과목명 : 시스템 프로그래밍

담당 교수명 : 김지환

<<Assignment 3>>

서강대학교 컴퓨터공학과

학번: 20171690

이름: 정유석

**목차**

1. 프로그램 개요
2. 프로그램 설명
3. 분할 c 파일
   1. **linkingLoader.c**
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : **progaddr()**
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : **loader()**
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : **loadMap()**
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : **HRecord()**
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : **DRecord()**
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : **RRecord()**
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : **TRecord()**
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : **MRecord()**
            1. 기능
            2. 사용 변수
         9. 모듈 이름 : **newTRaddr()**
            1. 기능
            2. 사용 변수
         10. 모듈 이름 : **freeTRHead()**
             1. 기능
             2. 사용 변수
         11. 모듈 이름 : **setMem()**
             1. 기능
             2. 사용 변수
         12. 모듈 이름 : **addES()**
             1. 기능
             2. 사용 변수
         13. 모듈 이름 : **addRnN()**
             1. 기능
             2. 사용 변수
         14. 모듈 이름 : **searchESTAB()**
             1. 기능
             2. 사용 변수
         15. 모듈 이름 : **searchRN()**
             1. 기능
             2. 사용 변수
         16. 모듈 이름 : **haltLinkingLoader()**
             1. 기능
             2. 사용 변수
         17. 모듈 이름 : **isObjFile()**
             1. 기능
             2. 사용 변수
         18. 모듈 이름 : **printES()**
             1. 기능
             2. 사용 변수
         19. 모듈 이름 : **freeRN()**
             1. 기능
             2. 사용 변수
   2. **execution.c**
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : **printReg()**
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : **executeProg()**
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : **setBP()**
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : **clearBP()**
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : **printBP()**
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : **searchBP()**
            1. 기능
            2. 사용 변수
   3. **opcodeAction.c**
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : **opAct()**
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : **getRegPtr()**
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : **write\_to\_memory()**
            1. 기능
            2. 사용 변수
4. 구조체 정의
   1. **ESNode**
   2. **EShead**
   3. **referNode**
   4. **newTR**
   5. **BPNode**
5. 전역 변수 정의
   1. **int PROGADDR**
   2. **Eshead\* ESTAB**
   3. **referNode\* referHead**
   4. **newTR\* TRHead**
   5. **BPNode\* BPHead**
   6. **EXEC\_ADDR**
   7. **EXEC\_LEN**
   8. **PROG\_START**
   9. **PROG\_END**
   10. **continuing**
6. 코드

1. 프로그램 개요

주어진 다수의 object file에 대해, 각 file들을 서로 link하고 load하는 프로그램을 작성하였다. project1에서 구현한 1MB Memory 에 link된 프로그램을 load한다. 또한 load한 프로그램을 실행시켜, 수업시간에 공부한 여러가지 operation code들에 상응하는 적절한 동작을 수행한다.

프로그햄 수행 시 SIC machine과 SIC/XE machine instruction 모두 호환 가능하며, WORD size는 3Byte이다. 메모리에 load된 object code는 BIG ENDIAN형식으로 저장되었으며, 각 byte를 operation code의 format에 따라 1 ~ 4 byte씩 읽어와 적절한 동작을 수행하도록 구현하였다.

입력되는 object file은 형식과 문법이 정확하다고 가정하였고, 그렇지 않을 경우 error message를 출력하며 linking load 및 프로그램 실행이 중단될 수 있다. 또한 정확히 처리해주지 못한 예외 및 예측하지 못한 상황에 대해 프로그램이 비정상적으로 동작할 수 있다.

2. 프로그램 설명

프로그램을 linking load하는 과정은 수업시간에 공부한 2 pass algorithm을 그대로 따랐다. Pass 1에서 각 control section이 load될 시작 주소를 얻고, Define record에 기술된, external symbols에 대해 External Symbol Table을 build한다. 이후 Pass2를 진행하며 적절한 위치의 memory에 object code를 load하고, R Record와 M Record를 참고해 memory에 load된 값을 적절히 modify하는 과정을 거친다.

이 때 프로그램 수행 시 에러 발생을 방지하기 위해, 각 Text Record가 시작하는 시작 주소를 별도의 linked list로 저장하였다. program run 수행 시, 프로그램이 잘못 작성되어 비정상적으로 상수나 변수에 접근하게 될 때, 해당 상수/ 변수가 WORD size가 아닐 경우 다음 읽어오는 object code를 잘못 해석할 여지가 있다.(proga.obj에서와 같이 생략된 코드가 있는 경우도 해당된다.) 따라서 Text Record에 기술된 각 object code의 시작 주소를 이용해, 각 주소를 읽어오는 데서 발생한 오차를 수정하는 작업을 수행한다.

program을 run할 때, 프로그램이 실행되는 주소부터 각 byte를 읽어오며 수행한다. 읽어온 byte에 상응하는 opcode가 있는 경우, n, i flag가 0인지 아닌지를 확인해 SIC insturction, SIC/XE instruction을 구분한다. 그리고 형식에 맞게 필요한 수의 byte를 메모리에서 더 읽어와 적절한 동작을 수행한다.

한편 상응하는 opcode가 없는 경우는 WORD size에 해당하는 3 byte를 건너띄고 프로그램 실행을 계속하게 된다.

만약 현재의 program couter가 저장된 break point와 같다면, continuing flag(5.10에 기술)을 1로 설정하고 프로그램을 중단한다. 이 때 다음 run 명령어에 의해 수행되는 주소는 중단된 program counter가 된다. breakpoint를 만나지 않고 프로그램이 모두 실행되고 종료되었다면 continuing flag를 0으로 설정하고, 프로그램 실행 주소를 실행했던 프로그램의 시작 주소로 설정한다.

3. 분할 c파일

**3.1 linkingLoader.c**

3.1.1 설명

object file을 link 시켜 메모리에 load하는 일을 수행하는데 필요한 모듈을 구현한 c file이다. link 가능한 object file은 최대 3개이며, project 1에서 구현한 1MB memory에 그 결과를 load한다.

3.1.2 모듈 정의

3.1.2.1 모듈 이름 : **progaddr(char\* addr)**

3.1.2.1.1 기능

명령어 중 하나로서, loader 또는 run 명령어를 수행할 때 시작하는 주소를 지정한다. default 값으로 0x00 주소로 저장되며, progaddr 4000 과 같이 hexa decimal 수로 지정할 수 있다. 지정 가능한 주소 범위 밖이면 error message를 출력한다. 주소가 지정되지 않고 progaddr 만 입력하면 0x00 번지로 reset된다.

3.1.2.1.2 사용 변수

**char\* addr** – 변경할 주소를 나타낸다. object file에서 읽어온 문자열 형태이며, 추후에 16진수 수로 변환하여 저장한다.

3.1.2.2 모듈 이름 : **loader(char\* param)**

3.1.2.2.1 기능

입력한 object file을 link하고 load하는 기능을 수행한다. 함수의 parameter로는 loader 명령어 뒤에 따르는, linking load할 object file들이 나열된 문자열이 입력된다.

이 때 프로그램이 load되는 주소는 progaddr 명령어로 지정한 주소이며, 지정해주지 않았다면 default로 0x00번지에 load된다. 강의자료에 나온 2 pass 로 linking load가 진행되며, 만약 수행에 성공한다면 load map을 출력한다.

입력한 파일이 3개를 초과하거나, 입력한 object file을 찾을 수 없다면 error message를 출력하고 종료한다. 또한 object file에 오류가 있는 경우에도 error message를 출력하고 종료한다. 각 Record에 대한 기능 수행과 예외 처리는 3.1.2.4 ~ 3.1.2.8에 기술하였다.

