숭실대학교 Soongsil University

설계(프로젝트) 보고서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다.
3. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

교과목	시스템프로그래밍
프로젝트 명	Project #1: SIC/XE 머신 어셈블러
교과목 교수	최 재 영
제출인	컴퓨터학부 학번: 20032572 출석번호: 219 성명: 박연호
제출일	2007년 5 월 10 일

설 계 채 점 표

과목명	시스템프로그래밍	학기	2007년 1학기

프로젝트 제목	Project #1: SIC/XE 머신 어셈블러
제출인	컴퓨터학부 학번: 20032572 출석번호: 219 성명: 박연호

평가내용

단계(가중치)	항목	배점	평가점수	비고
1단계 (20%)	요구사항 분석	3		
	관련연구 조사	2		
	개발환경 및 도구선정	3		
	제안일정의 적절성	2		
	소계	10		
2단계 (30%)	설계방법론 준수	3		
	자료구조의 적절성	4		
	모듈별 설계 내용	4		
	시스템 유기성	4		
	소계	15		
최종단계 (50%)	시스템 동작여부	6		
	시스템 효율성	4		
	시스템 안정성	4		
	프로그래밍 스타일	5		
	문서화	6		
	소계	25		
평가	총 계	40		등급

차 례

1장. 프로젝트 개요

1.1절 개발 배경 및 목적

07년 1학기 System Progamming시간에 SIC/XE 머신에 대해서 배우게 되었다. 머신이 어떻게 돌아가고 명령어 체계가 어떻게 되어있고 레지스터는 어떻게 이용되어지고 이용하는지에 대해서 배웠다. 하지만 이론적으로 배우면 많이 남지 않는다. 직접 어셈블러, 로더, 시뮬레이터를 만들자고 결정하고 우선 SIC/XE용 Control Section을 이용하는 어셈블러를 만들기로 하였다.

목표는 Base·PC Relative Addressing, Immediate Addressing, Indirect Addressing 등 모든 Addressing 방법을 지원하며 Literal지원과 Equate지원과 가장 중요한 Control Section 블록화 방법을 완벽히 지원하는 어셈블러를 만드는데 중점을 두겠다.

1.2절 추진체계 및 일정

■ 추진체계

- 전반적인 SIC/XE 머신에 대한 이해 확립
- 구현해야 하는 기능에 대한 파악
- 프로그램 구조에 대한 설계
- 기능별 함수 구현
- 완벽한 성능을 위한 철저한 테스팅

■ 개발 일정

◦ 개발 기간 : 2007년 4월 ~ 2007월 5월 (1개월)

구 분		4월		5월	
		3주	4주	1주	2주
설 계 단 계	자료 수집				
	자료 분석				
	기능 설계				
구 현 단 계	샘플코드제작				
	Interface설계				
	Function 설계				
	Function 코딩				
	Testing				
정 리 단 계	Debugging				
	보고서 정리				
	최종 마무리				

2장 배경 지식

2.1 주제에 관한 배경지식

- Directives

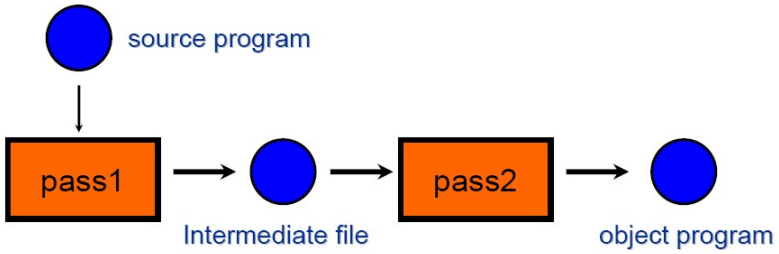
START: 프로그램의 이름과 시작주소를 정의 하는 Directives
END: 소스 프로그램의 끝을 알려주는 지시자, 옵션으로 처음 실행할 명령을 지시
BYTE: 16진수 캐리터 값을 생성하는 Directives
WORD: 한 WORD의 정수를 생성하는 Directives
RESB: 데이터 영역에 정하는 수만큼의 Byte공간을 예약하는 Directives
RESW: 데이터 영역에 정하는 수만큼의 Word공간을 예약하는 Directives
CSECT: 새로운 섹션의 시작을 알려주는 Directives
EXTDEF: 지금 섹션에서 정이되어있는 External symbol을 알려주는 Directives
EXTREF: 지금 섹션에서 다른섹션에서 선언한 Symbol을 사용하겠다고 알려주는 Directives

- Table

Operation Code Table (OPTAB) : 뉴머닉 명령 코드를 가지고 있는 테이블
Symbol Table (SYMTAB) : 레이블에 값을 저장하기 위해 사용되는 테이블
LITTAB : 리터럴 이름과 인자의 값과 길이를 가지는 테이블

- Addressing Mode

Indirect addressing (@) : 주소가 가리키는 주소의 값이 가리키는 값 참조
Immediate addressing (#) : Operand자체를 값으로 사용
Base·PC relative addressing : Base·PC Register값에 대한 상대적인 값 이용

