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 #2: SIC/XE 머신 Loader 및 Simulator
교과목 교수	최 재 영
제출인	컴퓨터학부 학번: 20032572 출석번호: 219 성명: 박연호
제출일	2007년 6 월 7 일

설 계 채 점 표

과목명	시스템프로그래밍	학기	2007년 1학기

프로젝트 제목	Project #2: SIC/XE 머신 Loader 및 Simulator
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	50		등급

차 례

1장 프로젝트 개요

1.1 개발 배경 및 목적

1.2 추진 체계 및 일정

2장 배경 지식

2.1 주제에 관한 배경지식

2.2 기술적 배경지식

3장 시스템 설계 내용

3.1 전체 시스템 설계 내용

3.2 모듈별 설계 내용

4장 시스템 구현 내용 (구현 화면 포함)

4.1 전체 시스템 구현 내용

4.2 모듈별 구현 내용

4.3 구현 화면

5장 기대효과 및 결론

첨부 프로그램 소스파일

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 만들었다.

다음으로 해야 할 것은 이전에 만든 어셈블러를 통해 생성한 오브젝트 코드들을 메모리에 올리고 오브젝트 코드가 제대로 만들어 졌는지를 확인해야 한다. 이러한 작업을 하기위해서는 로더와 시뮬레이터라는 것이 필요하다. 그래서 Linking과 Relocation이 가능한 로더와 Memory에서 오브젝트들을 가져올 Memory Management Unit과 메모리에서 가져온 Instruction을 실행하고 결과를 처리 할 Execution Unit을 가지는 GUI기반의 시뮬레이터를 구현하겠다. 추가로 오브젝트파일에 로드한 날짜를 출력해주는 간단한 Macro도 구현하겠다.

1.2절 추진체계 및 일정

■ 추진체계

- Relocation Loader에 대한 이해
- 각 Instruction에 대한 이해
- Simulator에 대한 이해
- Macro에 대한 이해
- 프로그램 구조에 대한 설계
- 기능별 함수 구현
- GUI 시뮬레이터 인터페이스 설계
- 테스팅 및 디버깅을 통한 안정성 확보

■ 개발 일정

◦ 개발 기간 : 2007년 5월 ~ 2007월 6월 (1개월)

구 분		5월			6월
		3주	4주	5주	1주
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

■ 용어

- Loading : 실행을 위해 오브젝트 프로그램을 메모리로 가져오는 것
- Relocation : 기존에 정의된 주소와 다른 주소에 오브젝트 프로그램을 올리도록 오브젝트 프로그램을 수정하는 것
- Linking : 오브젝트 프로그램끼리 서로 정보를 제공하도록 허락된 두 개 또는 그 이상의 오브젝트 프로그램들을 묶어주는 것
- Macro : 프로그래머의 반복성 편의를 위해 제공된 소스프로그램의 집합들
- MACRO : Macro 정의의 시작을 알려주는 Directive
- MEND : Macro 정의 끝을 알려주는 Directive

■ 자료 구조

- Operation Code Table (OPTAB) : 뉴머닉 명령 코드를 가지고 있는 테이블
- External Symbol Table (ESTAB) : External Symbol의 이름과 주소를 가지고 있는 테이블
- Program Load Address (PROGADDR) : Linked Program이 메모리 어디에 올리는지 나타내는 시작주소
- Control Section Address (CSADDR) : Loader에 의해 현재 읽혀지고 있는 Control Section의 시작주소

■ Addressing Mode

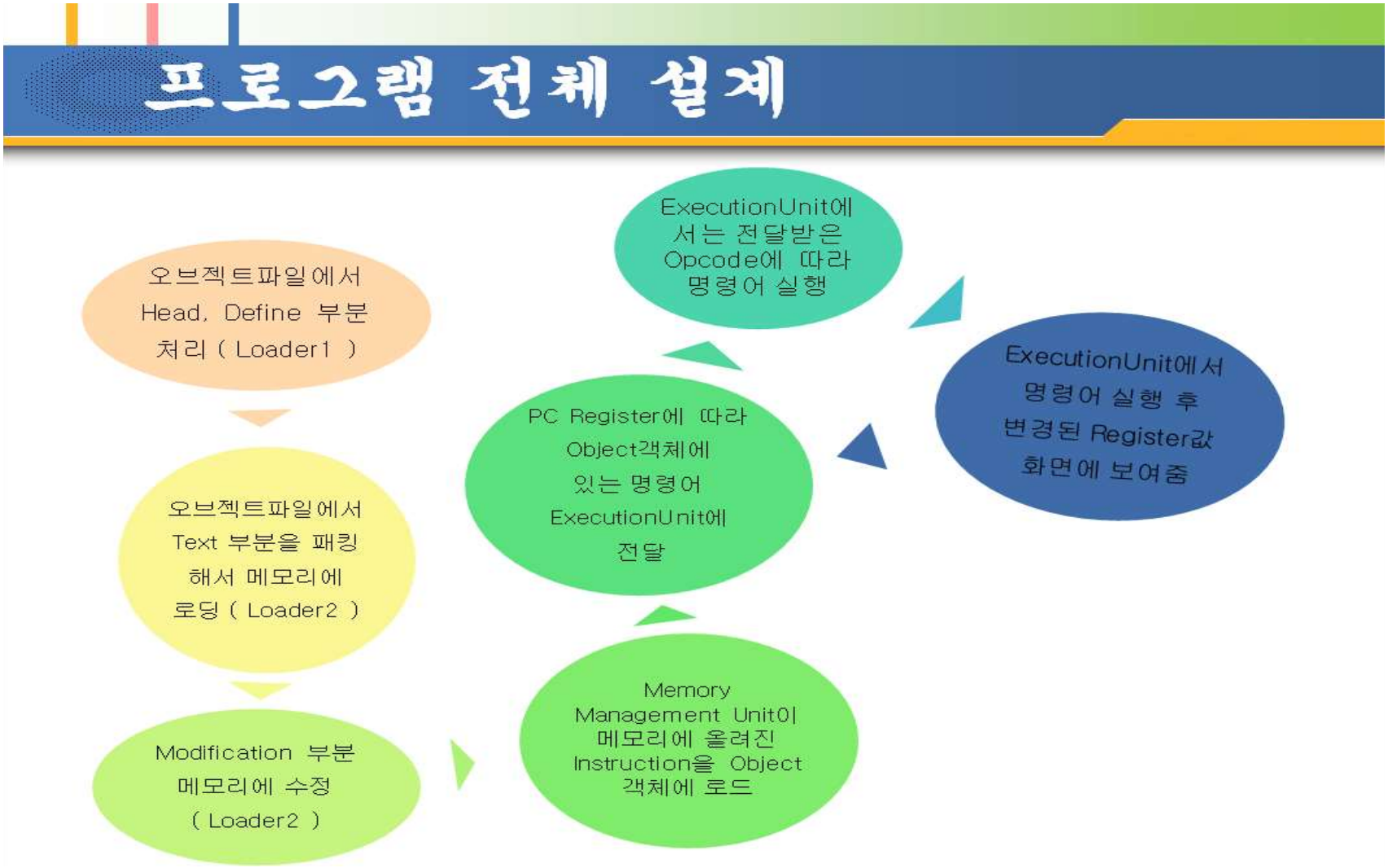
- Indirect addressing (@) : 주소가 가리키는 주소의 값이 가리키는 값 참조
- Immediate addressing (#) : Operand자체를 값으로 사용
- Base·PC relative addressing : Base·PC Register값에 대한 상대적인 값 이용

2.2 기술적 배경지식

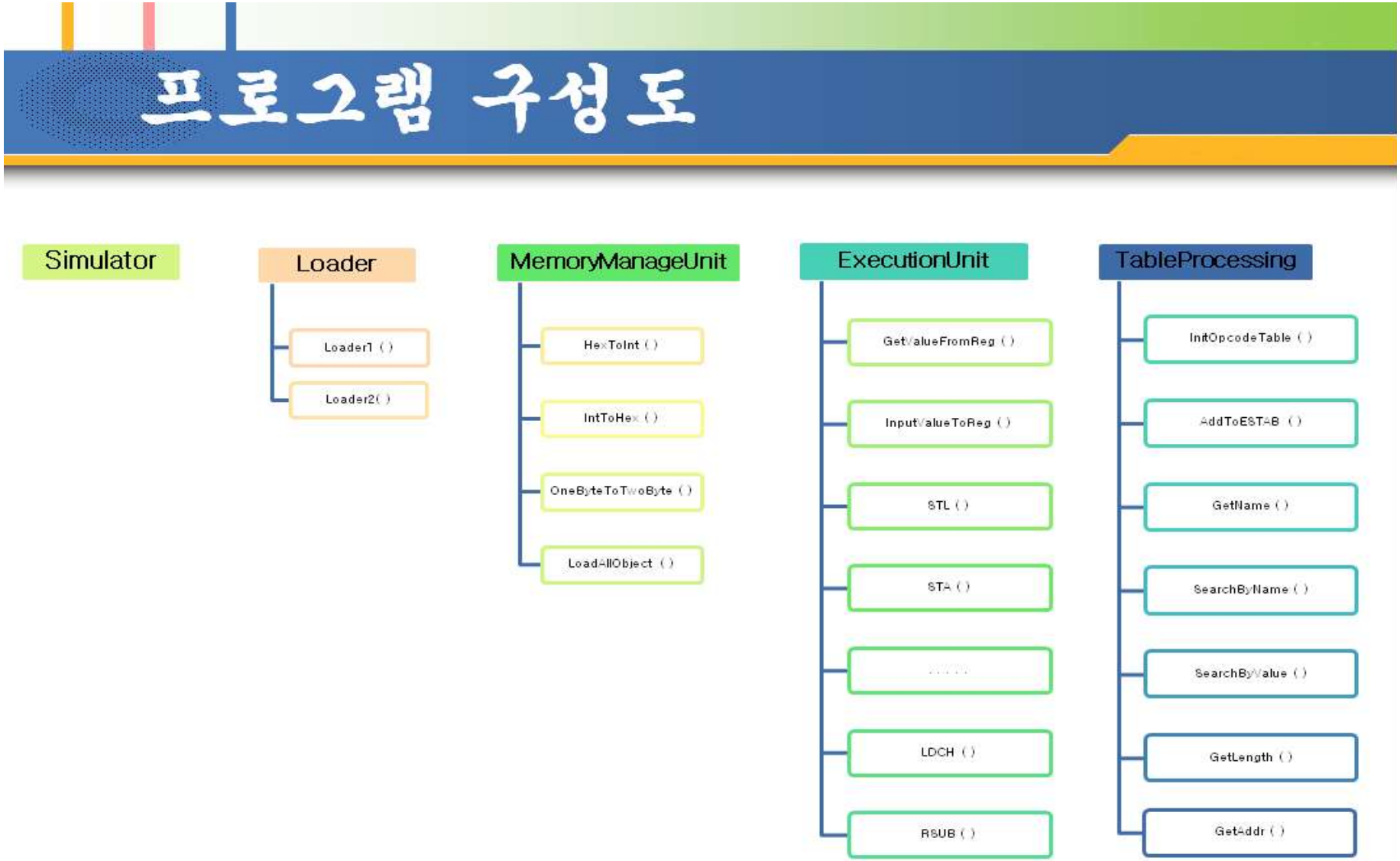
- Window API 프로그래밍
- Class 이용한 객체지향 프로그래밍 (사용하고자 노력)
- 포인터를 통한 능숙한 메모리 접근 및 사용
- 파일 접근 및 입 · 출력 다루기