3.1.2.2.2 사용 변수

**char\* param** – linking load할 object file들이 나열된 문자열이다. 각 file name은 space(공백)으로 구분된다.

**char\*\* objFile** – param에서 입력받은, linking load할 object file들을 저장한 문자열 배열이다.

**int objCnt** – linking load할 object file의 개수를 저장한다.

**int i, CS** – loop counter로 사용되었다.

**int flag** – linking load를 하는 과정에서 E record를 만났을 때 종료 flag를 설정한다.

**FILE\*\* objFP** – linking load할 object file들의 file pointer 배열이다.

**char line[MAX\_LINE\_LEN]** – object file에서 각 record를 저장하는 문자열이다.

3.1.2.3 모듈 이름 : **loadMap(int objCnt)**

3.1.2.3.1 기능

linking load과정이 성공적으로 끝났을 때 만들어진 load map을 출력한다. 이 때 load map은 각 control section의 시작 주소와 길이, 그리고 각 control section에서 external define된 symbol의 주소 정보를 포함한다.

3.1.2.3.2 사용 변수

**int objCnt** – linking load할 object file의 개수를 저장한다.

**int len** – 전체 프로그램의 길이를 저장한다.

3.1.2.4 모듈 이름 : **HRecord(char\* line, int currentCS, char\* file)**

3.1.2.4.1 기능

object file에서 Header Record에 대한 동작을 수행한다. program name을 읽고 program의 시작 주소와 길이를 저장한다.

3.1.2.4.2 사용 변수

**char\* line** – Header Record 한 줄을 저장하는 문자열이다.

**int currentCS** – linking load할 3개의 프로그램 중 현재 control section의 번호를 저장한다.

**char\* file** – object file의 파일명을 저장한다. error message를 출력할 때 사용된다.

**int charPtr** – Header Record를 읽을 때, 읽을 문자열의 pointer 위치를 저장한다.

**char tmp[7]** – Header Record에서 정보를 읽어올 때 각 문자열을 임시로 저장하는 문자열 변수이다.

**int csAddr** – 현재 control section의 시작 주소를 저장한다.

**int i** – loop counter로 사용되었다.

3.1.2.5 모듈 이름 : **DRecord(char\* line, int currentCS, char\* file)**

3.1.2.5.1 기능

object file에서 Define Record에 해당하는 동작을 수행한다. 현재 control section에서 정의된 external symbol의 이름과 주소를 ESTAB에 저장한다.

3.1.2.5.2 사용 변수

**char\* line** – Define Record 한 줄을 저장하는 문자열이다.

**int currentCS** – linking load할 3개의 프로그램 중 현재 control section의 번호를 저장한다.

**char\* file** - object file의 파일명을 저장한다. error message를 출력할 때 사용된다.

**int charPtr** – Define Record를 읽을 때, 읽을 문자열의 pointer 위치를 저장한다.

**char name[7]** – external symbol의 이름을 저장한다.

**char addr[7]** – external symbol의 주소를 저장한다.

3.1.2.6 모듈 이름 : **RRecord(char\* line, int currentCS, char\* file)**

3.1.2.6.1 기능

object file에서 Refer Record에 해당하는 동작을 수행한다. 외부 control section에서 정의된 symbol에 대해, reference number와 symbol name을 table에 저장하여 주소를 modify할 때 사용한다.

3.2.2.6.2 사용 변수

**char\* line** – Refer Record 한 줄을 저장하는 문자열이다.

**int currentCS** – linking load할 3개의 프로그램 중 현재 control section의 번호를 저장한다.

**char\* file** - object file의 파일명을 저장한다. error message를 출력할 때 사용된다.

**int charPtr** – Refer Record를 읽을 때, 읽을 문자열의 pointer 위치를 저장한다.

**int refAddr** – 외부 참조할 symbol의 주소를 저장한다.

**char referenceNum[7]** – R record에 기입된 reference number를 저장한다.

**char referenceName[7]** – R record에 기입된 external reference symbol name을 저장한다.

3.1.2.7 모듈 이름 : **TRecord(char\* line, int currentCS, char\* file)**

3.1.2.7.1 기능

object file에서 Text Record에 해당하는 동작을 수행한다. Text Record가 시작하는 주소를 읽고, Text Record의 길이동안 object code 를 읽어와 memory에 load하는 작업을 수행한다.

3.1.2.7.2 사용 변수

**char\* line** – Text Record 한 줄을 저장하는 문자열이다.

**int currentCS** – linking load할 3개의 프로그램 중 현재 control section의 번호를 저장한다.

**char\* file** – object file의 파일명을 저장한다. error message를 출력할 때 사용된다.

**int currentAddr** – Text Record가 시작하는 주소를 저장한다.

**int charPtr** – Text Record를 읽을 때, 읽을 문자열의 pointer 위치를 저장한다.

**char strAddr[7]** – Text Record에서, 시작 주소만을 저장하는 문자열이다.

**char tmp[7]** – Text Record에서 읽어온 정보를 임시로 저장하는 문자열이다.

**int tLen** – Text Record의 길이를 저장한다.

**int memVal** – Text Record에서 읽어온 1 byte의 정보를 16진수로 저장하는 변수이다.

**int setPtr** – Text Record의 시작주소 대비, 현재 읽어온 1 byte의 주소를 저장한다.

3.1.2.8 모듈 이름 : **MRecord(char\* line, int currentCS, char\* file)**

3.1.2.8.1 기능

object file에서 Modification Record에 해당하는 동작을 수행한다. M Record에 지정된 주소의 opcode를 modify하여 memory에 다시 저장한다.

3.1.2.8.2 사용 변수

**char\* line** – Modification Record 한 줄을 저장하는 문자열이다.

**int currentCS** – linking load할 3개의 프로그램 중 현재 control section의 번호를 저장한다.

**char\* file** - object file의 파일명을 저장한다. error message를 출력할 때 사용된다.

**int charPtr** – Modification Record를 읽을 때, 읽을 문자열의 pointer 위치를 저장한다.

**char \_mAddr[7]** – modify할 주소의 문자열을 저장한다.

**char tmp[7]** – M Record에서 읽어온 저장을 임시로 저장하는 문자열이다.

**int mAddr** – modify할 주소를 16진수 수로 저장한다.

**int mLen** – modify할 field의 길이를 half bytes로 저장한다.

**int mVal** – modify할 값을 저장한다.

**int prevVal** – modify하기 전 memory에 저장된 값을 나타낸다.

3.1.2.9 모듈 이름 : **newTRaddr(int addr)**

3.1.2.9.1 기능

Text Record가 시작하는 주소를 linked list로 저장하는 함수이다. 1 WORD size가 아닌 상수, 또는 변수가 memory에 load될 때, 이 프로그램을 실행하는 데 있어 오류가 발생할 수 있다. 따라서 Text Record에 저장된 각 record의 시작 주소를 저장하여 오류가 발생하지 않도록 하기 위해 활용하였다.

3.1.2.9.2 사용 변수

**int addr** – 새로운 Text Record가 시작되는 주소를 저장한다.

**newTR\* pNew** – 추가할 주소 정보가 포함된 노드 변수이다.

**newTR\* pMove** – 노드를 추가하기 위해 linked list를 탐색하는 변수이다.

3.1.2.10 모듈 이름 : **freeTRHead()**

3.1.2.10.1 기능

위에서 기술한, 새로운 Text Record의 시작주소를 저장하는 linked list를 free하는 함수이다.

3.1.2.10.2 사용 변수

**newTR\* pFree** – free할 노드를 가리킨다.

3.1.2.11 모듈 이름 : **setMem(int addr, int val, char\* file)**

3.1.2.11.1 기능

Memory의 1byte 값을 변경하는 함수이다.

3.1.2.11.2 사용 변수

**int addr** – 변경할 memory 주소를 나타낸다.

**int val** – 변경할 값을 나타낸다.

**char\* file** – object file의 파일명을 저장한다. error message를 출력할 때 사용된다.

3.1.2.12 모듈 이름 : **addES(EShead\* ES, char\* name, char\* loc)**

3.2.2.12.1 기능

EXTERNAL SYMBOL TABLE에 새로운 노드를 추가하는 함수이다. 각 노드에 대한 정보는 4.1 ESNode 에 자세히 기술하였다.

