과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**[20161631]**

**[임동진]**

목 차

1. 프로그램 개요 6

2. 프로그램 설명 6

2.1 프로그램 흐름도 6

3. 모듈 정의 8

3.1 모듈 이름 : execute\_progaddr(Command \*user\_command, State \*state\_store) 8

3.1.1 기능 8

3.1.2 사용 변수 8

3.2 모듈 이름: execute\_run(State \*state\_store) 8

3.2.1 기능 8

3.2.2 사용 변수 8

3.3 모듈이름execute\_bp(Command \*user\_command, State \*state\_store) 9

3.3.1 기능 9

3.3.2 사용변수 9

3.4 모듈이름: execute\_bp\_clear(State \*state\_store) 9

3.4.1 기능 9

3.4.2 사용변수 9

3.5 모듈이름: execute\_loader(Command \*user\_command, State \*state\_store) 9

3.5.1 기능 9

3.5.2 사용변수 9

3.6 모듈이름: execute\_bp\_list(State\* state\_store) 9

3.6.1 기능 9

3.6.2 사용변수 9

3.7 모듈이름: validate\_progaddr\_parameters(Command \*user\_command) 9

3.8 모듈이름: validate\_bp\_parameters(Command\* user\_command 9

3.8.1 기능 9

3.8.2 사용변수 9

3.9 모듈이름: validate\_bp\_clear\_parameters(Command\* user\_command) 10

3.9.1 기능 10

3.9.2 사용변수 10

3.10 모듈이름: validate\_loader\_parameters(Command\* user\_command) 10

3.10.1 기능 10

3.10.2 사용변수 10

3.11 모듈이름: op\_format\_by\_op\_num(int op\_num) 10

3.11.1 기능 10

3.11.2 사용변수 10

3.12 모듈이름: registers 구조체 11

3.12.1 기능 11

3.12.2 정의 내용 11

3.13 모듈이름: load\_info\_node구조체 11

3.13.1 기능 11

3.13.2 정의 내용 11

3.14 모듈이름: load\_info\_list구조체 11

3.14.1 기능 11

3.14.2 정의 내용 11

3.15 모듈이름: debugger구조체 12

3.15.1 기능 12

3.15.2 정의 내용 12

3.16 모듈이름: instruction구조체 13

3.16.1 기능 13

3.16.2 정의 내용 13

3.17 모듈이름: load\_info\_type ENUM 14

3.17.1 기능 14

3.17.2 정의 내용 14

3.18 모듈이름: Operator ENUM 14

3.18.1 기능 14

3.18.2 정의 내용 14

3.19 모듈이름: ADDRESSING\_MODE ENUM 15

3.19.1 기능 15

3.19.2 정의 내용 15

3.20 모듈이름: construct\_debugger() 15

3.20.1 기능 15

3.20.2 사용변수 15

3.21 모듈이름: destroy\_debugger(Debugger\*\* debugger) 15

3.21.1 기능 15

3.21.2 사용변수 15

3.22 모듈이름: construct\_registers() 15

3.22.1 기능 15

3.22.2 사용변수 15

3.23 모듈이름: destroy\_registers(Registers\*\* registers) 16

3.23.1 기능 16

3.23.2 사용변수 16

3.24 모듈이름: reset\_registers(Registers\* registers) 16

3.24.1 기능 16

3.24.2 사용변수 16

3.25 모듈이름: loader\_linker(Debugger \*debugger, Memories \*memories) 16

3.25.1 기능 16

3.25.2 사용변수 16

3.26 모듈이름: run(Debugger \*debugger, Memories \*memories) 17

3.26.1 기능 17

3.26.2 사용변수 17

3.27 모듈이름: loader\_linker\_pass1(Debugger \*debugger) 18

3.27.1 기능 18

3.27.2 사용변수 18

3.28 모듈이름: loader\_linker\_pass1\_one(Debugger \*debugger, int file\_num, int \*csaddr) 18

3.28.1 기능 18

3.28.2 사용변수 18

3.29 모듈이름: loader\_linker\_pass2(Debugger \*debugger, Memories \*memories) 19

3.29.1 기능 19

3.29.2 사용변수 19

3.30 모듈이름: loader\_linker\_pass2\_one(Debugger \*debugger, Memories \*memories, int file\_num , int \*csaddr) 19

3.30.1 기능 19

3.30.2 사용변수 19

3.31 모듈이름: construct\_load\_info\_list() 20

3.31.1 기능 20

3.31.2 사용변수 20

3.32 모듈이름: print\_load\_infos(LoadInfoList \*load\_infos) 20

3.32.1 기능 20

3.32.2 사용변수 20

3.33 모듈이름: execute\_operator(Debugger \*debugger, Memories \*memories, Instruction \*instruction) 21

3.33.1 기능 21

3.33.2 사용변수 21

3.34 모듈이름: calculate\_TA(Instruction\* instruction, Registers\* registers) 22

3.34.1 기능 22

3.34.2 사용변수 22

3.35 모듈이름: handling\_bp(Debugger \*debugger, int instruction\_size) 22

3.35.1 기능 22

3.35.2 사용변수 22

3.36 모듈이름: calculate\_addressing\_mode(Instruction\* instruction, bool jump\_op) 23

3.36.1 기능 23

3.36.2 사용변수 23

3.37 모듈이름: load\_from\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t \*value, size\_t bytes, bool jump\_op) 23

3.37.1 기능 23

3.37.2 사용변수 23

3.38 모듈이름: store\_to\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t value, size\_t bytes) 24

3.38.1 기능 24

3.38.2 사용변수 24

3.39 모듈이름: load\_from\_register(Debugger \*debugger, int reg\_id, uint32\_t \*val) 25

3.39.1 기능 25

3.39.2 사용변수 25

3.40 모듈이름: store\_to\_register(Debugger \*debugger, int reg\_id, uint32\_t val) 25

3.40.1 기능 25

3.40.2 사용변수 25

3.41 모듈이름: get\_reg\_by\_id(Registers \*registers, int reg\_id) 26

3.41.1 기능 26

3.41.2 사용변수 26

3.42 모듈이름: print\_registers(Registers\* registers) 26

3.42.1 기능 26

3.42.2 사용변수 26

4. 전역 변수 정의 26

4.1 long income[100] 26

4.2 int month[100], day[100], year[100] 26

4.3 int ctr 26

4.4 int cont 27

5. 코드 27

# 프로그램 개요

SIC/XE 의 Linking Loader를 구현한다. 여러 object file을 linking하여 메모리에 load하는 작업을 수행한다. 또한 load 된 프로그램을 실행하고, 브레이크 포인트를 통해 디버깅할 수 있는 디버깅 기능까지 구현되어있다.

# 프로그램 설명

## 프로그램 흐름도

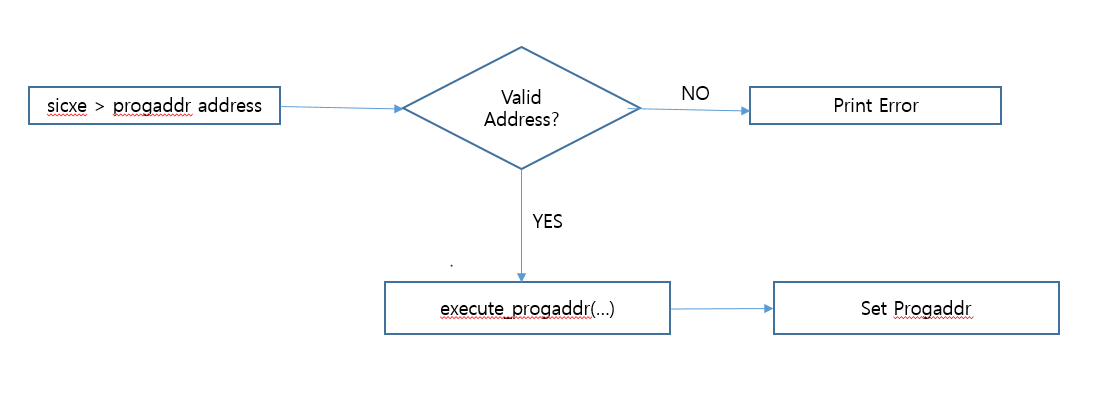


그림 1> progaddr 명령 흐름도

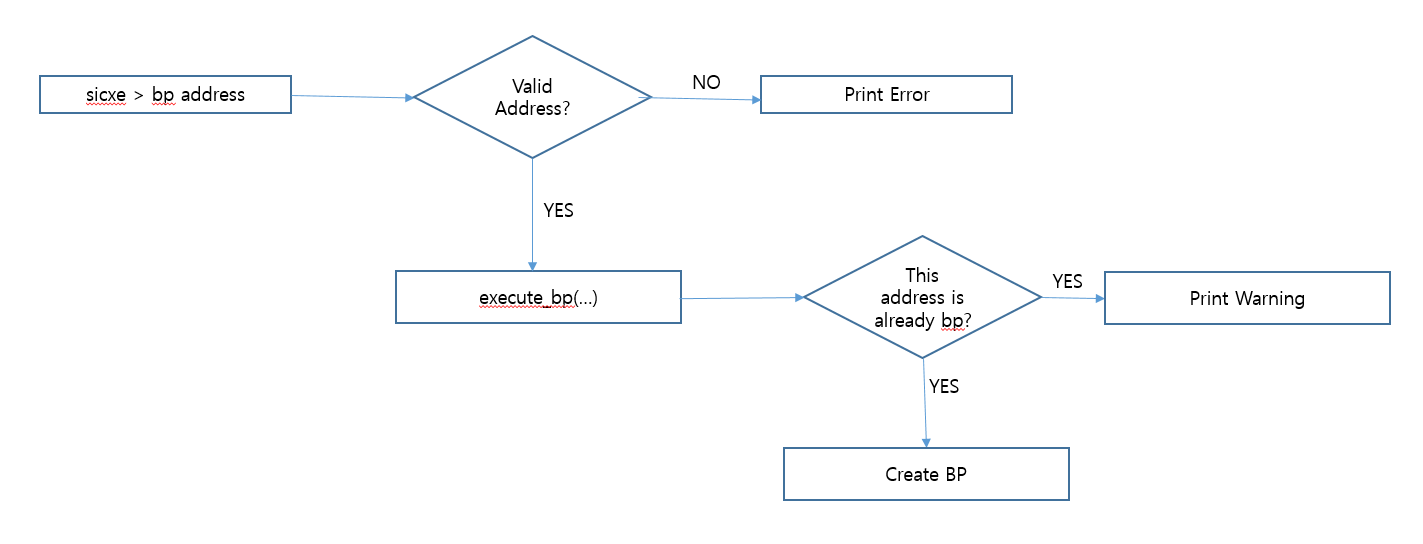


그림 2> bp [address] 명령 흐름도



그림 3> bp 명령 흐름도

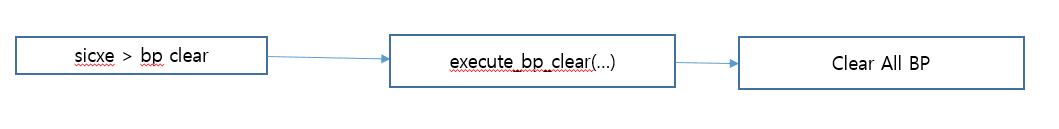


그림 4> bp clear 명령 흐름도

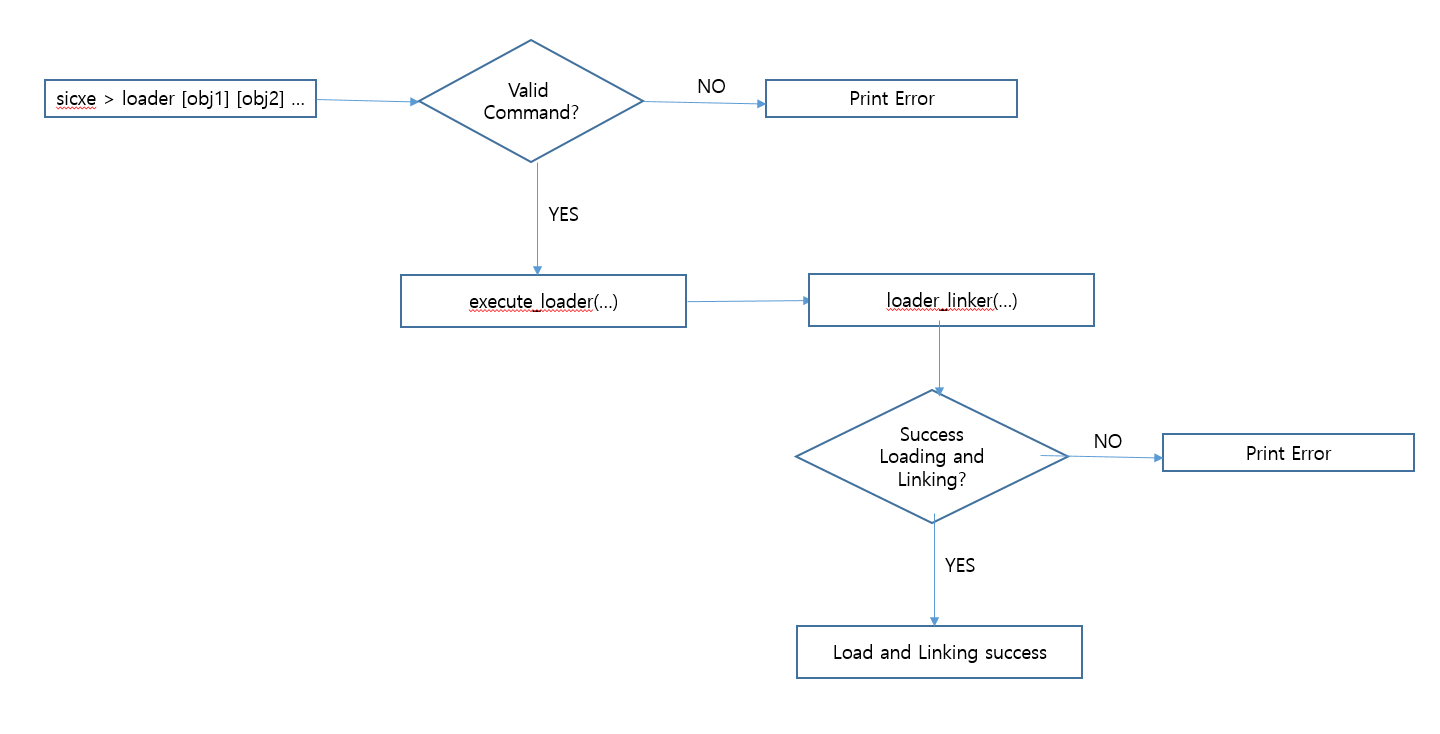


그림 5> loader [obj1] [obj2] … 명령 흐름도

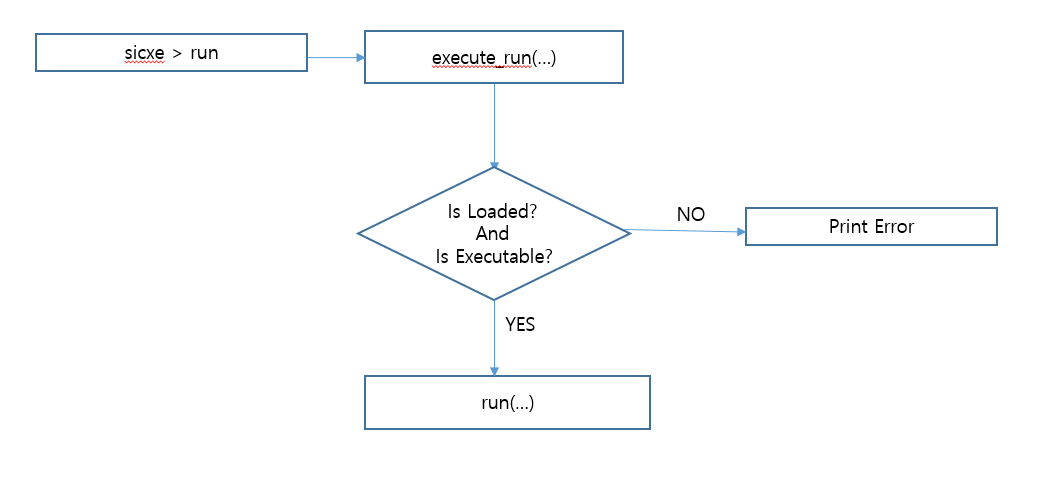


그림 6> run 명령 흐름도

# 모듈 정의

## 모듈 이름 : execute\_progaddr(Command \*user\_command, State \*state\_store)

### 기능

progaddr [address] 명령을 수행한다. 프로그램이 load 될 시작 주소를 지정하는 명령이다. ..

### 사용 변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |
| State \*state\_store | 현재 프로그램의 모든 상태정보를 저장한다. |
| int addr | Progaddr [address]에서 address로 들어온 값을 저장한다. |

## 모듈 이름: execute\_run(State \*state\_store)

### 기능

현재 프로그램이 run 될수 있는지 확인하고, run 할수있다면 run(…)함수를 호출하고, 아니라면 에러를 출력해준다.. 실질적인 run 명령 수행은 run(…) 함수에서 수행된다.

run(..) 함수 수행 성공 실패 여부에 따라서 적절한 에러 핸들링을 진행한다.

### 사용 변수

|  |  |
| --- | --- |
| bool status; | run(..)함수 수행의 성공 실패 여부를 저장한다.. |
| State \*state\_store | 현재 프로그램의 모든 상태정보를 저장한다. |

## 모듈이름execute\_bp(Command \*user\_command, State \*state\_store)

### 기능

bp [address] 명령을 수행한다. 브레이크 포인트를 설정한다. 이미 존재하는 bp라면 경고를 출력한다.

### 사용변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |
| State \*state\_store | 현재 프로그램의 모든 상태정보를 저장한다. |
| int addr | Progaddr [address]에서 address로 들어온 값을 저장한다. |

## 모듈이름: execute\_bp\_clear(State \*state\_store)

### 기능

bp clear 명령을 수행한다. 모든 브레이크 포인트를 삭제한다..

### 사용변수

|  |  |
| --- | --- |
| State \*state\_store | 현재 프로그램의 모든 상태정보를 저장한다. |
| int cnt | 반복문 수행을 위한 보조 변수 |

.

## 모듈이름: execute\_loader(Command \*user\_command, State \*state\_store)

### 기능

loader\_linker(…) 함수를 호출한다. 실질적인 loader [obj1] [obj2] … 명령 수행은 loader\_linker(…) 함수에서 수행된다. loader\_linker(..) 함수 수행 성공 실패 여부에 따라서 적절한 에러 핸들링을 진행한다

### 사용변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |
| State \*state\_store | 현재 프로그램의 모든 상태정보를 저장한다. |
| bool status | loader\_linker(…) 함수 수행의 성공 실패 여부를 저장한다. |

## 모듈이름: execute\_bp\_list(State\* state\_store)

### 기능

bp 들의 목록을 출력하는 명령인 bp 명령을 수행한다.

### 사용변수

|  |  |
| --- | --- |
| State \*state\_store | 현재 프로그램의 모든 상태정보를 저장한다. |
| int cnt | 반복문 수행을 위한 보조 변수 |

## 모듈이름: validate\_progaddr\_parameters(Command \*user\_command)

#### 기능

progaddr [address] 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID\_PARAMETERS 또는 INVALID\_PARAMETERS를 반환하여준다.

#### 사용변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |

## 모듈이름: validate\_bp\_parameters(Command\* user\_command

### 기능

bp [address] 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID\_PARAMETERS 또는 INVALID\_PARAMETERS를 반환하여준다.

### 사용변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |

## 모듈이름: validate\_bp\_clear\_parameters(Command\* user\_command)

### 기능

bp clear 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID\_PARAMETERS 또는 INVALID\_PARAMETERS를 리턴하여준다

### 사용변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |

## 모듈이름: validate\_loader\_parameters(Command\* user\_command)

### 기능

loader [obj1] [obj2] … 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID\_PARAMETERS 또는 INVALID\_PARAMETERS를 리턴하여준다

### 사용변수

|  |  |
| --- | --- |
| Command \*user\_command | .사용자가 입력한 명령에 대한 정보를 저장한다. |

## 모듈이름: op\_format\_by\_op\_num(int op\_num)

### 기능

Opcode 에 따라서 대응 되는 format(1 또는 2 또는 3/4)을 리턴한다.

### 사용변수

|  |  |
| --- | --- |
| int op\_num | Opcode 정보 |
| bool format3\_4[260] | Format3\_4에 해당하는 opcode 정보를 true로 하여 저장함 |
| bool format2[260] | Format2에 해당하는 opcode 정보를 true로 하여 저장함 |

## 모듈이름: registers 구조체

### 기능

레지스터 정보를 저장한다..

### 정의 내용

|  |
| --- |
| typedef struct registers {  uint32\_t A;  uint32\_t L;  uint32\_t X;  uint32\_t PC;  uint32\_t B;  uint32\_t S;  uint32\_t T;  uint32\_t SW;  }Registers; |

## 모듈이름: load\_info\_node구조체

### 기능

load info\_list 구조체의 node 구조체 역할을 한다.

### 정의 내용

|  |
| --- |
| typedef struct load\_info\_node {  enum load\_info\_type type;  char name[15];  int addr;  int length;  }LoadInfoNode; |

## 모듈이름: load\_info\_list구조체

### 기능

loader [obj1] [obj2] …명령의 로딩된 결과를 저장한다.

### 정의 내용

|  |
| --- |
| typedef struct load\_info\_list {  LoadInfoNode list[1001];  int count;  } LoadInfoList;; |

## 모듈이름: debugger구조체

### 기능

load, run, bp, 레지스터 관리 등의 역활을함. 즉 해당 명령들의 수행을 위해 필요한 정보들을 통합해서 관리한다.

### 정의 내용

|  |
| --- |
| typedef struct debugger {  uint32\_t start\_address;  int end\_address;  bool break\_points[MAX\_BP\_NUM];  int bp\_count;  Registers\* registers;  int run\_count;.  int previous\_bp;  char\* filenames[3];  int file\_count;  SymbolTable\* estab;  LoadInfoList\* load\_infos;  uint32\_t total\_length;  bool is\_running;  bool is\_loaded;  } Debugger; |

## 모듈이름: instruction구조체

### 기능

Instruction과 관련한 정보를 저장한다..

### 정의 내용

|  |
| --- |
| typedef struct instruction{  bool extend;  unsigned char opcode;  struct {  union{  struct{  uint16\_t r2 : 4;  uint16\_t r1 : 4;  uint16\_t opcode : 8;  } p2;  struct{  uint32\_t address: 12;  uint32\_t e : 1;  uint32\_t p : 1;  uint32\_t b : 1;  uint32\_t x : 1;  uint32\_t i : 1;  uint32\_t n : 1;  uint32\_t opcode : 6;  } p3;  struct{  uint32\_t address: 20;  uint32\_t e : 1;  uint32\_t p : 1;  uint32\_t b : 1;  uint32\_t x : 1;  uint32\_t i : 1;  uint32\_t n : 1;  uint32\_t opcode : 6;  } p4;  uint32\_t val;  } param;  bool extend;  } param;  } Instruction; |

## 모듈이름: load\_info\_type ENUM

### 기능

Load 된 정보의 유형을 enum으로 설정

### 정의 내용

|  |
| --- |
| enum load\_info\_type {  INFO\_TYPE\_CONTROL\_SECTION,  INFO\_TYPE\_SYMBOL  } LoadInfoType; |

## 모듈이름: Operator ENUM

### 기능

Operator 이름과 opcode를 매핑한 enum

### 정의 내용

|  |
| --- |
| typedef enum {  LDA = 0x00,  LDB = 0x68,  LDT = 0x74,  LDX = 0x04,  LDCH = 0x50,  STA = 0x0C,  STL = 0x14,  STX = 0x10,  STCH = 0x54,  JSUB = 0x48,  JEQ = 0x30,  JGT = 0x34,  JLT = 0x38,  J = 0x3C,  COMP = 0x28,  TD = 0xE0,  RD = 0xD8,  RSUB = 0x4C,  WD = 0xDC,  CLEAR = 0xB4,  COMPR = 0xA0,  TIXR = 0xB8  } Operator; |

## 모듈이름: ADDRESSING\_MODE ENUM

### 기능

Addressing 모드를 enum으로 매핑함.