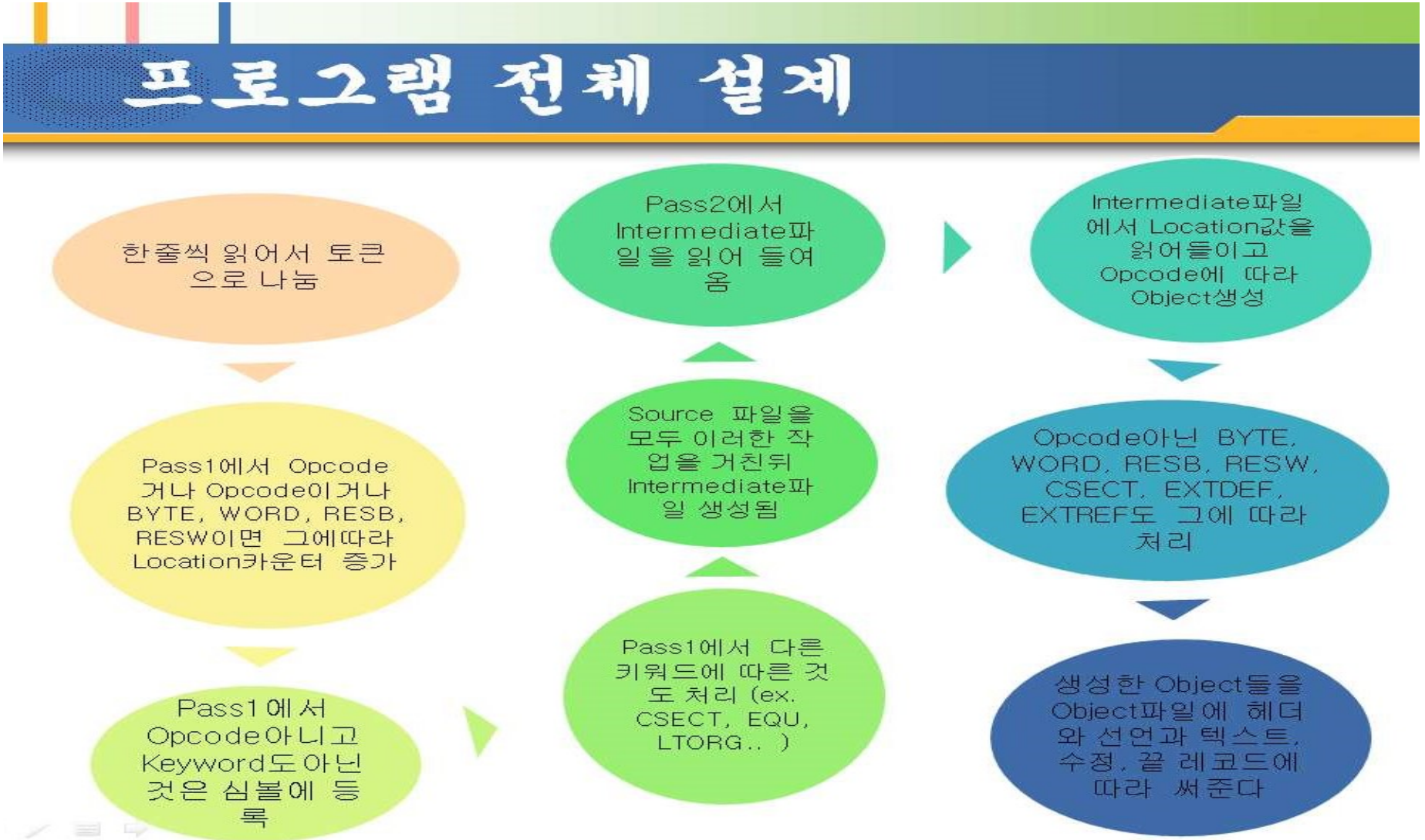


2.2 기술적 배경지식

- 파일 입·출력 다루기
- 포인터 원활하게 다루기
- 토큰별 파싱 다루기

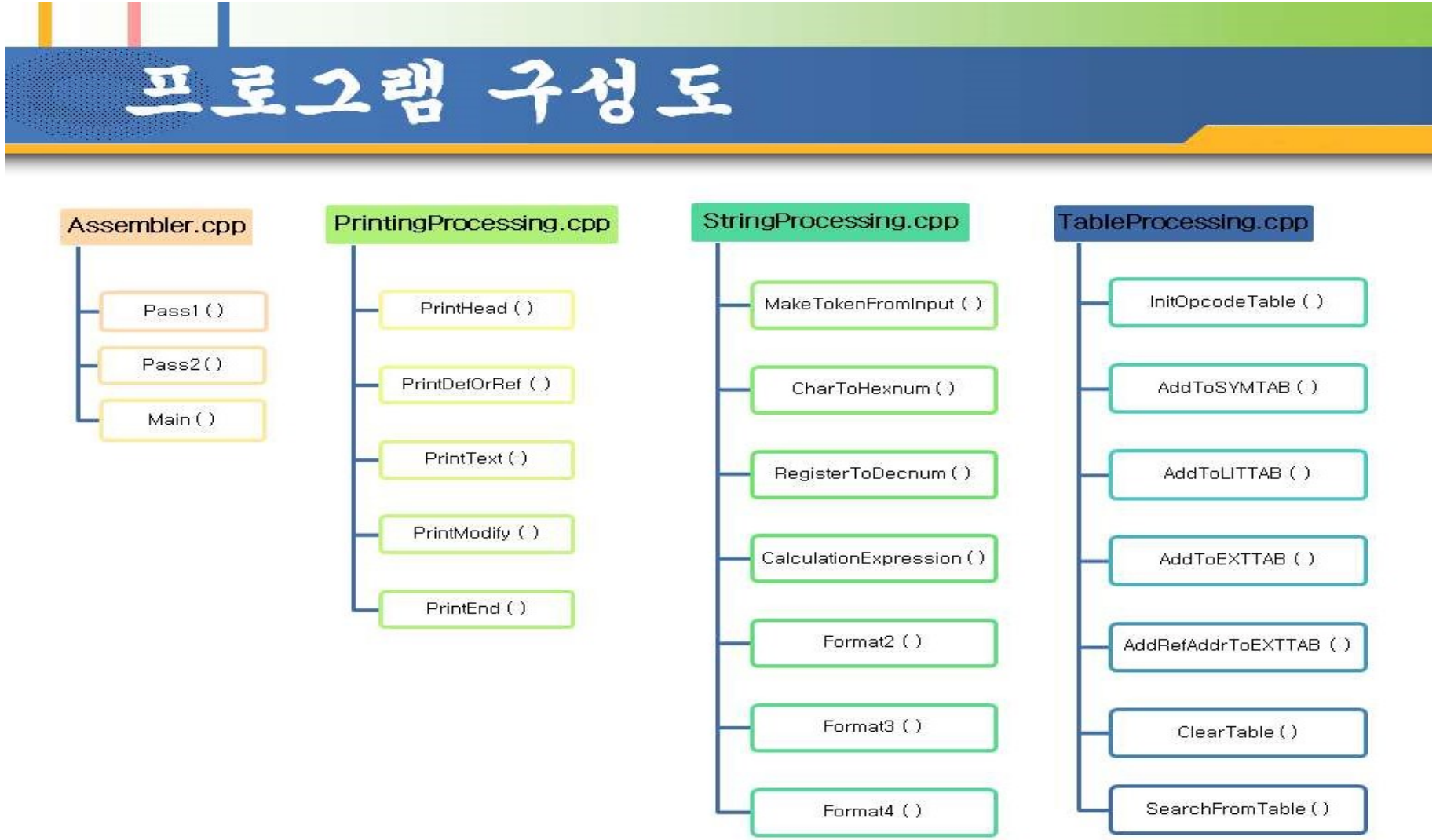
3장 시스템 설계 내용

3.1 전체 시스템 설계 내용



4장　시스템 구현 내용 (구현 화면 포함)

4.1 전체 시스템 구현 내용



```
//////////////////////////////////////////////////////////////////////
// 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!!!₩n", strFile );
        return;
    }

    // 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 )
            {
                // 이미 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;
            }
        }
```

```
    .       .       .
    .       .       .
FIRST   STL     RETADR
CLOOP   +JSUB   RDREC
        LDA     LENGTH
        COMP    #0
        JEQ     ENDFIL
        +JSUB   WRREC
        J       CLOOP
ENDFIL  LDA     =C'EOF'
        STA     BUFFER
        LDA     #3
    .       .       .
    .       .       .
```

< Source 일부분 >

소스파일을 한줄 읽어 저장할 버퍼에 메모리 할당

pSourceOneLine <= "First STL RETADR"

pTempSourceOneLine <= "FIRST STL RETADR"

입력받은 한줄을 토큰별로 저장하는 함수 호출

nNumOfToken = 3
pTokenOfOneLine[0] = "FIRST"
pTokenOfOneLine[1] = "STL"
pTokenOfOneLine[2] = "RETADR"

```
//////////////////////////////////////////////////////////////////////
// MakeTokenOfOneLine 함수
// pOut[] : 토큰을 나눠서 토큰별 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;

}
```

#define SEPERATOR ",.₩t₩n "

pTokenOfOneLine의 주소

pInputString <= "FIRST STL RETADR"

pStr => "FIRST"

pTokenOfOneLine[0] => "FIRST"

pStr => "STL"

pTokenOfOneLine[1] => "STL"

```
// 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, "%04X%t%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;
        }
        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;
        }
```

"0000 FIRST STL RETADR"이 중간파일에 써짐

nNumOfToken = 3
pTokenOfOneLine[0] = "FIRST"
pTokenOfOneLine[1] = "STL"
pTokenOfOneLine[2] = "RETADR"

두번째로 걸리는 곳
pTokenOfOneLine[1] = STL

```
////////////////////////////////////////////////////////////////////////
// MakeTokenOfOneLine 함수