3.1.2.12.2 사용 변수

**Eshead\* ES** – symbol을 추가할 control section의 head를 나타낸다.

**char\* name** – 추가할 symbol name을 나타낸다.

**char\* loc** – 추가할 symbol의 주소를 나타낸다.

**ESnode\* pNew** – 추가할 symbol의 정보를 저장하는 새로운 node이다.

**ESnode\* ptmp** – node를 새로 추가할 때, linked list를 탐색하는 node이다.

3.1.2.13 모듈 이름 : **addRN(char\* ref, int addr)**

3.1.2.13.1 기능

R Record에 정의되어 있는 external symbol에 대해, 각 symbol의 reference number와 주소 정보를 포함하는 노드를 linked list로 저장하는 함수이다. 이 때 이 list는 modification record에서 object code를 modify하는데 활용된다.

3.1.2.13.2 사용 변수

**char\* ref** – reference number를 문자열로 저장하는 변수이다.

**int addr** – external reference symbol의 주소를 저장한다.

**referNode\* pNew** – reference number와 그 주소를 포함하는, list에 추가될 node이다.

**int refNum** – reference number가 16진수 수로 저장된다.

3.1.2.14 모듈 이름 : **searchESTAB(char\* name, int objCnt)**

3.1.2.14.1 기능

External Symbol name을 parameter로 받아 해당 symbol이 있다면 그 symbol의 주소를, 없다면 -1을 return하는 함수이다. R Record에서 reference number에 대한 주소를 저장할 때 사용된다.

3.1.2.14.2 사용 변수

**char\* name** – search할 symbol name을 저장한다.

**int objCnt** – linking load할 object file의 개수를 저장한다.

**ESnode\* ptmp** – 탐색을 진행하기 위해 임의로 선언한 node이다.

3.1.2.15 모듈 이름 : **searchRN(char\* ref)**

3.1.2.15.1 기능

reference number를 이용해 external reference symbol의 주소를 찾는 기능을 수행한다. M Record에서 사용되며, 매순간 ESTAB 을 탐색해야 하는 비효율성을 해결한다.

3.1.2.15.2 사용 변수

**char\* ref** – search할 reference number의 문자열을 저장한다.

**referNode\* ptmp** – 탐색을 진행하기 위해 임의로 선언한 node이다.

**int refNum** – search할 reference number를 나타낸다.

3.1.2.16 모듈 이름 :  **haltLinkingLoader(char\*\* objFile, FILE\*\* objFP, int objCnt)**

3.1.2.16.1 기능

linking Loader 모듈이 비정상적으로 종료할 때, 혹은 정상적으로 종료할 때 할당했던 메모리를 회수하고 file pointer를 해제하는 역할을 수행한다.

3.1.2.16.2 사용 변수

**char\*\* objFile** – 해제할 object file name이 저장된 문자열 배열이다.

**FILE\*\* objFP** – 해제할 object file pointer가 저장된 file pointer 배열이다.

**int objCnt** – 해제할 object file의 개수를 저장한다.

**ESnode\* pFree** – ESTAB을 해제할 때 사용된, 해제할 노드를 가리킨다.

3.1.2.17 모듈 이름 :  **isObjFile(char\* file)**

3.1.2.17.1 기능

입력된 파일의 확장자가 .obj가 맞는지 아닌지 판단한다. 맞다면 1을, 아니면 0을 return한다.

3.1.2.17.2 사용 변수

**char\* file** – 검사할 file name을 나타낸다.

**int ptr** – 확장자가 나타난 문자열의 pointer 위치를 저장한다.

3.1.2.18 모듈 이름 :  **printES(int objCnt)**

3.1.2.18.1 기능

ESTAB에 저장된 내용을 임시로 출력하는 함수로서, 디버깅을 위해 작성하였다.

3.1.2.18.2 사용 변수

**int objCnt** – linking load한 object file의 총 개수를 저장한다.

**int i** – loop counter로 사용되었다.

**ESnode ptmp** – 정보를 출력하기 위해 탐색을 진행하는 node이다.

3.1.2.19 모듈 이름 :  **freeRN()**

3.1.2.19.1 기능

Reference number와 그 주소를 저장하는 linked list를 해제하는 역할을 수행한다.

3.1.2.19.2 사용 변수

**referNode\* pFree** – 해제할 노드를 가리키는 변수이다

**3.2 execution.c**

3.2.1 설명

Memory에 linking load된 프로그램을 실행시키는데 필요한 기능을 구현한 c file이다. 프로그램 실행 후 register의 상태를 화면에 출력한다. break point가 있다면 break point까지, 없다면 프로그램 끝까지 실행한다.

3.2.2 모듈 정의

3.2.2.1 모듈 이름 : **printReg()**

3.2.2.1.1 기능

Register 값을 출력한다. 이 때 출력하는 register는 A, X, L, PC, B, S, T register이다.

3.2.2.1.2 사용 변수

없음

3.2.2.2 모듈 이름 : **executeProg()**

3.2.2.2.1 기능

메모리에 load된 프로그램을 실행한다. progaddr로 지정된 주소, 혹은 break point로 중단된 지점 이후의 주소부터 실행되며 다음 break point가 있을 때까지, 없다면 프로그램이 종료할 때까지 실행한다.

이 때 SIC/XE machine 및 SIC machine instruction에 대해 수행이 가능하다.

3.2.2.2.2 사용 변수

**int opcode** – memory에서 읽은 object code의 operation code를 분리해 저장한다.

**unsigned char flags** – object code의 n, i, x, b, p, e flags들을 저장한다.

**unsigned char reg** – 2형식 object code의 경우 register 부분의 1byte 수를 저장한다.

**unsigned char ni** – 3형식 object code에서 n, i에 해당하는 부분의 수를 저장한다.

**int format** – 명령어의 format을 저장한다.

**int target** – 3형식 및 4형식 object code에서 displacement/ address를 저장한다.

**newTR\* addr\_of\_new\_TRecord** – 아직 나타나지 않은, 새로운 Text Record가 시작하는 주소를 저장하는 node를 가리킨다.

**int cnt** – 총 명령어 수행 횟수를 가리킨다.

**int MAX\_CNT** – 최대 명령어 수행 가능 횟수를 나타낸다. 전체 프로그램 길이 \* 1000으로 설정되었으며, 무한루프가 도는 등의 에러를 막기 위해 사용하였다.

**int endFlag** – 프로그램이 성공적으로 종료하였다면 0을 저장한다. 그렇지 않다면 1을 저장한다. 프로그램 정상 종료 후 필요한 작업을 수행하기 위해 사용하였다.

**int byteSise** – 현재 실행중인 object code의 메모리상 위치와, text record가 명시한 object code의 메모리 위치의 차가 1이상 4이하인 경우, 해당 변수의 값에 두 위치의 차를 저장하여 memory 위치를 조정하는데 사용하였다.

**int i** – loop counter로 사용되었다.

3.2.2.3 모듈 이름 : **setBP(char\* addr)**

3.2.2.3.1 기능

새로운 breakpoint를 추가한다.

3.2.2.3.2 사용 변수

**char\* addr** – 추가할 주소의 문자열 형태이다.

**int bpAddr**– 추가할 주소를 16진수 수로 저장한 변수이다.

3.2.2.4 모듈 이름 : **clearBP()**

3.2.2.4.1 기능

breakpoint를 모두 삭제한다.

3.2.2.4.2 사용 변수

**BPNode\* pFree** – 해제할 node를 가리킨다.

3.2.2.5 모듈 이름 : **printBP()**

3.2.2.5.1 기능

저장된 breakpoint를 모두 출력한다.