### 정의 내용

|  |
| --- |
| typedef enum {  ENUM\_IMMEDIATE\_ADDRESSING,  ENUM\_SIMPLE\_ADDRESSING,  ENUM\_INDIRECT\_ADDRESSING,  ENUM\_ADDRESSING\_ERROR  } ADDRESSING\_MODE; |

## 모듈이름: construct\_debugger()

### 기능

Debugger 구조체의 생성자(메모리를 할당받고 초기화를 진행하는) 함수..

### 사용변수

없음.

## 모듈이름: destroy\_debugger(Debugger\*\* debugger)

### 기능

Debugger 구조체와 그와 연관된 변수들에 대한 소멸자(메모리를 해제하는) 함수.

### 사용변수

|  |  |
| --- | --- |
| Debugger\*\* debugger | 해제할 debugger 를 포인팅함 |

## 모듈이름: construct\_registers()

### 기능

Registers 구조체의 생성자 함수.

### 사용변수

없음.

## 모듈이름: destroy\_registers(Registers\*\* registers)

### 기능

Registers 구조체의 소멸자 함수.

### 사용변수

|  |  |
| --- | --- |
| Registers\*\* registers | 해제할 registers 를 포인팅함 |

## 모듈이름: reset\_registers(Registers\* registers)

### 기능

Registers 구조체를 리셋함..

### 사용변수

|  |  |
| --- | --- |
| Registers\* registers | 레지스터 정보를 저장함. |

## 모듈이름: loader\_linker(Debugger \*debugger, Memories \*memories)

### 기능

loader [obj1] [obj2].. 명령을 실질적으로 수행함. Pass1 을 수행하는 함수와 pass2 를 수행하는 함수를 호출하고 성공적으로 load되었다면 load 된정보를 출력해서 보여주는 함수를 호출한다.

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| Memories \*memories | 메모리 정보를 저장함 |

## 모듈이름: run(Debugger \*debugger, Memories \*memories)

### 기능

run 명령을 실질적으로 수행함. Load 된 프로그램을 실행하고 중간에 브레이크포인트 가 있으면 이를 적절히 핸들링한다.

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| Memories \*memories | 메모리 정보를 저장함 |
| Registers\* registers | 레지스터 정보를 저장함. |
| uint32\_t tmp | Tmp 변수 |
| bool is\_continue | Continue 할지 여부를 저장함 |
| uint8\_t opcode | Opcode 를 저장함 |
| uint32\_t instruction\_val | Instruction의 값을 저장함. |
| uint8\_t memory\_val | 메모리 값을 저장함 |
| int instruction\_size | Instruction 의 크기를 저장함 |

## 모듈이름: loader\_linker\_pass1(Debugger \*debugger)

### 기능

Loader 의 Pass1 과정을 수행한다..

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| int csaddr | CSADDR 정보를 저장한다. |

## 모듈이름: loader\_linker\_pass1\_one(Debugger \*debugger, int file\_num, int \*csaddr)

### 기능

Loader 의 Pass1 과정중에 파일 하나에 대한 작업을 수행한다..

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| int\* csaddr | CSADDR 정보를 저장한다. |
| Int file\_num | 파일 번호를 저장한다. |
| FILE \*fp | 파일 포인터 |
| char buf[1010] | 버퍼 정보를 저장함 |
| bool is\_header | 헤더가 있는지 여부를 저장함 |
| int base\_address | base가 되는 주소를 저장함. 원래의 csaddr 주소 |

## 모듈이름: loader\_linker\_pass2(Debugger \*debugger, Memories \*memories)

### 기능

Loader 의 Pass2 과정을 수행한다...

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| int csaddr | CSADDR 정보를 저장한다. |
| Memories \*memories | 메모리 정보를 저장한다. |

## 모듈이름: loader\_linker\_pass2\_one(Debugger \*debugger, Memories \*memories, int file\_num , int \*csaddr)

### 기능

Loader 의 Pass2 과정중 파일 하나에 대한 작업을 수행한다...

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| int\* csaddr | CSADDR 정보를 저장한다. |
| Int file\_num | 파일 번호를 저장한다. |
| Memories \*memories | 메모리 정보를 저장한다. |
| FILE\* fp | 파일 포인터 |

## 모듈이름: construct\_load\_info\_list()

### 기능

Load\_info\_list 구조체의 생성자 함수

### 사용변수

없음

## 모듈이름: print\_load\_infos(LoadInfoList \*load\_infos)

### 기능

Load 된 정보를 출력해줌

### 사용변수

|  |  |
| --- | --- |
| LoadInfoList\* load\_infos | 로드된 정보를 저장함 |
| Unsigned int total\_length | 총 할당된 길이를 저장 |
| Const int count | 로드된 정보의 개수를 저장함 |

## 모듈이름: execute\_operator(Debugger \*debugger, Memories \*memories, Instruction \*instruction)

### 기능

Instruction에 대응되는 Operator(LDA, LDB, 등등)를 실행함

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| Memories \*memories | 메모리 정보를 저장한다. |
| Instruction \*instruction | Instruction 정보를 저장함. |
| Registers\* registers | 레지스터 정보를 저장한다. |
| uint32\_t value,val1,val2 | 값 정보를 저장하기 위한 변수 |
| static size\_t device\_input\_idx | Device Read Write 구현을 위한 변수 |
| char inputDevice[] | 가상 device 변수 |

## 모듈이름: calculate\_TA(Instruction\* instruction, Registers\* registers)

### 기능

Target Address 를 계산하여 리턴함.

### 사용변수

|  |  |
| --- | --- |
| Instruction \*instruction | Instruction 정보를 저장함. |
| Registers\* registers | 레지스터 정보를 저장한다. |
| uint32\_t TA | Target address 정보를 저장한다. |
| uint32\_t b,p,x | Instruction의 B,P,X 값을 저장함 |
| uint32\_t address | 주소를 저장함. |

## 모듈이름: handling\_bp(Debugger \*debugger, int instruction\_size)

### 기능

브레이크 포인트를 핸들링함. .

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버거 정보를 저장함 |
| Int instruction\_size | Instruction 크기를 저장함 |
| Bool is\_break | 브레이크 포인트가 있는지 여부를 저장함 |
| Unsigned int bp | Bp의 주소를 저장함 |

## 모듈이름: calculate\_addressing\_mode(Instruction\* instruction, bool jump\_op)

### 기능

Addressing 모드를 계산하여 리턴함. (immediate, simple, indirect). .

### 사용변수

|  |  |
| --- | --- |
| Instruction\* instruction | Instruction 정보를 저장함 |
| Bool jump\_op | 해당 instruction 이 jump를 하는 명령인지 여부 저장 |
| uint32\_t n, i | Instruction의 N, I 값을 저장함 |

## 모듈이름: load\_from\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t \*value, size\_t bytes, bool jump\_op)

### 기능

메모리로부터 값을 가져옴

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| Memories \*memories | 메모리 정보를 저장한다. |
| Instruction \*instruction | Instruction 정보를 저장함. |
| uint32 \*value | 메모리로부터 가져온 값을 저장할 변수 |
| Size\_t bytes | 크기 |
| Bool jump\_op | 점프인지 여부 |
| Uint32\_t TA | Target Address 정보 저장 |
| ADDRESSING\_MODE addr\_mode | 어드레싱 모드 정보 저장 |

## 모듈이름: store\_to\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t value, size\_t bytes)

### 기능

메모리에 값을 저장함

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| Memories \*memories | 메모리 정보를 저장한다. |
| Instruction \*instruction | Instruction 정보를 저장함. |
| uint32 \*value | 저장할 메모리 주소 |
| Size\_t bytes | 크기 |
| Uint32\_t TA | Target Address 정보 저장 |
| ADDRESSING\_MODE addr\_mode | 어드레싱 모드 정보 저장 |

## 모듈이름: load\_from\_register(Debugger \*debugger, int reg\_id, uint32\_t \*val)

### 기능

레지스터 로부터 값을 가져옴

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| uint32 \*val | 레지스터로부터 가져온 값을 저장할 변수 |
| Int reg\_id | 레지스터 번호 |
| Uint32\_t TA | Target Address 정보 저장 |
| ADDRESSING\_MODE addr\_mode | 어드레싱 모드 정보 저장 |
| uint32\_t\* reg | 레지스터 번호에 대응되는 레지스터 포인터 |

## 모듈이름: store\_to\_register(Debugger \*debugger, int reg\_id, uint32\_t val)

### 기능

레지스터에 값을 저장함

### 사용변수

|  |  |
| --- | --- |
| Debugger \*debugger | 디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함.. |
| uint32 val | 레지스터에 저장할 값 |
| Int reg\_id | 레지스터 번호 |
| uint32\_t\* reg | 레지스터 번호에 대응되는 레지스터 포인터 |

## 모듈이름: get\_reg\_by\_id(Registers \*registers, int reg\_id)

### 기능

레지스터 번호로 레지스터를 찾고 대응되는 포인터를 리턴함

### 사용변수

|  |  |
| --- | --- |
| Registers\* registers | 레지스터들을 저장하는 변수 |
| Int reg\_id | 레지스터 번호 |
| uint32\_t\* regs[10] | 레지스터 번호에 대응되는 레지스터 포인터를 매핑한 배열 |

## 모듈이름: print\_registers(Registers\* registers)

### 기능

현재의 레지스터 상태를 출력함.

### 사용변수

|  |  |
| --- | --- |
| Registers\* registers | 레지스터들을 저장하는 변수 |

# 전역 변수 정의

전역변수 선언,정의 하지 않았음.

# 코드

## 20161631.c

#include "20161631.h"  
  
**int** main() {  
 /\*  
 \* state\_store 에서는 명령어 히스토리, 가상 메모리 영역, Opcode 정보를 저장한다.  
 \*/  
 State\* state\_store = construct\_state();  
  
 /\*  
 \* 사용자가 quit(q) 명령을 입력하기전까지  
 \* 쉘을 통해 명령어를 입력, 수행할수있도록 한다.  
 \*/  
 command\_main(state\_store);  
  
 /\*  
 \* 동적 할당 받은 메모리를 모두 해제한다.  
 \*/  
 destroy\_state(&state\_store);  
  
 **return** 0;  
}

## 20161631.h

#ifndef \_\_20161631\_H\_\_  
#define \_\_20161631\_H\_\_  
  
#include <stdio.h>  
#include <dirent.h>  
#include <sys/stat.h>  
#include "command.h"  
#include "state.h"  
#include "dir.h"  
  
#endif

## command.c

#include "command.h"  
  
/\*  
 \* 사용자가 quit(q)를 명령을 입력하기 이전까지 쉘을 계속 수행한다.  
 \*/  
bool command\_main(State\* state\_store){  
 shell\_status status;  
 Command user\_command;  
  
 **while** (1){  
  
 render\_shell();  
  
 // 사용자로 부터 입력을 받는다.  
 status = read\_input(&user\_command.raw\_command);  
  
 // 잘못된 입력이라면 continue 한다.  
 **if**(!exception\_check\_and\_handling(status)) **continue**;  
  
 // 입력을 지정된 명령어에 있는지 확인하고, 매핑한다.  
 status = command\_mapping(&user\_command);  
 **if**(!exception\_check\_and\_handling(status)) **continue**;  
  
 // 매핑된 명령어를 수행한다.  
 status = command\_execute(&user\_command, state\_store);  
 **if**(check\_quit\_condition(&user\_command)) **break**;  
 **if**(!exception\_check\_and\_handling(status)) **continue**;  
 **if**(status == *EXECUTE\_FAIL*) **continue**;  
  
 // 실행이 완료된 명령어를 입력 그대로 히스토리에 추가한다.  
 add\_history(state\_store, user\_command.raw\_command);  
 }  
 **return** true;  
}  
  
/\*  
 \* status 파라미터에 넘어온 내용에 따라서  
 \* 에러에 해당한다면 적절한 에러문을 출력해주고 false 를 리턴한다.  
 \* 에러에 해당하지않는다면 true 를 리턴한다.  
 \*  
 \* 참고: 사용자에게 입력을 받거나, 토크나이징 하는 등의 함수들은  
 \* 성공, 실패 여부등에 따라서 shell\_status ( enum )을 리턴한다.  
 \*/  
bool exception\_check\_and\_handling(shell\_status status){  
 **switch**(status){  
 **case** *INPUT\_READ\_SUCCESS*:  
 **break**;  
 **case** *TOKENIZING\_SUCCESS*:  
 **break**;  
 **case** *VALID\_COMMAND\_TYPE*:  
 **break**;  
 **case** *VALID\_PARAMETERS*:  
 **break**;  
 **case** *EXECUTE\_SUCCESS*:  
 **break**;  
 **case** *TOO\_LONG\_WRONG\_INPUT*:  
 fprintf(stderr, "[ERROR] Too Long Input\n");  
 **return** false;  
 **case** *TOO\_MANY\_TOKEN*:  
 fprintf (stderr, "[ERROR] Too Many Tokens\n");  
 **return** false;  
 **case** *INVALID\_COMMAND\_TYPE*:  
 fprintf(stderr, "[ERROR] Invalid Command Type\n");  
 **return** false;  
 **case** *INVALID\_INPUT*:  
 fprintf(stderr, "[ERROR] Invalid Input\n");  
 **return** false;  
 **case** *INVALID\_PARAMETERS*:  
 fprintf(stderr, "[ERROR] Invalid Parameters\n");  
 **return** false;  
 **case** *MISSING\_REQUIRE\_PARAMETER*:  
 fprintf(stderr, "[ERROR] Missing Required Parameter\n");  
 **return** false;  
 **case** *EXECUTE\_FAIL*:  
 fprintf(stderr, "[ERROR] Invalid Input\n");  
 **break**;  
 **default**:  
 **break**;  
 }  
 **return** true;  
}  
  
/\*  
 \* quit(q) 명령이 들어왔다면 true 를 리턴하고,  
 \* 아니라면 false 를 리턴한다.  
 \*/  
bool check\_quit\_condition(Command\* user\_command){  
 **if**(user\_command->type == *TYPE\_QUIT*){  
 assert(user\_command->token\_cnt == 1);  
 **return** true;  
 }  
 **else**{  
 **return** false;  
 }  
}

## command.h

#ifndef \_\_COMMAND\_H\_\_  
#define \_\_COMMAND\_H\_\_  
  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <assert.h>  
#include <stdbool.h>  
  
#include "command\_macro.h"  
#include "command\_objects.h"  
#include "command\_shell.h"  
#include "command\_mapping.h"  
#include "command\_execute.h"  
#include "state.h"  
  
/\*  
 \* 사용자가 quit(q)를 명령을 입력하기 이전까지 쉘을 계속 수행한다.  
 \*/  
bool command\_main(State\* state\_store);  
  
/\*  
 \* status 파라미터에 넘어온 내용에 따라서  
 \* 에러에 해당한다면 적절한 에러문을 출력해주고 false 를 리턴한다.  
 \* 에러에 해당하지않는다면 true 를 리턴한다.  
 \*  
 \* 참고: 사용자에게 입력을 받거나, 토크나이징 하는 등의 함수들은  
 \* 성공, 실패 여부등에 따라서 shell\_status ( enum )을 리턴한다.  
 \*/  
bool exception\_check\_and\_handling(shell\_status status);  
  
/\*  
 \* quit(q) 명령이 들어왔다면 true 를 리턴하고,  
 \* 아니라면 false 를 리턴한다.  
 \*/  
bool check\_quit\_condition(Command\* user\_command);  
  
#endif

## command\_execute.c

#include "command\_execute.h"

/\*

\* 사용자가 입력한 명령어에 따른 실행 함수(execute\_\*\*\*())를 실행한다.

\* 또한 실행된 결과를 리턴해준다.

\*/

shell\_status command\_execute(Command \*user\_command, State \*state\_store) {

assert(user\_command);

assert(state\_store);

switch (user\_command->type){

case TYPE\_HELP:

execute\_help();

return EXECUTE\_SUCCESS;

case TYPE\_HISTORY:

return execute\_history(state\_store, user\_command->raw\_command);

case TYPE\_QUIT:

return execute\_quit();

case TYPE\_DIR:

return execute\_dir();

case TYPE\_EDIT:

return execute\_edit(user\_command, state\_store->memories\_state);

case TYPE\_FILL:

return execute\_fill(user\_command, state\_store->memories\_state);

case TYPE\_RESET:

return execute\_reset(state\_store->memories\_state);

case TYPE\_OPCODE:

return execute\_opcode(user\_command, state\_store);

case TYPE\_OPCODELIST:

return execute\_opcodelist(state\_store);

case TYPE\_DUMP:

return execute\_dump(user\_command, state\_store->memories\_state);

case TYPE\_ASSEMBLE:

return execute\_assemble(user\_command, state\_store);

case TYPE\_TYPE:

return execute\_type(user\_command);

case TYPE\_SYMBOL:

return execute\_symbol(state\_store);

case TYPE\_PROGADDR:

return execute\_progaddr(user\_command, state\_store);

case TYPE\_RUN:

return execute\_run(state\_store);

case TYPE\_BP:

return execute\_bp(user\_command, state\_store);

case TYPE\_BP\_CLEAR:

return execute\_bp\_clear(state\_store);

case TYPE\_BP\_LIST:

return execute\_bp\_list(state\_store);

case TYPE\_LOADER:

return execute\_loader(user\_command, state\_store);

default:

break;

}

return EXECUTE\_SUCCESS;

}

/\*

\* history 명령어

\* 실행되었던 명령어 히스토리를 출력한다

\*/

shell\_status execute\_history(State\* state\_store, char \*last\_command) {

assert(state\_store);

assert(last\_command);

print\_histories\_state(state\_store, last\_command);

return EXECUTE\_SUCCESS;

}

/\*

\* help 명령어

\* 사용할수있는 명령어들을 화면에 출력해서 보여준다,

\*/

void execute\_help(){

fprintf (stdout,"h[elp]\n"

"d[ir]\n"

"q[uit]\n"

"hi[story]\n"

"du[mp] [start, end]\n"

"e[dit] address, value\n"

"f[ill] start, end, value\n"

"reset\n"

"opcode mnemonic\n"

"opcodelist\n"

"assemble filename\n"

"type filename\n"

"symbol\n"

"loader [filename1] [filename2] ...\n"

"run\n"

"bp\n"

"progaddr [address]\n"

);

}

/\*

\* quit 명령어

\* QUIT 이라는 shell\_status(enum)을 리턴한다.

\*

\* 참고: command\_main 함수의 무한 루프는 QUIT 이라는 status 가 들어오면

\* break 되도록 설계되었음.

\*/

shell\_status execute\_quit(){

fprintf(stdout, "Bye :)\n");

return QUIT;

}

/\*

\* dir 명령어

\* 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.

\*/

shell\_status execute\_dir(){

if(!print\_dir()) return EXECUTE\_FAIL;

return EXECUTE\_SUCCESS;

}

/\*

\* dump 명령어

\* dump start, end : start~end 까지의 가상 메모리영역을 출력한다.

\* dump start : start 메모리 부터 10라인의 영역을 출력한다.

\* dump : 가장 마지막으로 실행되었던 메모리부터 10 라인의 영역 출력한다.

\*/

shell\_status execute\_dump(Command \*user\_command, Memories \*memories\_state) {

assert(memories\_state);

assert(user\_command);

assert(user\_command->token\_cnt < 4);

size\_t token\_cnt = user\_command->token\_cnt;

int start=0, end=0;

// 출력할 메모리 영역의 범위를 초기화 한다.

if(token\_cnt == 1){

// case1. dump

start = memories\_state->last\_idx + 1;

end = start + 159;

} else if(token\_cnt == 2){

// case2. dump start

start = (int)strtol(user\_command->tokens[1], NULL, 16);

if(start + 159 >= MEMORIES\_SIZE) end = MEMORIES\_SIZE - 1;

else end = start + 159;

} else if(token\_cnt == 3){

// case3. dump start, end

start = (int)strtol(user\_command->tokens[1], NULL, 16);

end = (int)strtol(user\_command->tokens[2], NULL, 16);

if(end >= MEMORIES\_SIZE) end = MEMORIES\_SIZE - 1;

}

// start ~ end 범위의 메모리 영역을 출력한다.

print\_memories(memories\_state, start, end);

// 출력된 마지막 메모리 주소를 저장한다.

// 만일 마지막 메모리 주소에 1을 더한 주소가 범위를 벗어났다면 -1을 저장한다.

if(end + 1 >= MEMORIES\_SIZE)

memories\_state->last\_idx = -1;

else

memories\_state->last\_idx = end;

return EXECUTE\_SUCCESS;

}

/\*

\* edit 명령어

\* edit addr, value: addr 주소의 값을 value 로 수정한다.

\*/

shell\_status execute\_edit(Command \*user\_command, Memories \*memories\_state) {

assert(user\_command);

assert(memories\_state);

assert(user\_command->token\_cnt == 3);

int addr = (int)strtol(user\_command->tokens[1], NULL, 16);

short value = (short)strtol(user\_command->tokens[2], NULL, 16);

edit\_memory(memories\_state, addr, value);

return EXECUTE\_SUCCESS;

}

/\*

\* fill 명령어

\* fill start, end, value: start~end 의 메모리 영역의 값들을 value 로 수정한다.

\*/

shell\_status execute\_fill(Command \*user\_command, Memories \*memories\_state) {

assert(user\_command);

assert(memories\_state);

assert(user\_command->token\_cnt == 4);

int start = (int)strtol(user\_command->tokens[1], NULL, 16);

int end = (int)strtol(user\_command->tokens[2], NULL, 16);

short value = (short)strtol(user\_command->tokens[3], NULL, 16);

int addr = 0;

assert(start <= end || start >= 0 || end >= 0 || value >= 0);

for(addr = start; addr <= end; addr++)

edit\_memory(memories\_state, addr, value);

return EXECUTE\_SUCCESS;

}

/\*

\* reset 명령어

\* 가상 메모리 영역의 모든 값들을 0 으로 바꾼다.

\*/

shell\_status execute\_reset(Memories \*memories\_state) {

assert(memories\_state);

int addr = 0, end = MEMORIES\_SIZE - 1;

for(addr=0;addr<=end;addr++)

edit\_memory(memories\_state, addr, 0);

return EXECUTE\_SUCCESS;

}

/\*

\* opcode 명령어

\* opcode mnemonic : mnemonic 의 value 를 출력

\* ex) opcode LDF => opcode is 70

\*/

shell\_status execute\_opcode(Command\* user\_command, State\* state\_store){

assert(user\_command->token\_cnt == 2);

Opcode\* opc = find\_opcode\_by\_name(state\_store->opcode\_table\_state, user\_command->tokens[1]);

if(!opc) return EXECUTE\_FAIL;

fprintf(stdout, "opcode is %X\n", opc->value);

return EXECUTE\_SUCCESS;

}

/\*

\* opcodelist 명령어

\* 해시테이블 형태로 저장된 opcode 목록을 출력해준다.