// pOut[] : 토큰을 나눠서 토큰별 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;
}
```

#define SEPERATOR ",.Wt%n "

pTokenOfOneLine의 주소

pInputString
<= "FIRST STL RETADR"

pStr => "FIRST"

pTokenOfOneLine[0] => "FIRST"

pStr => "STL"

pTokenOfOneLine[1] => "STL"

```
        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, "%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;
            }
            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] );
            }
        }

    }

}

// 프로그램작동완료 되었다고 화면에 표시!!!
printf ( "Pass1 : Intermediate File Created!!! Thank you!!!\n\n" );

// pSourceOneLine 메모리 해제
free ( pSourceOneLine );

// pTempSourceOneLine 메모리 해제
free ( pTempSourceOneLine );

// 열었던 파일 닫는 부분
fclose ( pIntermediateFile );
fclose ( pSourceFile );
```

첫번째로 loop=0 일때 pTokenOfOneLine[0] = FIRST

```
//////////////////////////////////////////////////////////////////////////////
// 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;
    }

}
```

"FIRST"
FIRST STL RETADR의 Location인 '0'

```
0003    CLOOP   +JSUB   RDREC
0007            LDA     LENGTH
000A            COMP    #0
000D            JEQ     ENDFIL
0010            +JSUB   WRREC
0014            J       CLOOP
0017    ENDFIL  LDA     =C'EOF'
001A            STA     BUFFER
001D            LDA     #3
```

```
//////////////////////////////////////////////////////////////////////
// Pass2 함수
// 기능 : 중간파일로 부터 오브젝트 파일을 만드는 함수
void Pass2 ( )
{
    OPTAB       pSearchReturnOPTAB;
    SYMTAB      pSearchReturnSYMTAB;
    FILE        *pIntermediateFile, *pObjectFile;
    char        *pInterOneLine, *pTempInterOneLine, strTextBuffer[70] = { '
    int         loop, loop1, nCheckExtent, nInstructionType, nBase_Loc = NC
    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 );

        // 첫번째 토큰은 무조건 LOCCTR이므로 nLocation에 저장해 논다.
        nLocation = 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 Instruction를 구성하는 함수 호출한다.
                nInstructionType = pSearchReturnOPTAB->nType;
                nOperandNum = nNumOfToken - loop - 1;
```