3.2.2.5.2 사용 변수

**BPNode\* ptmp** – 출력하며 탐색을 진행하는 node이다.

3.2.2.6 모듈 이름 : **searchBP()**

3.2.2.6.1 기능

현재 주소를 parameter로 받아와, 해당 주소가 breakpoint인지 아닌지 탐색한다. 맞다면 1을, 아니면 0을 return한다.

3.2.2.6.2 사용 변수

**BPNode\* pMv** - 탐색을 진행하며 사용되는 node이다.

**3.3 opcodeAction.c**

3.3.1 설명

memory에 load된 프로그램을 실행시킬 때, 각 operation code에 맞는 동작을 수행하는데 필요한 기능을 구현한 c file이다. 각 operation code에 따른 다른 기능을 구현하였기에 코드가 길어졌다.

3.3.2 모듈 설명

3.3.2.1 모듈 이름 : **opAct(int opcode, int format, int target, int flags)**

3.3.2.1.1 기능

operation code와 format, target address, flag 등 object code에서 필요한 정보를 받아와 적절한 동작을 수행한다.

3.3.2.1.2 사용 변수

**int opcode** – object code에 opcode에 해당하는 수를 저장한다.

**int format** – object code의 형식을 저장한다.

**int target** – object code에서 register/ displacement/ address 부분의 수를 저장한다.

**int flags** – 3형식 및 4형식 object code에서 n, i, x, b, p, e 부분의 수를 저장한다.

**unsigned char \_reg1, \_reg2** – 2형식 object code에 대해 각 register의 번호를 저장한다.

**int\* r1, \*r2** – 전역변수로 선언된 register의 포인터를 가리킨다.

**unsigned char ni, x, b, p, e** – flags에서 분리해낸 ni, x, b, p, e를 따로 저장하였다.

**int memVal** – memory 값을 참조할 때, 해당 메모리 값을 저장한다.

**int LOC** – memory 값에 접근할 때, 해당 메모리의 주소를 저장한다.

3.3.2.2 모듈 이름 : **getRegPtr(unsigned char reg)**

3.3.2.2.1 기능

2형식 object code를 실행할 때, register number에 따라 전역변수로 선언된 적절한 register의 pointer를 반환한다. 만약 상응하는 register가 없다면 NULL을 return하고 비정상 종료된다.

3.3.2.2.2 사용 변수

**unsigned char reg** – 탐색할 register number를 저장한다.

3.3.2.3 모듈 이름 : **write\_to\_memory(int LOC, int memVal)**

3.3.2.3.1 기능

3 byte object code를 memory에 write한다.

3.3.2.3.2 사용 변수

**int LOC** – write할 memory 위치를 저장한다.

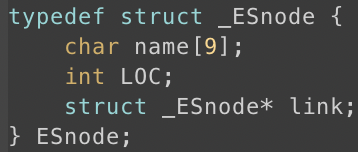
**int memVal** – 새로 write할 메모리 값을 저장한다.

**unsigned char** **b[3]** – 3byte 값을 각각 byte 단위로 저장하는 배열이다.

**int i** – loop counter로 사용되었다.

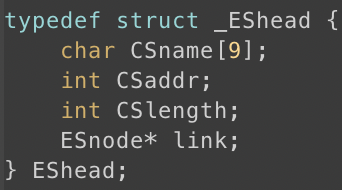
4. 구조체 정의

4.1 **ESNode**



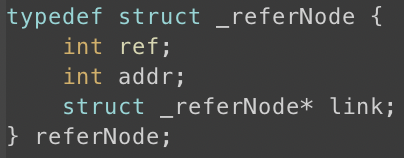
External symbol의 정보를 저장하는 노드이다. symbol의 이름과 주소 정보를 포함한다.

4.2 **EShead**



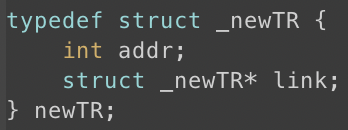
External symbol table을 구성할 때, 각 control section의 정보를 저장하는 노드이다. Control Section의 이름, 주소, 길이 정보를 포함하며, 4.1에 정의한 ESNode\* type을 link로 연결한다.

**4.3 referNode**



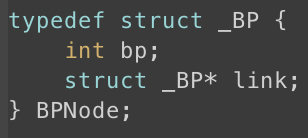
reference number와 그 주소 정보를 포함하는 node이다. linking load과정에서 R Record에 기술된 External reference symbol을 읽어와 정보를 저장하고, modification record에서 ESTAB을 모두 search할 필요 없이 각 control section에서 생성된 linked list를 탐색하여 modify하는 시간 복잡도를 줄일 수 있게 구현하였다.

**4.4 newTR**



object file에 기술된 T Record에서, 시작 주소를 저장하는 node이다. program을 run할 때 WORD size가 아닌 object code를 잘못 읽어와 오류가 발생하는 것을 방지한다.

**4.5 BPNode**



Break point를 저장하는 node이다. program을 run할 때 linked list를 탐색하며 break point에서 run을 일시 중단할 수 있도록 한다.

5. 전역 변수 정의

5.1 **int PROGADDR**

progaddr 명령어로 초기화한 프로그램 시작 주소를 저장한다. 지정해주지 않았다면 default값으로 0x00 번지가 저장된다.

프로그램을 load할 때나 run할 때 시작 주소가 된다.

5.2 **EShead\* ESTAB**

프로그램을 linking load할 때, 생성되는 External Symbol Table을 가리킨다.

5.3 **referNode\* referHead**

여러 개의 object file을 linking load할 때, 각 control section에서 External reference symbol의 reference number와 그 주소를 저장하는 linked list를 가리킨다.

5.4 **newTR\* TRHead**

object file에서 각 Text Record가 시작하는 주소를 저장한 linked list를 가리킨다.

5.5 **BPNode\* BPHead**

Breakpoint들을 저장한 linked list를 가리킨다.

5.6 **int EXEC\_ADDR**

run 명령어를 실행시키는 실행 주소에 해당한다. progaddr 명령어에 의해 변경되거나 bp에서 중단한 다음 시점으로 설정된다. 프로그램이 정상 종료되었다면 마지막으로 실행했던 프로그램의 시작 주소를 가리킨다.

5.7 **int EXEC\_LEN**

run 명령어로 실행시킬 프로그램의 길이를 나타낸다. 단순 주소의 범위를 저장한다.

5.8 **int PROG\_START**

breakpoint를 배제하고, 현재 실행하는 프로그램의 시작 주소를 나타낸다.

5.9 **int PROG\_END**

breakpoint를 배제하고, 현재 실행하는 프로그램이 종료되는 주소를 나타낸다.

5.10 **int continuing**

프로그램 실행 시 breakpoint에서 중단되었다면 해당 변수는 1로 설정된다. 프로그램을 처음부터 시작하였다면 0으로 초기화된다.

**6. 코드**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* linkingLoader.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190506 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

#include "linkingLoader.h"

#include "execution.h"

int progaddr(char\* addr) {

PROGADDR = strToHex(addr, 1);

if (PROGADDR == -1) {

// not a corect hexa decimal num

PROGADDR = 0;

return 0;

}

if ( !validAddr(PROGADDR) ) {

// out of valid address range

PROGADDR = 0;

printf("Invalid Address: out of range\n");

return 0;

}

EXEC\_ADDR = PROG\_START = PROGADDR;

continuing = 0;

printf("PROGADDR SET\n");

return 1;

}

int loader(char\* param) {

// Link & Load obj files

// if fail, return 0.

