTTAB, pTokenOfOperand[nNumOfOperand] ) == NULL )
                                {
                                        // 찾는 기본 심볼에도 없고 Extdef도 없으면 에러처리
                                        nCheckError++;
                                        printf ( "error : %dWt%s <= %s Not found Extref!!!Wn",
                                                nLOCCTR[nBlock], strOperand, pTokenOfOperand[nNumOfOperand] );
                                        return 0;

                                }
                                if ( nNumOfOperator == 0 )
                                {
                                        AddRefAddrToEXTTAB ( pTokenOfOperand[nNumOfOperand--], nLoc, 6, '+' );
                                }
                                else
                                {
                                        AddRefAddrToEXTTAB ( pTokenOfOperand[nNumOfOperand--], nLoc, 6,
                                                        *pTokenOfOperator[nNumOfOperator--] );
                                        free ( strTemp );
                                        free ( strTemp1 );
                                }
                                return 0;
                        }
                        result = pSearchReturnSYMTAB->nLocation;
                        switch ( pTokenOfOperator[loop][0] )
                        {
                        case '-':
                                result -= value;
                                break;
                        case '+':
                                result += value;
                                break;
                        case '*':
                                result *= value;
                                break;
                        case '/':
                                result /= value;
                                break;
                        }
                        value = result;
                }

                // Temp 메모리 해제
                free ( strTemp );
                free ( strTemp1 );
                return result;
        }

}
```

&&* TableProcessing.cpp 파일

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Assembler.h"

extern OPTAB		pOPTAB_Header, pOPTAB_Tail;
extern SYMTAB		pSYMTAB_Header, pSYMTAB_Tail;
extern LITTAB		pLITTAB_Header, pLITTAB_Tail;
extern EXTTAB		pEXTTAB_Header, pEXTTAB_Tail;
extern int		nBlock, nLOCCTR[MAX_SECTION], nCheckError;

////////////////////////////////////////////////////////////////////////
// InitOpcodeTable 함수
// 기능 : 테이블을 OPTAB 파일에서 자료를 읽어들여와 초기화 하다.
void 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!!!₩n" );
		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 = (OPTAB) malloc ( sizeof ( inst_unit ) );
		pNew->strName = (char *) malloc ( strlen ( pInst ) + 1 );

		// 데이터 입력 하는 부분
		strcpy ( pNew->xValue, pValue );
		strcpy ( pNew->strName, pInst );
		pNew->nType = nType;
		pNew->pNext = NULL;

		// 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 );
}
```

```
//////////////////////////////////////////////////////////////////////
// AddToSYMTAB 함수
// nLoc : 레이블의 Location 값을 받아들이는 인자
// strLab : 레이블의 이름을 받아들이는 인자
// 기능 : 주어진 문자열과 nLoc를 심볼테이블에 넣어준다
void AddToSYMTAB ( int nLoc, char *strLab )
{
	SYMTAB		pNew;

	// pNew에 메모리 할당
	pNew = (SYMTAB) malloc ( sizeof ( sym_unit ) );
	pNew->strName = (char *) malloc ( strlen ( strLab ) + 1 );

	// 데이터 입력 하는 부분
	strcpy ( pNew->strName, strLab );
	pNew->nLocation = nLoc;
	pNew->nBlockNum = nBlock;
	pNew->pNext = NULL;

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

}

//////////////////////////////////////////////////////////////////////
// AddToLITTAB 함수
// strLiteral : Literal로 판정된 Operand 전체를 넘겨받는 인자
// 기능 : strLiteral로 넘겨받은 Literal을 파싱해서 LITTAB에 자료별로 넣는다
void AddToLITTAB ( char *strLiteral )
{
	LITTAB	pNew;

	// pNew에 메모리 할당
	pNew = (LITTAB) malloc ( sizeof ( lit_unit ) );

	// Literal 전체를 나중에 찾기 위해 보관해둠
	pNew->strName = (char *) malloc ( strlen ( strLiteral ) + 1 );
	strcpy ( pNew->strName, strLiteral );

	// strLiteral에서 '='을 제거해 주는 부분
	strLiteral = strtok ( strLiteral, "=" );

	// Literal의 Type을 넣어주는 부분
	pNew->cType = strLiteral[0];

	// Literal의 Value를 넣어주는 부분
	// strLiteral에서 Type을 제거해주고 작은 따옴표도 없애주고서 Value값만 넣음
	strLiteral = strtok ( strLiteral, "CX" );
	strLiteral = strtok ( strLiteral, "'" );
	pNew->strValue = (char *) malloc ( strlen ( strLiteral ) + 1 );
	strcpy ( pNew->strValue, strLiteral );

	// LTORG에 의해 배정 되었는지 안되었는지 구분하기 위해
	// nLocation을 기본 -1을 넣어놈.