3장 시스템 설계 내용

3.1 전체 시스템 설계 내용

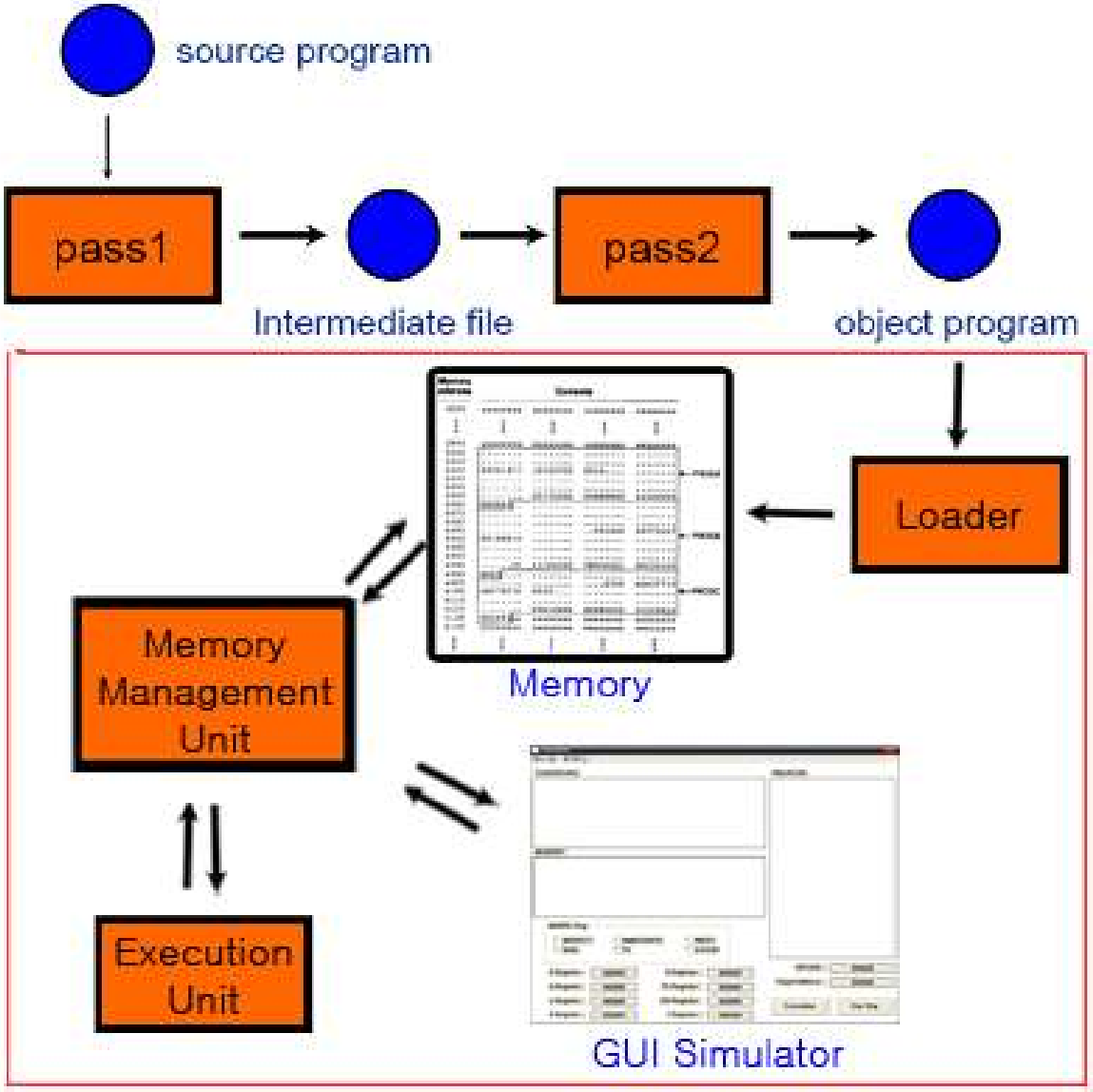


3.2 모듈별 설계 내용


프로그램 구성도
Simulator
Loader
Loader1 ( )
Loader2( )
MemoryManageUnit
HexToInt ( )
IntToHex ( )
OneByteToTwoByte ( )
LoadAllObject ( )
ExecutionUnit
GetValueFromReg ( )
InputValueToReg ( )
STL ( )
STA ( )
. . . . .
LDCH ( )
RSUB ( )
TableProcessing
InitOpcodeTable ( )
AddToESTAB ( )
GetName ( )
SearchByName ( )
SearchByValue ( )
GetLength ( )
GetAddr ( )

4장 시스템 구현 내용 (구현 화면 포함)

4.1 전체 시스템 구현 내용



< 전체 시스템 구성도 >

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

    // Exectuion한다.
    Execute ( OneObject );

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

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

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

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

OneObject – strOP = "STL"
– nDisp = 26
– nType = 3
– nNIXBPE = { 1, 1, 0, 0, 1, 0 }

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

    while ( 1 )
    {
        if ( nOffset > nLen )
        {
            return -1;
        }
        memset ( strObjectTwoByte, 'W0', 9 );
        memset ( strObjectOneByte, 'W0', 5 );

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

        // OPCODE 부분만 임시 저장해 놓는다.
        strncpy ( strOpcode, (char *)strObjectTwoByte, 2 );
        strOpcode[2] = 'W0';

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

        // Opcode 두번째 부분에서 N, I를 빼서 오리지널 Opcode만 구한다.
        nTemp1 -= ( 2 * nNIXBPE[INDIRECT] + nNIXBPE[IMMEDIATE] );
        if ( 0 <= nTemp1 && nTemp1 <= 9 )
        {
            strOpcode[1] = nTemp1 + '0';
        }
        else
        {
            strOpcode[1] = nTemp1 - 10 + 'A';
        }

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

strObjectTwoByte : "1720"