\*/

shell\_status execute\_opcodelist(State\* state\_store){

print\_opcodes(state\_store->opcode\_table\_state);

return EXECUTE\_SUCCESS;

}

/\*

\* assemble 명령어

\* ex.

\* assemble filename

\*/

shell\_status execute\_assemble(Command \*user\_command, State\* state\_store){

assert(user\_command->token\_cnt == 2);

if(assemble\_file(state\_store, user\_command->tokens[1]))

return EXECUTE\_SUCCESS;

else

return EXECUTE\_FAIL;

}

/\*

\* type 명령어

\*/

shell\_status execute\_type(Command\* user\_command){

assert(user\_command->token\_cnt == 2);

if(!user\_command->tokens[1])

return EXECUTE\_FAIL;

FILE\* fp = fopen(user\_command->tokens[1], "rt");

char buf[10000];

if(!fp){

fprintf(stderr, "[ERROR] Can't Open File\n");

return EXECUTE\_FAIL;

}

while (fgets (buf, sizeof(buf), fp))

fputs (buf, stdout);

fputs("\n", stdout);

fclose(fp);

return EXECUTE\_SUCCESS;

}

/\*

\* symbol 명령어

\*/

shell\_status execute\_symbol(State \*state\_store) {

if(!state\_store->is\_symbol\_table) return EXECUTE\_FAIL;

print\_symbols(state\_store->symbol\_table\_state);

return EXECUTE\_SUCCESS;

}

/\*

\* progaddr 명령어

\*/

shell\_status execute\_progaddr(Command \*user\_command, State \*state\_store) {

int addr = (int)strtol(user\_command->tokens[1], NULL, 16);

state\_store->debugger\_state->start\_address = (uint32\_t)addr;

fprintf(stdout, "Program starting address set to 0x%04X.\n", addr);

state\_store->debugger\_state->is\_loaded = false;

return EXECUTE\_SUCCESS;

}

/\*

\* run 명령어

\*/

shell\_status execute\_run(State \*state\_store){

bool status;

if(!state\_store->debugger\_state->is\_loaded){

fprintf(stderr, "[ERROR] Not Loaded\n");

return EXECUTE\_FAIL;

}

if(state\_store->debugger\_state->file\_count != 1){

fprintf(stderr, "[ERROR] Only Support One Object Run\n");

fprintf(stderr, "[ERROR] Must Re-Load\n");

return EXECUTE\_FAIL;

}

if(!state\_store->debugger\_state->is\_running){

state\_store->debugger\_state->registers->PC = state\_store->debugger\_state->start\_address;

}

status = run(state\_store->debugger\_state, state\_store->memories\_state);

if(!status){

state\_store->debugger\_state->is\_running = false;

state\_store->debugger\_state->is\_loaded = false;

return EXECUTE\_FAIL;

}

return EXECUTE\_SUCCESS;

}

/\*

\* bp 명령어

\*/

shell\_status execute\_bp(Command \*user\_command, State \*state\_store) {

int addr = (int)strtol(user\_command->tokens[1], NULL, 16);

if(state\_store->debugger\_state->break\_points[addr] == true){

fprintf(stdout, "[warning] breakpoint already at %04X\n", addr);

return EXECUTE\_SUCCESS;

}

state\_store->debugger\_state->break\_points[addr] = true;

state\_store->debugger\_state->bp\_count += 1;

fprintf(stdout, "[ok] create breakpoint %04X\n", addr);

return EXECUTE\_SUCCESS;

}

/\*

\* bp clear 명령어

\*/

shell\_status execute\_bp\_clear(State \*state\_store){

int cnt = 0;

for(int i = 0; i < MAX\_BP\_NUM; i++){

if(state\_store->debugger\_state->bp\_count == cnt)

break;

if(state\_store->debugger\_state->break\_points[i] == true){

state\_store->debugger\_state->break\_points[i] = false;

cnt++;

}

}

state\_store->debugger\_state->bp\_count = 0;

fprintf(stdout, "[ok] clear all breakpoints\n");

return EXECUTE\_SUCCESS;

}

/\*

\* loader 명령어

\*/

shell\_status execute\_loader(Command \*user\_command, State \*state\_store) {

// printf("loader\_linker execute\n");

bool status;

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

state\_store->debugger\_state->filenames[i] = NULL;

for(size\_t i= 1; i < user\_command->token\_cnt; i++)

state\_store->debugger\_state->filenames[i - 1] = user\_command->tokens[i];

state\_store->debugger\_state->file\_count = user\_command->token\_cnt - 1;

status = loader\_linker(state\_store->debugger\_state, state\_store->memories\_state);

if(!status){

for(int i = 0; i < 3; i++) state\_store->debugger\_state->filenames[i] = NULL;

state\_store->debugger\_state->file\_count = 0;

return EXECUTE\_FAIL;

}

state\_store->debugger\_state->run\_count = 0;

state\_store->debugger\_state->previous\_bp = -1;

// state\_store->debugger\_state->end\_address 설정 해주기.

state\_store->debugger\_state->end\_address = MAX\_BP\_NUM - 1;

state\_store->debugger\_state->is\_loaded = true;

return EXECUTE\_SUCCESS;

}

shell\_status execute\_bp\_list(State\* state\_store){

int cnt = 0;

if(state\_store->debugger\_state->bp\_count == 0){

fprintf(stdout, "no breakpoints set.\n");

return EXECUTE\_SUCCESS;

}

fprintf(stdout, "breakpoint\n");

fprintf(stdout, "----------\n");

for(int i = 0; i < MAX\_BP\_NUM; i++){

if(state\_store->debugger\_state->bp\_count == cnt)

break;

if(state\_store->debugger\_state->break\_points[i] == true) {

fprintf(stdout, "%04X\n", i);

cnt++;

}

}

return EXECUTE\_SUCCESS;

}

## command\_execute.h

#ifndef \_\_COMMAND\_EXECUTE\_H\_\_

#define \_\_COMMAND\_EXECUTE\_H\_\_

#include "command\_macro.h"

#include "command\_objects.h"

#include "state.h"

#include "dir.h"

/\*

\* 사용자가 입력한 명령어에 따른 실행 함수(execute\_\*\*\*())를 실행한다.

\* 또한 실행된 결과를 리턴해준다.

\*/

shell\_status command\_execute(Command \*user\_command, State \*state\_store);

/\*

\* help 명령어

\* 사용할수있는 명령어들을 화면에 출력해서 보여준다,

\*/

void execute\_help();

/\*

\* history 명령어

\* 실행되었던 명령어 히스토리를 출력한다

\*/

shell\_status execute\_history(State\* state\_store, char \*last\_command);

/\*

\* quit 명령어

\* QUIT 이라는 shell\_status(enum)을 리턴한다.

\*

\* 참고: command\_main 함수의 무한 루프는 QUIT 이라는 status 가 들어오면

\* break 되도록 설계되었음.

\*/

shell\_status execute\_quit();

/\*

\* dir 명령어

\* 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.

\*/

shell\_status execute\_dir();

/\*

\* dump 명령어

\* dump start, end : start~end 까지의 가상 메모리영역을 출력한다.

\* dump start : start 메모리 부터 10라인의 영역을 출력한다.

\* dump : 가장 마지막으로 실행되었던 메모리부터 10 라인의 영역 출력한다.

\*/

shell\_status execute\_dump(Command \*user\_command, Memories \*memories\_state);

/\*

\* edit 명령어

\* edit addr, value: addr 주소의 값을 value 로 수정한다.

\*/

shell\_status execute\_edit(Command \*user\_command, Memories \*memories\_state);

/\*

\* fill 명령어

\* fill start, end, value: start~end 의 메모리 영역의 값들을 value 로 수정한다.

\*/

shell\_status execute\_fill(Command \*user\_command, Memories \*memories\_state);

/\*

\* reset 명령어

\* 가상 메모리 영역의 모든 값들을 0 으로 바꾼다.

\*/

shell\_status execute\_reset(Memories \*memories\_state);

/\*

\* opcode 명령어

\* opcode mnemonic : mnemonic 의 value 를 출력

\* ex) opcode LDF => opcode is 70

\*/

shell\_status execute\_opcode(Command \*user\_command, State\* state\_store);

/\*

\* opcodelist 명령어

\* 해시테이블 형태로 저장된 opcode 목록을 출력해준다.

\*/

shell\_status execute\_opcodelist(State\* state\_store);

/\*

\* assemble 명령어

\*/

shell\_status execute\_assemble(Command \*user\_command, State\* state\_store);

/\*

\* type 명령어

\*/

shell\_status execute\_type(Command\* user\_command);

/\*

\* symbol 명령어

\*/

shell\_status execute\_symbol(State \*state\_store);

/\*

\* progaddr 명령어

\*/

shell\_status execute\_progaddr(Command \*user\_command, State \*state\_store);

/\*

\* run 명령어

\*/

shell\_status execute\_run(State \*state\_store);

/\*

\* bp <주소> 명령어

\*/

shell\_status execute\_bp(Command \*user\_command, State \*state\_store);

/\*

\* bp clear 명령어

\*/

shell\_status execute\_bp\_clear(State \*state\_store);

/\*

\* bp list 출력해주는 명령어

\*/

shell\_status execute\_bp\_list(State\* state\_store);

/\*

\* loader\_linker 명령어

\*/

shell\_status execute\_loader(Command \*user\_command, State \*state\_store);

#endif

## command\_macro.h

#ifndef \_\_COMMAND\_MACRO\_H\_\_  
#define \_\_COMMAND\_MACRO\_H\_\_  
  
#define COMPARE\_STRING(T, S) (strcmp ((T), (S)) == 0)  
#define COMMAND\_MAX\_LEN 510  
#define TOKEN\_MAX\_NUM 30  
  
#endif

## command\_mapping.c

#include "command\_mapping.h"

/\*

\* 사용자의 raw\_input 이 적절한 명령어인지 확인하고,

\* 적절하다면 지정된 명령어로 매핑하고, 성공했다는 shell\_status 를 리턴한다.

\* 적절하지않다면, 실패했다는 shell\_status 를 리턴한다.

\*/

shell\_status command\_mapping(Command\* user\_command){

assert(user\_command);

shell\_status status;

status = tokenizing(user\_command);

if(status != TOKENIZING\_SUCCESS) return status;

status = command\_mapping\_type(user\_command);

if(user\_command->type != TYPE\_LOADER

&& !validate\_tokenizing(user\_command->raw\_command,

(int) user\_command->token\_cnt,

TOKEN\_MAX\_NUM))

return INVALID\_INPUT;

else if(user\_command->type == TYPE\_LOADER

&& !validate\_tokenizing\_for\_not\_comma(user\_command->raw\_command,

(int) user\_command->token\_cnt,

TOKEN\_MAX\_NUM))

return INVALID\_INPUT;

if(status != VALID\_COMMAND\_TYPE) return status;

// 파라미터가 해당 명령어에 적절한지 확인한다.

// 예를들어 dump -1 과 같은 경우를 잡아낸다.

status = validate\_parameters(user\_command);

if(status != VALID\_PARAMETERS) return status;

return status;

}

/\*

\* 사용자의 raw\_input 을 토크나이징하여 user\_command->tokens 에 저장한다.

\* ex) 입력이 dump 1, 2 가 들어왔다면

\* tokens[0] = "dump"

\* tokens[1] = "1"

\* tokens[2] = "2"

\* 형태로 저장된다.

\*

\* @return TOKENIZING\_SUCCESS or INVALID\_INPUT

\*/

shell\_status tokenizing(Command\* user\_command){

assert(user\_command);

static char raw\_command[COMMAND\_MAX\_LEN];

strncpy (raw\_command, user\_command->raw\_command, COMMAND\_MAX\_LEN);

user\_command->token\_cnt = 0;

user\_command->tokens[user\_command->token\_cnt] = strtok(raw\_command, " ,\t\n");

while (user\_command->token\_cnt <= TOKEN\_MAX\_NUM && user\_command->tokens[user\_command->token\_cnt])

user\_command->tokens[++user\_command->token\_cnt] = strtok (NULL, " ,\t\n");

if(!user\_command->tokens[0]) return INVALID\_INPUT;

return TOKENIZING\_SUCCESS;

}

/\*

\* user\_command->tokens[0]에 저장된 문자열이

\* 적절한 명령어 타입인지 확인하고, user\_command->type 에 명령어 타입을 설정해준다.

\*

\* @return VALID\_COMMAND\_TYPE or INVALID\_COMMAND\_TYPE

\*/

shell\_status command\_mapping\_type(Command \*user\_command){

assert(user\_command);

char\* first\_token = user\_command->tokens[0];

if(COMPARE\_STRING(first\_token, "h") ||

COMPARE\_STRING(first\_token, "help")) {

user\_command->type = TYPE\_HELP;

}else if(COMPARE\_STRING(first\_token, "d") ||

COMPARE\_STRING(first\_token, "dir")) {

user\_command->type = TYPE\_DIR;

}else if(COMPARE\_STRING(first\_token, "q") ||

COMPARE\_STRING(first\_token, "quit")) {

user\_command->type = TYPE\_QUIT;

} else if(COMPARE\_STRING(first\_token, "hi") ||

COMPARE\_STRING(first\_token, "history")){

user\_command->type = TYPE\_HISTORY;

} else if(COMPARE\_STRING(first\_token, "du") ||

COMPARE\_STRING(first\_token, "dump")){

user\_command->type = TYPE\_DUMP;

} else if(COMPARE\_STRING(first\_token, "e") ||

COMPARE\_STRING(first\_token, "edit")){

user\_command->type = TYPE\_EDIT;

} else if(COMPARE\_STRING(first\_token, "f") ||

COMPARE\_STRING(first\_token, "fill")){

user\_command->type = TYPE\_FILL;

} else if(COMPARE\_STRING(first\_token, "reset")){

user\_command->type = TYPE\_RESET;

} else if(COMPARE\_STRING(first\_token, "opcode")){

user\_command->type = TYPE\_OPCODE;

} else if(COMPARE\_STRING(first\_token, "opcodelist")){

user\_command->type = TYPE\_OPCODELIST;

} else if(COMPARE\_STRING(first\_token, "assemble")){

user\_command->type = TYPE\_ASSEMBLE;

} else if(COMPARE\_STRING(first\_token, "type")){

user\_command->type = TYPE\_TYPE;

} else if(COMPARE\_STRING(first\_token, "symbol")){

user\_command->type = TYPE\_SYMBOL;

} else if(COMPARE\_STRING(first\_token, "progaddr")){

user\_command->type = TYPE\_PROGADDR;

} else if(COMPARE\_STRING(first\_token, "bp")){

if(user\_command->token\_cnt >= 2){

char\* second\_token = user\_command->tokens[1];

if(COMPARE\_STRING(second\_token, "clear"))

user\_command->type = TYPE\_BP\_CLEAR;

else

user\_command->type = TYPE\_BP;

} else {

user\_command->type = TYPE\_BP\_LIST;

}

} else if(COMPARE\_STRING(first\_token, "run")){

user\_command->type = TYPE\_RUN;

} else if(COMPARE\_STRING(first\_token, "loader")){

user\_command->type = TYPE\_LOADER;

} else {

return INVALID\_COMMAND\_TYPE;

}

return VALID\_COMMAND\_TYPE;

}

## command\_mapping.h

#ifndef \_\_COMMAND\_MAPPING\_H\_\_  
#define \_\_COMMAND\_MAPPING\_H\_\_  
  
#include "command\_macro.h"  
#include "command\_objects.h"  
#include "command\_validate\_util.h"  
#include "util.h"  
#include <string.h>  
#include <stdbool.h>  
#include <assert.h>  
#include <sys/types.h>  
#include <stdlib.h>  
  
/\*  
 \* 사용자의 raw\_input 이 적절한 명령어인지 확인하고,  
 \* 적절하다면 지정된 명령어로 매핑하고, 성공했다는 shell\_status 를 리턴한다.  
 \* 적절하지않다면, 실패했다는 shell\_status 를 리턴한다.  
 \*/  
shell\_status command\_mapping(Command\* user\_command);  
  
/\*  
 \* 사용자의 raw\_input 을 토크나이징하여 user\_command->tokens 에 저장한다.  
 \* ex) 입력이 dump 1, 2 가 들어왔다면  
 \* tokens[0] = "dump"  
 \* tokens[1] = "1"  
 \* tokens[2] = "2"  
 \* 형태로 저장된다.  
 \*  
 \* @return TOKENIZING\_SUCCESS or INVALID\_INPUT  
 \*/  
shell\_status tokenizing(Command\* user\_command);  
  
/\*  
 \* user\_command->tokens[0]에 저장된 문자열이  
 \* 적절한 명령어 타입인지 확인하고, user\_command->type 에 명령어 타입을 설정해준다.  
 \*  
 \* @return VALID\_COMMAND\_TYPE or INVALID\_COMMAND\_TYPE  
 \*/  
shell\_status command\_mapping\_type(Command \*user\_command);  
  
#endif

## command\_objects.h

#ifndef \_\_COMMAND\_OBJECTS\_H\_\_

#define \_\_COMMAND\_OBJECTS\_H\_\_

#include <sys/types.h>

#define TOKEN\_MAX\_NUM 30

// 명령어 타입을 enum 형태로 표현한다.

enum shell\_command\_type{

TYPE\_HELP, TYPE\_DIR, TYPE\_QUIT, TYPE\_HISTORY,

TYPE\_DUMP, TYPE\_EDIT, TYPE\_FILL, TYPE\_RESET,

TYPE\_OPCODE, TYPE\_OPCODELIST, TYPE\_ASSEMBLE,

TYPE\_TYPE, TYPE\_SYMBOL, TYPE\_PROGADDR, TYPE\_RUN, TYPE\_BP,

TYPE\_BP\_CLEAR, TYPE\_LOADER, TYPE\_BP\_LIST

};

// command 를 구조체로 구조화 하여 표현.

typedef struct command {

char\* raw\_command;

char\* tokens[TOKEN\_MAX\_NUM + 5];

size\_t token\_cnt;

enum shell\_command\_type type;

} Command;

// shell 의 상태를 표현함.

typedef enum SHELL\_STATUS {

INPUT\_READ\_SUCCESS, TOO\_LONG\_WRONG\_INPUT,

TOKENIZING\_SUCCESS, TOO\_MANY\_TOKEN,

INVALID\_INPUT, VALID\_COMMAND, INVALID\_COMMAND,

INVALID\_COMMAND\_TYPE, VALID\_COMMAND\_TYPE,

INVALID\_PARAMETERS, VALID\_PARAMETERS,

MISSING\_REQUIRE\_PARAMETER, EXECUTE\_SUCCESS, QUIT, EXECUTE\_FAIL

} shell\_status;

#endif

## command\_shell.c

#include "command\_shell.h"  
  
/\*  
 \* 사용자로 부터 입력을 받고 \*target 에 저장한다.  
 \* 입력이 너무 길 경우 에러를 리턴해준다.  
 \*  
 \* @return INPUT\_READ\_SUCCESS or TOO\_LONG\_WRONG\_INPUT  
 \*/  
shell\_status read\_input(**char**\*\* target){  
 **static char** input[COMMAND\_MAX\_LEN + 10];  
 fgets(input, COMMAND\_MAX\_LEN + 10, stdin);  
 **if**(strlen(input) >= COMMAND\_MAX\_LEN) **return** *TOO\_LONG\_WRONG\_INPUT*;  
 \*target = input;  
 **return** *INPUT\_READ\_SUCCESS*;  
}  
  
/\*  
 \* shell 을 출력함.  
 \*/  
**void** render\_shell(){  
 printf("sicsim > ");  
}

## command\_shell.h

#ifndef \_\_COMMAND\_SHELL\_H\_\_  
#define \_\_COMMAND\_SHELL\_H\_\_  
  
#include "command\_macro.h"  
#include "command\_objects.h"  
#include <string.h>  
#include <stdio.h>  
  
/\*  
 \* 사용자로 부터 입력을 받고 \*target 에 저장한다.  
 \* 입력이 너무 길 경우 에러를 리턴해준다.  
 \*  
 \* @return INPUT\_READ\_SUCCESS or TOO\_LONG\_WRONG\_INPUT  
 \*/  
shell\_status read\_input(**char**\*\* target);  
  
/\*  
 \* shell 을 출력함.  
 \*/  
**void** render\_shell();  
  
#endif

## command\_validate\_util.c

#include "command\_validate\_util.h"

/\*

\* 토크나이징이 적절하게 되었는지 검증한다.

\* 예를들어 du , 1 1 과 같이

\* 파라미터 사이에 콤마가 없거나 이상한 위치에 콤마가 있는 등의 문제를 잡아낸다.

\*

\* @return true or false

\*/

bool validate\_tokenizing(char \*str, int token\_cnt, int max\_token\_num) {

assert(str);

int length = (int)strlen(str);

int i=0, comma\_cnt = 0, flag = 0;

char cm;

// 토큰의 개수를 검증한다.

if(token\_cnt > max\_token\_num)

return false;

if(token\_cnt <= 0)

return false;

// 콤마의 개수를 계산한다.

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

if(str[i] == ',')

comma\_cnt++;

}

// 토큰의 개수가 두개이면서 콤마가 없는 경우는 적절한 경우다.

if(token\_cnt <= 2 && comma\_cnt == 0)

return true;

// 예시와 같은 경우의 에러를 잡아낸다.

// ex) du , 1 1 [x]

// ex) , du

flag = 0;

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

cm = str[i];

if(flag == 0){

// 첫번째 토큰 이전 문자열.

if(cm == ',') return false;

if(cm != ' ' && cm != '\t') flag = 1;

continue;

} else if(flag == 1){

// 첫번째 토큰을 지나가는 중

if(cm == ',') return false;

if(cm == ' ' || cm == '\t') flag = 2;

continue;

}

// 첫번째 토큰과 두번째 토큰 사이

if(str[i] == ' ') continue;

if(str[i] == '\t') continue;

if(str[i] != ',') break;

else return false;

}

// 예시와 같은 경우의 에러를 잡아낸다.

// ex) du 1 1 , [x]

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

if(str[i] == ' ') continue;

if(str[i] == '\t') continue;

if(str[i] != ',') break;

else return false;

}

// 토큰의 개수와 콤마의 개수를 비교한다.

if(token\_cnt != comma\_cnt + 2)

return false;

// ex) f 1 ,, 2 3 [x]

// , 다음에는 [ ] [\t]이 나오다가 [ ] [\t] [,]이 아닌 값이 나와야한다.

flag = 0; // 콤마가 나오면 flag 는 1 로 놓자.

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

cm = str[i];

if (flag == 1) {

if (cm == ',') return false;

if (cm == ' ' || cm == '\t') continue;

flag = 0;

continue;

}

if (cm == ',') {

flag = 1;

continue;

}

}

return true;

}

bool validate\_tokenizing\_for\_not\_comma(char \*str, int token\_cnt, int max\_token\_num){

assert(str);

int length = (int)strlen(str);

int comma\_cnt = 0;

// 토큰의 개수를 검증한다.

if(token\_cnt > max\_token\_num)

return false;

if(token\_cnt <= 0)

return false;

// 콤마의 개수를 계산한다.

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

if(str[i] == ',')

comma\_cnt++;

}

if(comma\_cnt != 0)

return false;

return true;

}

/\*

\* 사용자가 입력한 파라미터가 적절한 파라미터 값인지 검증한다.