pInterOneLine <= "LDA LENGTH"

pTokenOfOneLine[0] = "0007"
pTokenOfOneLine[1] = "LDA"
pTokenOfOneLine[2] = "LENGTH"
nNumOfToken = 2;

nLocation = 7

pTokenOfOneLine[1] = "LDA"

```
//////////////////////////////////////////////////////////////////////
// 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;

}
```

pTokenOfOneLing[0] = "0007"

```
    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
        }

    }
    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
        }
    }
    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 );
        }
    }
    // 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 );
        }
    }
```

nInstructionType = 3
nOperandNum = 1

nInstructionType이 3이고 nCheckExten가 0이므로 여기서 걸린다

nOperandNum이 1이었기때문에 3형식 함수 호출

strObject를 strTextBuffer에 모으고 있다 허용범위 넘으면 Object파일에 쓰는 함수

```
////////////////////////////////////////////////////////////////////////
// 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] = { '\0', };
    static char strSeperatorCode[MAX_ONELINE] = { '\0', };

    // 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 주어줄때라고 정한다.
    if ( strObject == NULL )
    {
        fprintf ( pObjectFile, "%7s%02X%s\n", strTextHeadBuffer, nLenOfTextBuffer / 2, strTextBuffer );
```

strTextBuffer에 아무것도 없으면 하는 일

```
////////////////////////////////////////////////////////////////////////
// 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 Instruction를 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         bInstruction[7] = { 0, };
    char        *strDisp;

    //strObject 메모리 초기화
    memset ( strObject, '\0', sizeof ( strObject ) );

    // OPTAB에서 strOpcode를 찾아서 그 값을 리턴받는다.
    pSearchReturnOPTAB = (OPTAB) SearchFromTable ( WHAT_OPTAB, strOpcode );
    if ( pSearchReturnOPTAB == NULL )
    {
        // 정확한 OPCODE가 아니므로 오류 출력
        return;
    }

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

        // Operand2의 유무에 따라 Index addressing 처리
        if ( strOperand2 !=NULL && strOperand2[0] == 'X' )
        {
            // 두번째 Operand가 있으므로 Index addressing 체크
            bInstruction[INDEX] = 1;
        }
        else
        {
            bInstruction[INDEX] = 0;
        }
        bInstruction[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 ( bInstruction[IMMEDIATE] == 1 && bInstruction[INDIRECT] == 0 )
            {
                // Immediate Addressing으로 직접 값을 사용 하는 경우
                bInstruction[PC] = 0;
                bInstruction[BASE] = 0;
```

"LDA" "LENGTH" "NULL" "7" "NOBASE"

Operand가 하나 있으므로 여기에 속하지 않는다. 스킵

3형식 가장 기본 형식 체크

두번째 Operand가 NULL이므로 Index Addressing 0체크함

즉시 상수값이 들어올때는 심볼테이블이나 리터럴테이블에 검색이 안되기때문에 둘다 NULL일때는 즉시상수값 처리

```
////////////////////////////////////////////////////////////////////////
// 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       ",.\t\n "
#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
```

```
            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], nLoc
            // 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[loo
            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 Object를 출력하고 Object파일 끝부분 마무리
PrintText ( pObjectFile, strTextBuffer, 0, NULL );
PrintModify ( pObjectFile );
PrintEnd ( pObjectFile, nSTART );
// 프로그램작동완료 되었다고 화면에 표시!!!
printf ( "Pass2 : Object File Created!!! Thank you!!!₩n₩n" );
// pSourceOneLine 메모리 해제
free ( pInterOneLine );
// strObject 메모리 해제
free ( strObject );
// 열었던 파일 닫는 부분
fclose ( pIntermediateFile );
fclose ( pObjectFile );
```

```
            fprintf ( pObjectFile, "%s₩n", strSeperatorCode );
            memset ( strSeperatorCode, '₩0', MAX_ONELINE );
            memset ( strTextBuffer, '₩0', MAX_ONELINE );
            memset ( strTextHeadBuffer, '₩0', MAX_ONELINE );
        }
        else
        {
            // strObject의 길이를 구해준다.
            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, "%7sX02X%s₩n", strTextHeadBuffer, nLenOfTextBuffer / 2, strTextBuffer );
                // 맨앞에 '^'로 구분해주는것 출력
                fprintf ( pObjectFile, "%s%s", strSeperatorCode );
                memset ( strSeperatorCode, '₩0', MAX_ONELINE );
                memset ( strTextBuffer, '₩0', MAX_ONELINE );
                memset ( strTextHeadBuffer, '₩0', MAX_ONELINE );
                PrintText ( pObjectFile, strTextBuffer, nStartAddress, strObject );
            }
        }
    }
}
```

nLenOfObject = 6

14+6 < 60이기때문에 strTextBuffer에 붙여쓴다

```
            nDisp = atoi ( strOperand1 );
        }
        else {
            binstruction[PC] = 0;
            binstruction[BASE] = 0;
            nDisp = 0;
        }
    }
    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 3를 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 바꿔줘
                // 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
            }
        }
        else
        {
            // PC도 안되고 BASE도 안되므로 에러 처리
        }
    }
}

// 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 );
```

"LENGTH"는 심볼테이블에 존재하기 때문에 이곳에서 계속된다.