nNIXBPE[INDIRECT] = 1, nNIXBPE[IMMEDIATE] = 1

nTemp1 = 14

pSearchReturnOPTAB = "STL"

```
    nNIXBPE[INDEX] = nNIXBPE[BASE] = nNIXBPE[PC] = nNIXBPE[EXTENT] = 0;
    for ( loop = 4 ; loop > 0 ; loop-- )
    {
        nNIXBPE[loop+1] = nTemp % 2;
        nTemp /= 2;
    }
    switch ( pSearchReturnOPTAB->GetType ( ) )
    {
    case 1:
        if ( nNIXBPE[INDIRECT] == 0 && nNIXBPE[IMMEDIATE] == 0 )
        {
            nType = 1;
            strncpy ( strOP, strOpcode, 3 );
        }
        else
        {
            nOffset += 1;
            continue;
        }
        break;
    case 2:
        nType = 2;
        for ( loop = 0 ; loop < 6 ; loop++ )
        {
            nNIXBPE[loop] = 0;
        }
        strncpy ( strOP, strOpcode, 3 );
        break;
    case 3:
        if ( nNIXBPE[EXTENT] == 1 && ( nNIXBPE[BASE] == 1 || nNIXBPE[PC] == 1 ) )
        {
            nOffset += 1;
            continue;
        }
        else if ( nNIXBPE[EXTENT] == 1 && ( nNIXBPE[INDIRECT] == 1 || nNIXBPE[IMMEDIATE] == 1 ) )
        {
            nType = 4;
            strncpy ( strOP, strOpcode, 3 );
        }
        else if ( nNIXBPE[INDIRECT] == 0 && nNIXBPE[IMMEDIATE] == 0 )
        {
            nOffset += 1;
            continue;
        }
        else if ( nNIXBPE[BASE] == 1 && nNIXBPE[IMMEDIATE] == 1 )
        {
            nOffset += 1;
            continue;
        }
        else if ( nNIXBPE[INDIRECT] == 1 && nNIXBPE[INDEX] == 1 )
        {
            nOffset += 1;
            continue;
        }
        else
        {
            // 나머지는 다 3형식으로 16진수 6개 멤버 변수에 복사
            nType = 3;
            strncpy ( strOP, strOpcode, 3 );
        }
        break;
    }

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

nNIXBPE[PC] = 1, others = 0

nDisp = 27

strObjectTwoByte = "172027"

< GetObject 멤버 함수 >

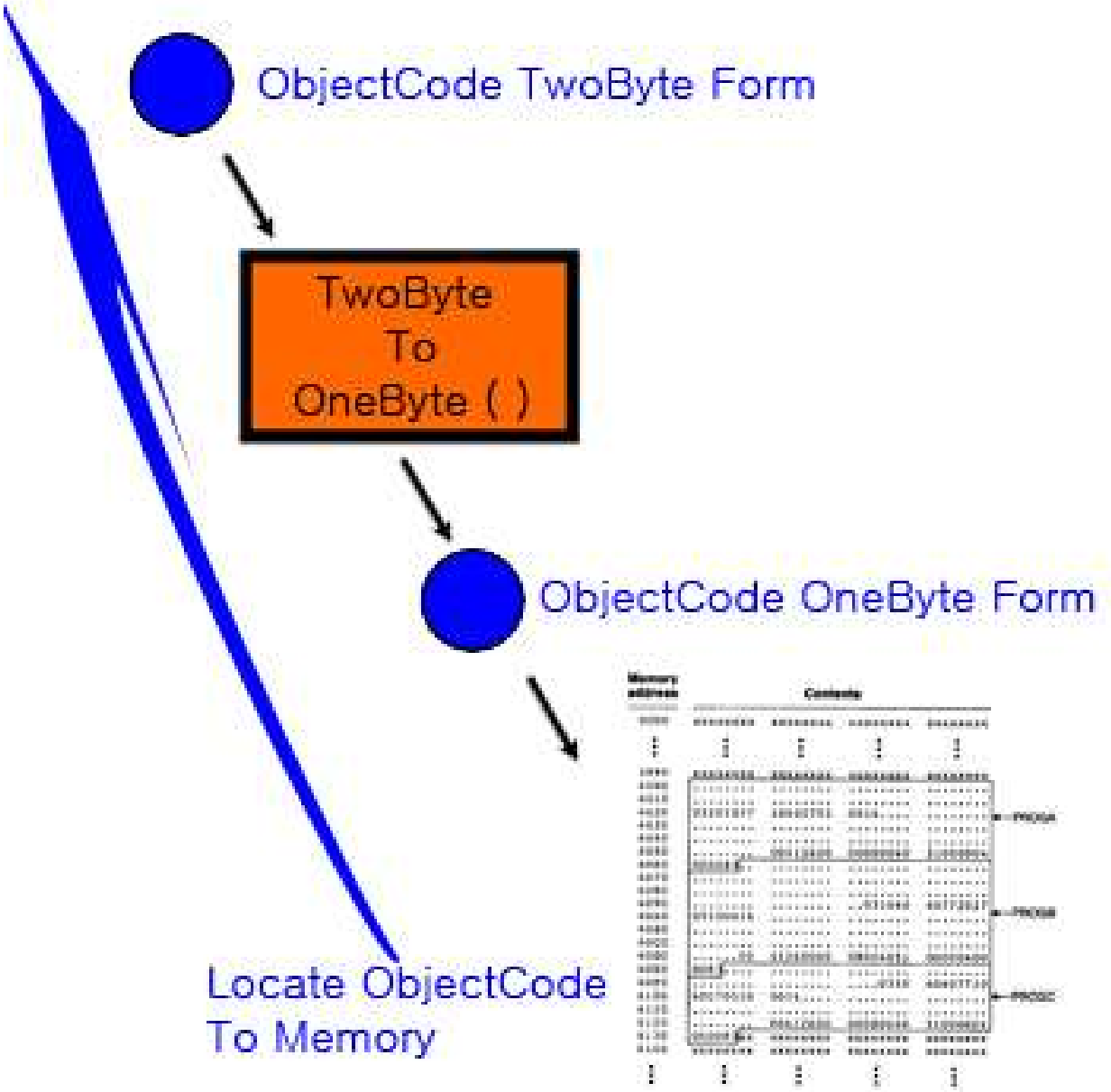
```
////////////////////////////////////////////////////////////////////////////
// STL 함수
// Instruction : STL 명령을 포함하는 전체 Instruction를 받아들이는 인자
// 기능 : L Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STL ( ObjectCode *Instruction )
{
    int         nAddr = 0;
    char        strVal[7];

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

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

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

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

< STL 함수 >

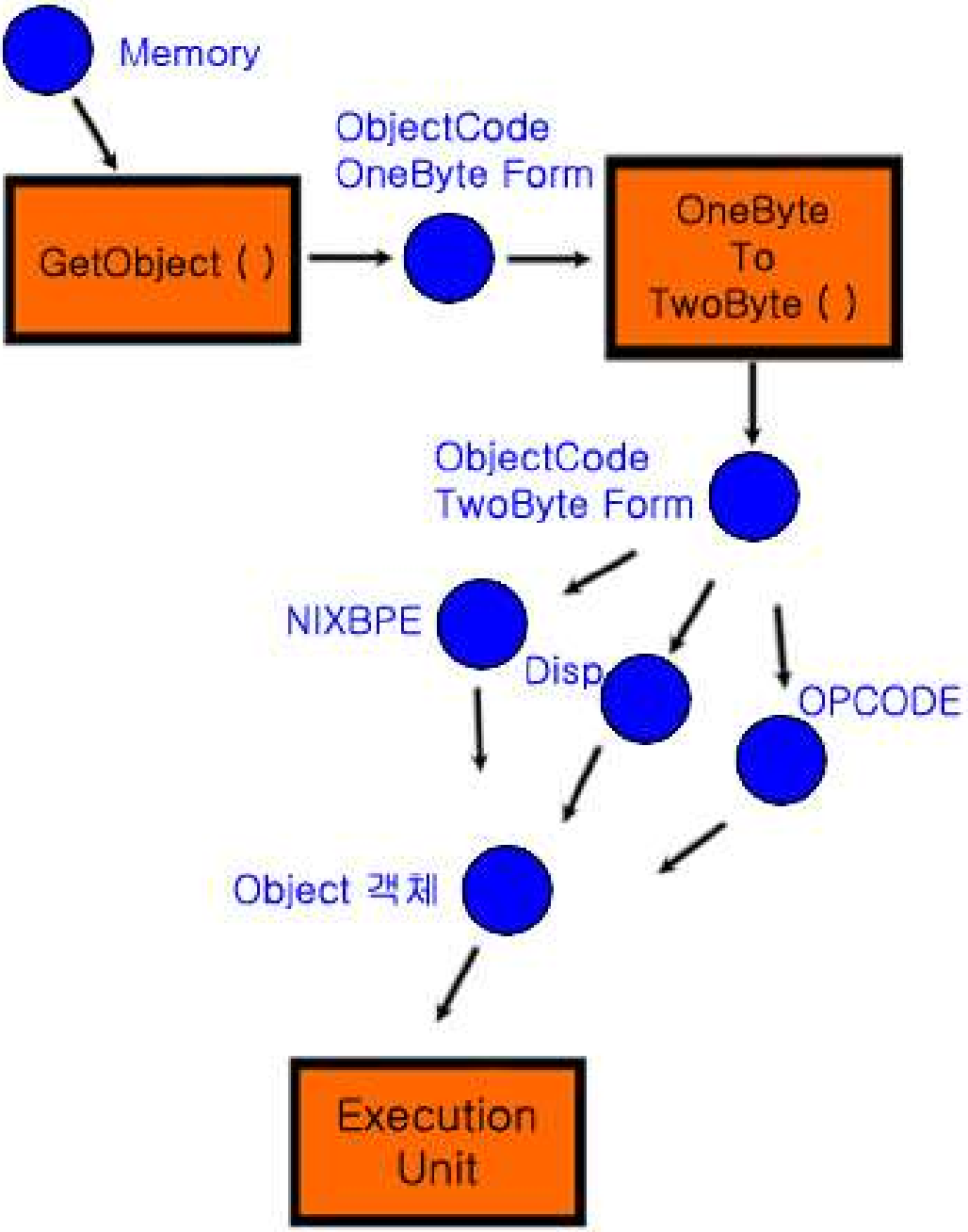
```
////////////////////////////////////////////////////////////////////////////
// Execute 함수
// pObject : ObjectCode 객체를 입력 받는 인자
// 기능 : ObjectCode 객체를 받아서 객체 속에 OPOCDE를 보고 그에 따른 실행 함수 호출한다.
void Execute ( ObjectCode *pObject )
{
    char        *pOPCODE;
    OPTAB       *pSearchReturnOPTAB;

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

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

< Execute 함수 >

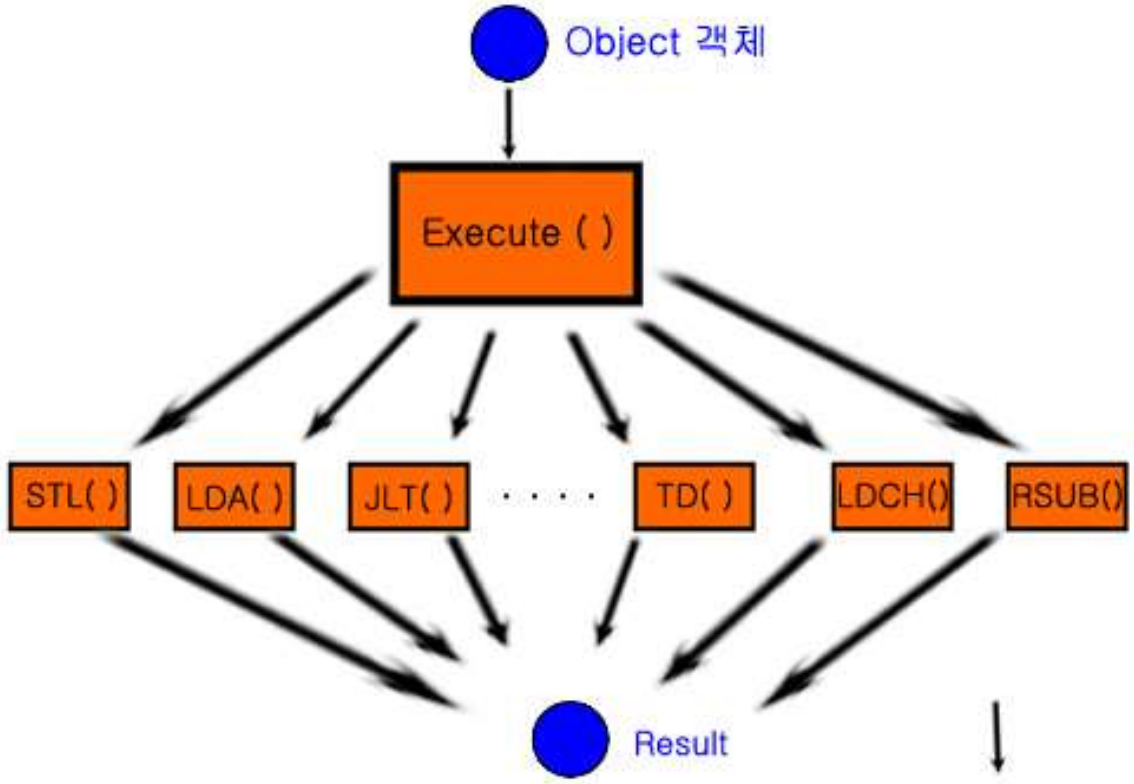
4.2 모듈별 구현 내용



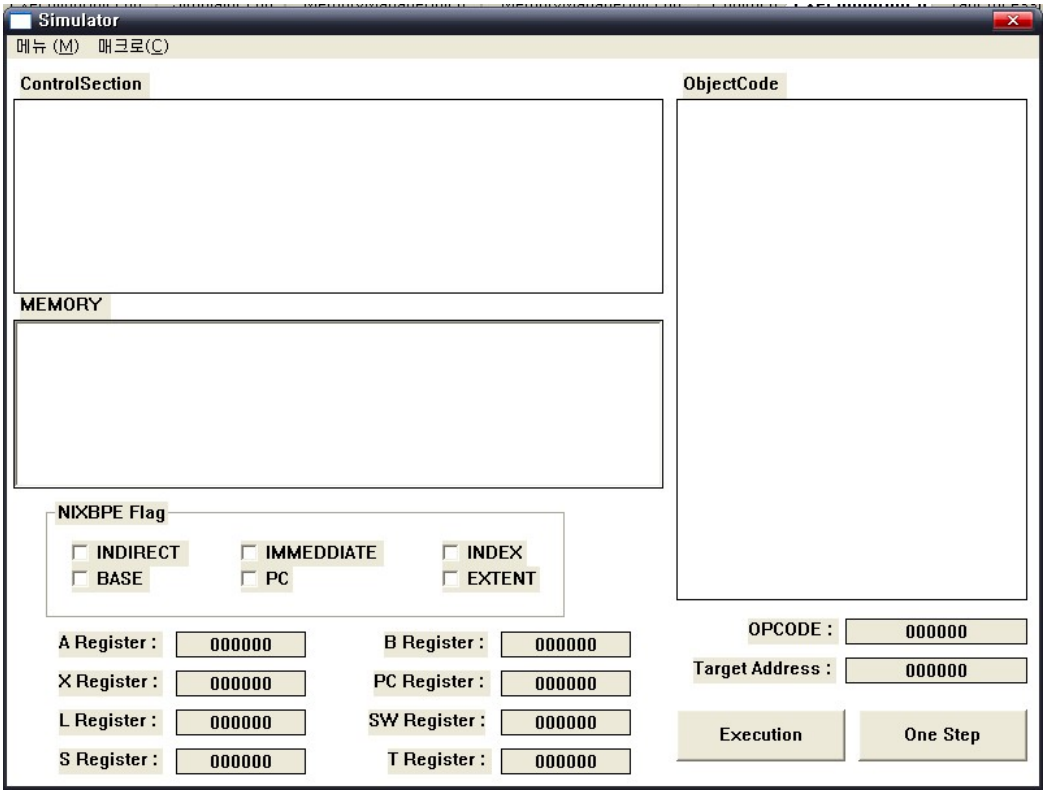
< Object Loading Unit >



< Memory Management Unit >



< Execution Unit >



< GUI Interface >