\* (명령어에 따른 파라미터 개수, 크기, 범위 등)

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_parameters(Command \*user\_command){

assert(user\_command);

if((user\_command->type == TYPE\_QUIT ||

user\_command->type == TYPE\_HELP ||

user\_command->type == TYPE\_HISTORY ||

user\_command->type == TYPE\_DIR ||

user\_command->type == TYPE\_RESET ||

user\_command->type == TYPE\_OPCODELIST ||

user\_command->type == TYPE\_SYMBOL ||

user\_command->type == TYPE\_RUN ||

user\_command->type == TYPE\_BP\_LIST

) &&

user\_command->token\_cnt > 1)

return INVALID\_PARAMETERS;

if(user\_command->type == TYPE\_TYPE && user\_command->token\_cnt != 2)

return INVALID\_PARAMETERS;

if(user\_command->type == TYPE\_OPCODE)

return validate\_opcode\_parameters(user\_command);

if(user\_command->type == TYPE\_EDIT)

return validate\_edit\_parameters(user\_command);

if(user\_command->type == TYPE\_FILL)

return validate\_fill\_parameters(user\_command);

if(user\_command->type == TYPE\_DUMP)

return validate\_dump\_parameters(user\_command);

if(user\_command->type == TYPE\_ASSEMBLE)

return validate\_assemble\_parameters(user\_command);

if(user\_command->type == TYPE\_PROGADDR)

return validate\_progaddr\_parameters(user\_command);

if(user\_command->type == TYPE\_BP)

return validate\_bp\_parameters(user\_command);

if(user\_command->type == TYPE\_BP\_CLEAR)

return validate\_bp\_clear\_parameters(user\_command);

if(user\_command->type == TYPE\_LOADER)

return validate\_loader\_parameters(user\_command);

return VALID\_PARAMETERS;

}

/\*

\* dump 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_dump\_parameters(Command \*user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_DUMP);

int tok1, tok2;

if(user\_command->token\_cnt > 1) {

// 적절한 주소값인지 확인한다.

if (!is\_valid\_address(user\_command->tokens[1], MB))

return INVALID\_PARAMETERS;

else if (user\_command->token\_cnt == 2)

return VALID\_PARAMETERS;

// 적절한 주소값인지 확인한다.

if (!is\_valid\_address(user\_command->tokens[2], MB))

return INVALID\_PARAMETERS;

tok1 = (int) strtol(user\_command->tokens[1], NULL, 16);

tok2 = (int) strtol(user\_command->tokens[2], NULL, 16);

// invalid area

if (tok1 > tok2) return INVALID\_PARAMETERS;

}

// dump 1, 2, 3과 같이 파라미터가 세개 이상이 되는 경우는 에러이다.

if(user\_command->token\_cnt > 3)

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

/\*

\* opcode 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_opcode\_parameters(Command \*user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_OPCODE);

if(user\_command->token\_cnt != 2) return INVALID\_PARAMETERS;

if(strlen(user\_command->tokens[1]) > 14) return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

/\*

\* edit 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_edit\_parameters(Command \*user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_EDIT);

int value;

if(user\_command->token\_cnt != 3)

return INVALID\_PARAMETERS;

// 적절한 주소값인지 검증

if(!is\_valid\_address(user\_command->tokens[1], MB))

return INVALID\_PARAMETERS;

// 적절한 16진수 값인지 검증한다.

if(!is\_valid\_hex(user\_command->tokens[2]))

return INVALID\_PARAMETERS;

value = (int) strtol(user\_command->tokens[2], NULL, 16);

// 범위 확인

if(!(0 <= value && value <= 0xFF))

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

/\*

\* fill 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_fill\_parameters(Command \*user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_FILL);

int value, start, end;

if(user\_command->token\_cnt != 4)

return INVALID\_PARAMETERS;

if(!is\_valid\_address(user\_command->tokens[1], MB))

return INVALID\_PARAMETERS;

if(!is\_valid\_address(user\_command->tokens[2], MB))

return INVALID\_PARAMETERS;

start = (int) strtol(user\_command->tokens[1], NULL, 16);

end = (int) strtol(user\_command->tokens[2], NULL, 16);

if (start > end) return INVALID\_PARAMETERS;

value = (int) strtol(user\_command->tokens[3], NULL, 16);

if(!is\_valid\_hex(user\_command->tokens[3]))

return INVALID\_PARAMETERS;

if(!(0 <= value && value <= 0xFF)) return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

/\*

\* assemble 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_assemble\_parameters(Command \*user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_ASSEMBLE);

if(user\_command->token\_cnt != 2)

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

/\*

\* progaddr 명령어의 파라미터를 검증한다

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_progaddr\_parameters(Command \*user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_PROGADDR);

if(user\_command->token\_cnt != 2)

return INVALID\_PARAMETERS;

// 적절한 주소 값인지 확인한다.

if(!is\_valid\_address(user\_command->tokens[1], MB))

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

/\*

\* bp 명령어의 파라미터를 검증한다

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_bp\_parameters(Command\* user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_BP);

if(user\_command->token\_cnt != 2) {

return INVALID\_PARAMETERS;

}

// 적절한 주소 값인지 확인한다.

if(!is\_valid\_address(user\_command->tokens[1], MB))

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

shell\_status validate\_bp\_clear\_parameters(Command\* user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_BP\_CLEAR);

if(user\_command->token\_cnt != 2)

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

shell\_status validate\_loader\_parameters(Command\* user\_command){

assert(user\_command);

assert(user\_command->type == TYPE\_LOADER);

if(user\_command->token\_cnt > 4 || user\_command->token\_cnt == 1)

return INVALID\_PARAMETERS;

return VALID\_PARAMETERS;

}

## command\_validate\_util.h

#ifndef \_\_COMMAND\_VALIDATE\_UTIL\_H\_\_

#define \_\_COMMAND\_VALIDATE\_UTIL\_H\_\_

#include <stdbool.h>

#include <string.h>

#include <assert.h>

#include "command\_objects.h"

#include "util.h"

#define MB (1024\*1024)

/\*

\* 토크나이징이 적절하게 되었는지 검증한다.

\* 예를들어 du , 1 1 과 같이

\* 파라미터 사이에 콤마가 없거나 이상한 위치에 콤마가 있는 등의 문제를 잡아낸다.

\*

\* @return true or false

\*/

bool validate\_tokenizing(char \*str, int token\_cnt, int max\_token\_num);

bool validate\_tokenizing\_for\_not\_comma(char \*str, int token\_cnt, int max\_token\_num);

/\*

\* 사용자가 입력한 파라미터가 적절한 파라미터 값인지 검증한다.

\* (명령어에 따른 파라미터 개수, 크기, 범위 등)

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_parameters(Command \*user\_command);

/\*

\* dump 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_dump\_parameters(Command \*user\_command);

/\*

\* opcode 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_opcode\_parameters(Command \*user\_command);

/\*

\* edit 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_edit\_parameters(Command \*user\_command);

/\*

\* fill 명령어의 파라미터를 검증한다.

\*

\* @return VALID\_PARAMETERS or INVALID\_PARAMETERS

\*/

shell\_status validate\_fill\_parameters(Command \*user\_command);

shell\_status validate\_assemble\_parameters(Command \*user\_command);

shell\_status validate\_progaddr\_parameters(Command \*user\_command);

shell\_status validate\_bp\_parameters(Command\* user\_command);

shell\_status validate\_bp\_clear\_parameters(Command\* user\_command);

shell\_status validate\_loader\_parameters(Command\* user\_command);

#endif

## dir.c

#include "dir.h"  
  
/\*  
 \* 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.  
 \*/  
bool print\_dir(){  
 DIR\* dir = opendir(".");  
 **struct** dirent \*ent;  
 **struct** stat stat;  
 **char**\* ent\_dname;  
 **char**\* format;  
 **char** path[1025];  
 **int** i = 0;  
  
 **if**(!dir){  
 fprintf(stderr, "[ERROR] Can't open directory");  
 **return** false;  
 }  
 ent = readdir(dir);  
 **while** (ent){  
 ent\_dname = ent->d\_name;  
 lstat(ent\_dname, &stat);  
  
 **if**(S\_ISDIR(stat.st\_mode)) format = "%s/";  
 **else if**(S\_IXUSR & stat.st\_mode) format = "%s\*";  
 **else** format = "%s ";  
  
 sprintf(path, format, ent->d\_name);  
 printf("%-20s", path);  
  
 **if**(++i % 5 == 0) printf("\n");  
 ent = readdir(dir);  
 }  
 **if** (i % 5 != 0) printf ("\n");  
 closedir(dir);  
 **return** true;  
}

## dir.h

#ifndef \_\_DIR\_H\_\_  
#define \_\_DIR\_H\_\_  
  
#include <stdio.h>  
#include <dirent.h>  
#include <sys/stat.h>  
#include <stdbool.h>  
#include <unistd.h>  
#include <string.h>  
  
/\*  
 \* 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.  
 \*/  
bool print\_dir();  
  
#endif

## history.c

#include "history.h"  
  
/\*  
 \* Histories 구조체를 생성(할당)하고 HistoryList 도 생성(할당)한다.  
 \*/  
Histories\* construct\_histories(){  
 Histories\* hists = (Histories\*)malloc(**sizeof**(\*hists));  
  
 hists->list = (HistoryList\*)malloc(**sizeof**(HistoryList));  
 hists->list->head = (HistoryNode\*)malloc(**sizeof**(HistoryNode));  
 hists->list->tail = (HistoryNode\*)malloc(**sizeof**(HistoryNode));  
 hists->list->head->prev = NULL;  
 hists->list->head->next = hists->list->tail;  
 hists->list->tail->prev = hists->list->head;  
 hists->list->tail->next = NULL;  
  
 hists->list->head->data = construct\_history();  
 hists->list->tail->data = construct\_history();  
  
 hists->size = 0;  
 **return** hists;  
}  
  
/\*  
 \* History 구조체를 생성(할당)한다.  
 \*/  
History\* construct\_history(){  
 History\* hist = (History\*)malloc(**sizeof**(\*hist));  
  
 **return** hist;  
}  
  
/\*  
 \* Histories 구조체를 생성하면서 할당했던 모든 메모리를 해제한다.  
 \*/  
bool destroy\_histories(Histories \*\*histories\_state){  
 HistoryNode \*cur;  
 **int** i;  
 cur = ((\*histories\_state)->list->head);  
 **for**(i=0;i<(\*histories\_state)->size + 1;i++){  
 cur = (\*histories\_state)->list->head;  
 (\*histories\_state)->list->head = (\*histories\_state)->list->head->next;  
 free(cur->data);  
 free(cur);  
 }  
  
 free((\*histories\_state)->list->tail->data);  
 free((\*histories\_state)->list->tail);  
  
 free((\*histories\_state)->list);  
  
 free((\*histories\_state));  
  
 **return** true;  
}  
  
/\*  
 \* History 구조체를 생성(할당)하고 value 에 str 문자열을 저장한다.  
 \*/  
History\* construct\_history\_with\_string(**char**\* str){  
 History\* hist = construct\_history();  
 strncpy(hist->value, str, HISTORY\_MAX\_LEN);  
  
 **return** hist;  
}  
  
/\*  
 \* Histories 에 target History 구조체를 저장한다.  
 \*/  
bool push\_history(Histories\* histories\_store, History\* target){  
 assert(histories\_store);  
 assert(target);  
 assert(histories\_store->list);  
 assert(histories\_store->list->head);  
 assert(histories\_store->list->tail);  
  
 HistoryNode\* hist\_node = (HistoryNode\*)malloc(**sizeof**(HistoryNode));  
 hist\_node->data = target;  
  
 hist\_node->prev = histories\_store->list->tail->prev;  
 hist\_node->next = histories\_store->list->tail;  
 histories\_store->list->tail->prev->next = hist\_node;  
 histories\_store->list->tail->prev = hist\_node;  
 histories\_store->list->size += 1;  
  
 histories\_store->size += 1;  
 **return** true;  
}  
  
/\*  
 \* Histories 에 저장된 명령어 히스토리를 출력한다.  
 \*/  
**void** print\_history(Histories \*histories\_store, **char** \*last\_command) {  
 assert(histories\_store);  
 assert(last\_command);  
 assert(histories\_store->list);  
 assert(histories\_store->list->head);  
 assert(histories\_store->list->tail);  
  
 HistoryNode\*\* cur = &histories\_store->list->head;  
 **int** i = 0;  
 **for**(i=0;i<histories\_store->size + 1;i++){  
 **if**(i == 0){  
 cur = &((\*cur)->next);  
 **continue**;  
 }  
 printf("%-4d %s", i, (**char**\*)(\*cur)->data);  
 cur = &((\*cur)->next);  
  
 }  
 printf("%-4d %s", i, last\_command);  
 printf("\n");  
}

## history.h

#ifndef \_\_HISTORY\_H\_\_  
#define \_\_HISTORY\_H\_\_  
#define HISTORY\_MAX\_LEN 501  
#include "util.h"  
#include <stdlib.h>  
#include <stdbool.h>  
#include <string.h>  
#include <stdio.h>  
  
/\*  
 \* 히스토리에 저장될 하나의 명령어 문자열을  
 \* history 구조체에 wrapping 해서 저장하게된다.  
 \*/  
**typedef struct** history {  
 **char** value[HISTORY\_MAX\_LEN];  
} History;  
  
/\*  
 \* history\_list 의 Node 역활을 한다.  
 \*/  
**typedef struct** history\_node{  
 History\* data;  
 **struct** history\_node\* prev;  
 **struct** history\_node\* next;  
} HistoryNode;  
  
/\*  
 \* 명령어 히스토리를 링크드 리스트 형태로 저장하는 구조체.  
 \*/  
**typedef struct** history\_list {  
 **struct** history\_node\* head;  
 **struct** history\_node\* tail;  
 **int** size;  
} HistoryList;  
  
/\*  
 \* 히스토리에 담긴 명령어의 개수를 저장 하고  
 \* 실질적으로 히스토리를 저장하는 history\_list\* 를 멤버로 갖고있다.  
 \* 일종의 wrapper struct 이다.  
 \*/  
**typedef struct** histories {  
 **int** size;  
 HistoryList\* list;  
} Histories;  
  
/\*  
 \* Histories 구조체를 생성(할당)하고 HistoryList 도 생성(할당)한다.  
 \*/  
Histories\* construct\_histories();  
  
/\*  
 \* History 구조체를 생성(할당)한다.  
 \*/  
History\* construct\_history();  
  
/\*  
 \* History 구조체를 생성(할당)하고 value 에 str 문자열을 저장한다.  
 \*/  
History\* construct\_history\_with\_string(**char**\* str);  
  
/\*  
 \* Histories 에 target History 구조체를 저장한다.  
 \*/  
bool push\_history(Histories\* histories\_store, History\* target);  
  
/\*  
 \* Histories 에 저장된 명령어 히스토리를 출력한다.  
 \*/  
**void** print\_history(Histories \*histories\_store, **char** \*last\_command);  
  
/\*  
 \* Histories 구조체를 생성하면서 할당했던 모든 메모리를 해제한다.  
 \*/  
bool destroy\_histories(Histories \*\*histories\_state);  
  
#endif

## memory.c

#include "memory.h"  
  
/\*  
 \* Memories 구조체를 생성(할당)한다.  
 \*/  
Memories\* construct\_memories(){  
 Memories\* virtual\_memories = (Memories\*)malloc(**sizeof**(\*virtual\_memories));  
 virtual\_memories->last\_idx = -1;  
  
 **return** virtual\_memories;  
}  
  
/\*  
 \* Memories 구조체를 생성하면서 할당했던 메모리를 해제한다.  
 \*/  
bool destroy\_memories(Memories\*\* memories\_state){  
 free(\*memories\_state);  
 **return** true;  
}  
  
/\*  
 \* start ~ end 메모리 영역을 출력한다.  
 \* start 와 end 는 10진수로 변환한 값이다.  
 \*/  
**void** print\_memories(Memories\* memories\_state, **int** start, **int** end){  
 assert(memories\_state);  
 assert(start >= 0);  
 assert(end >= 0);  
 assert(start <= end);  
 assert(start < MEMORIES\_SIZE);  
 assert(end < MEMORIES\_SIZE);  
  
 **int** start\_row = (start / 16)\*16;  
 // ex, [dump 11] 에서 start 는 17이 되고, start\_row 는 16이 됨.  
 // ex, [dump AA] 에서 start 는 170이 되고, start\_row 는 160이 됨. ( 10 == 0xA0)  
  
 **int** end\_row = (end / 16)\*16;  
 **int** y, x, n;  
 **short** value;  
 **for**(y=start\_row;y<=end\_row;y+=16){  
 printf("%05X ", y);  
 **for**(x=0;x<16;x++){  
 n = y + x;  
 **if**(n < start || n > end) printf(" ");  
 **else** printf("%02X ", memories\_state->data[n].value);  
 }  
 printf("; ");  
 **for**(x=0;x<16;x++){  
 n = y + x;  
 **if**(n < start || n > end) printf(".");  
 **else** {  
 value = memories\_state->data[n].value;  
 **if**(0x20 <= value && value <= 0x7E) printf("%c", value);  
 **else** printf(".");  
 }  
 }  
 printf("\n");  
 }  
}  
  
/\*  
 \* 메모리 영역의 address 주소의 값을 value 로 수정한다.  
 \*/  
**void** edit\_memory(Memories\* memories\_state, **int** address, **short** value){  
 assert(address < MEMORIES\_SIZE && address >= 0);  
 assert(value >= 0 && value <= 0xFF);  
  
 memories\_state->data[address].value = value;  
}

## memory.h

#ifndef \_\_MEMORY\_H\_\_  
#define \_\_MEMORY\_H\_\_  
  
#define MEMORIES\_SIZE (1024 \* 1024) // 1MB  
#include <stdlib.h>  
#include <stdio.h>  
#include <assert.h>  
#include <stdbool.h>  
  
/\*  
 \* 메모리 값 하나를 의미하는 wrapper struct  
 \*/  
**typedef struct** memory {  
 **short** value;  
} Memory;  
  
/\*  
 \* 메모리 영역을 저장하는 구조체  
 \*/  
**typedef struct** memories {  
 Memory data[MEMORIES\_SIZE];  
 **int** last\_idx;  
} Memories;  
  
/\*  
 \* Memories 구조체를 생성(할당)한다.  
 \*/  
Memories\* construct\_memories();  
  
/\*  
 \* start ~ end 메모리 영역을 출력한다.  
 \* start 와 end 는 10진수로 변환한 값이다.  
 \*/  
**void** print\_memories(Memories\* memories\_state, **int** start, **int** end);  
  
/\*  
 \* 메모리 영역의 address 주소의 값을 value 로 수정한다.  
 \*/  
**void** edit\_memory(Memories\* memories\_state, **int** address, **short** value);  
  
/\*  
 \* Memories 구조체를 생성하면서 할당했던 메모리를 해제한다.  
 \*/  
bool destroy\_memories(Memories\*\* memories\_state);  
  
#endif

## opcode.c

#include "opcode.h"

/\*

\* Opcode 구조체를 생성(할당)한다.

\*/

Opcode\* construct\_opcode(){

Opcode\* op = (Opcode\*)malloc(sizeof(Opcode));

op->value = -1;

return op;

}

/\*

\* op\_node 를 생성(할당)한다.

\*/

struct op\_node\* construct\_opnode(){

struct op\_node\* node = (struct op\_node\*)malloc(sizeof(struct op\_node));

return node;

}

/\*

\* OpcodeTable 을 생성(할당)한다.

\*/

OpcodeTable\* construct\_opcode\_table(){

OpcodeTable\* table = (OpcodeTable\*)malloc(sizeof(OpcodeTable));

table->list = (OpLinkedList\*\*)malloc(sizeof(OpLinkedList\*)\*20);

for(int i = 0; i<20;i++) {

table->list[i] = (OpLinkedList\*)malloc(sizeof(OpLinkedList));

table->list[i]->head = construct\_opnode();

table->list[i]->tail = construct\_opnode();

table->list[i]->head->prev = NULL;

table->list[i]->head->next = table->list[i]->tail;

table->list[i]->tail->prev = table->list[i]->head;

table->list[i]->tail->next = NULL;

table->list[i]->head->data = construct\_opcode();

table->list[i]->tail->data = construct\_opcode();

table->list[i]->size = 0;

}

table->size = 20;

return table;

}

/\*

\* opcode 파일을 읽어서 OpcodeTable 에 적절히 저장한다

\*/

bool build\_opcode\_table(OpcodeTable\* table){

FILE\* fp = fopen("opcode.txt", "rt");

if(!fp) {

fprintf(stderr, "[ERROR] opcode file not exists\n");

return false;

}

char format\_name[20];

char name[20];

unsigned int value;

while (fscanf(fp, "%X %6s %5s",

&value, name, format\_name) != EOF){

Opcode\* opc = construct\_opcode();

strncpy(opc->mnemonic\_name, name, 10);

opc->value = value;

if(opc->value > 0xFF) {

fprintf(stderr, "[ERROR] %X %6s %5s is Invalid Input!\n", value, name, format\_name);

continue;

}

// printf("%s %d\n", opc->mnemonic\_name,opc->value);

if(COMPARE\_STRING(format\_name, "1")){

opc->format = OP\_FORMAT\_1;

}else if(COMPARE\_STRING(format\_name, "2")){

if(COMPARE\_STRING(opc->mnemonic\_name, "CLEAR") ||

COMPARE\_STRING(opc->mnemonic\_name, "TIXR"))

opc->format = OP\_FORMAT\_2\_ONE\_REG;

else if(COMPARE\_STRING(opc->mnemonic\_name, "SHIFTL") ||

COMPARE\_STRING(opc->mnemonic\_name, "SHIFTR"))

opc->format = OP\_FORMAT\_2\_REG\_N;

else if(COMPARE\_STRING(opc->mnemonic\_name, "SVC"))

opc->format = OP\_FORMAT\_2\_ONE\_N;

else

opc->format = OP\_FORMAT\_2\_GEN;

} else if(COMPARE\_STRING(format\_name, "3/4")){

if(COMPARE\_STRING(opc->mnemonic\_name, "RSUB"))

opc->format = OP\_FORMAT\_3\_4\_NO\_OPERAND;

else

opc->format = OP\_FORMAT\_3\_4\_GEN;

}

insert\_opcode(table, opc);

}

fclose(fp);

return true;

}

/\*

\* OpcodeTable 을 해제한다.

\*/

bool destroy\_opcode\_table(OpcodeTable\*\* table){

assert(table);

OpNode \*cur;

OpNode \*next;

OpLinkedList\*\* list;

int i, j;

list = (\*table)->list;

for(i=0;i<(\*table)->size;i++){

cur = list[i]->head;

for(j=0;j<list[i]->size+1;j++){

next = cur->next;

free(cur->data);

free(cur);

cur = next;

}

free(list[i]);

}

free(\*table);

return true;

}

/\*

\* OpcodeTable 에 Opcode 를 추가한다.

\*/

bool insert\_opcode(OpcodeTable\* table, Opcode\* opc){

OpNode\* op\_node = construct\_opnode();

op\_node->data = opc;

int hash = (int)hash\_string(opc->mnemonic\_name, 20);

op\_node->prev = table->list[hash]->tail->prev;

op\_node->next = table->list[hash]->tail;

table->list[hash]->tail->prev->next = op\_node;

table->list[hash]->tail->prev = op\_node;

table->list[hash]->size += 1;

// assert(opc->value >= 0);

// assert(table->list[hash]->tail->prev->data->value >= 0);

return true;

}

/\*

\* OpcodeTable 에서 name 문자열 이름의 mnemonic 을 가진

\* Opcode 를 찾고 리턴한다.