// else, 1

char\*\* objFile = (char\*\*)malloc(3 \* sizeof(char\*));

int objCnt = 0;

objFile[objCnt++] = strtok(param, " \t");

do {

objFile[objCnt] = strtok(NULL, " \t");

} while (objFile[objCnt++]);

objCnt--;

freeTRHead();

if (objCnt > 3) {

// consider upto 3 files

printf("Too many .obj file. Available only up to 3, you have %d\n", objCnt);

return 0;

}

int i, flag = 0;

FILE\*\* objFP = (FILE\*\*)malloc(objCnt \* sizeof(FILE\*));

for (i = 0; i < objCnt; i++) {

objFP[i] = fopen(objFile[i], "r");

if ( !objFP[i] ) {

printf("FILE NOT FOUND - %s\n", objFile[i]);

flag = 1;

}

}

if (flag) {

free(objFile); free(objFP);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LINKING & LOADING \*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// PASS 1

ESTAB = (EShead\*)malloc(objCnt \* sizeof(EShead));

char line[MAX\_LINE\_LEN];

for (int CS = 0; CS < objCnt; CS++) {

fgets(line, MAX\_LINE\_LEN, objFP[CS]); // get first line

if (line[0] != 'H') {

// wrong obj file

printf("Error occured in file [%s] - NO H RECORD\n", objFile[CS]);

haltLinkingLoader(objFile, objFP, objCnt);

return 0;

}

if ( !HRecord(line, CS, objFile[CS]) ) {

haltLinkingLoader(objFile, objFP, objCnt);

return 0;

}

flag = 0;

while (1) {

memset(line, '\0', sizeof(line));

fgets(line, MAX\_LINE\_LEN, objFP[CS]);

switch (line[0]) {

case 'H':

printf("Error occured in file [%s] - Multiple H Records\n", objFile[CS]);

haltLinkingLoader(objFile, objFP, objCnt);

return 0;

case 'R': case 'T': case 'M': case '.':

break;

case 'E':

flag = 1;

break;

case 'D':

if ( !DRecord(line, CS, objFile[CS]) ) {

haltLinkingLoader(objFile, objFP, objCnt);

return 0;

}

break;

default:

if ( isBlankLine(line) ) break;

printf("Error occured in file [%s] - Wrong format\n", objFile[CS]);

haltLinkingLoader(objFile, objFP, objCnt);

return 0;

}

if (flag) break;

}

}

// PASS 2

for (int CS = 0; CS < objCnt; CS++) {

fseek(objFP[CS], 0, SEEK\_SET);

addRN("01", ESTAB[CS].CSaddr);

flag = 0;

while (1) {

memset(line, '\0', sizeof(line));

fgets(line, MAX\_LINE\_LEN, objFP[CS]);

switch (line[0]) {

case 'H': case 'D': case '.':

// skip for H record & D records

break;

case 'R': // REFER Record

if ( !RRecord(line, objCnt, objFile[CS]) )

return 0;

break;

case 'T': // TEXT Record

if ( !TRecord(line, ESTAB[CS], objFile[CS]) )

return 0;

break;

case 'M': // MODIFICATION Record

if ( !MRecord(line, ESTAB[CS], objFile[CS]) )

return 0;

break;

case 'E': // END Record

flag = 1;

if ( (int)strlen(line) >= 7 ) {

// get the first executable address

char \_\_addr[7];

strncpy(\_\_addr, line + 1, 6);

/\*\* EXEC\_ADDR = strToHex(\_\_addr, 0) + PROGADDR; \*/

/\*\* if (EXEC\_ADDR == -1) { \*/

/\*\* printf("Error occured in file [%s] - Wrong format in E Record**\n**", objFile[CS]); \*/

/\*\* haltLinkingLoader(objFile, objFP, objCnt); \*/

/\*\* return 0; \*/

/\*\* } \*/

}

break;

default:

if ( isBlankLine(line) ) break;

printf("Error occured in file [%s] - Wrong format\n", objFile[CS]);

haltLinkingLoader(objFile, objFP, objCnt);

return 0;

}

if (flag) break;

}

freeRN();

}

loadMap(objCnt);

haltLinkingLoader(objFile, objFP, objCnt);

/\*\* if (EXEC\_ADDR == -1) EXEC\_ADDR = ESTAB[0].CSaddr; // no executable addr specified in E Record \*/

EXEC\_ADDR = PROG\_START = ESTAB[0].CSaddr;

EXEC\_LEN = ESTAB[objCnt-1].CSaddr + ESTAB[objCnt-1].CSlength - EXEC\_ADDR;

PROG\_END = PROG\_START + EXEC\_LEN;

continuing = 0;

/\*\* printf("EXEC - %06X %06X**\n**", EXEC\_ADDR, EXEC\_LEN); \*/

//printTRHead();

return 1;

}

void loadMap(int objCnt) {

int len = 0;

printf("\t\t control\tsymbol\t\taddress\t\tlength\n");

printf("\t\t section\tname\n");

printf("\t\t --------------------------------------------------------\n");

for (int i = 0; i < objCnt; i++) {

printf("\t\t %s\t\t\t\t%04X\t\t%04X\n", ESTAB[i].CSname, ESTAB[i].CSaddr, ESTAB[i].CSlength);

len += ESTAB[i].CSlength;

ESnode\* ptmp = ESTAB[i].link;

for ( ; ptmp; ptmp = ptmp->link) {

printf("\t\t \t\t%s\t\t%04X\n", ptmp->name, ptmp->LOC);

}

}

printf("\t\t --------------------------------------------------------\n");

printf("\t\t \t\t\t\ttotal length\t%04X\n", len);

}

int HRecord(char\* line, int currentCS, char\* file) {

// ACTION for H Record

// save informations about Control Section

// if success, return 1

// else, return 0;

if ( (int)strlen(line) < 19 ) {

printf("Error occured in file [%s] - Wrong format in H Record\n", file);

return 0;

}

int charPtr = 1;

char tmp[7];

strncpy(tmp, line + charPtr, 6);

strcpy(ESTAB[currentCS].CSname, removeSpace(tmp)); // get Control Section Name

charPtr += 6;

strncpy(tmp, line + charPtr, 6); // get starting address of current Control Section

charPtr += 6;

int csAddr = strToHex(tmp, 0);

if (csAddr == -1) {

printf("Error occured in file [%s] - wrong Starting Address in H Record\n", file);

return 0;

}

ESTAB[currentCS].CSaddr = csAddr + PROGADDR;

for (int i = 0; i < currentCS; i++)

ESTAB[currentCS].CSaddr += ESTAB[i].CSlength;

memset(tmp, '\0', sizeof(tmp));

strncpy(tmp, line + charPtr, 6); // get length of obj file

ESTAB[currentCS].CSlength = strToHex(tmp, 0);

ESTAB[currentCS].link = NULL;

if ( !validAddr(ESTAB[currentCS].CSaddr) ) {

printf("Error occured in file [%s] - Invalid Address in H Record.\nPlease set PROGADDR again\n", file);

return 0;

}

return 1;

}

int DRecord(char\* line, int currentCS, char\* file) {

if ( (int)strlen(line) < 19 ) {

printf("Error occured in file [%s] - Wrong format in D Record\n", file);

return 0;

}

int charPtr = 1;

char name[7], addr[7];

while (charPtr <= (int)strlen(line) - 6 - 1) {

memset(name, '\0', sizeof(name));

memset(addr, '\0', sizeof(addr));

strncpy(name, line + charPtr, 6);

charPtr += 6;

strncpy(addr, line + charPtr, 6);

charPtr += 6;

addES(&(ESTAB[currentCS]), name, addr);

}

return 1;

}

int RRecord(char\* line, int objCnt, char\* file) {

// ACTION for R Record

// build Reference Number list

// if error occured, return 0

// else, return 1

if ( (int)strlen(line) < 4 ) {

printf("Error occured in file [%s] - Wrong format in R Record\n", file);

return 0;

}

int charPtr = 1;

int refAddr;

char referenceNum[3], referenceName[7];

while (charPtr < (int)strlen(line) - 1) {

memset(referenceNum, '\0', sizeof(referenceNum));

memset(referenceName, '\0', sizeof(referenceName));

strncpy(referenceNum, line + charPtr, 2);

charPtr += 2;

strncpy(referenceName, line + charPtr, 6);

charPtr += 6;

refAddr = (searchESTAB(removeSpace(referenceName), objCnt));

if (refAddr == -1) {

printf("Error occured in file [%s] - No External Refernece defined: %s\n", file, referenceName);

return 0;