```
//////////////////////////////////////////////////////////////////////////
// Loader1 함수
// pObjectFile : Object파일의 포인터를 받아들이는 인자
int Loader1 ( FILE *pObjectFile )
```
=> Loader1 함수는 Loading 작업의 Pass1을 행하는 함수이다. 오브젝트의 Head부분과 Definition을 읽어 와서 ESTAB에 넣어서 External Symbol의 정보를 모아두게 한다.

```
//////////////////////////////////////////////////////////////////////////
// Loader2함수
// pObjectFile : Object파일의 포인터를 받아들이는 인자
void Loader2 ( FILE *pObjectFile )
```
=> Loader2는 Loading 작업의 Pass2를 행하는 함수이다. 오브젝트의 Text부분과 Modification부분 처리를 한다. Text부분은 두개의 16진수 문자열을 하나의 문자열로 패킹해서 메모리에 올린다. 그리고 Modification부분에 쓰여 있는 데로 수정할 주소를 찾아가서 수정할 값을 더하거나, 뺀다.

```
//////////////////////////////////////////////////////////////////////////
// GetObjectCode 멤버 함수
// strObj : Object의 시작 주소를 받아들이는 인자
// 기능 : Object의 앞을 보고 타입을 판단하고 그에 맞게 멤버변수에 넣어준다.
int ObjectCode::GetObjectCode ( char *strObj, int nLen )
```
=> GetObjectCode 함수는 첫 번째 인자로 주어진 메모리 주소부터 Instruction을 가져와서 Object 객체에 넣어주는 함수이다. Object객체 속에는 OPCODE, Disp, NIXBPE 등 Instruction의 정보를 가지는 멤버변수가 있다. 그리고 두 번째 인자로 주어진 nLen은 프로그램의 길이가 들어가는데, Instruction을 찾는데 있어서 프로그램 범위를 넘지 않도록 하기 위해서 쓰여진다.

```
//////////////////////////////////////////////////////////////////////////
// HexToInt 함수
// strHex : 문자열 Hex값을 받아들이는 인자
// nLength : 문자열의 길이를 받아들이는 인자
// 기능 : 문자열 Hex값을 받아들여서 int형으로 반환해주는 기능
int HexToInt ( char *strHex, int nLength )
```
=> HexToInt 함수는 16진수 문자열을 받아들여서 이것을 정수형 값으로 돌려주는 기능을 하고 있다. 첫 번째 인자인 strHex가 16진수문자열을 받아들이는 인자이고, nLength가 16진수 문자열의 길이를 받아들이는 인자이다.

```
//////////////////////////////////////////////////////////////////////////
// IntToHex 함수
// strHex : 주어진 Int형 수를 16진수로 저장할 포인터를 받아들이는 인자
// nInt : 16진수로 변환할 int형 값을 받아들이는 인자
// nLength : 16진수로 변환할 수 있는 최대 길이를 받아들이는 인자
void IntToHex ( char *strHex, int nInt, int nLength )
```
=> IntToHex 함수는 정수형 값을 받아들여서 주어진 버퍼로 16진수문자열로 출력해주는 함수이다. 첫 번째 인자로 strHex가 들어오는데 이것은 출력 결과인 16진수 문자열을 저장할 포인터를 받아들이는 인자이다. 그리고 nInt는 16진수 문자열로 변환할 정수형 값을 받아들이는 인자이고, 세 번째 인자인 nLength는 16진수 문자열로 출력할 때 출력문자열의 길이를 받아들이는 인자이다.

```
//////////////////////////////////////////////////////////////////////////
// OneByteToTwoByte 전역 오버로딩 함수
// strOneByte : 팩킹된 문자열을 받아들이는 인자
// strTwoByte : 패킹한 것을 언팩해서 저장할 인자
// nLen : 길이 값을 받아들이는 인자
// 기능 : 1바이트로 팩킹된 문자열을 2바으트로 바꾸어서 저장해준다.
void OneByteToTwoByte ( char *strOneByte, char *strTwoByte, int nLen )
=> OneByteToTwoByte함수는
```

```
//////////////////////////////////////////////////////////////////////////
// LoadAllObject 함수
// AllObject : 메모리에서 얻어온 Object를 저장할 배열 주소를 받아들이는 인자
// nLen : Program의 길이를 받아들이는 인자
// 기능 : Object배열에 Object를 하나씩 메모리에서 읽어서 넣어준다
int LoadAllObject ( ObjectCode *AllObject[], int nLen )
=> SIC/XE 메모리에 올라가있는 프로그램에서 Instruction만을 뽑아서 Object객체 배열에 하나씩 하나씩 집어 넣어준다.
```

```
//////////////////////////////////////////////////////////////////////////
// AddToESTAB 멤버 함수
// 기능 : ESTAB에 현 객체를 연결해주는 기능
void ESTAB::AddToESTAB ( )
=> ESTAB_Header부분에 현재 객체를 붙여준다. ESTAB에 새로운 Symbol을 넣어준다.
```

```
//////////////////////////////////////////////////////////////////////////
// Search 멤버 함수
// strNa : 찾고자 하는 External 이름을 받아들이는 인자
ESTAB* ESTAB::Search ( char *strNa )
=> Symbol의 이름을 가지고 ESTAB에 존재하는 Symbol중에 strNa와 같은 이름을 가진 Symbol이 존재하는지 확인해서 존재하면 그것의 포인터를 리턴해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// ESTAB 생성자
// strName : External Symbol의 이름을 받아들이는 인자
// nAddr : External Symbol의 주소를 받아들이는 인자
// nControlSec : External Symbol의 컨트롤 섹션값을 받아들이는 인자
ESTAB::ESTAB ( char *strNa, int nAdd, int nLen, int nControl )
=> ESTAB 객체 안에 이름과, 주소, 길이, 그리고 섹션 번호를 배정해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// InitOpcodeTable 멤버함수
// 기능 : 테이블을 OPTAB 파일에서 자료를 읽어들여와 초기화 하다.
void OPTAB::InitOpcodeTable ( )
=> OPCODE들이 들어있는 파일에서 OPCODE를 하나씩 읽어 와서 OPTAB을 구성한다.
```

```
//////////////////////////////////////////////////////////////////////////
// OPTAB 생성자 오버로딩 멤버 함수
// pInst : Instruction의 포인터를 받아들이는 인자
// pValue : Instruction의 값의 포인터를 받아들이는 인자
// nType : Instruction의 타입을 받아들이는 인자
// 기능 : 주어진 인자가지고 객체를 초기화해서 생성해 준다.
OPTAB::OPTAB ( char *pInst, char *pValue, int nTyp )
=> OPTAB 객체를 초기화 해주는 함수이다. 첫 번째 인자는 OPCODE의 이름을 받아들이는 인자이고, 두 번째 인자인 pValue는 OPCODE의 값을 받아들이는 인자이다. 그리고 마지막 nType인자는 OPCODE의 타입을 받아들이는 인자이다. 이렇게 세 개의 인자를 받아서 OPTAB 객체를 초기화한다.
```