\*/

Opcode\* find\_opcode\_by\_name(OpcodeTable\* table, char\* name){

assert(strlen(name) <= 14);

assert(table);

int hash = (int)hash\_string(name, table->size);

int i = 0;

OpNode\*\* cur = &(table->list[hash]->head);

Opcode\* opc;

// DEFAULT OPCODE only just hard coding

if(COMPARE\_STRING(name, "START")

|| COMPARE\_STRING(name, "END")

|| COMPARE\_STRING(name, "BYTE")

|| COMPARE\_STRING(name, "WORD")

|| COMPARE\_STRING(name, "RESB")

|| COMPARE\_STRING(name, "RESW")

|| COMPARE\_STRING(name, "BASE")

|| COMPARE\_STRING(name, "NOBASE")){

opc = construct\_opcode();

strncpy(opc->mnemonic\_name, name, 10);

opc->value = 0;

opc->format = OP\_DEFAULT;

return opc;

}

for(i=0;i<(table->list[hash]->size)+1;i++){

opc = (\*cur)->data;

if(opc->value == -1){

cur = &((\*cur)->next);

continue;

}

if(COMPARE\_STRING(name, opc->mnemonic\_name)){

return opc;

}

cur = &((\*cur)->next);

}

return NULL;

}

/\*

\* OpcodeTable 에 저장된 opcode 목록을 출력한다.

\*/

void print\_opcodes(OpcodeTable\* table){

int size = table->size;

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

printf("%2d : ", i);

OpNode\*\* cur = &(table->list[i]->head);

Opcode\* opc;

for(int j=0;j<table->list[i]->size+1;j++){

opc = (\*cur)->data;

// if(opc->value == -1) cnt -= 1;

if(opc->value == -1){

cur=&((\*cur)->next);

continue;

}

printf("[%s,%02X] ",opc->mnemonic\_name,opc->value);

// printf("gogo %s %d\n", opc->mnemonic\_name, opc->value);

// cnt += 1;

if(j != table->list[i]->size)

printf(" -> ");

cur = &((\*cur)->next);

}

printf("\n");

}

}

// format 을 매핑하는 함수

enum op\_format op\_format\_by\_op\_num(int op\_num){

bool format3\_4[260] = {

[0x00] = true,

[0x68] = true,

[0x74] = true,

[0x04] = true,

[0x50] = true,

[0x0C] = true,

[0x14] = true,

[0x10] = true,

[0x54] = true,

[0x48] = true,

[0x30] = true,

[0x34] = true,

[0x38] = true,

[0x3C] = true,

[0x28] = true,

[0xE0] = true,

[0xD8] = true,

[0x4C] = true,

[0xDC] = true

};

bool format2[260] = {

[0xB4] = true,

[0xA0] = true,

[0xB8] = true

};

if(format3\_4[op\_num]) return OP\_FORMAT\_3\_4\_GEN;

if(format2[op\_num]) return OP\_FORMAT\_2\_GEN;

return OP\_FORMAT\_1;

}

## opcode.h

#ifndef \_\_OPCODE\_H\_\_

#define \_\_OPCODE\_H\_\_

#include <stdbool.h>

#include "util.h"

/\*

\* opcode 의 format 을 enum 으로 표현한다. (구현중)

\*/

enum op\_format {

OP\_FORMAT\_1,

OP\_FORMAT\_2\_ONE\_REG, OP\_FORMAT\_2\_REG\_N,

OP\_FORMAT\_2\_ONE\_N, OP\_FORMAT\_2\_GEN,

OP\_FORMAT\_3\_4\_NO\_OPERAND, OP\_FORMAT\_3\_4\_GEN,

OP\_DEFAULT

};

/\*

\* opcode 의 mnemonic 을 enum 으로 표현한다. (구현중)

\*/

enum op\_mnemonic {

OP\_ADD, OP\_ADDR

};

/\*

\* opcode 정보 하나를 나타낸다.

\*/

typedef struct opcode {

enum op\_format format;

enum op\_mnemonic mnemonic;

char mnemonic\_name[10];

int value;

} Opcode;

/\*

\* op\_linked\_list 의 Node 역활

\*/

typedef struct op\_node{

Opcode\* data;

struct op\_node\* prev;

struct op\_node\* next;

} OpNode;

/\*

\* 링크드 리스트

\*/

typedef struct op\_linked\_list {

struct op\_node\* head;

struct op\_node\* tail;

int size;

}OpLinkedList;

/\*

\* opcode 정보들을 해시테이블 형태로 저장하기 위한 구조체.

\*/

typedef struct opcode\_table {

OpLinkedList\*\* list;

int size;

} OpcodeTable;

/\*

\* Opcode 구조체를 생성(할당)한다.

\*/

Opcode\* construct\_opcode();

/\*

\* op\_node 를 생성(할당)한다.

\*/

struct op\_node\* construct\_opnode();

/\*

\* OpcodeTable 을 생성(할당)한다.

\*/

OpcodeTable\* construct\_opcode\_table();

/\*

\* opcode 파일을 읽어서 OpcodeTable 에 적절히 저장한다

\*/

bool build\_opcode\_table(OpcodeTable\* table);

/\*

\* OpcodeTable 을 해제한다.

\*/

bool destroy\_opcode\_table(OpcodeTable\*\* table);

/\*

\* OpcodeTable 에 Opcode 를 추가한다.

\*/

bool insert\_opcode(OpcodeTable\* table, Opcode\* opc);

/\*

\* OpcodeTable 에서 name 문자열 이름의 mnemonic 을 가진

\* Opcode 를 찾고 리턴한다.

\*/

Opcode\* find\_opcode\_by\_name(OpcodeTable\* table, char\* name);

/\*

\* OpcodeTable 에 저장된 opcode 목록을 출력한다.

\*/

void print\_opcodes(OpcodeTable\* table);

// format 을 매핑하는 함수

enum op\_format op\_format\_by\_op\_num(int op\_num);

#endif

## state.c #include "state.h" #include "util.h" #include "assemble.h" /\* \* History, 가상 Memory, Opcode 정보가 초기화(및 저장)된 State\* 을 리턴한다. \*/ State\* construct\_state(){ State\* state\_obj = (State\*)malloc(sizeof(\*state\_obj)); state\_obj->histories\_state = construct\_histories(); state\_obj->memories\_state = construct\_memories(); state\_obj->opcode\_table\_state = construct\_opcode\_table(); build\_opcode\_table(state\_obj->opcode\_table\_state); state\_obj->symbol\_table\_state = construct\_symbol\_table(); state\_obj->is\_symbol\_table = false; return state\_obj; } /\* \* state\_store 가 동적 할당한 모든 메모리를 해제한다. \*/ bool destroy\_state(State \*\*state\_store){ destroy\_histories(&((\*state\_store)->histories\_state)); destroy\_memories(&((\*state\_store)->memories\_state)); destroy\_opcode\_table(&(\*state\_store)->opcode\_table\_state); destroy\_symbol\_table(&((\*state\_store)->symbol\_table\_state)); free(\*state\_store); return true; } /\* \* history\_str 문자열을 명령어 히스토리에 기록한다. \*/ bool add\_history(State \*state\_store, char\* history\_str){ return push\_history(state\_store->histories\_state, construct\_history\_with\_string(history\_str)); } /\* \* 명령어 히스토리를 출력한다. \*/ void print\_histories\_state(State\* state\_store, char\* last\_command){ print\_history(state\_store->histories\_state, last\_command); } /\* \* file 을 assemble 하여 state 변경 및 성공 오류 여부 리턴 \*/ bool assemble\_file(State \*state\_store, char \*asm\_file\_name){ assert(strlen(asm\_file\_name) < MAX\_ASM\_FILENAME\_LENGTH); free(state\_store->symbol\_table\_state); state\_store->symbol\_table\_state = construct\_symbol\_table(); if(!assemble\_pass1(state\_store, asm\_file\_name)) { state\_store->is\_symbol\_table = false; return false; } if(!assemble\_pass2(state\_store, asm\_file\_name)){ state\_store->is\_symbol\_table = false; return false; } state\_store->is\_symbol\_table = true; return true; } /\* \* assembler 의 pass1 과정을 구현하였다. \*/ bool assemble\_pass1(State \*state\_store, char \*asm\_file\_name) { FILE\* asm\_fp = fopen(asm\_file\_name, "r"); char\* tmp\_file\_name, \*prefix, \*lst\_file\_name, \*obj\_file\_name; int location\_counter = 0, stmt\_size = 0, line\_num = -1; Statement stmt; FILE\* tmp\_fp = NULL; prefix = before\_dot(asm\_file\_name, MAX\_ASM\_FILENAME\_LENGTH); tmp\_file\_name = concat\_n(prefix, ".tmp", MAX\_ASM\_FILENAME\_LENGTH); lst\_file\_name = concat\_n(prefix, ".lst", MAX\_ASM\_FILENAME\_LENGTH); obj\_file\_name = concat\_n(prefix, ".obj", MAX\_ASM\_FILENAME\_LENGTH); if(!tmp\_file\_name){ fprintf(stderr, "[ERROR] Invalid File Format\n"); error\_handling\_pass1or2(NULL, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } tmp\_fp = fopen(tmp\_file\_name, "wt"); if(!tmp\_fp){ fprintf(stderr, "[ERROR] Can't Create tmp file\n"); error\_handling\_pass1or2(NULL, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } if(!asm\_fp){ fprintf(stderr, "[ERROR] Can't Open File\n"); error\_handling\_pass1or2(NULL, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num = 5; if(!read\_statement(state\_store->opcode\_table\_state, asm\_fp, tmp\_fp, &stmt, false, &location\_counter, &stmt\_size)) { error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } if(!stmt.comment && COMPARE\_STRING(stmt.opcode->mnemonic\_name, "START")){ location\_counter = strtol (stmt.tokens[0], NULL, 16); fprintf (tmp\_fp, "%04X\t0\t%s\n", location\_counter, stmt.raw\_input); if (!read\_statement(state\_store->opcode\_table\_state, asm\_fp, tmp\_fp, &stmt, false, &location\_counter, &stmt\_size)){ error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num += 5; } while (1){ if (is\_comment\_stmt(&stmt)){ record\_stmt\_for\_pass1(&stmt, tmp\_fp, &location\_counter, NULL); if(is\_end\_condition(&stmt, asm\_fp)) break; if (!read\_statement(state\_store->opcode\_table\_state, asm\_fp, tmp\_fp, &stmt, false, NULL, NULL)) { error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num += 5; continue; } int old\_location\_counter = location\_counter; if (stmt.raw\_symbol){ if(find\_symbol\_by\_name(state\_store->symbol\_table\_state, stmt.raw\_symbol)) { error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } Symbol\* symbol = construct\_symbol(); strncpy (symbol->label, stmt.raw\_symbol, 10); symbol->location\_counter = location\_counter; // [TODO] symbol insert 테스트 필요 insert\_symbol(state\_store->symbol\_table\_state, symbol); } update\_location\_counter\_by\_format(&stmt, &location\_counter); if(!update\_location\_counter\_by\_mnemonic\_name(&stmt, &location\_counter)){ error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } if(!update\_location\_counter\_by\_plus\_and\_format(&stmt, &location\_counter)){ error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } // fprintf (tmp\_fp, "%04X\t%X\t%s\n", // (unsigned int) old\_location\_counter, // (unsigned int)(location\_counter - old\_location\_counter), // stmt.raw\_input); record\_stmt\_for\_pass1(&stmt, tmp\_fp, &location\_counter, &old\_location\_counter); if(is\_end\_condition(&stmt, asm\_fp)) break; if (!read\_statement(state\_store->opcode\_table\_state, asm\_fp, tmp\_fp, &stmt, false, NULL, NULL)) { error\_handling\_pass1or2(&stmt, asm\_fp, tmp\_fp, NULL, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num += 5; } fclose(asm\_fp); fclose(tmp\_fp); free(lst\_file\_name); free(tmp\_file\_name); free(obj\_file\_name); // print\_symbols(state\_store->symbol\_table\_state); return true; } bool assemble\_pass2(State \*state\_store, char \*asm\_file\_name) { FILE \*tmp\_fp, \*lst\_fp, \*obj\_fp; char \*tmp\_file\_name, \*lst\_file\_name, \*obj\_file\_name; char\* prefix = before\_dot(asm\_file\_name, MAX\_ASM\_FILENAME\_LENGTH); int line\_num = -1; Statement stmt; int location\_counter = 0, stmt\_size = 0, obj\_code, r\_lc, start\_lc; int\* location\_counters = malloc(sizeof(int)\*1001); char\* b\_buf = malloc(sizeof(char)\*1001); int location\_counter\_cnt = 0; bool is\_base = false; int base; char symb[11] = {0, }; char\* obj\_buf = malloc(sizeof(char)\*1001); char\* rec\_head = malloc(sizeof(char)\*31); tmp\_file\_name = concat\_n(prefix, ".tmp", MAX\_ASM\_FILENAME\_LENGTH); lst\_file\_name = concat\_n(prefix, ".lst", MAX\_ASM\_FILENAME\_LENGTH); obj\_file\_name = concat\_n(prefix, ".obj", MAX\_ASM\_FILENAME\_LENGTH); tmp\_fp = fopen(tmp\_file\_name, "rt"); lst\_fp = fopen(lst\_file\_name, "wt"); obj\_fp = fopen(obj\_file\_name, "wt"); if(!tmp\_fp || !lst\_fp || !obj\_fp) { fprintf(stderr, "[ERROR] Can't Open Files \n"); error\_handling\_pass1or2(NULL, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } start\_lc = r\_lc = location\_counter; obj\_buf[0] = '\0'; line\_num = 5; if(!read\_statement(state\_store->opcode\_table\_state, NULL, tmp\_fp, &stmt, true, &location\_counter, &stmt\_size)) { error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } if(!stmt.comment && COMPARE\_STRING(stmt.opcode->mnemonic\_name,"START")){ if(stmt.raw\_symbol) strncpy(symb, stmt.raw\_symbol, 11); fprintf (lst\_fp, "%d\t%04X%s\n", line\_num, location\_counter, stmt.raw\_input); fprintf (obj\_fp, "%19s\n", ""); if (!read\_statement(state\_store->opcode\_table\_state, NULL, tmp\_fp, &stmt, true, &location\_counter, &stmt\_size)){ error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num += 5; } while (1){ if(stmt.comment){ fprintf (lst\_fp, "%d\t%s\n", line\_num, stmt.raw\_input); if(is\_end\_condition(&stmt, tmp\_fp)) break; if(!read\_statement(state\_store->opcode\_table\_state, NULL, tmp\_fp, &stmt, true, &location\_counter, &stmt\_size)) { error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num += 5; continue; } if(is\_format(&stmt, 1) && !handling\_format1(&stmt, &obj\_code)) { error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } else if(is\_format(&stmt, 2) && !handling\_format2(&stmt, &obj\_code)){ error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; }else if(is\_format(&stmt, 3) && !handling\_format3(state\_store->symbol\_table\_state, &stmt, &obj\_code, &location\_counter, &location\_counters, &location\_counter\_cnt, stmt\_size, &is\_base, &base)) { error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } else if(is\_format(&stmt, 0) && !handling\_format\_default(state\_store->symbol\_table\_state, &stmt, &obj\_code, &is\_base, &base, &b\_buf)) { error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } record\_stmt\_for\_pass2(&stmt, &obj\_code, &location\_counter, &r\_lc, &line\_num, lst\_fp, obj\_fp, &obj\_buf, &b\_buf, &rec\_head); if(is\_end\_condition(&stmt, tmp\_fp)) break; if(!read\_statement(state\_store->opcode\_table\_state, NULL, tmp\_fp, &stmt, true, &location\_counter, &stmt\_size)) { error\_handling\_pass1or2(&stmt, lst\_fp, tmp\_fp, obj\_fp, tmp\_file\_name, lst\_file\_name, obj\_file\_name, line\_num); return false; } line\_num += 5; } location\_counter += stmt\_size; if(location\_counter + 30 > r\_lc + 30){ snprintf(rec\_head, 30, "T%06X%02X", r\_lc, (uint8\_t)strlen(obj\_buf) / 2); fprintf (obj\_fp, "%s%s\n", rec\_head, obj\_buf); r\_lc = location\_counter; obj\_buf[0] = '\0'; } for (int i = 0; i < location\_counter\_cnt; ++i){ snprintf (rec\_head, 30, "M%06X05", location\_counters[i]); fprintf (obj\_fp, "%s\n", rec\_head); } snprintf (rec\_head, 30, "E%06X", start\_lc); fprintf (obj\_fp, "%s\n", rec\_head); snprintf (rec\_head, 30, "H%-6s%06X%06X", symb, start\_lc, location\_counter - start\_lc); fseek (obj\_fp, 0, SEEK\_SET); fprintf (obj\_fp, "%s\n", rec\_head); if(tmp\_fp) fclose(tmp\_fp); if(lst\_fp) fclose(lst\_fp); if(obj\_fp) fclose(obj\_fp); remove(tmp\_file\_name); free(tmp\_file\_name); free(location\_counters); free(b\_buf); free(obj\_buf); free(rec\_head); fprintf(stdout, "\toutput file : [%s], [%s]\n", lst\_file\_name, obj\_file\_name); free(lst\_file\_name); free(obj\_file\_name); return true; }

## state.h

#ifndef \_\_STATE\_H\_\_

#define \_\_STATE\_H\_\_

#include "history.h"

#include "memory.h"

#include "opcode.h"

#include "symbol.h"

#include "assemble.h"

#include "util.h"

#include "debug.h"

#include <stdlib.h>

#include <stdint.h>

#define MAX\_ASM\_FILENAME\_LENGTH 300

/\*

\* state 구조체에서는 명령어 히스토리, 가상 메모리 영역, Opcode 정보를 저장하는

\* 구조체 포인터들을 멤버 변수로 갖고있다.

\*/

typedef struct state {

// 명령어 히스토리

Histories\* histories\_state;

// 가상 메모리 영역

Memories\* memories\_state;

// opcode 파일을 읽어 들인 내용들

OpcodeTable\* opcode\_table\_state;

// symbol 정보 저장

SymbolTable\* symbol\_table\_state;

// run, bp, progaddr 등의 정보 저장

Debugger\* debugger\_state;

bool is\_symbol\_table;

} State;

/\*

\* History, 가상 Memory, Opcode 정보가 초기화(및 저장)된 State\* 을 리턴한다.

\*/

State\* construct\_state();

/\*

\* state\_store 가 동적 할당한 모든 메모리를 해제한다.

\*/

bool destroy\_state(State \*\*state\_store);

/\*

\* history\_str 문자열을 명령어 히스토리에 기록한다.

\*/

bool add\_history(State \*state\_store, char\* history\_str);

/\*

\* 명령어 히스토리를 출력한다.

\*/

void print\_histories\_state(State\* state\_store, char\* last\_command);

/\*

\* file 을 assemble 하여 state 변경 및 성공 오류 여부 리턴

\*/

bool assemble\_file(State \*state\_store, char \*asm\_file\_name);

/\*

\* assembler 의 pass1 과정을 구현하였다.

\*/

bool assemble\_pass1(State \*state\_store, char \*asm\_file\_name);

/\*

\* assembler 의 pass2 과정을 구현하였다.

\*/

bool assemble\_pass2(State \*state\_store, char \*asm\_file\_name);

#endif

## util.c

#include "state.h"  
#include "util.h"  
  
/\*  
 \* hashtable 구현을 위한 hash function  
 \*/  
size\_t hash\_string (char \*str, int hash\_size){  
 int32\_t hash = 2829;  
 int32\_t c;  
 size\_t res;  
 while((c = \*str++)){  
 hash = (hash \* 615) + c;  
 }  
 res = (size\_t)hash % hash\_size;  
 return res;  
}  
  
/\*  
 \* 문자열 str 이 0으로 변환될수있는지 확인한다.  
 \* ex, is\_zero\_str("0000") => return true  
 \* ex, is\_zero\_str("A00") => return false  
 \*/  
bool is\_zero\_str(char\* str){  
 assert(str);  
 int len = (int)strlen(str);  
 int i;  
 for(i=0;i<len;i++)  
 if(str[i] != '0')  
 return false;  
 return true;  
}  
  
/\*  
 \* 문자열 str 이 16 진수인지 확인한다.  
 \* ex, is\_valid\_hex("00F1") => return true  
 \* ex2, is\_valid\_hex("FZ") => return false  
 \*/  
bool is\_valid\_hex(char\* str){  
 assert(str);  
 int l = (int)strlen(str), i;  
 for(i=0;i<l;i++) {  
 if ('0' <= str[i] &&  
 str[i] <= '9')  
 continue;  
 if('A' <= str[i] &&  
 str[i] <= 'F')  
 continue;  
 if('a' <= str[i] &&  
 str[i] <= 'f')  
 continue;  
 return false;  
 }  
 return true;  
}  
  
  
/\*  
 \* 문자열 str 이 [0 ~ max\_size-1] 범위의 적절한 주소값인지 확인한다.  
 \* ex, is\_zero\_str("00F", 100) => return true  
 \* ex, is\_zero\_str("FG", 100000) => return false  
 \*/  
bool is\_valid\_address(char \*str, int max\_size) {  
 assert(str);  
 int target = (int)strtol(str, NULL, 16);  
  
 if(target < 0) return false; // 0 보다 큰지 검증  
 if(target == 0 && !is\_zero\_str(str)) return false; // 올바른 hex 값인지 검증  
 if(target >= max\_size) return false; // 범위 내에 있는지 검증  
 if(!is\_valid\_hex(str)) return false; // 올바른 hex 값인지 검증  
  
 return true;  
}  
  
/\*  
 \* 문자열에서 . 이전의 문자열 을 찾아서 리턴한다.  
 \* 예를들어 before\_dot(2\_5.asm) 은 2\_5가 리턴된다.  
 \*/  
char \*before\_dot(char \*name, int size) {  
 char \*pre;  
 char\* dot;  
 pre = malloc(sizeof(char)\*size);  
  
 strncpy(pre, name, size);  
  
 dot = strrchr (pre,'.');  
  
 if(dot == NULL){  
 return NULL;  
 }  
 \*dot = '\0';  
  
 return pre;  
}  
  
char \*concat\_n(char \*name, char \*name2, int max\_size) {  
 char\* res;  
 res = malloc(sizeof(char)\*max\_size);  
  
 snprintf (res, max\_size, "%s%s", name, name2);  
  
 return res;  
}

## util.h

#ifndef \_\_UTIL\_H\_\_  
#define \_\_UTIL\_H\_\_  
#include <stdlib.h>  
#include <assert.h>  
#include <stdio.h>  
#include <stdbool.h>  
#include <string.h>  
  
#define COMPARE\_STRING(T, S) (strcmp ((T), (S)) == 0)  
  
/\*  
 \* hashtable 구현을 위한 hash function  
 \*/  
size\_t hash\_string (char \*str, int hash\_size);  
  
/\*  
 \* 문자열 str 이 0으로 변환될수있는지 확인한다.  
 \* ex, is\_zero\_str("0000") => return true  
 \* ex, is\_zero\_str("A00") => return false  
 \*/  
bool is\_zero\_str(char\* str);  
  
/\*  
 \* 문자열 str 이 16 진수인지 확인한다.  
 \* ex, is\_valid\_hex("00F1") => return true  
 \* ex2, is\_valid\_hex("FZ") => return false  
 \*/  
bool is\_valid\_hex(char\* str);  
  
/\*  
 \* 문자열 str 이 [0 ~ max\_size-1] 범위의 적절한 주소값인지 확인한다.  
 \* ex, is\_zero\_str("00F", 100) => return true  
 \* ex, is\_zero\_str("FG", 100000) => return false  
 \*/  
bool is\_valid\_address(char \*str, int max\_size);  
  