LENGTH가 리터럴테이블에서 검색된지 아니면 심볼테이블에서 검색되었는지 확인하여 심볼과 리터럴 구분

PC Relartive Addressing 가능 범위인지 확인

nDisp가 음수인 경우 %X로 출력시... int가 4Byte이기때문에 2의 보수가 부호확장으로 인하여 F가 5개가 더 붙는다. 이 것을 제거하는 루틴. 우리는 12bit만 필요하므로..

strObject에 "032023" 넣고 끝냄

BASE 레지스터가 지정되었을시.. 그리고 PC Relartive가 불가능시...

4.2 모듈별 구현 내용

```
//////////////////////////////////////////////////////////////////////////
// Pass1 함수
// 기능: 소스를중간파일로만드는함수
void Pass1 ( char *strFile )
```

=> 이 함수는 소스파일을 한 줄 씩 읽어들어 들여서 토큰별로 나누어서 처리한다. Opcode, Symbol, BYTE, WORD, RESB, RESW, START, END, CSECT, LTORG, EQU등에 따라 분기해서 그에 따른 처리 작업을 한다. 작업완료 후에는 중간파일이 생성되는데 그 속에는 Pass2에서 사용 할 Location 값과 소스 원문 그대로 출력되게 된다.

```
//////////////////////////////////////////////////////////////////////////
// Pass2 함수
// 기능: 중간파일로부터오브젝트파일을만드는함수
void Pass2 ( )
```

=> 이 함수는 Pass1에서 작업한 중간파일을 읽어들여와서 Pass1과 같이 토큰을 나누고 Opcode 일때는 포맷 형식에 따라 오브젝트 코드를 생성하고, BYTE, WORD, RESB, RESW, START, END, CSECT, LTORG, EQU에 따라 적절한 오브젝트를 생성한다. 이렇게 생성 오브젝트들은 오브젝트파일에 Head, Define, Reference, Text, Modify, End 레코드에 맞게 출력해서 들어가게 된다.

```
//////////////////////////////////////////////////////////////////////////
// MakeTokenOfOneLine 함수
// pOut[] : 토큰을나눠서토클별주소를넣어가져갈인자
// pInputString : 문자열한줄의주소를받아들이는인자
// strSep : 어떤구분자로나눌지받아들이는인자
// 기능: 문자열한줄에서토큰별로나눠서pTokenOfOneLine 전역변수에저장하는기능
int MakeTokenFromInput ( char *pOut[], char *pInputString, char *strSep )
```

=> 이 함수는 인자를 3개 가지고 리턴을 하나 한다. 인풋 인자로서는 토큰을 나눠서 그 토큰들의 포인터를 따로 가지고 있을 pOut인자와 그리고 소스의 한줄을 입력받는 pInputString인자와 그리고 토큰을 나눌 문자열인 strSep를 입력 인자로 받아 들인다.

주어진 구분문자열인 strSep에 따라 pInputString을 토큰별로 나누어서 pOut에 포인터들을 차례차례 저장하게 된다. 그리고 나누어진 토큰의 수를 int형으로 리턴하게 된다.

```
//////////////////////////////////////////////////////////////////////////
// Format2 함수
// strOpcode : 문자열명령어를받아들이는인자
// strOperand1 : 첫번째Operand를받아들이는인자
// strOperand2 : 두번째Operand를받아들이는인자
// 기능: 2형식Instruction을만들어주는기능
void Format2 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2 )
```

=> Format2 함수는 4개의 인자를 인풋인자로 받는다. 첫 번째 인풋인자는 strObject로서 Format2 함수에서 생성된 Object를 가리키는 캐릭터형 포인터변수이고, 두 번째 인자는 명령어를 받아들이는 strOpcode이고, 세 번째 인자로는 첫 번째 Operand값을 받아들이는 strOperand1이다. 그리고 마지막 인자로는 두 번째 Operand를 받아들이는 strOperand2이다. Format2는 레지스터 대 레지스터를 처리하는 함수이므로 별다른 처리 없이 Opcode와 함께 레지스터 번호, 레지스터 번호 이렇게해서 오브젝트를 만들어서 strObject로 전달해준다.

```
//////////////////////////////////////////////////////////////////////
// Format3 함수
// strObject : Format3 함수에서생성한코드를받아갈인자
// strOpcode : 문자열명령어를받아들이는인자
// strOperand1 : 첫번째Operand를받아들이는인자
// strOperand2 : 두번째Operand를받아들이는인자
// nLocation : 현재의LOCATION을받아들이는인자For PC Relative
// nBas : BASE 레지스터값을받아들이는인자For BASE Relative
// 기능: 주어진인자가지고PC Relative나Base Relative 형식으로Instruction을구성하는기능
void Format3 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2,
int nLocation, int nBas )
```

=> Format3함수는 다른 Format함수에 비해 입력 인자가 많다. 총 6개의 인자를 받아들이게 되는데 첫 번째 인자는 Format3에서 완성된 Object코드를 받아나갈 strObject이고, 두 번째, 세 번째, 네 번째 인자는 Format2와 같이 Opcode, 그리고 첫 번째 Operand1, 두 번째 Operand2를 받아들이는 인자이다. 그리고 Format3가 불려진 Location 카운터가 들어오게 되고, 만약 Base Register가 정의되어 있다면 Base레지스터 값이 들어오게 된다.

이렇게 들어온 인자가지고 Immediate, Indirect, PC Relative, Base Relative 등 여러 어드레싱 방식으로 Distance를 계산해서 Object 코드를 생성해서 strObject에게 넘겨 준다.

```
//////////////////////////////////////////////////////////////////////
// Format4 함수
// strObject : Format4함수에서구성한ObjectCode를가져갈인자
// strOpcode : 문자열명령어를받아들이는인자
// strOperand1 : 첫번째Operand를받아들이는인자
// strOperand2 : 두번째Operand를받아들이는인자
// nLoc : 현재Format4가사용되는Location을받아들이는인자
// 기능: 주어진인자가지고4형식Instruction을만들어서리턴해주는기능
void Format4 ( char *strObject, char *strOpcode, char *strOperand1, char *strOperand2, int nLoc )
```

=> Format4 함수는 다른 Format함수와 같이 Object코드를 받아 나갈 strObject와, Opcode를 입력받는 strOpcode, 그리고 첫 번째 Operand를 받아올 strOperand1, 두 번째 Operand를 받아올 strOperand2, 그리고 Format4를 현재 명령어의 Location 카운터를 받아들이는 nLoc가 있다.