```
//////////////////////////////////////////////////////////////////////////
// SearchByName 멤버 함수
// strNa : 찾고자 하는 OPCODE이름을 받아들이는 인자
OPTAB* OPTAB::SearchByName ( char *strNa )
=> OPCODE의 이름을 가지고 OPTAB에서 찾고자하는 OPCODE가 있으면 그것의 포인터를 리턴해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// SearchByValue 멤버 함수
// strVal : 찾고자 하는 OPCODE의 값을 받아들이는 인자
OPTAB* OPTAB::SearchByValue ( char *strVal )
=> OPCODE의 값을 가지고 OPTAB에서 찾고자하는 OPCODE를 찾는 것이다. 만약 찾고자하는 값과 일치한 OPCODE가 있으면 그것의 포인터를 리턴해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// GetValueFromReg 함수
// nReg : 값을 넣을 Register 번호를 받아들이는 인자
// 기능 : 원하는 Register에 원하는 값을 리턴해준다.
int GetValueFromReg ( int nReg )
=> GetValueFromReg함수는 인자로 주어진 nReg를 통해서 그에 맞는 레지스터의 값을 리턴해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// InputValueToReg 함수
// nRegNum : 값을 넣을 Register 번호를 받아들이는 인자
// nVal : 원하는 Register에 넣을 값을 받아들이는 인자
// 기능 : 원하는 Register에 원하는 값을 넣어 준다.
void InputValueToReg ( int nReg, int nVal )
=> InputValueToReg 함수는 첫 번째 인자로 주어진 nReg가 가리키는 Register에 두 번째 인자로 주어진 nVal값을 넣어준다.
```

```
//////////////////////////////////////////////////////////////////////////
// LDT 함수
// Instruction : LDT명령을 포함하는 Instruction 전체를 받아들이는 인자
// 기능 : T Register에 Instruction에 포함된 데이터를 저장한다.
void LDT ( ObjectCode *Instruction )
=> 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 목적지 주소에서 데이터를 읽어와서 T 레지스터에 저장해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// LDA 함수
// Instruction : LDA명령을 포함하는 Instruction 전체를 받아들이는 인자
// 기능 : A Register에 Instruction에 포함된 데이터를 저장한다.
void LDA ( ObjectCode *Instruction )
=> 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 목적지 주소에서 데이터를 읽어 와서 A 레지스터에 저장해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// STA 함수
// Instruction : STA 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : A Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STA ( ObjectCode *Instruction )
=> 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 A 레지스터에 있는 데이터를 목적지 주소에다가 저장해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// STL 함수
// Instruction : STL 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : L Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STL ( ObjectCode *Instruction )
=> 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 L 레지스터에 있는 데이터를 목적지 주소에다가 저장해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// STX 함수
// Instruction : STX 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : X Register에 들어있는 값을 Instruction에 포함된 주소에 쓴다
void STX ( ObjectCode *Instruction )
=> 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 X 레지스터에 들어있는 데이터를 목적지 주소에다가 저장해준다.
```

```
//////////////////////////////////////////////////////////////////////////
// J 함수
// Instruction : J 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : PC Register에 Instruction에 포함된 값을 쓴다.
void J ( ObjectCode *Instruction )
```
=> J함수는 강제 분기를 할때 스이는 함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 PC 레지스터에 목적지 주소가 가리키는 값을 쓴다.

```
//////////////////////////////////////////////////////////////////////////
// JSUB 함수
// Instruction : JSUB 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : 현재 PC값을 L Register에 저장하고 PC Register에 Instruction에 포함된 값을 넣는다.
void JSUB ( ObjectCode *Instruction )
```
=> JSUB 함수는 서브루틴을 호출할 때 쓰이는 함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 현재의 PC 레지스터 값을 L Register에다가 쓰고 목적지 주소에 있는 값을 PC 레지스터에 써준다.

```
//////////////////////////////////////////////////////////////////////////
// JEQ 함수
// Instruction : JEQ 명령을 포함하는 전체 Instructoin을 받아들이는 인자
// 기능 : Condition Code를 보고 0이면 Instruction에 포함한 값을 PC Register에 넣는다
void JEQ ( ObjectCode *Instruction )
```
=> JEQ함수는 조건분기문 함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 Condition Code를 보고 Condition Code가 0이면 목적지 주소에 있는 값을 PC 레지스터에 써준다.

```
//////////////////////////////////////////////////////////////////////////
// JLT 함수
// Instruction : JLT 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Condition Code를 보고 음수이면 Instruction에 포함된 값을 PC Register에 넣는다.
void JLT ( ObjectCode *Instruction )
```
=> JLT함수는 조건분기문 함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 Condition Code값이 음수이면 목적지 주소에 있는 값을 PC 레지스터에 써준다.

```
//////////////////////////////////////////////////////////////////////////
// COMP 함수
// Instruction : COMP 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : A Register값과 Instruction에 포함된 값을 비교하여 Condition Code 값을 변경한다
void COMP ( ObjectCode *Instruction )
```
=> COMP는 A 레지스터와 목적지 주소에 값과 비교해서 Condition Code 값을 변경해준다. A 레지스터 값이 더 크면 음수를, 목적지 주소에 값이 더 크면 양수를 같으면 Condition Code값을 0으로 변경 시켜준다.

```
//////////////////////////////////////////////////////////////////////////
// COMPR 함수
// Instruction : COMPR 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 두개의 레지스터 값을 비교해서 Condition Code값을 변화시킨다
void COMPR ( ObjectCode *Instruction )
```

=> COMPR은 레지스터와 레지스터간의 비교함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 주어진 두 개의 레지스터를 비교해서 왼쪽 레지스터 값이 더 크면 음수로, 오른쪽 레지스터값이 더 크면 양수로, 두 개의 레지스터 값이 같으면 Condition Code값을 0으로 써준다.

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

=> CLEAR 함수는 주어진 레지스터를 0으로 초기화해주는 함수로써, 주어진 레지스터를 간단히 0으로 초기화 시켜준다.

```
//////////////////////////////////////////////////////////////////////////
// TD 함수
// Instruction : TD 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 디바이스가 사용가능 한지 테스트한다
void TD ( ObjectCode *Instruction )
```

=> TD 함수는 디바이스가 준비되었는지 안되었는지를 테스트 하는 함수로써, 준비가 되어 있지 않으면 Condition Code를 0으로 만들고 준비가 제대로 되어있다면 Condition Code값을 -1로 써준다.

```
//////////////////////////////////////////////////////////////////////////
// WD 함수
// Instruction : WD 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Device로 A Register의 가장 오른쪽 데이터부터 쓴다.
void WD ( ObjectCode *Instruction )
```

=> WD 함수는 디바이스에 A 레지스터의 가장 오른쪽 1바이트를 써주는 함수로써, 여기서는 디바이스 이름으로 주어진 파일명에다가 데이터를 써준다.

```
//////////////////////////////////////////////////////////////////////////
// RD 함수
// Instruction : RD 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Device로 부터 한바이트씩 값을 읽어 A Register의 가장 오른쪽부터 쓴다.
void RD ( ObjectCode *Instruction )
```

=> RD 함수는 디바이스에서 1바이트를 읽어와서 A 레지스터에 가장 오른쪽 부분에 써주는 함수로써, 디바이스 이름으로 주어진 파일에서 1바이트를 읽어온다.

```
//////////////////////////////////////////////////////////////////////////
// LDCH 함수
// Instruction : LDCH 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : Instruction에 주어진 버퍼로 부터 한바이트를 읽어와서 A Register의
//                         가장 오른쪽에 저장한다.
void LDCH (ObjectCode *Instruction )
```

=> LDCH 함수는 LD계열 함수와 비슷하지만 워드 단위로 읽어오는 것이 아니라 1바이트씩 읽어오는 함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 목적지 주소에 있는 값을 1바이트 읽어서 A 레지스터에 써준다.