	pNew->nLocation = -1;

	pNew->pNext = NULL;
```

```
        // LITTAB에 붙여 주는 부분
        if ( pLITTAB_Header == NULL )
        {
                // 헤더가 NULL일경우. 아직 리스트에 아무것도 없는 경우처리
                pLITTAB_Header = pNew;
                pLITTAB_Tail = pNew;
        }
        else
        {
                // 기존 리스트 끝에 연결하는 부분
                pLITTAB_Tail->pNext = pNew;
                pLITTAB_Tail = pNew;
        }
}


////////////////////////////////////////////////////////////////////////
// SearchFromTable 함수
// nWhatTable : str을 검색할 테이블이 무엇인지 받아들이는 인자
// str : 찾을 문자열을 받아들이는 인자
// 기능 : 찾고자 하는 테이블에서 찾고자 하는 문자열을 검색해서
//                      찾는 결과과 있으면 그 주소값을 리턴하고 아니면 NULL리턴
void *SearchFromTable ( int nWhatTable, char *str )
{
        OPTAB           pTempOPTAB;
        SYMTAB          pTempSYMTAB;
        LITTAB          pTempLITTAB;
        EXTTAB  pTempEXTTAB;

        // 헤더를 움직이면서 찾아야 하기때문에
        // 대신 움직일 임시변수 배정 무슨테이블이냐에 따라 다르게 배정
        switch ( nWhatTable )
        {
        case WHAT_OPTAB :
                pTempOPTAB = pOPTAB_Header;
                // 리스트의 끝까지 루프돌면서 원하는 값 찾음
                while ( pTempOPTAB != NULL )
                {
                        if ( strcmp ( pTempOPTAB->strName, str ) == 0 )
                        {
                                return pTempOPTAB;
                        }
                        else
                        {
                                pTempOPTAB = pTempOPTAB->pNext;
                        }
                }
                return NULL;
        case WHAT_SYMTAB :
                pTempSYMTAB = pSYMTAB_Header;
                // 리스트의 끝까지 루프돌면서 원하는 값 찾음
                while ( pTempSYMTAB != NULL )
                {
                        if ( strcmp ( pTempSYMTAB->strName, str ) == 0 && pTempSYMTAB->nBlockNum == nBlock )
                        {
                                return pTempSYMTAB;
                        }
                        else
                        {
                                pTempSYMTAB = pTempSYMTAB->pNext;
                        }
                }
                return NULL;
        case WHAT_LITTAB :
                pTempLITTAB = pLITTAB_Header;
                // 리스트의 끝까지 루프돌면서 원하는 값 찾음
                while ( pTempLITTAB != NULL )
                {
                        if ( strcmp ( pTempLITTAB->strName, str ) == 0 )
```

```
                        {
                                return pTempLITTAB;
                        }
                        else
                        {
                                pTempLITTAB = pTempLITTAB->pNext;
                        }
                }
                return NULL;
        case WHAT_EXTTAB :
                pTempEXTTAB = pEXTTAB_Header;
                // 리스트의 끝까지 루프돌면서 원하는 값 찾음
                while ( pTempEXTTAB != NULL )
                {
                        if ( strcmp ( pTempEXTTAB->strExtSymName, str ) == 0 && pTempEXTTAB->nDefOrRef == EXTDEF
)
                        {
                                return pTempEXTTAB;
                        }
                        else
                        {
                                pTempEXTTAB = pTempEXTTAB->pNext;
                        }
                }
                return NULL;
        }

        return NULL;
 }



 ////////////////////////////////////////////////////////////////////////
 // ClearTable 함수
 // nWhatTable : 어떤 테이블을 Clear할지 받아들이는 인자
 // 기능 : 모든 자료들을 삭제하고 메모리 해제해줌
 void ClearTable ( int nWhatTable )
 {
        OPTAB           pTempOPTAB;
        SYMTAB          pTempSYMTAB;
        LITTAB          pTempLITTAB;
        EXTTAB          pTempEXTTAB;

        switch ( nWhatTable )
        {
        case WHAT_SYMTAB :
                // 리스트의 끝까지 가면서 하나씩 메모리 해제
                while ( pSYMTAB_Header != NULL )
                {
                        pTempSYMTAB = pSYMTAB_Header;
                        pSYMTAB_Header = pSYMTAB_Header->pNext;
                        free ( pTempSYMTAB->strName );
                        free ( pTempSYMTAB );
                }
                // 리스트를 다 지우고 헤더와 테일 변수 초기화
                pSYMTAB_Header = pSYMTAB_Tail = NULL;
                break;
        case WHAT_OPTAB :
                // 리스트의 끝까지 가면서 하나씩 메모리 해제
                while ( pOPTAB_Header != NULL )
                {
                        pTempOPTAB = pOPTAB_Header;
                        pOPTAB_Header = pOPTAB_Header->pNext;