이 함수는 4형식 명령어를 처리 하기 때문에 3형식 보다는 조금 간단하고 2형식 보다는 약간 복잡하다. 처리 해줘야 할 것은 두가지이다. 하나는 그냥 Direct Addressing으로 간단하고 다른 하나는 '#'이 붙었을 경우 Immediate Addressing을 처리해주는 것이다.

```
//////////////////////////////////////////////////////////////////////
// CharToHexnum 함수
// strVal : 16진수문자열을받아들이는인자
// 기능: 16진수를10진수로변환해리턴해주는기능
int CharToHexnum ( char *strVal )
```

=> 이 함수의 인자는 하나이다. 16진수 값을 가지는 문자열이다. 이 함수는 문자열 수 만큼 루프를 돌면서 16, 16^2, 16^3.... 이런식으로 곱해서 더해서 10진수 값으로 전환 해주는 함수 이다.

```
////////////////////////////////////////////////////////////////////////
// RegisterToDecnum 함수
// cReg : Register 캐릭터문자를받아오는부분
// 기능: Register 캐릭터문자를받아와서대응하는숫자로리턴해주는함수
int     RegisterToDecnum ( char *strReg )
```
=> 이 함수는 입력 인자로서 레지스터 이름을 문자열 값으로 받아들인다. 받아들인 문자열을 비교문을 통해서 그 레지스터 문자열과 맞는 레지스터 넘버를 리턴해주게 된다.

```
////////////////////////////////////////////////////////////////////////
// InitOpcodeTable 함수
// 기능 : 테이블을 OPTAB 파일에서 자료를 읽어들여와 초기화 하다.
void InitOpcodeTable ( )
```
=> 이 함수는 OPTAB 파일을 읽어들여와서 그 속에 들어있는 명령어의 이름과 명령어의 Opcode와 명령어의 타입을 테이블에 모두 채워주는 역할을 한다.

```
////////////////////////////////////////////////////////////////////////
// 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 )
```
=> 이 함수는 Symbol Table에 함수 인자로 들어온 Location값과 Symbol이름을 가지고 기존에 존재하던 테이블뒤에다가 이어붙여 주는 함수이다.

```
////////////////////////////////////////////////////////////////////////
// AddToLITTAB 함수
// strLiteral : Literal로 판정된 Operand 전체를 넘겨받는 인자
// 기능 : strLiteral로 넘겨받은 Literal을 파싱해서 LITTAB에 자료별로 넣는다
void AddToLITTAB ( char *strLiteral )
```
=> 이 함수는 Literal Table에 함수 인자로 들어온 리터럴 전체 문자열을 파싱해서
문자열 전체는 리터럴 이름으로 집어 넣고, 파싱해서 나온 값은 그 리터럴의 값으로 넣어서
기존에 존재하던 Literal Table뒤에 이어 붙여주는 역할을 한다.

```
////////////////////////////////////////////////////////////////////////
// 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 )
```
=> 이 함수가 하는 역할은 함수의 첫 번째 인자에 따라 달라 지게 된다. 첫 번째 인자로 WHAT_SYMTAB이라고 정의된 값이 들어오면 Symbol Table에서 주어진 str과 같은 링크를 리턴해 주게 되고, WHAT_LITTAB이라고 정의된 값이 첫 번째 인자로 들어오게 되면 Liter Table에서 두 번째 인자로 주어진 문자열과 같은 값을 가지는 링크를 찾아 주소를 리턴하게 된다.
  WHAT_EXTTAB이 첫 번째 인자로 들어오면 역시나 Exteral Table에서 검색하게 된다.

```
////////////////////////////////////////////////////////////////////////
// ClearTable 함수
// nWhatTable : 어떤 테이블을 Clear할지 받아들이는 인자
// 기능 : 모든 자료들을 삭제하고 메모리 해제해줌
void ClearTable ( int nWhatTable )
```
=> 프로그램을 돌리면서 생성된 테이블에는 무수한 메모리들이 잡혀져 있다. 그렇기 때문에 프로그램이 종료될때는 사용하던 메모리를 해제해주어야 한다. 이 함수 역시 Table을 Clear해주는데 있어서 첫 번째 인자에 따라 Clear하는 테이블이 달라진다. WHAT_SYMTAB이 들어오면 Symbol Table에 선언된 메모리를 모두 해제하게 되고 WHAT_LITTAB이 들어오면 Literal Table에 선언된 메모리를 모두 해제한다. OPTAB과 EXTTAB역시 같은 방법으로 행해진다.

```
////////////////////////////////////////////////////////////////////////
// AddToEXTTAB 함수
// strExt : EXTREF로 선언된 문자열을 받아들이는 인자
void AddToEXTTAB ( char *strExt, int nDefOrRef )
```
=> Extdef또는 Extref선언되는 Symbol들을 EXTTAB에 저장하는 함수이다. Symbol Table에 저정되어있는 것들중에 소스에서 Extdef하거나 Extref하는 것들을 따로 모아두는 EXTTAB에 선언형식과 함께 선언된 블록등 정보를 담아서 EXTTAB 끝에 연결해주는 함수이다.

```
////////////////////////////////////////////////////////////////////////
// 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 )
```
=> 이 함수는 Extdef된 심볼들을 명령어에서 사용할 때 나중에 Modification 레코드에 넣을수 있게 그 사용되어지는 주소를 저장하는데 쓰인다. 첫 번째 인자는 Extref하는 문자열 값을 받아들이는 함수이고, 두 번째는 사용되어지는 곳의 Location값을 받아들이는 인자이고, 세 번째는 수정되는 레코들의 길이를 받아들이는 인자이다. 마지막으로 수정할 때 더할 것인지 뺄것인지 결정하는 것을 받아들이는 인자이다.

```
////////////////////////////////////////////////////////////////////////
// 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 )
```
=> 오브젝트 파일에 Head부분을 담당하는 함수로서 첫 번째 인자는 파일포인터 인자값이고, 두 번째 인자는 프로그램의 이름을 받아들이는 인자, 세 번째 인자는 프로그램의 시작주소값을 받아들이는 인자이다. 그리고 마지막으로 이 프로그램의 길이를 받아들이는 인자가 있다.
  받아들인 인자들을 규칙에 맞게 파일에 써주게 하는 함수이다.