```
//////////////////////////////////////////////////////////////////////////
// STCH 함수
// Instruction : STCH 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : A Register의 가장 오른쪽 한바이트를 Instruction에 주어진 버퍼에 저장한다.
void STCH (ObjectCode *Instruction )
```

=> STCH 함수는 ST계열 함수와 비슷하지만 워드 단위로 쓰는 것이 아니라 1바이트씩 쓰는 함수로써, 첫 번째 인자로 Object 객체를 받아들인다. 그 속에는 Disp와 NIXBPE 등의 기본 정보가 들어있어서 이를 기반으로 목적지 주소에 A 레지스터의 가장 오른쪽 부분 1바이트를 써준다.

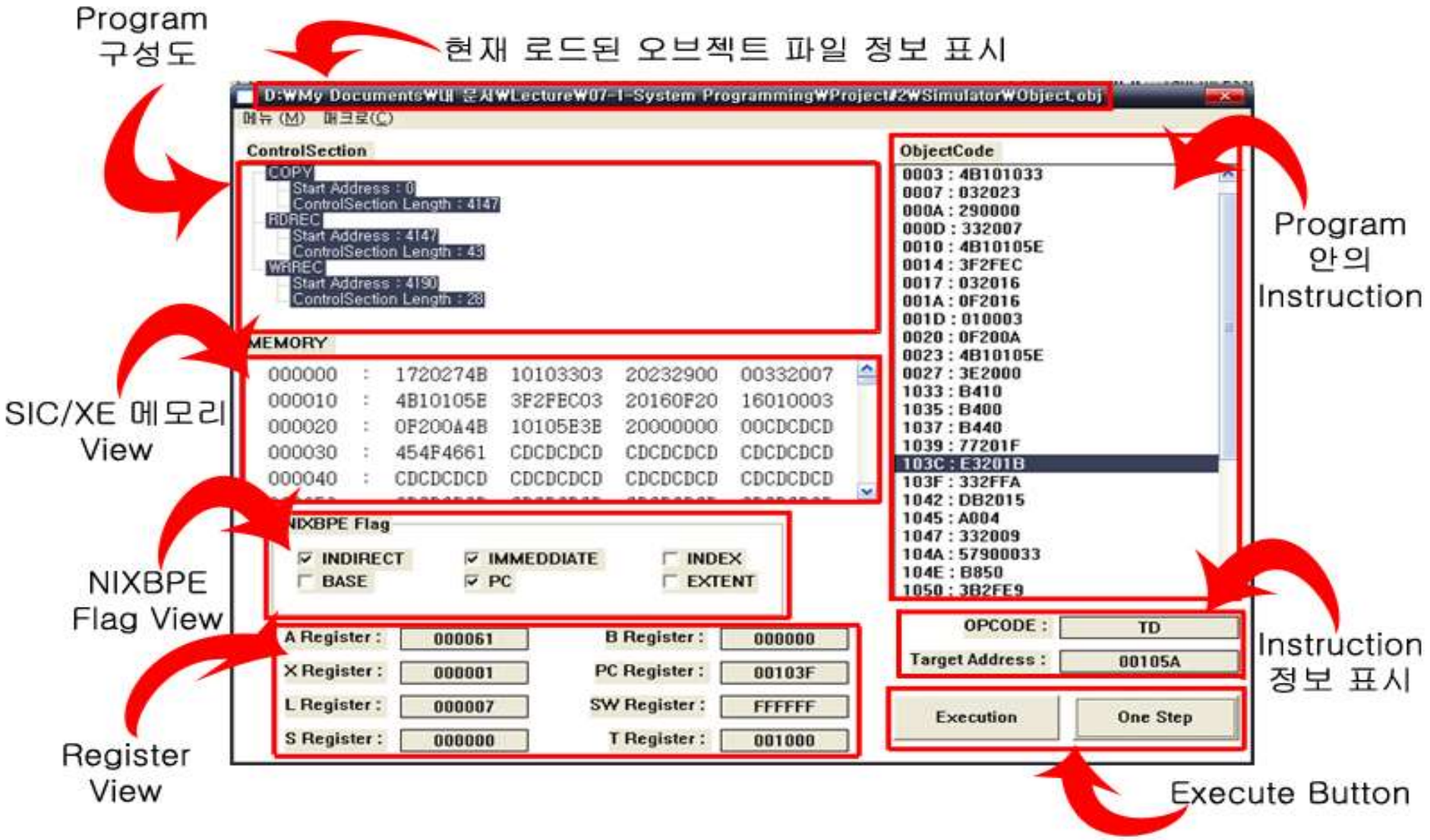
```
//////////////////////////////////////////////////////////////////////////
// TIXR 함수
// Instruction : TIXR 명령을 포함하는 전체 Instruction을 받아들이는 인자
// 기능 : X Register 값을 하나 증가 시키고 Instruction에 주어진 Register와 비교해서
//                         Condition Code를 변화 시켜준다.
void TIXR ( ObjectCode *Instruction )
```

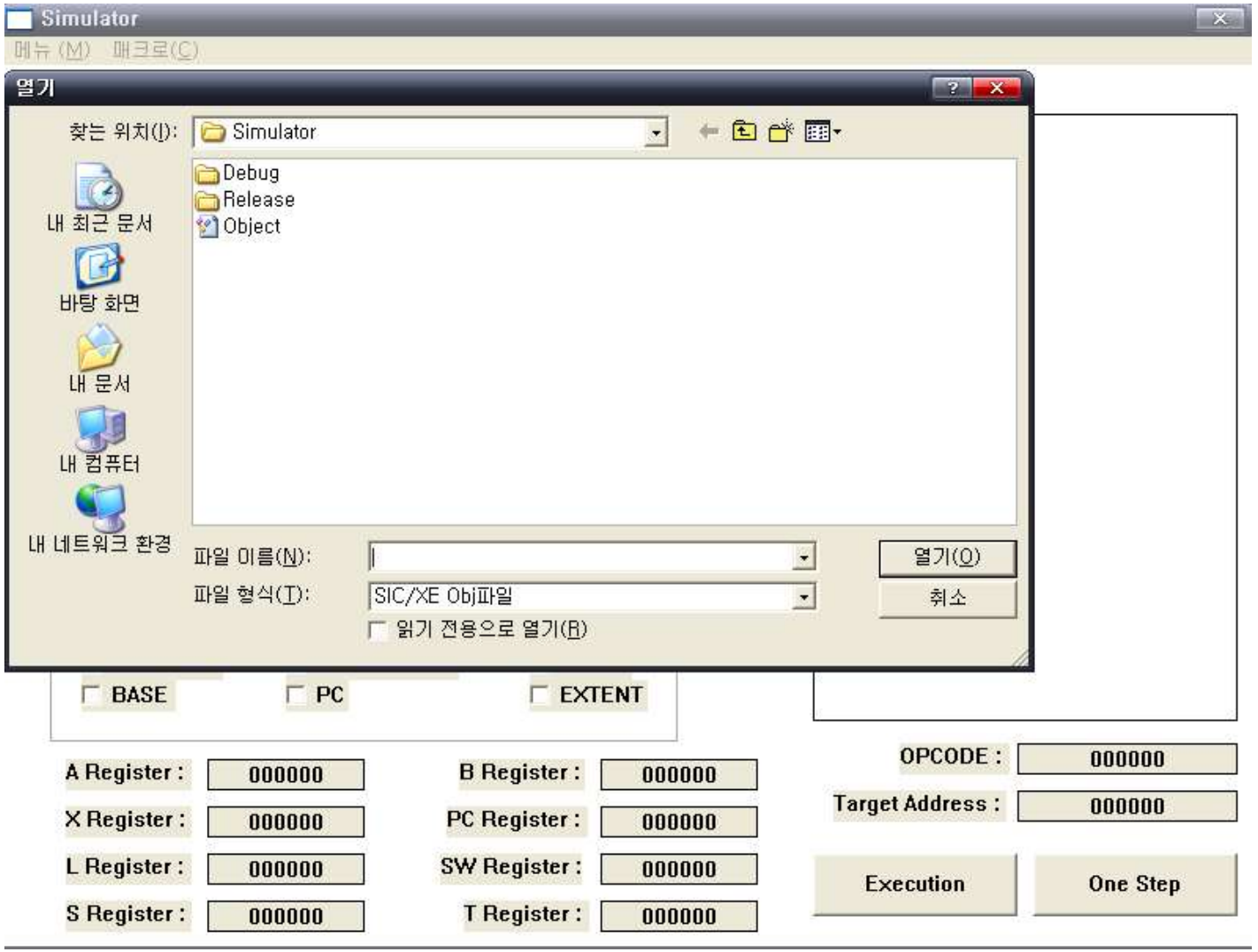
=> TIXR 함수는 X 레지스터 값을 증가하고 주어진 레지스터와 비교해서 Condition Code를 변경시켜준다. 1이 증가된 X 레지스터가 주어진 레지스터값 보다 작으면 음수, 크면 Condition Code를 양수로 변경시켜준다.

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

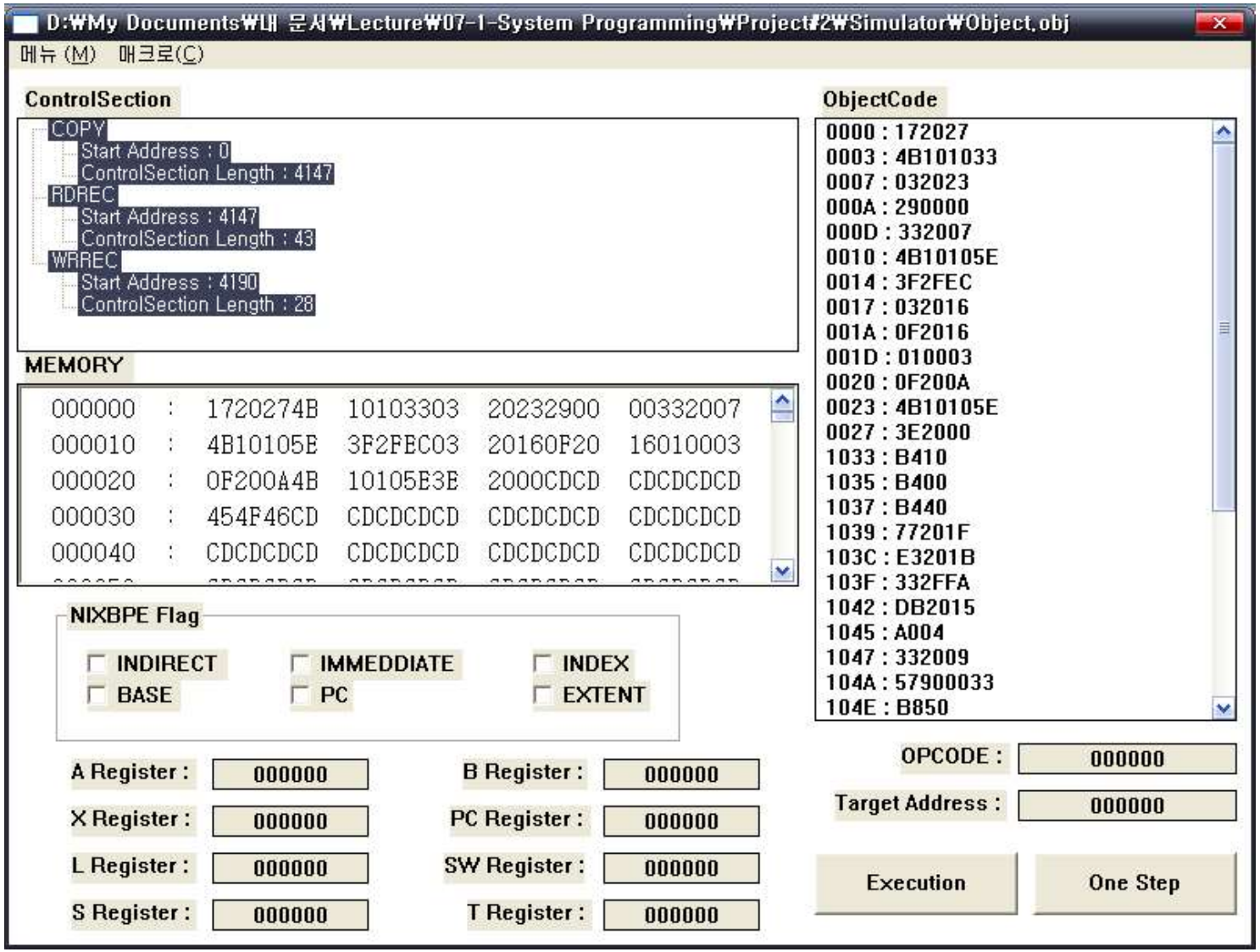
=> RSUB 함수는 서브루틴에서 서브루틴을 호출한 곳으로 돌아갈 때 쓰이는 함수로써, 이전에 서브루틴 호출할 때 넣어두었던 L 레지스터에 들어있던 호출했던 곳의 주소를 이제는 다시 PC 레지스터에 써준다.

4.3 구현 화면

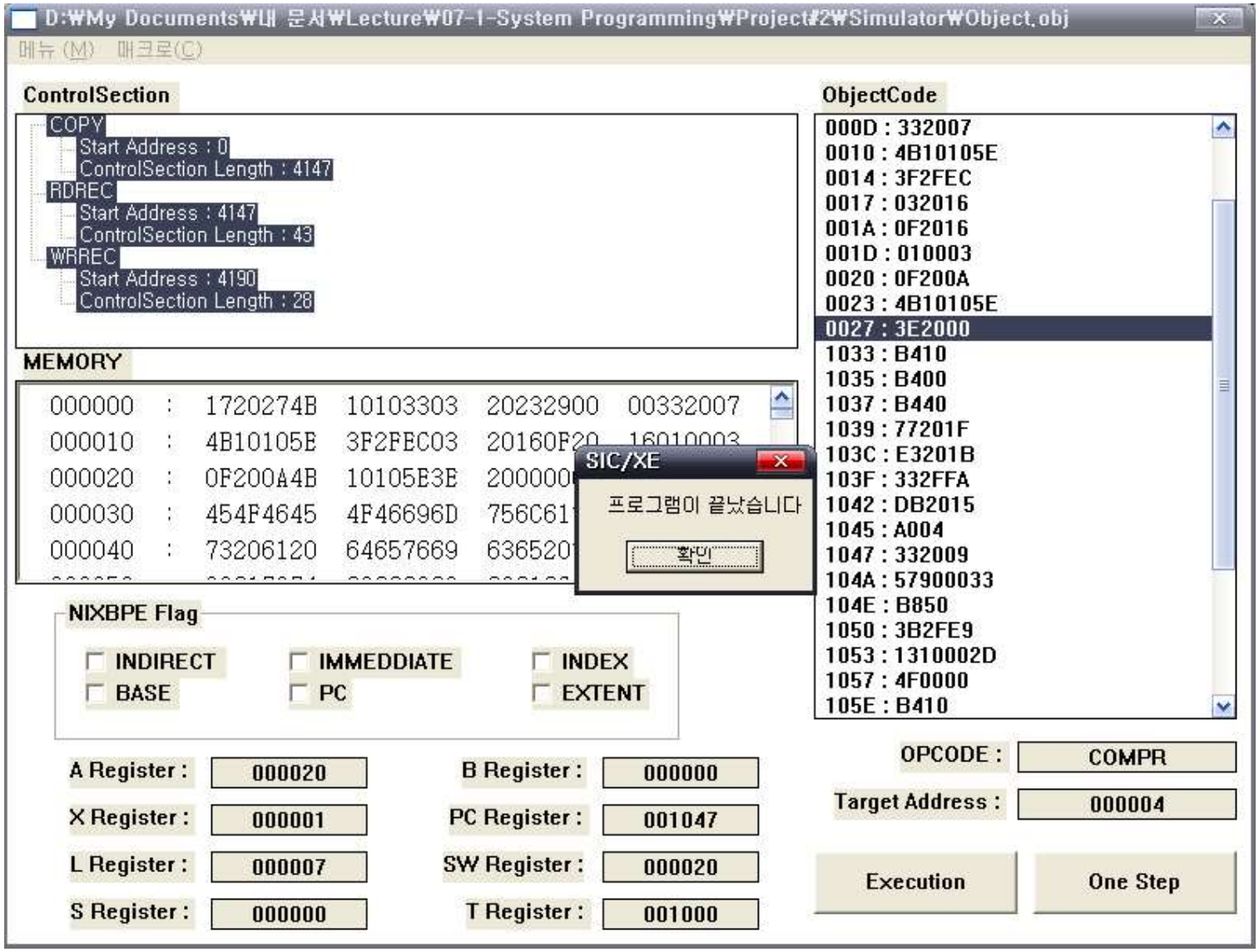

Program
구성도
현재 로드된 오브젝트 파일 정보 표시
D:₩My Documents₩내 문서₩Lecture₩07-1-System Programming₩Project#2₩Simulator₩Object.obj
메뉴(M) 매크로(C)
ControlSection
COPY
Start Address : 0
ControlSection Length : 4147
RDREC
Start Address : 4147
ControlSection Length : 43
WRREC
Start Address : 4190