}

addRN(referenceNum, refAddr);

}

return 1;

}

int TRecord(char\* line, EShead CShead, char\* file) {

// ACTION for T Record

// Set Memory Value

// if error occured, return 0

// else, return 1

if ( (int)strlen(line) < 11 ) {

printf("Error occured in file [%s] - Wrong format in T Record\n", file);

return 0;

}

int currentAddr, charPtr = 1;

char strAddr[7], tmp[3];

int tLen, memVal;

strncpy(strAddr, line + charPtr, 6); // start addr of T record

charPtr += 6;

currentAddr = strToHex(strAddr, 0);

if (currentAddr == -1) {

printf("Error occured in file [%s] - Wrong starting address in T Record\n", file);

return 0;

}

currentAddr += CShead.CSaddr;

newTRaddr(currentAddr); // store the addr of new T Record

strncpy(tmp, line + charPtr, 2); // length of T record

tLen = strToHex(tmp, 0);

charPtr += 2;

if (tLen == -1) {

printf("Error occured in file [%s] - Wrong Length of T Record\n", file);

return 0;

}

int setPtr = 0;

while (setPtr < tLen) {

if (charPtr >= (int)strlen(line)) {

printf("Error occured in file [%s] - Wrong format in T Record\n", file);

return 0;

}

memset(tmp, '\0', sizeof(tmp));

strncpy(tmp, line + charPtr, 2); // get one byte

charPtr +=2;

memVal = strToHex(tmp, 0);

if ( !setMem(currentAddr + setPtr++, memVal, file) )

return 0;

}

return 1;

}

int MRecord(char\* line, EShead CShead, char\* file) {

// ACTION for M Record

// Modify Memory value

// if error occured, return 0

// else, return 1

if ( (int)strlen(line) < 12 ) {

printf("Error occured in file [%s] - Wrong format in M Record\n", file);

return 0;

}

int charPtr = 1;

char \_mAddr[7], tmp[3];

int mAddr, mLen, mVal;

strncpy(\_mAddr, line + charPtr, 6); // addr to modify

charPtr += 6;

mAddr = strToHex(\_mAddr, 0);

if (mAddr == -1) {

printf("Error occred in file [%s] - Wrong address in M Record: %s\n", file, \_mAddr);

return 0;

}

strncpy(tmp, line + charPtr, 2); // length to modify

charPtr += 2;

mLen = strToHex(tmp, 0);

if (mLen == -1) {

printf("Error occured in file [%s] - Wrong format for length to be modified in M Record: %s\n", file, tmp);

}

memset(tmp, '\0', sizeof(tmp));

strncpy(tmp, line + charPtr + 1, 2); // Reference number

mVal = searchRN(tmp); // value to be either added or subtracted

if (mVal == -1) {

printf("Error occured in file [%s] - Cannot find reference number at %06X: %s\n", file, mAddr, tmp);

return 0;

}

int prevVal = 0;

mAddr += CShead.CSaddr;

prevVal += (\*(MEMORY + mAddr) << 16);

prevVal += (\*(MEMORY + mAddr + 1) << 8);

prevVal += (\*(MEMORY + mAddr + 2));

mVal = (line[charPtr] == '+') ? prevVal + mVal : prevVal - mVal;

setMem(mAddr, mVal >> 16, file);

setMem(mAddr + 1, (mVal % (2 << 16)) >> 8, file);

setMem(mAddr + 2, mVal % (2 << 8), file);

return 1;

}

void newTRaddr(int addr) {

newTR\* pNew = (newTR\*)malloc(sizeof(newTR));

pNew->addr = addr; pNew->link = NULL;

if (!TRHead) TRHead = pNew;

else {

newTR\* pMove;

for (pMove = TRHead; pMove->link; pMove = pMove->link) ;

pMove->link = pNew;

}

}

void printTRHead() {

newTR\* pMv;

for (pMv = TRHead; pMv; pMv = pMv->link) {

printf("%06X\n", pMv->addr);

}

}

int searchTR(int LOC) {

if (!TRHead) return 0;

newTR\* pMv;

int byteSize;

for (pMv = TRHead; pMv; pMv = pMv->link) {

byteSize = pMv->addr - LOC;

if (byteSize == 0) return 0;

if (byteSize >= 1 && byteSize <= 2) return byteSize;

}

return 0;

}

void freeTRHead() {

if (!TRHead) return ;

newTR\* pFree = TRHead;

while (TRHead) {

pFree = TRHead;

TRHead = TRHead->link;

free(pFree);

}

TRHead = NULL;

}

int setMem(int addr, int val, char\* file) {

// set Memory to value

// simailar with EDIT func in MEMORY.c

// parameter as int

// if success, return 1

// else, return 0

if ( !validAddr(addr) ) {

printf("Error occured in file [%s] - Segmentation falut!\n", file);

return 0;

}

unsigned char value = val & ONE\_BYTE;

memcpy(MEMORY + addr, &value, 1);

return 1;

}

void addES(EShead\* ES, char\* name, char\* loc) {

ESnode\* pNew = (ESnode\*)malloc(sizeof(ESnode));

strcpy(pNew->name, removeSpace(name));

pNew->LOC = strToHex(loc, 0) + ES->CSaddr;

pNew->link = NULL;

if ( ES->link == NULL ) {

ES->link = pNew;

return ;

}

ESnode\* ptmp;

for (ptmp = ES->link; ptmp->link; ptmp = ptmp->link) ;

ptmp->link = pNew;

}

void addRN(char\* ref, int addr) {

int refNum = strToDecimal(ref);

referNode\* pNew = (referNode\*)malloc(sizeof(referNode));

pNew->ref = refNum; pNew->addr = addr;

pNew->link = NULL;

pNew->link = referHead;

referHead = pNew;

}

int searchESTAB(char\* name, int objCnt) {

// search ESTAB

// parameter: External Symbol name

// return LOC

ESnode\* ptmp;

for (int i = 0; i < objCnt; i++) {

ptmp = ESTAB[i].link;

while (ptmp) {

if (strcmp(ptmp->name, name) == 0) return ptmp->LOC;

ptmp = ptmp->link;

}

}

return -1;

}

int searchRN(char\* ref) {

// saerch Reference number

// return address if success

// else, return -1

referNode\* ptmp = referHead;

int refNum = strToDecimal(ref);

while (ptmp) {

if (ptmp->ref == refNum)

return ptmp->addr;

ptmp = ptmp->link;

}

return -1;

}

void printES(int objCnt) {

for (int i = 0; i < objCnt; i++) {

printf("[%s %06X %06X] ", ESTAB[i].CSname, ESTAB[i].CSaddr, ESTAB[i].CSlength);

ESnode\* ptmp = ESTAB[i].link;

for ( ; ptmp; ptmp = ptmp->link) {

printf("- [%s, %06X] ", ptmp->name, ptmp->LOC);

}

printf("\n");

}

}

int isObjFile(char\* file) {

// chech the file name

// it must be "\*.obj" form

int ptr;

if (!file) return 0;

ptr = strlen(file) - 4;

if (ptr <= 0) return 0;

if ( strcmp(file + ptr, ".obj") != 0 ) return 0;

return 1;

}

void haltLinkingLoader(char\*\* objFile, FILE\*\* objFP, int objCnt) {

free(objFile); free(objFP);

if ( ESTAB ) {

ESnode\* pFree;

for (int i = 0; i < objCnt; i++) {

pFree = ESTAB[i].link;

while (ESTAB[i].link) {

pFree = ESTAB[i].link;

ESTAB[i].link = pFree->link;

free(pFree);

}

}

free(ESTAB);