/\*  
 \* 문자열에서 . 이전의 문자열 을 찾아서 리턴한다.  
 \* 예를들어 before\_dot(2\_5.asm) 은 2\_5가 리턴된다.  
 \*/  
char \*before\_dot(char \*name, int size);  
  
/\*  
 \* 두 문자열을 합치는 concat 함수  
 \*/  
char \*concat\_n(char \*name, char \*name2, int max\_size);  
  
#endif

## symbol.h

#ifndef \_\_SYMBOL\_H\_\_  
#define \_\_SYMBOL\_H\_\_  
  
#include <stdbool.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
typedef struct symbol {  
 char label[11];  
 int location\_counter;  
}Symbol;  
  
/\*  
 \* 링크드 리스트 구현을 위한 노드  
 \*/  
typedef struct sym\_node {  
 Symbol\* data;  
 struct sym\_node\* prev;  
 struct sym\_node\* next;  
}SymNode;  
  
/\*  
 \* 링크드 리스트  
 \*/  
typedef struct sym\_linked\_list {  
 SymNode\* head;  
 SymNode\* tail;  
 int size;  
}SymLinkedList;  
  
/\*  
 \* symbol 정보들을 해시테이블 형태로 저장하기 위한 구조체.  
 \*/  
typedef struct symbol\_table {  
 SymLinkedList\* list[40];  
 int size;  
}SymbolTable;  
  
/\*  
 \* Symbol table 생성자 함수  
 \*/  
SymbolTable\* construct\_symbol\_table();  
  
/\*  
 \* Symbol table 소멸자 함수  
 \*/  
bool destroy\_symbol\_table(SymbolTable\*\* table);  
  
/\*  
 \* SymNode 생성자 함수  
 \*/  
SymNode\* construct\_symbol\_node();  
  
/\*  
 \* Symbol 생성자 함수  
 \*/  
Symbol\* construct\_symbol();  
  
/\*  
 \* SymbolTable 구조체의 해시테이블에 Symbol 을 추가한다.  
 \*/  
bool insert\_symbol(SymbolTable\* table, Symbol\* symbol);  
  
/\*  
 \* Symbol 을 찾는 함수  
 \*/  
Symbol\* find\_symbol\_by\_name(SymbolTable \*table, char \*name);  
  
/\*  
\* SymbolTable 에 저장된 Symbol 들을 내림차순으로 출력한다  
\*/  
void print\_symbols(SymbolTable\* table);  
  
/\*  
 \* Sort Comparator 함수  
 \*/  
int symbol\_comparator(const void \*a, const void \*b);  
  
#endif

## symbol.c

#include "symbol.h"  
#include "util.h"  
  
/\*  
 \* Symbol table 생성자 함수  
 \*/  
SymbolTable\* construct\_symbol\_table(){  
 SymbolTable\* table = malloc(sizeof(\*table));  
 int i;  
 for(i = 0; i < 40;i++){  
 table->list[i] = (SymLinkedList\*)malloc(sizeof(SymLinkedList));  
 table->list[i]->head = construct\_symbol\_node();  
 table->list[i]->tail = construct\_symbol\_node();  
  
 table->list[i]->head->prev = NULL;  
 table->list[i]->head->next = table->list[i]->tail;  
 table->list[i]->tail->prev = table->list[i]->head;  
 table->list[i]->tail->next = NULL;  
  
 table->list[i]->head->data = construct\_symbol();  
 table->list[i]->tail->data = construct\_symbol();  
 table->list[i]->size = 0;  
 }  
  
 table->size = 40;  
 return table;  
}  
  
/\*  
 \* Symbol table 소멸자 함수  
 \*/  
bool destroy\_symbol\_table(SymbolTable\*\* table){  
 SymNode \*cur;  
 SymNode \*next;  
 SymLinkedList\*\* list;  
 int i, j;  
  
 list = (\*table)->list;  
 for(i=0;i<(\*table)->size;i++){  
 cur = list[i]->head;  
  
 for(j=0;j<list[i]->size+1;j++){  
 next = cur->next;  
 free(cur->data);  
 free(cur);  
 cur = next;  
 }  
 free(list[i]);  
 }  
 free(\*table);  
  
 return true;  
}  
  
/\*  
 \* SymNode 생성자 함수  
 \*/  
SymNode\* construct\_symbol\_node(){  
 SymNode\* node = (SymNode\*)malloc(sizeof(SymNode));  
  
 return node;  
}  
  
/\*  
 \* Symbol 생성자 함수  
 \*/  
Symbol\* construct\_symbol(){  
 Symbol\* symb = (Symbol\*)malloc(sizeof(Symbol));  
 strncpy(symb->label, "---nono--", 11);  
  
 return symb;  
}  
  
/\*  
 \* SymbolTable 구조체의 해시테이블에 Symbol 을 추가한다.  
 \*/  
bool insert\_symbol(SymbolTable\* table, Symbol\* symbol){  
 SymNode \*node = malloc(sizeof(SymNode));  
 int hash = (int)hash\_string(symbol->label, 40);  
  
 node->data = symbol;  
  
 node->prev = table->list[hash]->tail->prev;  
 node->next = table->list[hash]->tail;  
 table->list[hash]->tail->prev->next = node;  
 table->list[hash]->tail->prev = node;  
// fprintf(stdout, "[SUCCESS?] insert symbol %s\n", table->list[hash]->tail->prev->data->label);  
 table->list[hash]->size += 1;  
  
// fprintf(stdout, "[SUCCESS] insert symbol %s\n", symbol->label);  
  
 return true;  
}  
  
/\*  
 \* Symbol 을 찾는 함수  
 \*/  
Symbol \* find\_symbol\_by\_name(SymbolTable \*table, char \*name){  
 int hash = (int)hash\_string(name, 40);  
 int i;  
  
 SymNode\*\* cur = &(table->list[hash]->head);  
 Symbol\* symb;  
  
 for(i=0;i<(table->list[hash]->size)+1;i++){  
 symb = (\*cur)->data;  
 if(COMPARE\_STRING(symb->label, "---nono--")){  
 cur = &((\*cur)->next);  
 continue;  
 }  
  
 if(COMPARE\_STRING(symb->label, name)){  
 return symb;  
 }  
 cur = &((\*cur)->next);  
 }  
 return NULL;  
}  
  
/\*  
\* SymbolTable 에 저장된 Symbol 들을 내림차순으로 출력한다  
\*/  
void print\_symbols(SymbolTable\* table){  
 int size = table->size;  
 Symbol \*list[1200] = {0};  
 int i = 0;  
 int num = 0;  
 for(i = 0;i < size;i++){  
 SymNode\*\* cur = &(table->list[i]->head);  
 Symbol\* symb;  
 for(int j=0;j<table->list[i]->size+1;j++){  
 symb = (\*cur)->data;  
  
 if(COMPARE\_STRING(symb->label, "---nono--")){  
 cur=&((\*cur)->next);  
 continue;  
 }  
 list[num++] = symb;  
 cur = &((\*cur)->next);  
 }  
 }  
// symbol\_comparator(NULL, NULL);  
 qsort(list, num, sizeof(Symbol \*), symbol\_comparator);  
  
 for (int k = 0; k < num; ++k){  
 if(!list[k])  
 return;  
 printf ("\t%-5s\t%04X\n", list[k]->label, list[k]->location\_counter);  
 }  
}  
  
/\*  
 \* Sort Comparator 함수  
 \*/  
int symbol\_comparator(const void \*a, const void \*b){  
 if(!a || !b) return 0;  
  
 Symbol \*left = \*(Symbol \*\*) a;  
 Symbol \*right = \*(Symbol \*\*) b;  
  
 if(!left) return 0;  
 if(!right) return 0;  
  
 return -1\*strcmp(left->label, right->label);  
}

## assemble.h

#ifndef \_\_ASSEMBLE\_H\_\_  
#define \_\_ASSEMBLE\_H\_\_  
  
#include <stdlib.h>  
#include <stdbool.h>  
#include "opcode.h"  
#include "symbol.h"  
#include <stdint.h>  
#include <stdio.h>  
  
#define MAX\_TOKENS\_LENGTH 8  
  
/\*  
 \* 어셈블리 코드 한줄 Statement 라고 정의한다.  
 \* 코드의 정보를 이 구조체에 저장한다.  
 \*/  
typedef struct statement {  
 char\* raw\_input;  
 int token\_cnt;  
 char\* tokens[MAX\_TOKENS\_LENGTH];  
 bool comment;  
 Opcode\* opcode;  
 char\* raw\_symbol;  
 bool tmp\_bool;  
 bool plus;  
}Statement;  
  
/\*  
 \* format 2 인 Statement 일때,  
 \* 비트 값으로 저장하기 위한 변수이다.  
 \*/  
typedef union bits\_format2{  
 struct  
 {  
 uint16\_t r2 : 4;  
 uint16\_t r1 : 4;  
 uint16\_t opcode : 8;  
 } bits;  
 uint16\_t res;  
}BitsFormat2;  
  
/\*  
 \* format 3 인 Statement 일때,  
 \* 비트 값으로 저장하기 위한 변수이다.  
 \*/  
typedef union bits\_format3{  
 struct{  
 uint32\_t disp : 12;  
 uint32\_t e : 1;  
 uint32\_t p : 1;  
 uint32\_t b : 1;  
 uint32\_t x : 1;  
 uint32\_t i : 1;  
 uint32\_t n : 1;  
 uint32\_t opcode : 6;  
 } bits;  
 uint32\_t res;  
}BitsFormat3;  
  
/\*  
 \* format 4 인 Statement 일때,  
 \* 비트 값으로 저장하기 위한 변수이다.  
 \*/  
typedef union bits\_format4{  
 struct{  
 uint32\_t addr: 20;  
 uint32\_t e : 1;  
 uint32\_t p : 1;  
 uint32\_t b : 1;  
 uint32\_t x : 1;  
 uint32\_t i : 1;  
 uint32\_t n : 1;  
 uint32\_t opcode : 6;  
 } bits;  
 uint32\_t res;  
}BitsFormat4;  
  
/\*  
 \* register mnemonic 을 숫자값으로 변환한다  
 \*/  
int reg\_mnemonic\_num (char \*reg\_mnemonic);  
  
/\*  
 \* Statement 의 format 을 검사한다.  
 \* ex)  
 \* is\_format(&stmt, 0) : END, BYTE, WORD, RESB 등인지 검사  
 \* is\_format(&stmt, 1) : format 1 인지 검사  
 \* is\_format(&stmt, 2) : format 2 인지 검사  
 \* is\_format(&stmt, 3) : format 3/4 인지 검사  
 \*/  
bool is\_format(Statement\* stmt, int num);  
  
  
/\*  
 \* Statement 의 format 에 따라서 적절히  
 \* obj\_code, location\_counter, line\_num 등을 .obj, .list 파일에 기록한다  
 \*/  
bool record\_stmt\_for\_pass2(Statement \*stmt,  
 const int \*obj\_code,  
 const int \*location\_counter,  
 int \*r\_lc,  
 const int \*line\_num,  
 FILE \*lst\_fp,  
 FILE \*obj\_fp,  
 char \*\*obj\_buf,  
 char \*\*byte\_buf,  
 char \*\*rec\_head);  
  
/\*  
 \* Default format (WORD, BASE 등등) 의 statement 를  
 \* 적절히 handling 하여 is\_base, obj\_code, base, b\_buf 등의 값을 조정한다.  
 \*/  
bool handling\_format\_default(SymbolTable \*symbol\_table,  
 Statement \*stmt,  
 int \*obj\_code,  
 bool \*is\_base,  
 int \*base,  
 char \*\*b\_buf);  
  
/\*  
 \* format 3/4 의 statement 를  
 \* 적절히 handling 하여 obj\_code 를 조정한다.  
 \*/  
bool handling\_format3(SymbolTable \*symbol\_table,  
 Statement \*stmt,  
 int \*obj\_code,  
 const int \*location\_counter,  
 int \*\*location\_counters,  
 int \*location\_counter\_cnt,  
 int stmt\_size,  
 const bool \*is\_base,  
 const int \*base);  
  
/\*  
 \* format 2 의 statement 를  
 \* 적절히 handling 하여 obj\_code 를 조정한다.  
 \*/  
bool handling\_format2(Statement \*stmt, int \*obj\_code);  
  
/\*  
 \* format 1 의 statement 를  
 \* 적절히 handling 하여 obj\_code 를 조정한다.  
 \*/  
bool handling\_format1(Statement \*stmt, int\* obj\_code);  
  
/\*  
 \* input 을 토크나이징하여 stmt->tokens 와 stmt->token\_cnt 를 조정함  
 \*/  
bool tokenizing\_stmt\_tokens(Statement\* stmt, char\* input);  
  
/\*  
 \* Statement 가 주석인지 아닌지 확인한다  
 \*/  
bool is\_comment\_stmt(Statement\* stmt);  
  
/\*  
 \* Statement 구조체에 이 statement 가 주석이라는 표시를 한다.  
 \*/  
bool mark\_comment\_stmt(Statement\* stmt);  
  
/\*  
 \* +JSUB과 같이 opcode 앞에 +가 붙었는지 확인  
 \*/  
bool is\_plus\_stmt(Statement \*stmt, int str\_idx);  
  
/\*  
 \* Statement 가 +JSUB 과 같이 opcode 앞에 +가 붙은 경우에  
 \* stmt->plus = true 로 설정함  
 \*/  
bool mark\_plus\_true\_or\_false(Statement \*stmt, int str\_idx);  
  
/\*  
 \* Statement 의 format 에 따라서 적절히 location\_counter 값을 증가시킴  
 \*/  
void update\_location\_counter\_by\_format(Statement \*stmt,  
 int \*location\_counter);  
  
/\*  
 \* Statement 의 opcode 의 mnemonic 에 따라서 적절히 location\_counter 를 증가시킴  
 \*/  
bool update\_location\_counter\_by\_mnemonic\_name(Statement \*stmt,  
 int \*location\_counter);  
  
/\*  
 \* Statement 가 plus 이면서 format 3/4 인경우에 location\_counter 를 1 증가 시킴  
 \*/  
bool update\_location\_counter\_by\_plus\_and\_format(Statement \*stmt, int \*location\_counter);  
  
/\*  
 \* pass 2 를 끝낼 시점인지 아닌지 확인함  
 \*/  
bool is\_end\_condition(Statement \*stmt, FILE \*fp);  
  
/\*  
 \* 에러일 경우 이 함수가 실행된다.  
 \* 파일들을 전부 close 하고, 파라미터로 보낸 이름의 파일들을 삭제한다.  
 \* 에러가 난 Line 을 출력해준다.  
 \*/  
bool error\_handling\_pass1or2(Statement \*stmt,  
 FILE \*fp1,  
 FILE \*fp2,  
 FILE \*fp3,  
 char \*rm\_file\_name1,  
 char \*rm\_file\_name2,  
 char \*rm\_file\_name3,  
 int line\_num);  
  
/\*  
 \* symbol 인지 아닌지에 따라서 적절히 handling 한다.  
 \*/  
bool symbol\_handling(OpcodeTable \*opcode\_table,  
 Statement \*stmt,  
 char \*name);  
  
/\*  
 \* 파일로 부터 한줄을 읽어서 Statement 변수 에 적절히 초기화하여 저장한다.  
 \*/  
bool read\_statement(OpcodeTable \*opcode\_table,  
 FILE \*asm\_fp,  
 FILE \*tmp\_fp,  
 Statement \*stmt,  
 bool is\_tmp,  
 int \*location\_counter,  
 int \*stmt\_size);  
  
/\*  
 \* Statement 의 종류에 따라서 적절히 tmp 파일에 기록한다  
 \*/  
bool record\_stmt\_for\_pass1(Statement \*stmt, FILE \*fp, int \*location\_counter, int \*old\_location\_counter);  
  
#endif

## assemble.c

#include "assemble.h"  
  
/\*  
 \* register mnemonic 을 숫자값으로 변환한다  
 \*/  
int reg\_mnemonic\_num (char \*reg\_mnemonic){  
 if (COMPARE\_STRING(reg\_mnemonic, "A")) return 0;  
 if (COMPARE\_STRING(reg\_mnemonic, "X")) return 1;  
 if (COMPARE\_STRING(reg\_mnemonic, "L")) return 2;  
 if (COMPARE\_STRING(reg\_mnemonic, "B")) return 3;  
 if (COMPARE\_STRING(reg\_mnemonic, "S")) return 4;  
 if (COMPARE\_STRING(reg\_mnemonic, "T")) return 5;  
 if (COMPARE\_STRING(reg\_mnemonic, "F")) return 6;  
 if (COMPARE\_STRING(reg\_mnemonic, "PC")) return 8;  
 if (COMPARE\_STRING(reg\_mnemonic, "SW")) return 9;  
 return -1;  
}  
  
/\*  
 \* Statement 의 format 을 검사한다.  
 \* ex)  
 \* is\_format(&stmt, 0) : END, BYTE, WORD, RESB 등인지 검사 ( DEFAULT format 이라 부르겠다)  
 \* is\_format(&stmt, 1) : format 1 인지 검사  
 \* is\_format(&stmt, 2) : format 2 인지 검사  
 \* is\_format(&stmt, 3) : format 3/4 인지 검사  
 \*/  
bool is\_format(Statement\* stmt, int num){  
 if(num == 1) {  
 return stmt->opcode->format == OP\_FORMAT\_1;  
 }else if(num == 2){  
 return (stmt->opcode->format == OP\_FORMAT\_2\_ONE\_N ||  
 stmt->opcode->format == OP\_FORMAT\_2\_REG\_N ||  
 stmt->opcode->format == OP\_FORMAT\_2\_ONE\_REG ||  
 stmt->opcode->format == OP\_FORMAT\_2\_GEN  
 );  
 }else if(num == 3){  
 return (stmt->opcode->format == OP\_FORMAT\_3\_4\_GEN ||  
 stmt->opcode->format == OP\_FORMAT\_3\_4\_NO\_OPERAND  
 );  
 }else if(num == 0){  
 return stmt->opcode->format == OP\_DEFAULT;  
 }else{  
 return false;  
 }  
}  
  
/\*  
 \* Statement 의 format 에 따라서 적절히  
 \* obj\_code, location\_counter, line\_num 등을 .obj, .list 파일에 기록한다  
 \*/  
bool record\_stmt\_for\_pass2(Statement \*stmt,  
 const int \*obj\_code,  
 const int \*location\_counter,  
 int \*r\_lc,  
 const int \*line\_num,  
 FILE \*lst\_fp,  
 FILE \*obj\_fp,  
 char \*\*obj\_buf,  
 char \*\*byte\_buf,  
 char \*\*rec\_head) {  
 const char \*format;  
 if (is\_format(stmt, 1)){  
 format = "%d\t%04X%-30s%02X\n";  
 if(\*location\_counter + 1 > \*r\_lc + 30){  
 snprintf(\*rec\_head, 30, "T%06X%02X", \*r\_lc, (uint8\_t) strlen (\*obj\_buf) / 2);  
 fprintf (obj\_fp, "%s%s\n", \*rec\_head, \*obj\_buf);  
 \*r\_lc = \*location\_counter;  
 (\*obj\_buf)[0] = '\0';  
 }  
 sprintf ((\*obj\_buf) + strlen (\*obj\_buf), "%02X", (\*obj\_code));  
 }else if (is\_format(stmt, 2)){  
 format = "%d\t%04X%-30s%04X\n";  
 if(\*location\_counter + 2 > \*r\_lc + 30){  
 snprintf(\*rec\_head, 30, "T%06X%02X", \*r\_lc, (uint8\_t) strlen (\*obj\_buf) / 2);  
 fprintf (obj\_fp, "%s%s\n", \*rec\_head, \*obj\_buf);  
 \*r\_lc = \*location\_counter;  
 (\*obj\_buf)[0] = '\0';  
 }  
 sprintf ((\*obj\_buf) + strlen (\*obj\_buf), "%04X", (\*obj\_code));  
 }else if (is\_format(stmt, 3)){  
 if (stmt->plus){  
 format = "%d\t%04X%-30s%08X\n";  
 if(\*location\_counter + 4 > \*r\_lc + 30){  
 snprintf(\*rec\_head, 30, "T%06X%02X", \*r\_lc, (uint8\_t) strlen (\*obj\_buf) / 2);  
 fprintf (obj\_fp, "%s%s\n", \*rec\_head, \*obj\_buf);  
 \*r\_lc = \*location\_counter;  
 (\*obj\_buf)[0] = '\0';  
 }  
 sprintf ((\*obj\_buf) + strlen (\*obj\_buf), "%08X", (\*obj\_code));  
 }  
 else{  
 format = "%d\t%04X%-30s%06X\n";  
 if(\*location\_counter + 3 > \*r\_lc + 30){  
 snprintf(\*rec\_head, 30, "T%06X%02X", \*r\_lc, (uint8\_t) strlen (\*obj\_buf) / 2);  
 fprintf (obj\_fp, "%s%s\n", \*rec\_head, \*obj\_buf);  
 \*r\_lc = \*location\_counter;  
 (\*obj\_buf)[0] = '\0';  
 }  
 sprintf ((\*obj\_buf) + strlen (\*obj\_buf), "%06X", (\*obj\_code));  
 }  
 }else if (COMPARE\_STRING(stmt->opcode->mnemonic\_name,"BYTE")){  
 fprintf (lst\_fp, "%d\t%04X%-30s%s\n", (\*line\_num), (\*location\_counter), stmt->raw\_input, (\*byte\_buf));  
 format = NULL;  
 if(\*location\_counter + (int)strlen(\*byte\_buf) > \*r\_lc + 30){  
 snprintf(\*rec\_head, 30, "T%06X%02X", \*r\_lc, (uint8\_t) strlen (\*obj\_buf) / 2);  
 fprintf (obj\_fp, "%s%s\n", \*rec\_head, \*obj\_buf);  
 \*r\_lc = \*location\_counter;  
 (\*obj\_buf)[0] = '\0';  
 }  
 sprintf ((\*obj\_buf) + strlen (\*obj\_buf), "%s", (\*byte\_buf));  
 }else if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "WORD")){  
 format = "%d\t%04X%-30s%06X\n";  
 if(\*location\_counter + 3 > \*r\_lc + 30){  
 snprintf(\*rec\_head, 30, "T%06X%02X", \*r\_lc, (uint8\_t) strlen (\*obj\_buf) / 2);  
 fprintf (obj\_fp, "%s%s\n", \*rec\_head, \*obj\_buf);  
 \*r\_lc = \*location\_counter;  
 (\*obj\_buf)[0] = '\0';  
 }  
 sprintf ((\*obj\_buf) + strlen (\*obj\_buf), "%06X", (\*obj\_code));  
 }else{  
 fprintf (lst\_fp, "%d\t%s\n",(\*line\_num), stmt->raw\_input);  
 format = NULL;  
 }  
  
 if (format)  
 fprintf (lst\_fp, format, (\*line\_num), (\*location\_counter), stmt->raw\_input, (\*obj\_code));  
 return true;  
}  
  