ControlSection Length : 28
ObjectCode
0003 : 4B101033
0007 : 032023
000A : 290000
000D : 332007
0010 : 4B10105E
0014 : 3F2FEC
0017 : 032016
001A : 0F2016
001D : 010003
0020 : 0F200A
0023 : 4B10105E
0027 : 3E2000
1033 : B410
1035 : B400
1037 : B440
1039 : 77201F
103C : E3201B
103F : 332FFA
1042 : DB2015
1045 : A004
1047 : 332009
104A : 57900033
104E : B850
1050 : 3B2FE9
Program
안의
Instruction
MEMORY
000000 : 1720274B 10103303 20232900 00332007
000010 : 4B10105E 3F2FEC03 20160F20 16010003
000020 : 0F200A4B 10105E3E 20000000 00CDCDCD
000030 : 454F4661 CDCDCDCD CDCDCDCD CDCDCDCD
000040 : CDCDCDCD CDCDCDCD CDCDCDCD CDCDCDCD
SIC/XE 메모리
View
NIXBPE Flag
INDIRECT
IMMEDDIATE
INDEX
BASE
PC
EXTENT
NIXBPE
Flag View
A Register : 000061
B Register : 000000
X Register : 000001
PC Register : 00103F
L Register : 000007
SW Register : FFFFFF
S Register : 000000
T Register : 001000
Register
View
OPCODE : TD
Target Address : 00105A
Instruction
정보 표시
Execution
One Step
Execute Button



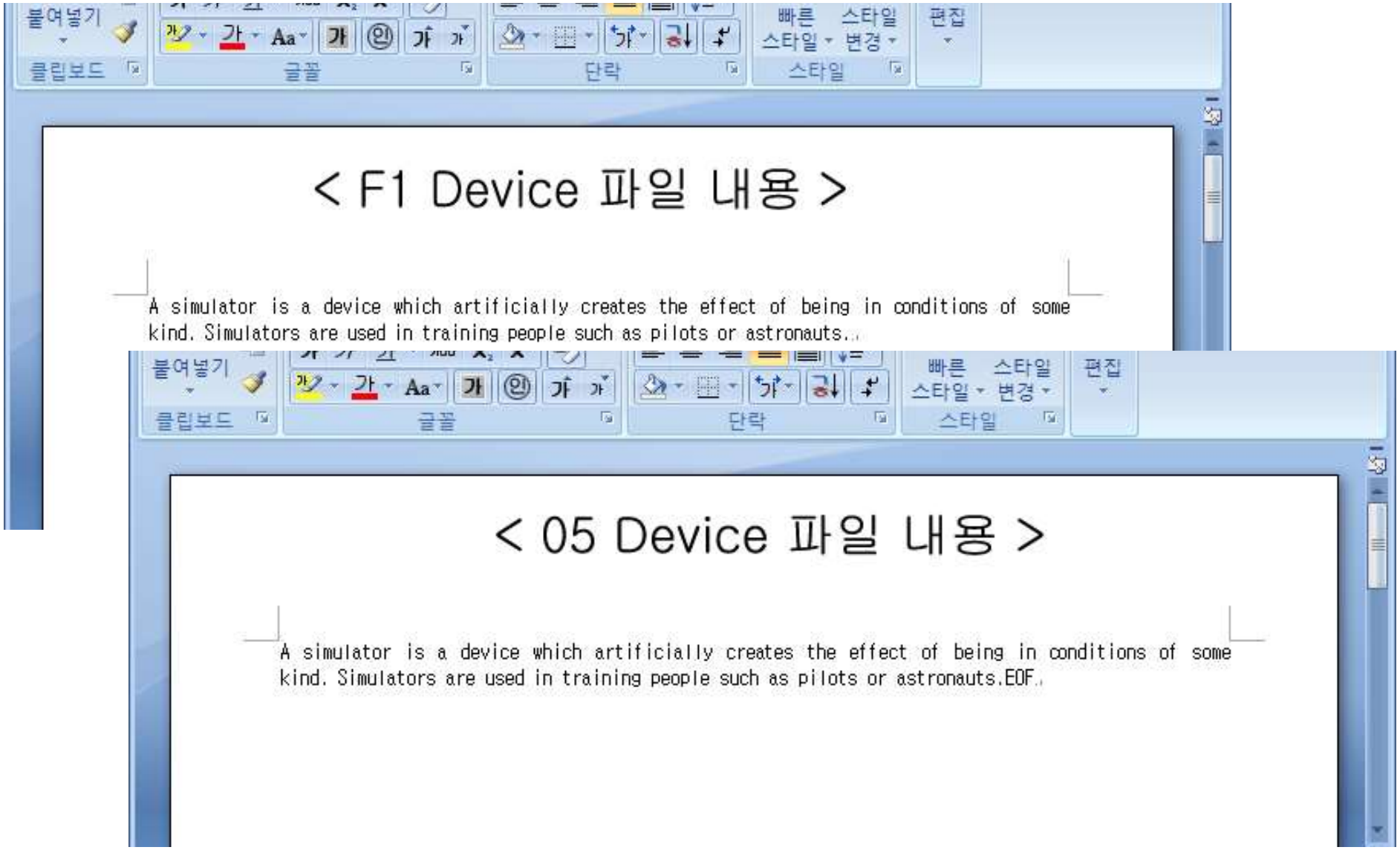
< Object 파일 열기 화면 >



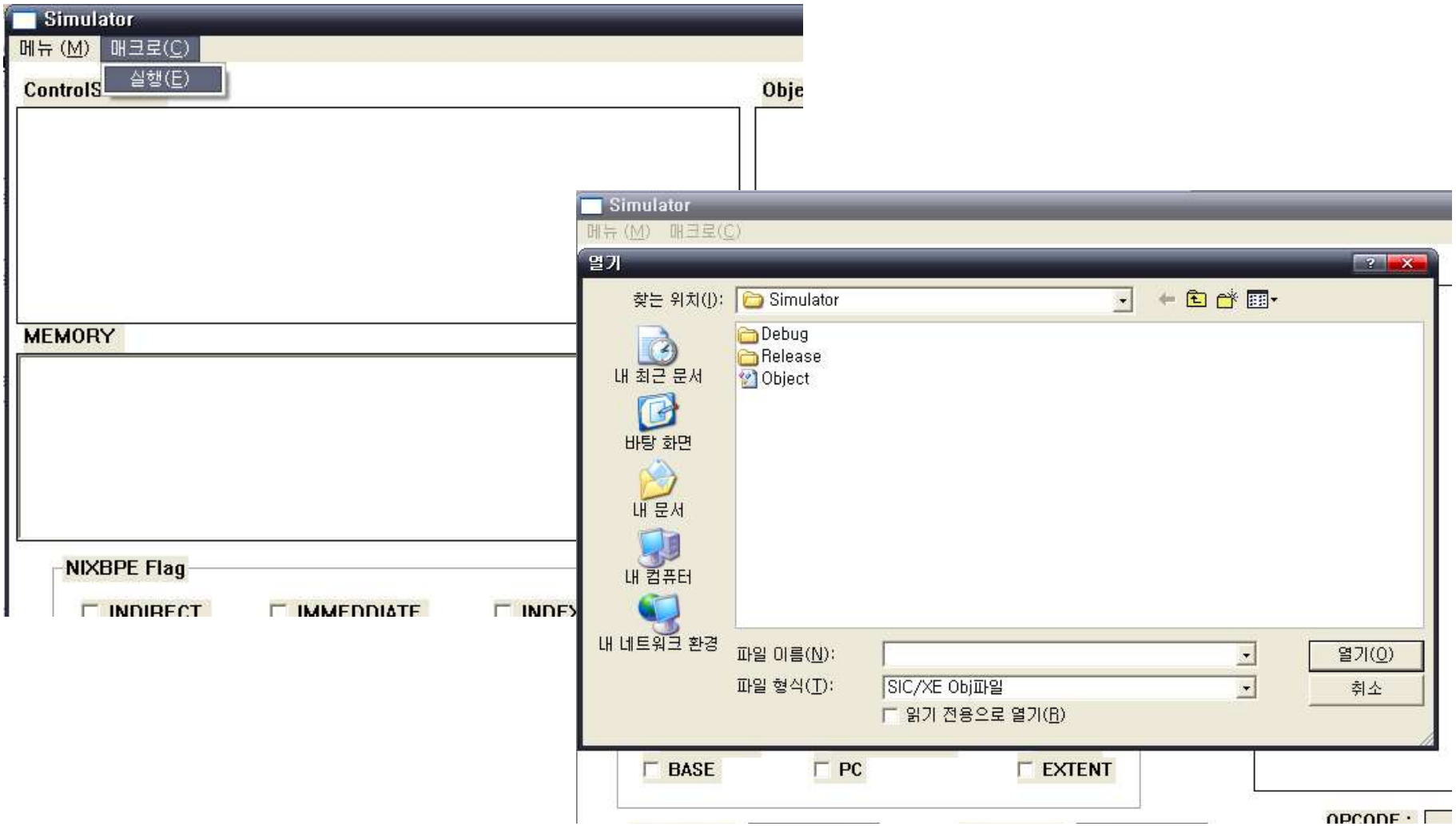
< Object 파일 Load한 화면 >


D:₩My Documents₩내 문서₩Lecture₩07-1-System Programming₩Project#2₩Simulator₩Object.obj
메뉴 (M)
매크로(C)
ControlSection
COPY
Start Address : 0
ControlSection Length : 4147
RDREC
Start Address : 4147
ControlSection Length : 43
WRREC
Start Address : 4190
ControlSection Length : 28
ObjectCode
000D : 332007
0010 : 4B10105E
0014 : 3F2FEC
0017 : 032016
001A : 0F2016
001D : 010003
0020 : 0F200A