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

=> Define과 Reference 레코드의 출력을 담당하는 함수이다. EXTTAB을 돌면서 두 번째 인자가 EXTDEF이면 Define된 것만 골라서 오브젝트 파일에 써주게 되고, EXTREF이면 Reference로 되어있는 것만 골라서 오브젝트 파일에 써주게 된다.

```
//////////////////////////////////////////////////////////////////////////
// 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 )
```

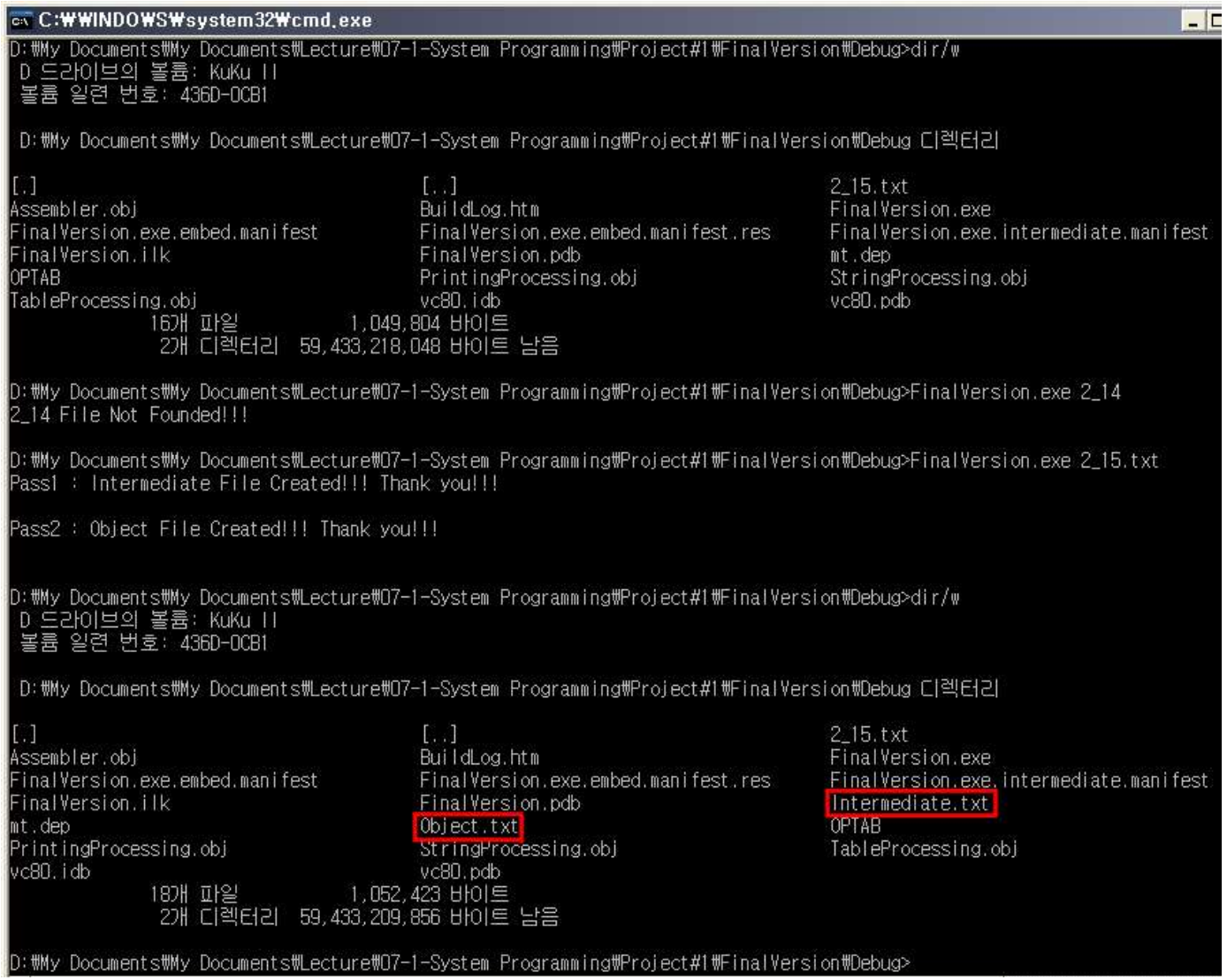
=> Object 파일에 핵심을 출력해주는 함수이다. 첫 번째 인자로는 파일포인터가 들어오고, 두 번째로는 Object 코드들을 저장하고 있을 strTextBuffer이고, 세 번째는 라인의 처음 Object 코드의 시작주소를 받아들이는 인자이고, 마지막 인자는 매번 들어오는 Object 코드이다.