/\*  
 \* Default format (WORD, BASE 등등) 의 statement 를  
 \* 적절히 handling 하여 is\_base, obj\_code, base, b\_buf 등의 값을 조정한다.  
 \*/  
bool handling\_format\_default(SymbolTable \*symbol\_table,  
 Statement \*stmt,  
 int \*obj\_code,  
 bool \*is\_base,  
 int \*base,  
 char \*\*b\_buf) {  
 if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "BASE")){  
 if(stmt->token\_cnt != 1) return false;  
 Symbol\* symb = find\_symbol\_by\_name(symbol\_table, stmt->tokens[0]);  
 if(!symb) return false;  
 \*is\_base = true;  
 \*base = symb->location\_counter;  
 return true;  
 }  
 if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "NOBASE")){  
 (\*is\_base) = false;  
 return true;  
 }  
 if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "BYTE")){  
 if(stmt->token\_cnt != 1) return false;  
  
 const char \*operand = stmt->tokens[0];  
 int len = strlen (operand);  
  
 if (len > 500) return false;  
  
 if (operand[0] == 'C'){  
 int idx = 0;  
 for (int i = 2; i < len-1; ++i){  
 unsigned char ch = operand[i];  
 uint8\_t val[2] = { ch / 16 , ch % 16 };  
 for (int j = 0; j < 2; ++j, ++idx){  
 if (val[j] <= 9)  
 (\*b\_buf)[idx] = val[j] + '0';  
 else  
 (\*b\_buf)[idx] = val[j] - 10 + 'A';  
 }  
 }  
 (\*b\_buf)[idx] = '\0';  
 }  
 else if (operand[0] == 'X'){  
 int i;  
 for (i = 2; i < len-1; ++i) (\*b\_buf)[i-2] = operand[i];  
 (\*b\_buf)[i-2] = '\0';  
 }  
 else {  
 return false;  
 }  
 return true;  
 }else if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "WORD")){  
 if(stmt->token\_cnt != 1) return false;  
 int val = strtol (stmt->tokens[0], NULL, 10);  
 \*obj\_code = val;  
  
 return true;  
 }  
  
 return true;  
}  
  
/\*  
 \* format 3/4 의 statement 를  
 \* 적절히 handling 하여 obj\_code 를 조정한다.  
 \*/  
bool handling\_format3(SymbolTable \*symbol\_table,  
 Statement \*stmt,  
 int \*obj\_code,  
 const int \*location\_counter,  
 int \*\*location\_counters,  
 int \*location\_counter\_cnt,  
 int stmt\_size,  
 const bool \*is\_base,  
 const int \*base) {  
 BitsFormat3 bitsFormat3; bitsFormat3.res = 0;  
 BitsFormat4 bitsFormat4;  
  
 if(stmt->plus){  
 bitsFormat4.bits.opcode = (stmt->opcode->value >> 2);  
 bitsFormat4.bits.e = stmt->plus;  
 }else{  
 bitsFormat3.bits.opcode = (stmt->opcode->value >> 2);  
 bitsFormat3.bits.e = stmt->plus;  
 }  
  
 if (stmt->opcode->format == OP\_FORMAT\_3\_4\_NO\_OPERAND){  
 if (stmt->token\_cnt != 0) return false;  
 if(stmt->plus){  
 bitsFormat4.bits.n = 1;  
 bitsFormat4.bits.i = 1;  
 bitsFormat4.bits.x = 0;  
 bitsFormat4.bits.b = 0;  
 bitsFormat4.bits.p = 0;  
 bitsFormat4.bits.addr = 0;  
 }else {  
 bitsFormat3.bits.n = 1;  
 bitsFormat3.bits.i = 1;  
 bitsFormat3.bits.x = 0;  
 bitsFormat3.bits.b = 0;  
 bitsFormat3.bits.p = 0;  
 bitsFormat3.bits.disp = 0;  
 }  
 } else {  
 if (stmt->token\_cnt > 2 || stmt->token\_cnt < 1) return false;  
  
 /\* Index Mode \*/  
 if (stmt->token\_cnt == 2){  
 if (strcmp (stmt->tokens[1], "X") != 0) return false;  
 if(stmt->plus) bitsFormat4.bits.x = 1;  
 else bitsFormat3.bits.x = 1;  
 }else{  
 if(stmt->plus) bitsFormat4.bits.x = 0;  
 else bitsFormat3.bits.x = 0;  
 }  
  
 const char \*operand = stmt->tokens[0];  
  
 /\* Addressing mode \*/  
 bool operand\_is\_constant = false;  
  
 // Immediate addressing  
 if (operand[0] == '#'){  
 if(stmt->plus){  
 bitsFormat4.bits.n = 0;  
 bitsFormat4.bits.i = 1;  
 }else{  
 bitsFormat3.bits.n = 0;  
 bitsFormat3.bits.i = 1;  
 }  
 if ('0' <= operand[1] && operand[1] <= '9')  
 operand\_is\_constant = true;  
 ++operand;  
 }  
 // Indirect addressing  
 else if (operand[0] == '@')  
 {  
 if(stmt->plus){  
 bitsFormat4.bits.n = 1;  
 bitsFormat4.bits.i = 0;  
 }else{  
 bitsFormat3.bits.n = 1;  
 bitsFormat3.bits.i = 0;  
 }  
 ++operand;  
 }  
 // simple addressing  
 else{  
 if(stmt->plus){  
 bitsFormat4.bits.n = 1;  
 bitsFormat4.bits.i = 1;  
 }else{  
 bitsFormat3.bits.n = 1;  
 bitsFormat3.bits.i = 1;  
 }  
 }  
  
 uint32\_t operand\_value;  
  
 if (operand\_is\_constant){  
 operand\_value = strtol (operand, NULL, 10);  
 }  
 else{  
 const Symbol \*symb = find\_symbol\_by\_name(  
 symbol\_table,  
 (char\*)operand);  
 if(!symb){  
 return false;  
 }  
 operand\_value = symb->location\_counter;  
 }  
  
 if (stmt->plus){  
 bitsFormat4.bits.b = 0;  
 bitsFormat4.bits.p = 0;  
 bitsFormat4.bits.addr = operand\_value;  
 if (!operand\_is\_constant)  
 (\*location\_counters)[(\*location\_counter\_cnt)++] = (\*location\_counter)+1;  
 } else if (operand\_is\_constant){  
 bitsFormat3.bits.b = 0;  
 bitsFormat3.bits.p = 0;  
 bitsFormat3.bits.disp = operand\_value;  
 } else {  
 /\* Displacement \*/  
 int32\_t disp;  
  
 /\* PC relative \*/  
 const size\_t PC = (\*location\_counter) + stmt\_size;  
  
 disp = operand\_value - PC;  
  
 if (-(1 << 11) <= disp && disp < (1 << 11)){  
 // PC relative O case  
 bitsFormat3.bits.b = 0;  
 bitsFormat3.bits.p = 1;  
 bitsFormat3.bits.disp = disp;  
 }  
 else{  
 // PC relative X case  
  
 /\* Base relative check \*/  
 if ((\*is\_base) == false){  
 // BASE X => error  
 return false;  
 }  
  
 disp = operand\_value - (\*base);  
  
 if (0 <= disp && disp < (1 << 12)){  
 // BASE relative O  
 bitsFormat3.bits.b = 1;  
 bitsFormat3.bits.p = 0;  
 bitsFormat3.bits.disp = disp;  
 }  
 else{  
 // Base relative X  
 return false;  
 }  
  
 }  
 }  
 }  
  
 if (stmt->plus)  
 \*obj\_code = bitsFormat4.res;  
 else  
 \*obj\_code = bitsFormat3.res;  
  
 return true;  
}  
  
/\*  
 \* format 3/4 의 statement 를  
 \* 적절히 handling 하여 obj\_code 를 조정한다.  
 \*/  
bool handling\_format2(Statement \*stmt, int \*obj\_code) {  
 BitsFormat2 bits;  
 if(stmt->opcode->format == OP\_FORMAT\_2\_GEN){  
 if(stmt->token\_cnt != 2) return false;  
 int reg\_no\_1, reg\_no\_2;  
  
 reg\_no\_1 = reg\_mnemonic\_num (stmt->tokens[0]);  
 reg\_no\_2 = reg\_mnemonic\_num (stmt->tokens[1]);  
  
 if(reg\_no\_1 == -1 || reg\_no\_2 == -1) return false;  
  
 bits.bits.opcode = stmt->opcode->value;  
 bits.bits.r1 = reg\_no\_1;  
 bits.bits.r2 = reg\_no\_2;  
  
 } else if(stmt->opcode->format == OP\_FORMAT\_2\_ONE\_REG){  
 if(stmt->token\_cnt != 1) return false;  
  
 int reg\_no = reg\_mnemonic\_num (stmt->tokens[0]);  
 if(reg\_no == -1) return false;  
  
 bits.bits.opcode = stmt->opcode->value;  
 bits.bits.r1 = reg\_no;  
 bits.bits.r2 = 0;  
  
 } else if(stmt->opcode->format == OP\_FORMAT\_2\_REG\_N){  
 if(stmt->token\_cnt != 2) return false;  
  
 int reg\_no = reg\_mnemonic\_num(stmt->tokens[0]);  
 char \*endptr;  
 long int n = strtol (stmt->tokens[1], &endptr, 16);  
  
 if(reg\_no == -1 || \*endptr != '\0' || n > 0xF || n < 0)  
 return false;  
 bits.bits.opcode = stmt->opcode->value;  
 bits.bits.r1 = reg\_no;  
 bits.bits.r2 = n;  
 } else if(stmt->opcode->format == OP\_FORMAT\_2\_ONE\_N){  
  
 if(stmt->token\_cnt != 1) return false;  
 char \*endptr;  
 long int n = strtol (stmt->tokens[0], &endptr, 16);  
  
 if (\*endptr != '\0' || n > 0xF || n < 0) return false;  
  
 bits.bits.opcode = stmt->opcode->value;  
 bits.bits.r1 = n;  
 bits.bits.r2 = 0;  
 } else{  
 assert(false);  
 }  
 \*obj\_code = bits.res;  
  
 return true;  
}  
  
/\*  
 \* format 1 의 statement 를  
 \* 적절히 handling 하여 obj\_code 를 조정한다.  
 \*/  
bool handling\_format1(Statement\* stmt, int\* obj\_code){  
 if (stmt->token\_cnt != 0) return false;  
 \*obj\_code = stmt->opcode->value;  
  
 return true;  
}  
  
/\*  
 \* input 을 토크나이징하여 stmt->tokens 와 stmt->token\_cnt 를 조정함  
 \*/  
bool tokenizing\_stmt\_tokens(Statement\* stmt, char\* input){  
 stmt->token\_cnt = 0;  
 stmt->tokens[stmt->token\_cnt] = strtok (input, " ,\t\n");  
 while (stmt->token\_cnt <= 15 && stmt->tokens[stmt->token\_cnt])  
 stmt->tokens[++stmt->token\_cnt] = strtok (NULL, " ,\t\n");  
  
 return true;  
}  
  
/\*  
 \* Statement 가 주석인지 아닌지 확인한다  
 \*/  
bool is\_comment\_stmt(Statement\* stmt){  
 if(stmt->tokens[0][0] != '.') return false;  
 if(stmt->comment) return true;  
 return true;  
}  
  
/\*  
 \* Statement 구조체에 이 statement 가 주석이라는 표시를 한다.  
 \*/  
bool mark\_comment\_stmt(Statement\* stmt){  
 assert(is\_comment\_stmt(stmt));  
  
 stmt->comment = true;  
 stmt->opcode = NULL;  
  
 return true;  
}  
  
/\*  
 \* +JSUB 과 같이 opcode 앞에 +가 붙었는지 확인  
 \*/  
bool is\_plus\_stmt(Statement \*stmt, int str\_idx) {  
 if(stmt->tokens[str\_idx][0] != '+') return false;  
  
 return true;  
}  
  
/\*  
 \* Statement 가 +JSUB 과 같이 opcode 앞에 +가 붙은 경우에  
 \* stmt->plus = true 로 설정함  
 \*/  
bool mark\_plus\_true\_or\_false(Statement \*stmt, int str\_idx) {  
 if(is\_plus\_stmt(stmt, str\_idx))  
 stmt->plus = true;  
 else  
 stmt->plus = false;  
  
 return true;  
}  
  
/\*  
 \* Statement 의 format 에 따라서 적절히 location\_counter 값을 증가시킴  
 \*/  
void update\_location\_counter\_by\_format(Statement \*stmt, int \*location\_counter) {  
 if (is\_format(stmt, 1)){  
 \*location\_counter += 1;  
 return;  
 }  
 else if (is\_format(stmt, 2)){  
 \*location\_counter += 2;  
 return;  
 }  
 else if (is\_format(stmt, 3)){  
 \*location\_counter += 3;  
 }  
}  
  
/\*  
 \* Statement 의 opcode 의 mnemonic 에 따라서 적절히 location\_counter 를 증가시킴  
 \*/  
bool update\_location\_counter\_by\_mnemonic\_name(Statement \*stmt,  
 int \*location\_counter){  
 int len, b;  
 if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "BYTE")){  
 if (stmt->token\_cnt != 1) return false;  
  
 const char \*operand = stmt->tokens[0];  
  
 if (operand[1] != '\'') return false;  
  
 len = strlen (operand);  
  
 if (operand[0] == 'C') b = len - 3;  
 else if (operand[0] == 'X') b = (len - 3) / 2;  
 else return false;  
  
 if (operand[len-1] != '\'') return false;  
  
 \*location\_counter += b;  
 } else if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "WORD")){  
 if (stmt->token\_cnt != 1) return false;  
 \*location\_counter += 3;  
 } else if (COMPARE\_STRING(stmt->opcode->mnemonic\_name,"RESB")){  
 if (stmt->token\_cnt != 1) return false;  
 int cnt = strtol (stmt->tokens[0], NULL, 10);  
 \*location\_counter += cnt;  
 } else if (COMPARE\_STRING(stmt->opcode->mnemonic\_name, "RESW")){  
 if (stmt->token\_cnt != 1) return false;  
 int cnt = strtol (stmt->tokens[0], NULL, 10);  
 \*location\_counter += cnt \* 3;  
 }  
  
 return true;  
}  
  
/\*  
 \* Statement 가 plus 이면서 format 3/4 인경우에 location\_counter 를 1 증가 시킴  
 \*/  
bool update\_location\_counter\_by\_plus\_and\_format(Statement \*stmt, int \*location\_counter){  
 if (stmt->plus){  
 if (is\_format(stmt, 3)) ++(\*location\_counter);  
 else return false;  
 }  
  
 return true;  
}  
  
/\*  
 \* pass 2 를 끝낼 시점인지 아닌지 확인함  
 \*/  
bool is\_end\_condition(Statement \*stmt, FILE \*fp) {  
 if (feof (fp) != 0)  
 return true;  
 else if (!stmt->comment && COMPARE\_STRING(stmt->opcode->mnemonic\_name,"END"))  
 return true;  
  
 return false;  
}  
  
/\*  
 \* 에러일 경우 이 함수가 실행된다.  
 \* 파일들을 전부 close 하고, 파라미터로 보낸 이름의 파일들을 삭제한다.  
 \* 에러가 난 Line 을 출력해준다.  
 \*/  
bool error\_handling\_pass1or2(Statement \*stmt,  
 FILE \*fp1,  
 FILE \*fp2,  
 FILE \*fp3,  
 char \*rm\_file\_name1,  
 char \*rm\_file\_name2,  
 char \*rm\_file\_name3,  
 int line\_num){  
 if(fp1) fclose(fp1);  
 if(fp2) fclose(fp2);  
 if(fp3) fclose(fp3);  
  
 if(rm\_file\_name1) remove(rm\_file\_name1);  
 if(rm\_file\_name2) remove(rm\_file\_name2);  
 if(rm\_file\_name3) remove(rm\_file\_name3);  
  
 if(line\_num != -1 && stmt && stmt->raw\_input) {  
 fprintf(stderr, "[ERROR] Line %d: %s \n", line\_num, stmt->raw\_input);  
 }  
  
 return true;  
}  
  
/\*  
 \* symbol 인지 아닌지에 따라서 적절히 handling 한다.  
 \*/  
bool symbol\_handling(OpcodeTable \*opcode\_table,  
 Statement \*stmt,  
 char \*name) {  
  
 Opcode\* opc = find\_opcode\_by\_name(opcode\_table, name);  
  
 int offset;  
  
 if(opc){  
 stmt->raw\_symbol = NULL;  
 stmt->opcode = opc;  
 offset = 1;  
 } else {  
 if (stmt->token\_cnt <= 1) return false;  
  
 mark\_plus\_true\_or\_false(stmt, 1);  
 if(stmt->plus) name = &stmt->tokens[1][1];  
 else name = stmt->tokens[1];  
  
 opc = find\_opcode\_by\_name (opcode\_table, name);  
 if (!opc) return false;  
  
 offset = 2;  
 stmt->opcode = opc;  
 stmt->raw\_symbol = stmt->tokens[0];  
 }  
 for (size\_t i = offset; i < (size\_t)stmt->token\_cnt; ++i)  
 stmt->tokens[i - offset] = stmt->tokens[i];  
  
 stmt->token\_cnt -= offset;  
  
 return true;  
}  
  
/\*  
 \* 파일로 부터 한줄을 읽어서 Statement 변수 에 적절히 초기화하여 저장한다.  
 \*/  
bool read\_statement(OpcodeTable \*opcode\_table,  
 FILE \*asm\_fp,  
 FILE \*tmp\_fp,  
 Statement \*stmt,  
 bool is\_tmp,  
 int \*location\_counter,  
 int \*stmt\_size) {  
 FILE\* fp;  
 static char raw\_input[220];  
 static char tmp\_input[200];  
 int length = 0;  
 char \*op\_tok;  
 int offset, i;  
  
 if(is\_tmp) fp = tmp\_fp;  
 else fp = asm\_fp;  
  
 if(!fgets(raw\_input, 220, fp)){  
// fprintf(stderr,"[DEBUG]");  
 return false;  
 }  
  
 length = strlen(raw\_input);  
 if(!feof(fp) && raw\_input[length - 1] != '\n') return false;  
  
 raw\_input[length - 1] = '\0';  
  
// printf("%s\n", raw\_input);  
  
 if(is\_tmp){  
 sscanf (raw\_input, "%X\t%X%n", location\_counter, stmt\_size, &offset);  
 for (i = 0; raw\_input[offset + i]; i++)  
 raw\_input[i] = raw\_input[offset + i];  
 raw\_input[i] = 0;  
 }  
  
 strncpy (tmp\_input, raw\_input, 200);  
 stmt->raw\_input = raw\_input;  
  
 tokenizing\_stmt\_tokens(stmt, tmp\_input);  
  
 if (is\_comment\_stmt(stmt) && mark\_comment\_stmt(stmt)) return true;  
  
 stmt->comment = false;  
  
 if (stmt->token\_cnt == 0 || stmt->token\_cnt > MAX\_TOKENS\_LENGTH)  
 return false;  
  
 mark\_plus\_true\_or\_false(stmt, 0);  
  
 if(stmt->plus) op\_tok = &stmt->tokens[0][1];  
 else op\_tok = stmt->tokens[0];  
  
 if(!symbol\_handling(opcode\_table,  
 stmt,  
 op\_tok)) return false;  
  
 return true;  
}  
  
/\*  
 \* Statement 의 종류에 따라서 적절히 tmp 파일에 기록한다  
 \*/  
bool record\_stmt\_for\_pass1(Statement \*stmt,  
 FILE \*fp,  
 int \*location\_counter,  
 int \*old\_location\_counter) {  
 if(is\_comment\_stmt(stmt)){  
 fprintf(fp, "%04X\t0\t%s\n", \*location\_counter, stmt->raw\_input);  
 return true;  
 }  
 fprintf (fp, "%04X\t%X\t%s\n",  
 (unsigned int) \*old\_location\_counter,  
 (unsigned int)(\*location\_counter - \*old\_location\_counter),  
 stmt->raw\_input);  
 return true;  
}

## debug.c

#include "debug.h"

#include <stdint.h>

/\*

\* Struct or Enum Declarations and Definition

\*/

// Instruction 정보를 저장함.

typedef struct instruction{

bool extend;

unsigned char opcode;

struct {

union{

struct{

uint16\_t r2 : 4;

uint16\_t r1 : 4;

uint16\_t opcode : 8;

} p2;

struct{

uint32\_t address: 12;

uint32\_t e : 1;

uint32\_t p : 1;

uint32\_t b : 1;

uint32\_t x : 1;

uint32\_t i : 1;

uint32\_t n : 1;

uint32\_t opcode : 6;

} p3;

struct{

uint32\_t address: 20;

uint32\_t e : 1;

uint32\_t p : 1;

uint32\_t b : 1;

uint32\_t x : 1;

uint32\_t i : 1;

uint32\_t n : 1;

uint32\_t opcode : 6;

} p4;

uint32\_t val;

} param;

bool extend;

} param;

} Instruction;

// Operator 이름과 opcode 를 매핑

typedef enum {

LDA = 0x00,

LDB = 0x68,

LDT = 0x74,

LDX = 0x04,

LDCH = 0x50,

STA = 0x0C,

STL = 0x14,

STX = 0x10,

STCH = 0x54,

JSUB = 0x48,

JEQ = 0x30,

JGT = 0x34,

JLT = 0x38,

J = 0x3C,

COMP = 0x28,

TD = 0xE0,

RD = 0xD8,

RSUB = 0x4C,

WD = 0xDC,

CLEAR = 0xB4,

COMPR = 0xA0,

TIXR = 0xB8

} Operator;

// Addressing mode

typedef enum {

ENUM\_IMMEDIATE\_ADDRESSING,

ENUM\_SIMPLE\_ADDRESSING,

ENUM\_INDIRECT\_ADDRESSING,

ENUM\_ADDRESSING\_ERROR

} ADDRESSING\_MODE;

/\*

\* Static Function Declarations

\*/

/\* loader\_linker 함수 구현을 위한 함수들 \*/

// pass1

static bool loader\_linker\_pass1(Debugger \*debugger);

// pass2

static bool loader\_linker\_pass2(Debugger \*debugger, Memories \*memories);

// pass1 sub function

static bool loader\_linker\_pass1\_one(Debugger \*debugger, int file\_num, int \*csaddr);

// pass2 sub function

static bool loader\_linker\_pass2\_one(Debugger \*debugger, Memories \*memories, int file\_num , int \*csaddr);

// 생성자 함수

static LoadInfoList\* construct\_load\_info\_list();

// load 된 정보를 출력함.

static void print\_load\_infos(LoadInfoList \*load\_infos);

/\* run 함수를 구현을 위한 함수들 \*/

// 레지스터 상태를 출력함.

static void print\_registers(Registers\* registers);

// 명령을 실행함.

static bool execute\_operator(Debugger \*debugger, Memories \*memories, Instruction \*instruction);

// TA 계산함.

static uint32\_t calculate\_TA(Instruction\* instruction, Registers\* registers);

// Addressing Mode 계산함

static ADDRESSING\_MODE calculate\_addressing\_mode(Instruction\* instruction, bool jump\_op);

// reg\_id 로 레지스터를 찾아서 리턴함.

static uint32\_t \*get\_reg\_by\_id(Registers \*registers, int reg\_id);

// BP 를 핸들링함.

static bool handling\_bp(Debugger \*debugger, int instruction\_size);

// 메모리로 부터 값을 가져옴.

static bool load\_from\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t \*value,

size\_t bytes, bool jump\_op);

// 메모리에 값을 저장함.

static bool store\_to\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t value,

size\_t bytes);

// 레지터로 부터 값을 가져옴.

static bool load\_from\_register(Debugger \*debugger, int reg\_id, uint32\_t \*val);

// 레지스터에 값을 저장함.

static bool store\_to\_register(Debugger \*debugger, int reg\_id, uint32\_t val);

/\*

\* 함수 상세 설명(주석)은 declaration 파트 참고

\* Function Definitions (public)