0023 : 4B10105E
0027 : 3E2000
1033 : B410
1035 : B400
1037 : B440
1039 : 77201F
103C : E3201B
103F : 332FFA
1042 : DB2015
1045 : A004
1047 : 332009
104A : 57900033
104E : B850
1050 : 3B2FE9
1053 : 1310002D
1057 : 4F0000
105E : B410
MEMORY
000000 : 1720274B 10103303 20232900 00332007
000010 : 4B10105E 3F2FEC03
000020 : 0F200A4B 10105E3E
000030 : 454F4645 4F46696D
000040 : 73206120 64657669
SIC/XE
프로그램이 끝났습니다
확인
NIXBPE Flag
INDIRECT
IMMEDDIATE
INDEX
BASE
PC
EXTENT
A Register : 000020
B Register : 000000
X Register : 000001
PC Register : 001047
L Register : 000007
SW Register : 000020
S Register : 000000
T Register : 001000
OPCODE : COMPR
Target Address : 000004
Execution
One Step

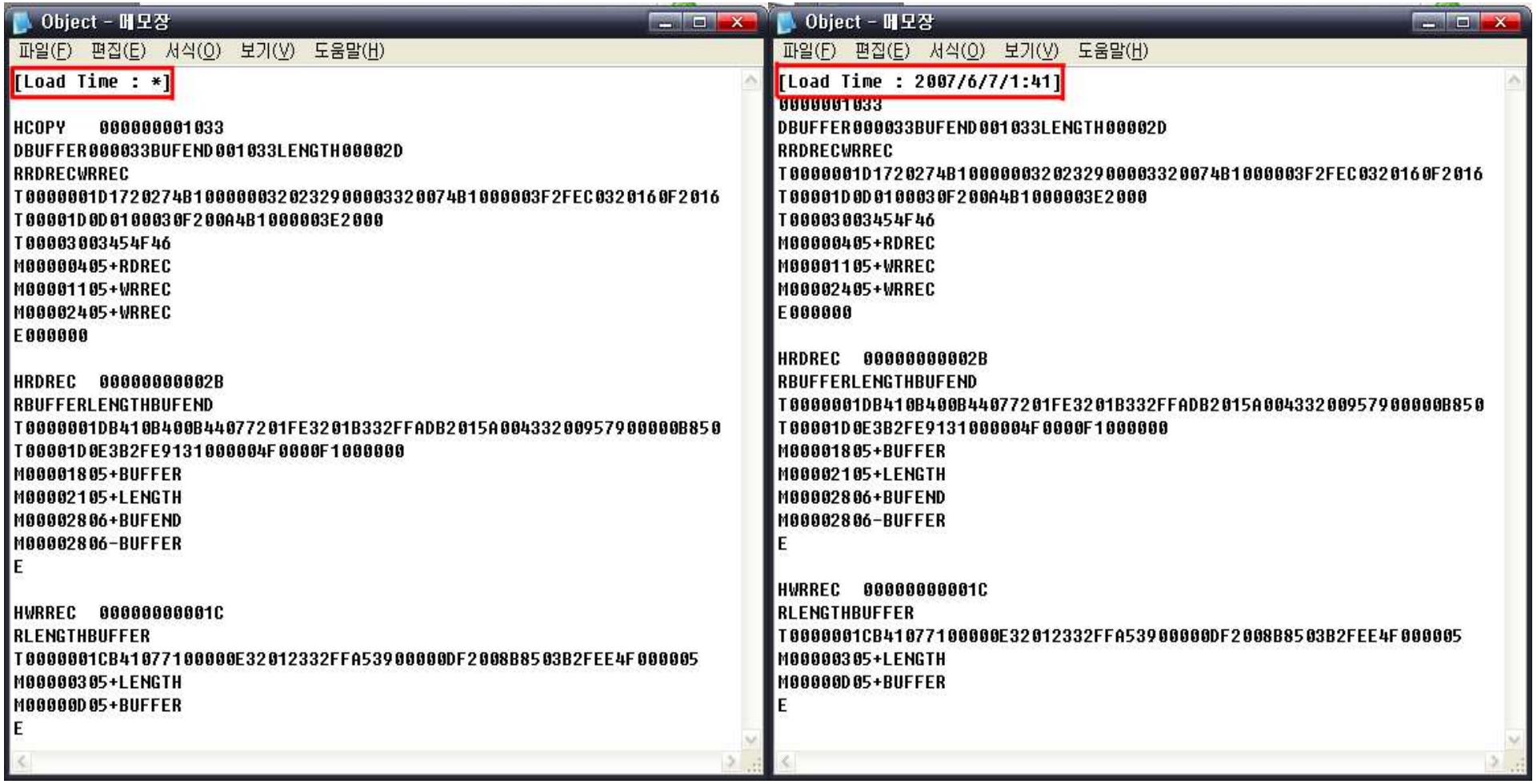
< 실행이 종료 된 화면 >



< Simulator 실행 후 결과 >



< Macro 실행 하는 화면 >



< Macro 실행 전 Object파일과 실행 후 Object파일 화면 >

5장 결론

지금까지 링킹로더와 시뮬레이터를 만들어보았다. 그리고 추가로 매크로를 이용한 간단한 로딩날짜 삽입하기. 막상 우리가 컴퓨터를 사용하면서, 그리고 프로그래밍을 배우면서, 소스파일을 컴파일하면 오브젝트파일이 나오고 바이너리 파일이 나온다. 이렇게만 간단히 알고 있었다. 그런데 이렇게 시스템프로그래밍과목 시간에 더욱 깊이 들어가서 명령어의 형식, 그리고 위치마다 의미하는 것들을 보면서 조금 더 많은 것을 알게 되었다. 지금은 간단하게 컴퓨터가 어떻게 돌아가는지 이해를 돕기 위한 가상의 기계인 SIC/XE 용 어셈블러와, 로더, 그리고 시뮬레이터를 만들어보았다. 간단한 머신도 이렇게 힘든데 현재 존재하는 머신들 기반에서 구현한다면 얼마나 복잡하고 힘들까? 하는 생각이 들었다.

이번 프로젝트1과 프로젝트2를 통해서, 군대 갔다 와서 잊고 있던 많은 것들을 다시 보고 배우게 해주었고, 떠오르게 해주었다. 그리고 2년 전에 해보았던 윈도우 프로그래밍을 요번 프로젝트2를 위해서 다시 해보게 되어서 많이 잊어먹었던 것들도 다시 생각나게 해주었고 하면서 많은 새로운 것들도 배우게 해주었다. 처음에 시스템프로그래밍 수업을 신청하면서 주변의 말들을 많이 들어서 처음부터 겁먹고 시작했다. 하지만 힘들게 수강한 만큼 다른 수업보다 더 많이 얻어 가는 것 같아서 너무 좋았다.