}

}

void freeRN() {

referNode\* pFree;

while (referHead) {

pFree = referHead;

referHead = referHead->link;

free(pFree);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* execution.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190506 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

#include "execution.h"

#include "linkingLoader.h"

int executeProg() {

// execute last linked & loaded program

// its starting addr & len is stored in EXEC\_ADDR & EXEC\_LEN

// if execute successfully, return 1

// else, return 0

if (EXEC\_LEN == -1) {

printf("Please load the program first\n");

return 0;

}

PC = EXEC\_ADDR;

int opcode;

unsigned char flags, reg, ni;

int format;

int target;

int cnt = 0;

int MAX\_CNT = (PROG\_END - PROG\_START) \* 1000;

int endFlag = 0;

if (EXEC\_LEN + EXEC\_ADDR >= 0x100000) {

printf("Segmentation Fault!\n");

return 0;

}

if (!continuing)

L = PROG\_END;

while (PC >= PROG\_START && PC < PROG\_END) {

if ( !validAddr(PC + 2) ) {

printf("Segmentation Fault!\n - %06X\n", PC);

return 0;

}

opcode = \*(MEMORY + PC);

ni = opcode & (unsigned char)0x03;

opcode &= (unsigned char)0xFC;

format = searchWithOpcode(opcode);

int byteSize;

if ( format == 0 ) {

// WORD or BYTE CONST

// no change of REG or MEM

PC += 3;

/\*\* printf("**\n**"); \*/

}

else if (format == 3 && ni == 0) {

// SIC MACHINE COMPATIBLE

/\*\* printf("[PC %06X], **\n**", PC); \*/

PC++;

flags = \*(MEMORY + PC++);

target = (flags & (unsigned char)0x7F) << 8;

flags = flags >> 7 << 3;

target += \*(MEMORY + PC++);

if ( !opAct(opcode, format, target, flags) )

return 0;

byteSize = searchTR(PC);

PC += byteSize;

}

else {

// if opcode exists

/\*\* printf("[PC %06X], ", PC); \*/

PC++;

switch (format) {

case 1:

opAct(opcode, format, 0, 0);

/\*\* printf("**\n**"); \*/

break;

case 2:

reg = \*(MEMORY + PC++);

opAct(opcode, format, reg, 0);

/\*\* printf("**\n**"); \*/

break;

case 3:

/\*\* if (ni == 0) { \*/

/\*\* PC += 2; \*/

/\*\* [> printf("**\n**"); <] \*/

/\*\* if ( ++cnt >= MAX\_CNT) { \*/

/\*\* printf("Program Halted at PC: [%05X] - please set progaddr again, or correct input file**\n**", PC); \*/

/\*\* return 0; \*/

/\*\* } \*/

/\*\* break; \*/

/\*\* } \*/

flags = \*(MEMORY + PC++);

target = (flags & (unsigned char)0x0F) << 8;

flags = (flags >> 4) + (ni << 4);

if (flags % 2) {

// if e flag set

format = 4;

target = (target << 8);

}

for (int i = (format == 3) ? 0 : 1; i >= 0; i--) {

if ( !validAddr(PC) ) {

printf("Segmentation Fault! - %06X\n", PC);

return 0;

}

target += (\*(MEMORY + PC++) << (i \* 8));

}

/\*\* printf("\t[opcode %02X], [format %d], [flags %01X], [target %05X]**\n**", opcode, format, flags, target); \*/

if ( !opAct(opcode, format, target, flags) )

return 0;

break;

}

/\*\* printReg(); \*/

}

if ( ++cnt == MAX\_CNT) {

printf("Program Halted at PC: [%05X] - please set progaddr again, or correct input file\n", PC);

return 0;

}

if (BPHead) {

if (searchBP(PC)) {

printReg();

endFlag = 1;

printf("\t Stop at checkpoint [%05X]\n", PC);

EXEC\_LEN -= (PC - EXEC\_ADDR);

EXEC\_ADDR = PC;

continuing = 1;

break;

}

}

}

if (endFlag == 0) {

printReg();

printf("\t End Program\n");

EXEC\_ADDR = PROG\_START;

EXEC\_LEN = PROG\_END - PROG\_START;

continuing = 0;

}

return 1;

}

void printReg() {

printf("\t\tA : %06X X : %06X\n", A, X);

printf("\t\tL : %06X PC: %06X\n", L, PC);

printf("\t\tB : %06X S : %06X\n", B, S);

printf("\t\tT : %06X\n", T);

}

int setBP(char\* addr) {

// set BreakPoint

// if success, return 1

// else, return 0

int bpAddr = strToHex(addr, 0);

if (bpAddr == -1 || !validAddr(bpAddr)) {

// wrong parameter

printf("\t invalid break point\n");

return 0;

}

BPNode\* pNew = (BPNode\*)malloc(sizeof(BPNode));

pNew->bp = bpAddr; pNew->link = NULL;

printf("\t [ok] create breakpoint %05X\n", bpAddr);

if (!BPHead) {

BPHead = pNew;

return 1;

}

if (bpAddr < BPHead->bp) {

// head - pNew - ...

pNew->link = BPHead;

BPHead = pNew;

}

else if (bpAddr == BPHead->bp) {

printf("\t bp already set\n");

return 0;

}

else {

// head - node - pNew - ...

pNew->link = BPHead->link;

BPHead->link = pNew;

// head - pMv1 - pNew - pMv2

BPNode\* pMv1 = BPHead;

BPNode\* pMv2 = pNew->link;

while (pMv2) {

if (pMv2->bp == bpAddr) {

printf("\t bp already set\n");

return 0;

}

else if (pMv2->bp < bpAddr) {

pMv1->link = pMv2;

pNew->link = pMv2->link;

pMv2->link = pNew;

pMv1 = pMv1->link;

pMv2 = pNew->link;

}

else break;

}

}

return 1;

}

int clearBP() {

// clear all breakpoints

if (!BPHead) {

printf("\t BP Already Cleared!\n");

return 0;

}

BPNode\* pFree;

while (BPHead) {

pFree = BPHead;

BPHead = BPHead->link;

free(pFree);

}

BPHead = NULL;

printf("\t [ok] clear all breakpoints\n");

return 1;

}

void printBP() {

// print all breakpoints

if (!BPHead) {

printf("\t No Breakpoint Set\n");

return ;

}

printf("\t breakpoint\n");

printf("\t ----------\n");

BPNode\* ptmp = BPHead;

for ( ; ptmp; ptmp = ptmp->link)

printf("\t %05X\n", ptmp->bp);

}

int searchBP(int PC) {

// Check whether PC is breakpoint or not

// if it is, return 1

// else, return 0

BPNode\* pMv = BPHead;

while (pMv) {

if (pMv->bp == PC)

return 1;

else

pMv = pMv->link;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* opcodeAction.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190506 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

#include "execution.h"

int opAct(int opcode, int format, int target, int flags) {

// operate proper action

// format always 1 ... 4

// target, either reg or disp/addr

// flags get nixbpe info

// if DONE successfully, return 1

// else, return 0

if (format == 1) {

switch (opcode) {

case 0xC4: // FIX

A = (int)F;

break;

case 0xC0: // FLOAT

F = (float)A;

break;

/\*\* case 0xF4: // HIO \*/

/\*\* break; \*/

/\*\* case 0xC8: // NORM \*/

/\*\* break; \*/

/\*\* case 0xF0: // SIO \*/

/\*\* break; \*/

/\*\* case 0xF8: // TIO \*/

/\*\* break; \*/

}

}

else if (format == 2) {

unsigned char \_reg1 = target >> 4;

unsigned char \_reg2 = target & (unsigned char)0x0F;

int\* r1 = getRegPtr(\_reg1);

int\* r2 = getRegPtr(\_reg2);

switch (opcode) {

case 0x90: // ADDR r1, r2

if (!r1 || !r2) return 0;

\*r2 += \*r1;

break;

case 0xB4: // CLEAR r1

if (!r1) return 0;