                        free ( pTempOPTAB->strName );
                        free ( pTempOPTAB );
                }
                // 리스트를 다 지우고 헤더와 테일 변수 초기화
                pOPTAB_Header = pOPTAB_Tail = NULL;
                break;
```

```
        case WHAT_LITTAB :
                // 리스트의 끝까지 가면서 하나씩 메모리 해제
                while ( pLITTAB_Header != NULL )
                {
                        pTempLITTAB = pLITTAB_Header;
                        pLITTAB_Header = pLITTAB_Header->pNext;
                        free ( pTempLITTAB->strName );
                        free ( pTempLITTAB->strValue );
                        free ( pTempLITTAB );
                }
                // 리스트를 다 지우고 헤더와 테일 변수 초기화
                pLITTAB_Header = pLITTAB_Tail = NULL;
                break;
        case WHAT_EXTTAB :
                // 리스트의 끝까지 가면서 하나씩 메모리 해제
                while ( pEXTTAB_Header != NULL )
                {
                        pTempEXTTAB = pEXTTAB_Header;
                        pEXTTAB_Header = pEXTTAB_Header->pNext;
                        free ( pTempEXTTAB->strExtSymName );
                        free ( pTempEXTTAB );
                }
                // 리스트를 다 지우고 헤더와 테일 변수 초기화
                pEXTTAB_Header = pEXTTAB_Tail = NULL;
                break;
        }
}


////////////////////////////////////////////////////////////////////////
// AddToEXTTAB 함수
// strExt : EXTREF로 선언된 문자열을 받아들이는 인자
void AddToEXTTAB ( char *strExt, int nDefOrRef )
{
        EXTTAB                  pNew;
        EXTTAB                  pSearchReturnEXTTAB;
        SYMTAB                  pSearchReturnSYMTAB;

        if ( nDefOrRef == EXTDEF )
        {
                if ( ( pSearchReturnSYMTAB = (SYMTAB) SearchFromTable ( WHAT_SYMTAB, strExt ) ) == NULL )
                {
                        // 심볼테이블에 strExt가 없으면 EXTDEF 할수 없는 심볼이므로 에러 처리
                        nCheckError++;
                        printf ( "error : %dWt%s <= Not found Symbol!!!Wn", nLOCCTR[nBlock], strExt );
                        return;
                }

                // pNew에 메모리 할당
                pNew = (EXTTAB) malloc ( sizeof ( ext_unit ) );
                pNew->strExtSymName = (char *) malloc ( strlen ( strExt ) + 1 );

                // 데이터 입력 하는 부분
                strcpy ( pNew->strExtSymName, strExt );
                pNew->nLocation = pSearchReturnSYMTAB->nLocation;
                pNew->nBlock = nBlock;
                pNew->nDefOrRef = nDefOrRef;
                pNew->pNext = NULL;
        }
        else
        {
                if ( ( pSearchReturnEXTTAB = (EXTTAB) SearchFromTable ( WHAT_EXTTAB, strExt ) ) == NULL )
                {
                        // EXTTAB에 EXTDEF된 것이 없으면 EXTREF 할 수 없는 것이므로 에러 처리
                        nCheckError++;
                        printf ( "error : %dWt%s <= Not found Extdef!!!Wn", nLOCCTR[nBlock], strExt );
                        return;
                }
```

```
                // pNew에 메모리 할당
                pNew = (EXTTAB) malloc ( sizeof ( ext_unit ) );
                pNew->strExtSymName = (char *) malloc ( strlen ( strExt ) + 1 );

                // 데이터 입력 하는 부분
                strcpy ( pNew->strExtSymName, strExt );
                pNew->nLocation = pSearchReturnEXTTAB->nLocation;
                pNew->nBlock = nBlock;
                pNew->nDefOrRef = nDefOrRef;
                pNew->pNext = NULL;
        }

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

}


////////////////////////////////////////////////////////////////////////
// AddRefAddrToEXTTAB 함수
// strExt : Extref되는 심볼이름을 받아들이는 인자
// nLocation : Extref되는 심볼의 위치값을 받아들이는 인자
// cOperator : Extref되는 심볼의 주소값을 더 할 것인지 뺄것인지 받아들이는 인자
// 기능 : Extref되는 심볼들을 Extref로 선언되어있는 테이블 값에
//                사용되는 곳의 위치를 계속 이어 붙인다.
void AddRefAddrToEXTTAB ( char *strExt, int nLocation, int nLength, char cOperator )
{
        EXTTAB                  pNew;

        // pNew에 메모리 할당
        pNew = (EXTTAB) malloc ( sizeof ( ext_unit ) );
        pNew->strExtSymName = (char *) malloc ( strlen ( strExt ) + 1 );

        // 데이터 입력 하는 부분
        strcpy ( pNew->strExtSymName, strExt );
        pNew->nLocation = nLocation;
        pNew->nBlock = nBlock;
        pNew->cOperator =cOperator;
        pNew->nLength = nLength;
        pNew->nDefOrRef = EXTREFED;
        pNew->pNext = NULL;

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

}
```