Object 코드를 받아서 지금 존재하는 코드들과 더해도 정해진 크기 만큼 넘치지 않으면 계속 붙여나가는 것이고 만약 정해진 크기보다 넘어간다면 지금까지 저장했던 것을 파일에다가 쓰고 새로운 라인에 넘치던것을 다시 쓰면서 시작한다.

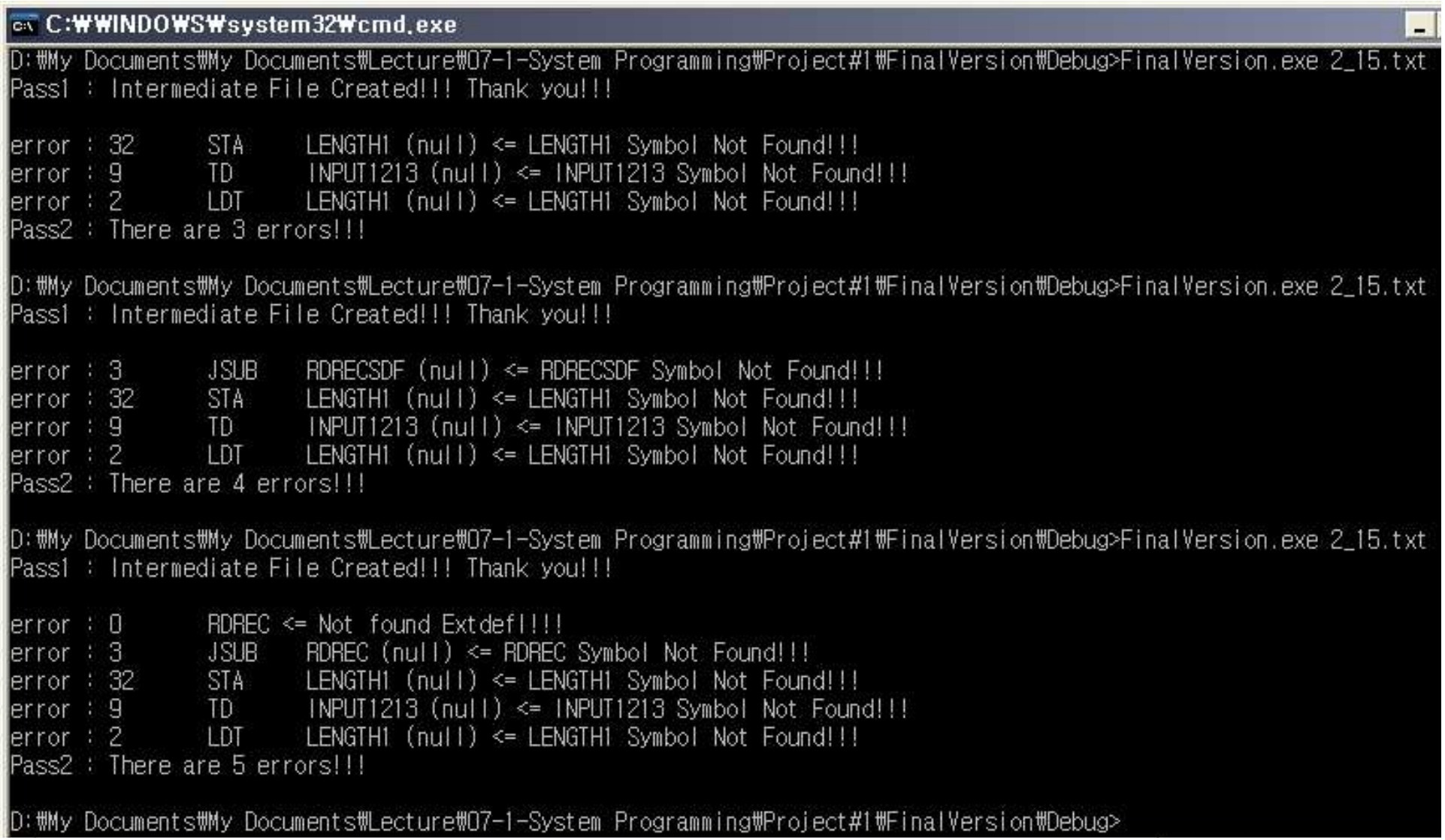
```
//////////////////////////////////////////////////////////////////////////
// PrintModify 함수
// pObjectFile : Object들을 쓸 파일 포인터를 받아들이는 인자
void PrintModify ( FILE *pObjectFile )
```

=> Modification 레코드를 출력해주는 함수로서, 그 블록에서 사용된 Extref들을 출력해주는 기능 한다.

4.3 구현 화면



< 프로그램 실행후 생긴 Intermediate파일과 Object파일 >



< 잘 못된 코드 컴파일시 오류메시지 출력해주는 모습 >

< Object파일을 열어 확인 한 모습 >

5장 결론

SIC/XE 머신의 어셈블러를 만들어 보았다. 완벽한 어셈블러는 아니었지만 그래도 머신의 명령어 체계에 대해 이해하고, 컴퓨터 내부적으로 어떻게 명령어를 해독하고 처리하는지 그리고 기계어 코드는 어떻게 만드는가에 대해 알게 되었다. 요번에 간단한 머신에 어셈블러를 만드는 것도 아니 조금 더 작은 부분을 위한 어셈블러를 만드는 것도 힘든데 전체 어셈블러를 만든 사람들을 생각하니 존경스러울 뿐이다. 이번 기회를 통해 지금껏 배운 내용에 대해 조금 더 몸으로 느낄 수 있었고, 많이 생각하고, 머리를 썼던 것 같아서 끝내고 나니 기분은 좋았다.