\*/

Debugger\* construct\_debugger(){

Debugger\* debugger = (Debugger\*)malloc(sizeof(\*debugger));

debugger->start\_address = 0; // default: 0

debugger->bp\_count = 0; // default: 0

debugger->run\_count = 0; // default: 0

debugger->registers = construct\_registers();

debugger->previous\_bp = -1;

debugger->end\_address = MAX\_BP\_NUM - 1;

return debugger;

}

bool destroy\_debugger(Debugger\*\* debugger){

destroy\_registers(&((\*debugger)->registers));

free(\*debugger);

return true;

}

Registers\* construct\_registers(){

Registers\* registers = (Registers\*)malloc(sizeof(\*registers));

registers->A = 0;

registers->B = 0;

registers->L = 0;

registers->PC = 0;

registers->S = 0;

registers->T = 0;

registers->X = 0;

registers->SW = 0;

return registers;

}

bool destroy\_registers(Registers\*\* registers){

free(\*registers);

return true;

}

void reset\_registers(Registers\* registers){

memset (registers, 0, sizeof (\*registers));

registers->L = 0x00FFFFFF;

}

bool loader\_linker(Debugger \*debugger, Memories \*memories){

assert(debugger->filenames[0]);

assert(debugger->file\_count > 0);

bool status;

status = loader\_linker\_pass1(debugger);

if(!status) return false;

status = loader\_linker\_pass2(debugger, memories);

if(!status) return false;

print\_load\_infos(debugger->load\_infos);

destroy\_symbol\_table(&(debugger->estab));

return true;

}

bool run(Debugger \*debugger, Memories \*memories){

Registers\* registers = debugger->registers;

uint32\_t tmp;

bool is\_continue = debugger->is\_running;

if(!debugger->is\_running){

tmp = registers->PC;

reset\_registers(registers);

registers->PC = tmp;

}

debugger->is\_running = true;

while (registers->PC != 0x00FFFFFF){

uint8\_t opcode;

uint32\_t instruction\_val = 0;

uint8\_t memory\_val;

int instruction\_size;

enum op\_format format;

memory\_val = (uint8\_t)memories->data[registers->PC].value;

instruction\_val = memory\_val;

opcode = memory\_val & 0xFC;

format = op\_format\_by\_op\_num(opcode);

switch (format){

case OP\_FORMAT\_1:

instruction\_size = 1;

break;

case OP\_FORMAT\_2\_GEN:

memory\_val = (unsigned char)memories->data[registers->PC + 1].value;

instruction\_size = 2;

instruction\_val = (instruction\_val << 8) + memory\_val;

break;

case OP\_FORMAT\_3\_4\_GEN:

memory\_val = (unsigned char)memories->data[registers->PC + 1].value;

instruction\_val = (instruction\_val << 8) + memory\_val;

memory\_val = (unsigned char)memories->data[registers->PC + 2].value;

instruction\_val = (instruction\_val << 8) + memory\_val;

instruction\_size = 3;

if (instruction\_val & (1 << 12)){

memory\_val = (unsigned char)memories->data[registers->PC + 3].value;

instruction\_val = (instruction\_val << 8) + memory\_val;

++instruction\_size;

}

break;

default:

return false;

}

if (!is\_continue){

bool is\_break = handling\_bp(debugger, instruction\_size);

if(is\_break) return true;

}

Instruction instruction;

instruction.opcode = opcode;

instruction.extend = (instruction\_size == 4);

instruction.param.param.val = instruction\_val;

bool status;

registers->PC += instruction\_size;

status = execute\_operator(debugger, memories, &instruction);

if(!status) return false;

is\_continue = false;

}

registers->PC = debugger->start\_address + (uint32\_t)debugger->load\_infos->list[0].length;

print\_registers(registers);

debugger->previous\_bp = -1;

printf("End program\n");

debugger->is\_running = false;

return true;

}

/\*

\* 함수 상세 설명(주석)은 declaration 파트 참고

\* Static Function Definitions

\*/

static bool loader\_linker\_pass1(Debugger \*debugger) {

debugger->estab = construct\_symbol\_table();

debugger->load\_infos = construct\_load\_info\_list();

int csaddr = (int)debugger->start\_address;

for(int i = 0; i < debugger->file\_count; i++){

int status;

status = loader\_linker\_pass1\_one(debugger, i, &csaddr);

if(!status) return false;

}

return true;

}

static bool loader\_linker\_pass1\_one(Debugger \*debugger, int file\_num, int \*csaddr) {

assert(file\_num >= 0 && file\_num <= 2);

assert(debugger->filenames[file\_num]);

FILE \*fp;

char buf[1010];

bool is\_header = false;

int base\_address = \*csaddr;

fp = fopen (debugger->filenames[file\_num], "rt");

if(!fp){

fprintf(stderr, "[ERROR] Can't file open\n");

return false;

}

while (fgets (buf, 1000, fp) != NULL){

if(buf[0] != 'H' && buf[0] != 'D') continue;

if (buf[0] == 'H'){

if (is\_header) return false;

else is\_header = true;

int tmp;

LoadInfoList\* load\_infos = debugger->load\_infos;

load\_infos->list[load\_infos->count].type = INFO\_TYPE\_CONTROL\_SECTION;

sscanf (buf, "H%6s%06X%06X",

load\_infos->list[load\_infos->count].name,

&tmp, &load\_infos->list[load\_infos->count].length);

load\_infos->list[load\_infos->count].addr = base\_address;

\*csaddr += load\_infos->list[load\_infos->count].length;

load\_infos->count += 1;

continue;

}

if (buf[0] == 'D'){

int rec\_len = strlen(buf);

rec\_len -= 2;

if (rec\_len % 12 != 0) return false;

LoadInfoList\* load\_infos = debugger->load\_infos;

for (int i = 0; i < rec\_len / 12; i++){

int offset;

Symbol\* symbol = construct\_symbol();

load\_infos->list[load\_infos->count].type = INFO\_TYPE\_SYMBOL;

sscanf (buf + 1 + i \* 12,

"%6s%06X",

load\_infos->list[load\_infos->count].name,

&offset);

load\_infos->list[load\_infos->count].addr = base\_address + offset;

strncpy (symbol->label,

load\_infos->list[load\_infos->count].name,

10);

symbol->location\_counter = load\_infos->list[load\_infos->count].addr;

insert\_symbol(debugger->estab, symbol);

load\_infos->count += 1;

}

continue;

}

}

if(!is\_header){

fprintf(stderr, "[ERROR] Object file is wrong\n");

return false;

}

fclose(fp);

return true;

}

static bool loader\_linker\_pass2(Debugger \*debugger, Memories \*memories){

int csaddr = (int)debugger->start\_address;

for(int i = 0; i < debugger->file\_count; i++){

int status;

status = loader\_linker\_pass2\_one(debugger, memories, i, &csaddr);

if(!status) return false;

}

return true;

}

static bool loader\_linker\_pass2\_one(Debugger \*debugger, Memories \*memories, int file\_num , int \*csaddr){

assert(file\_num >= 0 && file\_num <= 2);

assert(debugger->filenames[file\_num]);

FILE \*fp;

fp = fopen (debugger->filenames[file\_num], "rt");

if(!fp){

fprintf(stderr, "[ERROR] Can't file open\n");

return false;

}

char ch\_stat[128] = {};

ch\_stat['H'] = 0;

ch\_stat['R'] = 0;

ch\_stat['T'] = 1;

ch\_stat['M'] = 2;

char buf[1001];

int state = 0;

int num\_to\_address[256];

bool num\_validation[256] = { false, };

int base\_addr = \*csaddr;

num\_validation[1] = true;

num\_to\_address[1] = base\_addr;

while (fgets (buf, 1000, fp) != NULL){

if(buf[0] != 'H' && buf[0] != 'R' && buf[0] != 'T' && buf[0] != 'M')

continue;

if(state > ch\_stat[(unsigned char)buf[0]]) return false;

state = ch\_stat[(unsigned char)buf[0]];

if (buf[0] == 'H'){

char name[8];

unsigned int hex, length;

sscanf (buf, "H%6s%06X%06X", name, &hex, &length);

\*csaddr += length;

continue;

}

if (buf[0] == 'R'){

int num;

char label[8];

int cnt;

char \*ptr = buf + 1;

while (sscanf (ptr, "%02X%6s%n", &num, label, &cnt) > 0){

ptr += cnt;

const Symbol \*symbol = find\_symbol\_by\_name (debugger->estab, label);

if (!symbol) return false;

num\_to\_address[num] = symbol->location\_counter;

num\_validation[num] = true;

}

continue;

}

if (buf[0] == 'T'){

int start\_offset;

int length;

sscanf (buf, "T%06X%02X", &start\_offset, &length);

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

unsigned int val\_c;

sscanf (buf + 9 + i \* 2, "%02X", &val\_c);

edit\_memory (memories, base\_addr + start\_offset + i, val\_c);

}

continue;

}

if (buf[0] == 'M'){

int offset;

int length, num = 0xFFFFFFFF;

char op\_char = 0;

sscanf (buf, "M%06X%02X%c%02X", &offset, &length, &op\_char, &num);

if ((unsigned)num == 0xFFFFFFFF){

op\_char = '+';

num = 0x01;

}

if (!num\_validation[num]) return false;

const unsigned int old\_value = num\_to\_address[num];

unsigned int new\_value = 0;

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

new\_value <<= 8;

new\_value += (unsigned char)memories->data[base\_addr+offset+i].value;

}

const int sum\_new\_old\_value = new\_value + old\_value;

const int sub\_new\_old\_value = new\_value - old\_value;

if (length == 5){

if (op\_char == '+')

new\_value = (sum\_new\_old\_value & 0xFFFFF) + ((new\_value >> 20) << 20);

else if (op\_char == '-')

new\_value = (sub\_new\_old\_value & 0xFFFFF) + ((new\_value >> 20) << 20);

else return false;

}else if (length == 6){

if (op\_char == '+') new\_value = sum\_new\_old\_value;

else if (op\_char == '-') new\_value = sub\_new\_old\_value;

else return false;

}

edit\_memory (memories, base\_addr + offset, (unsigned char)(new\_value >> 16));

edit\_memory (memories, base\_addr + offset + 1, (unsigned char)(new\_value >> 8));

edit\_memory (memories, base\_addr + offset + 2,(unsigned char)new\_value);

}

}

fclose(fp);

return true;

}

static LoadInfoList\* construct\_load\_info\_list(){

LoadInfoList\* loadInfoList;

loadInfoList = (LoadInfoList\*)malloc(sizeof(LoadInfoList));

loadInfoList->count = 0;

return loadInfoList;

}

static void print\_load\_infos(LoadInfoList \*load\_infos){

unsigned int total\_length = 0;

const int count = load\_infos->count;

printf ("%-15s %-15s %-15s %-15s\n", "Control", "Symbol", "Address", "Length");

printf ("%-15s %-15s %-15s %-15s\n", "Section", "Name", "", "");

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

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

const LoadInfoNode node = load\_infos->list[i];

if (node.type == INFO\_TYPE\_CONTROL\_SECTION){

printf ("%-15s %-15c %04X%11c %04X%11c\n",

node.name,

' ', node.addr,

' ', node.length, ' ');

total\_length += node.length;

continue;

}

if (node.type == INFO\_TYPE\_SYMBOL){

printf ("%-15c %-15s %04X%11c %-15c\n", ' ',

node.name, node.addr,

' ', ' ');

}

}

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

printf ("%-15c %-15c %-15s %04X%11c\n", ' ', ' ', "Total Length", total\_length, ' ');

}

static bool execute\_operator(Debugger \*debugger, Memories \*memories, Instruction \*instruction) {

assert(debugger);

assert(memories);

assert(instruction);

Registers\* registers = debugger->registers;

uint32\_t value;

uint32\_t val1, val2;

static size\_t device\_input\_idx = 0;

char inputDevice[] = " I am Test Device\0\0";// virtual input device

switch (instruction->opcode){

case LDA:

load\_from\_memory(debugger, memories, instruction, &registers->A, 3, false);

break;

case LDB:

load\_from\_memory(debugger, memories, instruction, &registers->B, 3, false);

break;

case LDT:

load\_from\_memory(debugger, memories, instruction, &registers->T, 3, false);

break;

case LDX:

load\_from\_memory(debugger, memories, instruction, &registers->X, 3, false);

break;

case LDCH:

load\_from\_memory(debugger, memories, instruction, &value, 1, false);

registers->A = (registers->A & 0xFFFFFF00) + (value & 0xFF);

break;

case STA:

store\_to\_memory(debugger, memories, instruction, registers->A, 3);

break;

case STL:

store\_to\_memory(debugger, memories, instruction, registers->L, 3);

break;

case STX:

store\_to\_memory(debugger, memories, instruction, registers->X, 3);

break;

case STCH:

store\_to\_memory(debugger, memories, instruction, registers->A & 0xFF, 1);

break;

case JSUB:

load\_from\_memory(debugger, memories, instruction, &value, 3, true);

registers->L = registers->PC;

registers->PC = value;

break;

case JEQ:

if (registers->SW == 0) load\_from\_memory(debugger, memories, instruction, &registers->PC, 3, true);

break;

case JGT:

if ((int) registers->SW > 0) load\_from\_memory(debugger, memories, instruction, &registers->PC, 3, true);

break;

case JLT:

if ((int)registers->SW < 0) load\_from\_memory(debugger, memories, instruction, &registers->PC, 3, true);

break;

case J:

load\_from\_memory(debugger, memories, instruction, &registers->PC, 3, true);

break;

case COMP:

load\_from\_memory(debugger, memories, instruction, &value, 3, false);

if (registers->A > value)

registers->SW = 1;

else if (registers->A < value)

registers->SW = -1;

else

registers->SW = 0;

break;

case TD:

registers->SW = 1;

break;

case RD:

registers->A = (registers->A & 0xFFFFFF00) + (unsigned char)inputDevice[device\_input\_idx++];

if (device\_input\_idx >= sizeof(inputDevice) / sizeof(char))

device\_input\_idx = 0;

break;

case RSUB:

registers->PC = registers->L;

break;

case WD:

// is done!

break;

case CLEAR:

store\_to\_register(debugger, instruction->param.param.p2.r1, 0);

break;

case COMPR:

load\_from\_register(debugger, instruction->param.param.p2.r1, &val1);

load\_from\_register(debugger, instruction->param.param.p2.r2, &val2);

if (val1 > val2) registers->SW = 1;

else if (val1 < val2) registers->SW = -1;

else registers->SW = 0;

break;

case TIXR:

load\_from\_register(debugger, instruction->param.param.p2.r1, &value);

++registers->X;

if (registers->X > value) registers->SW = 1;

else if (registers->X < value) registers->SW = -1;

else registers->SW = 0;

break;

default:

return false;

}

return true;

}

static uint32\_t calculate\_TA(Instruction\* instruction, Registers\* registers){

uint32\_t TA;

uint32\_t b = (instruction->extend ? instruction->param.param.p4.b : instruction->param.param.p3.b);

uint32\_t p = (instruction->extend ? instruction->param.param.p4.p : instruction->param.param.p3.p);

uint32\_t address = (instruction->extend ? instruction->param.param.p4.address : instruction->param.param.p3.address);

uint32\_t x = (instruction->extend ? instruction->param.param.p4.x : instruction->param.param.p3.x);

if (b == 1 && p == 0){

// Base Relative

TA = address + registers->B;

} else if (b == 0 && p == 1){

// PC Relative

int32\_t val;

uint32\_t bound;

if(instruction->extend) bound = (1 << 19);

else bound = (1 << 11);

if(address >= bound) val = address - (bound << 1);

else val = address;

TA = registers->PC + val;

}else {

TA = address;

}

if (x == 1){

TA += registers->X;

}

return TA;

}

static bool handling\_bp(Debugger \*debugger, int instruction\_size) {

bool is\_break = false;

unsigned int bp;

for(unsigned int i = debugger->registers->PC; i < debugger->registers->PC + instruction\_size; i++){

if(debugger->break\_points[i]){

if(debugger->previous\_bp != -1 && i == (unsigned int)debugger->previous\_bp)

continue;

is\_break = true;

bp = i;

debugger->previous\_bp = bp;

break;

}

}

if(is\_break){

print\_registers(debugger->registers);

printf ("Stop at checkpoint[%04X]\n", bp);

return true;

}

return false;

}

static ADDRESSING\_MODE calculate\_addressing\_mode(Instruction\* instruction, bool jump\_op){

uint32\_t n = (instruction->extend ? instruction->param.param.p4.n : instruction->param.param.p3.n);

uint32\_t i = (instruction->extend ? instruction->param.param.p4.i : instruction->param.param.p3.i);

if(jump\_op){

if(n == 1 && i == 1) return ENUM\_IMMEDIATE\_ADDRESSING;

if(n == 1 && i == 0) return ENUM\_SIMPLE\_ADDRESSING;

return ENUM\_ADDRESSING\_ERROR;

}

if(n == 0 && i == 1) return ENUM\_IMMEDIATE\_ADDRESSING;

if(n == 1 && i == 1) return ENUM\_SIMPLE\_ADDRESSING;

if(n == 1 && i == 0) return ENUM\_INDIRECT\_ADDRESSING;

return ENUM\_ADDRESSING\_ERROR;

}

static bool load\_from\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t \*value,

size\_t bytes, bool jump\_op){

uint32\_t TA;

ADDRESSING\_MODE addr\_mode;

TA = calculate\_TA(instruction, debugger->registers);

addr\_mode = calculate\_addressing\_mode(instruction, jump\_op);

if (addr\_mode == ENUM\_IMMEDIATE\_ADDRESSING){

\*value = TA;

return true;

}

if (addr\_mode == ENUM\_SIMPLE\_ADDRESSING){

uint8\_t memory\_val;

\*value = 0;

for (size\_t k = 0; k < bytes; ++k){

memory\_val = (uint8\_t)memories->data[TA + k].value;

\*value = (\*value << 8) + memory\_val;

}

return true;

}

if (addr\_mode == ENUM\_INDIRECT\_ADDRESSING) {

uint32\_t indirect\_address = 0;

uint8\_t memory\_val;

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

memory\_val = (uint8\_t)memories->data[TA + k].value;

indirect\_address = (indirect\_address << 8) + memory\_val;

}

\*value = 0;

for (size\_t k = 0; k < bytes; ++k){

memory\_val = (uint8\_t)memories->data[indirect\_address + k].value;

\*value = (\*value << 8) + memory\_val;

}

return true;

}

return false;

}

static bool store\_to\_memory(Debugger \*debugger, Memories \*memories, Instruction \*instruction, uint32\_t value,

size\_t bytes){

uint32\_t TA;

ADDRESSING\_MODE addr\_mode;

TA = calculate\_TA(instruction, debugger->registers);

addr\_mode = calculate\_addressing\_mode(instruction, false);

if(addr\_mode == ENUM\_IMMEDIATE\_ADDRESSING){

return false;

}

if(addr\_mode == ENUM\_SIMPLE\_ADDRESSING){

for (int k = bytes-1; k >= 0; --k){

edit\_memory(memories, TA + k, (uint8\_t)value);

value >>= 8;

}

return true;

}

if(addr\_mode == ENUM\_INDIRECT\_ADDRESSING){

uint8\_t mem\_val;

uint32\_t addr = 0;

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

mem\_val = (uint8\_t)memories->data[TA + k].value;

addr = (addr << 8) + mem\_val;

}

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

edit\_memory(memories, addr + k, (uint8\_t)value);

value >>= 8;

}

return true;

}

return false;

}

static bool load\_from\_register(Debugger \*debugger, int reg\_id, uint32\_t \*val){

uint32\_t\* reg = get\_reg\_by\_id(debugger->registers, reg\_id);

if(reg == NULL) return false;

\*val = \*reg;

return true;

}

static uint32\_t \*get\_reg\_by\_id(Registers \*registers, int reg\_id) {

if(reg\_id == 6 || reg\_id == 7) return NULL;

if(reg\_id < 0 || reg\_id > 9) return NULL;

uint32\_t\* regs[10] = {

[0] = &registers->A,

[1] = &registers->X,

[2] = &registers->L,

[3] = &registers->B,

[4] = &registers->S,

[5] = &registers->T,

[6] = NULL,

[7] = NULL,

[8] = &registers->PC,

[9] = &registers->SW

};

return regs[reg\_id];

}

static bool store\_to\_register(Debugger \*debugger, int reg\_id, uint32\_t val){

uint32\_t\* reg = get\_reg\_by\_id(debugger->registers, reg\_id);

if(reg == NULL) return false;

\*reg = val;

return true;

}

static void print\_registers(Registers\* registers){

printf (

"A : %06X X : %06X \n"

"L : %06X PC: %06X \n"

"B : %06X S : %06X \n"

"T : %06X \n",

registers->A, registers->X,

registers->L, registers->PC,

registers->B, registers->S,

registers->T

);

}

## debug.h

#ifndef \_\_DEBUG\_H\_\_

#define \_\_DEBUG\_H\_\_

#include <stdlib.h>

#include <stdbool.h>

#include <stdint.h>

#include "memory.h"

#include "symbol.h"

#include "opcode.h"

#include <string.h>

#define MAX\_BP\_NUM (1024 \* 1024) // 1MB

// 레지스터 값을 저장함.

typedef struct registers {

uint32\_t A;

uint32\_t L;

uint32\_t X;

uint32\_t PC;

uint32\_t B;

uint32\_t S;

uint32\_t T;

uint32\_t SW;

}Registers;

// load info 정보의 유형

enum load\_info\_type {

INFO\_TYPE\_CONTROL\_SECTION,

INFO\_TYPE\_SYMBOL

} LoadInfoType;

// load info 정보 한 단위

typedef struct load\_info\_node {

enum load\_info\_type type;

char name[15];

int addr;

int length;

}LoadInfoNode;

// load 되는 정보를 저장함.

typedef struct load\_info\_list {

LoadInfoNode list[1001];

int count;

} LoadInfoList;

// load, run, bp, 레지스터 관리 등의 역활을함.

typedef struct debugger {

uint32\_t start\_address; // 10 진수로 변환하여 저장함.

int end\_address;

bool break\_points[MAX\_BP\_NUM]; // 해당 셀에 브레이크포인트가 걸렸는지 안걸렸는지 여부 저장

int bp\_count; // 브레이크 포인트 개수

Registers\* registers; // 레지스터 정보

int run\_count; // run 실행 횟수.

int previous\_bp; // 이전 bp 주소

char\* filenames[3]; // 파일 이름들

int file\_count; // 파일 개수

SymbolTable\* estab;

LoadInfoList\* load\_infos;

uint32\_t total\_length;

bool is\_running;

bool is\_loaded;

} Debugger;

// debugger 의 생성자 함수

Debugger\* construct\_debugger();

// debugger 의 소멸자 함수

bool destroy\_debugger(Debugger\*\* debugger);

// registers 의 생성자 함수

Registers\* construct\_registers();

// registers 의 소멸자 함수

bool destroy\_registers(Registers\*\* registers);

// register 를 리셋함.

void reset\_registers(Registers\* registers);

// loader 명령을 실행하는 함수

bool loader\_linker(Debugger \*debugger, Memories \*memories);

// run 명령을 실행하는 함수

bool run(Debugger \*debugger, Memories \*memories);

#endif

## Makefile

SOURCES=20161631.c command.c command\_mapping.c command\_shell.c command\_execute.c history.c memory.c state.c util.c dir.c command\_validate\_util.c opcode.c symbol.c assemble.c debug.c

TARGET=20161631.out

all: $(TARGET)

$(TARGET): $(SOURCES)

gcc -std=gnu99 -W -Wall $(SOURCES) -o $(TARGET)

clean:

rm -f $(TARGET)