\*r1 = 0;

break;

case 0xA0: // COMPR r1, r2

if (\*r1 == \*r2) CC = 0;

else if (\*r1 > \*r2) CC = 1;

else CC = -1;

break;

case 0x9C: // DIVR r1, r2

if (!r1 || !r2) return 0;

if ( \*r1 == 0 ) {

printf("Divided by 0\n");

return 0;

}

\*r2 /= \*r1;

break;

case 0x98: // MULR r1, r2

if (!r1 || !r2) return 0;

\*r2 \*= \*r1;

break;

case 0xAC: // RMO r1, r2

if (!r1 || !r2) return 0;

\*r2 = \*r1;

break;

case 0xA4: // SHIFTL r1, n

if (!r1) return 0;

\*r1 = (\*r1 << (\_reg2+1));

break;

case 0xA8: // SHIFTR r1, n

if (!r1) return 0;

\*r1 = (\*r1 >> (\_reg2+1));

break;

case 0x94: // SUBR r1, r2

if (!r1 || !r2) return 0;

\*r2 -= \*r1;

break;

/\*\* case 0xB0: // SVC n \*/

/\*\* break; \*/

case 0xB8: // TIXR r1

X += 1;

if (!r1) return 0;

if (X == \*r1) CC = 0;

else if (X > \*r1) CC = 1;

else CC = -1;

break;

}

}

else {

// format 3 or 4

unsigned char ni, x, b, p, e;

ni = flags >> 4;

x = (flags >> 3) % 2;

b = (flags >> 2) % 2;

p = (flags >> 1) % 2;

e = flags % 2;

if (p)

target = (signed char)target + PC;

else if (b)

target += B;

if (x)

target += X;

/\*\* printf("%02X - %d %d %d %d %d**\n**", flags, ni, x, b, p, e); \*/

int memVal = 0;

int LOC = target;

switch (ni) {

case 1: // immediate addressing

memVal = target;

break;

case 2: // indirect addressing

for (int j = 0; j < 2; j++) {

// READ MEMORY

// READ VALUE AS ADDR OF MEMORY AGAIN!

memVal = 0;

for (int i = 2; i >= 0; i--) {

// READ ONE WORD SIZE from the MEMORY

if ( !validAddr(LOC) ) {

printf("%06X\n", LOC);

printf("Segmentation Fault!\_1\n");

return 0;

}

memVal += (\*(MEMORY + LOC++) << (i \* 8));

}

if (j == 0) {

LOC = memVal;

target = memVal;

}

/\*\* printf("%06X**\n**", memVal); \*/

}

break;

case 3: // simple addressing

for (int i = 2; i >= 0; i--) {

// READ ONE WORD SIZE from the MEMORY

if ( !validAddr(LOC) ) {

printf("Segmentation Fault!\_2\n");

return 0;

}

memVal += ((\*(MEMORY + LOC++) << (i \* 8)));

/\*\* printf("%06X ", memVal); \*/

}

/\*\* printf("**\n**"); \*/

break;

}

switch (opcode) {

// memory operand is stored in memVal variable

case 0x18: // ADD m

A += memVal;

break;

/\*\* case 0x58: // ADDF m \*/

/\*\* break; \*/

case 0x40: // AND m

A &= memVal;

break;

case 0x28: // COMP m

if (A == memVal) CC = 0;

else if (A > memVal) CC = 1;

else CC = -1;

break;

/\*\* case 0x88: // COMPF m \*/

/\*\* break; \*/

case 0x24: // DIV m

if (memVal == 0) {

printf("Divided by 0\n");

return 0;

}

A /= memVal;

break;

/\*\* case 0x64: // DIVF m \*/

/\*\* break; \*/

case 0x3C: // J m

PC = target;

break;

case 0x30: // JEQ m

if (CC == 0) PC = target;

break;

case 0x34: // JGT m

if (CC == 1) PC = target;

break;

case 0x38: // JLT m

if (CC == -1) PC = target;

break;

case 0x48: // JSUB m

L = PC;

PC = target;

break;

case 0x00: // LDA m

/\*\* printf("%06X %06X**\n**", A, target); \*/

A = memVal;

break;

case 0x68: // LDB m

B = memVal;

break;

case 0x50: // LDCH m

memVal = (unsigned)(memVal >> 16);// | (unsigned char)0xFFFF00;

A = (A >> 8) << 8;

A += memVal;

break;

/\*\* case 0x70: // LDF m \*/

/\*\* break; \*/

case 0x08: // LDL m

L = memVal;

break;

case 0x6C: // LDS m

S = memVal;

break;

case 0x74: // LDT m

T = memVal;

break;

case 0x04: // LDX m

X = memVal;

break;

/\*\* case 0xD0: // LPS m \*/

/\*\* break; \*/

case 0x20: // MUL m

A \*= memVal;

break;

/\*\* case 0x60: // MULF m \*/

/\*\* break; \*/

case 0x44: // OR m

A |= memVal;

break;

case 0xD8: // RD m

CC = 1;

break;

case 0x4C: // RSUB

PC = L;

break;

/\*\* case 0xEC: // SSK m \*/

/\*\* break; \*/

case 0x0C: // STA m

LOC -= 3;

if ( !write\_to\_memory(LOC, A) )

return 0;

break;

case 0x78: // STB m

LOC -= 3;

if ( !write\_to\_memory(LOC, B) )

return 0;

break;

case 0x54: // STCH m

LOC -= 3;

if ( !validAddr(LOC) ) {

printf("Segmentation Fault!\_STCH\n");

return 0;

}

memVal = A & (unsigned char)0x0FF;

memcpy(MEMORY + LOC, &memVal, 1);

break;

/\*\* case 0x80: // STF m \*/

/\*\* break; \*/

/\*\* case 0xD4: // STI m \*/

/\*\* break; \*/

case 0x14: // STL m

LOC -= 3;

if ( !write\_to\_memory(LOC, L) )

return 0;

break;

case 0x7C: // STS m

LOC -= 3;

if ( !write\_to\_memory(LOC, S) )

return 0;

break;

case 0xE8: // STSW m

LOC -= 3;

if ( !write\_to\_memory(LOC, SW) )

return 0;

break;

case 0x84: // STT m

LOC -= 3;

if ( !write\_to\_memory(LOC, T) )

return 0;

break;

case 0x10: // STX m

LOC -= 3;

if ( !write\_to\_memory(LOC, X) )

return 0;

break;

case 0x1C: // SUB m

A -= memVal;

break;

/\*\* case 0x5C: // SUBF m \*/

/\*\* break; \*/

case 0xE0: // TD m

CC = 1;

break;

case 0x2C: // TIX m

X += 1;

if (X == memVal) CC = 0;

else if (X > memVal) CC = 1;

else CC = -1;

break;

/\*\* case 0xDC: // WD m \*/

/\*\* break; \*/

}

}

return 1;

}

int\* getRegPtr(unsigned char reg) {

switch (reg) {

case 0x00:

return &A;

case 0x01:

return &X;

case 0x02:

return &L;

case 0x03:

return &B;

case 0x04:

return &S;

case 0x05:

return &T;

case 0x06:

return &F;

case 0x08:

return &PC;

case 0x09:

return &SW;

default:

return NULL;

}

}

int write\_to\_memory(int LOC, int memVal) {

// Write ONE WORD SIZE value to memory

// if success, return 1

// else, return 0

if ( !validAddr(LOC + 2) ) {

printf("Segmentation Fault!\_writing\n");

return 0;

}

unsigned char b[3];

b[0] = memVal >> 16;

b[1] = (memVal >> 8) & (unsigned char)0x00FF;

b[2] = memVal & (unsigned char)0x00FF;

for (int i = 0; i < 3; i++) {

b[i] &= ONE\_BYTE;

memcpy(MEMORY + LOC++, &(b[i]), 1);

}

return 1;

}