**과목명: 시스템프로그래밍**

**CSE4100-01**

**<<Project #1>>**

**서강대학교 컴퓨터공학부**

**20151523**

**김동현**

목 차

1. **프로그램 개요**
   1. 프로그램에 관하여
   2. 프로그램 구성
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명
      1. help
      2. type
      3. assemble
      4. obj
      5. make\_obj
      6. make\_lst
      7. symbol
4. **전역 변수 정의**
   1. 구조체 및 변수
5. **코드 설명**
   1. 모듈
      1. main
      2. help
      3. type
      4. assemble
      5. obj
      6. make\_obj
      7. make\_lst
      8. symbol

**1. 프로그램 개요**

* 1. **프로그램에 관하여**

프로젝트1에서 구현한 셸에 assemble 기능을 추가하는 프로그램이다. SIC/XE의 assembly program source 파일을 입력 받아서 object파일과 lst파일을 생성하고, 어셈블리 과정 중에 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공한다.

* 1. **프로그램 구성**

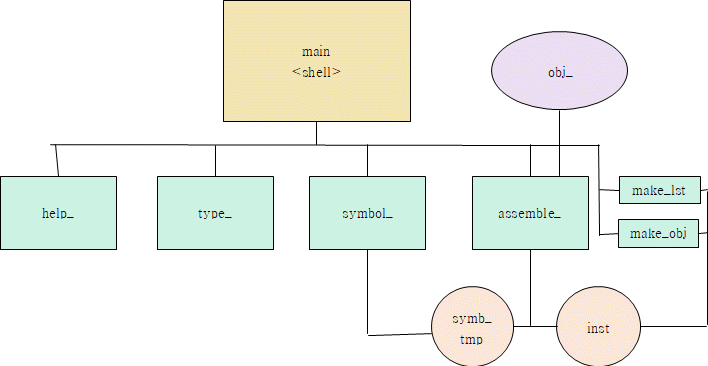
- Shell 관련 명령어들(help, type)

- assembler (assemble)

- assemble 관련 명령어들 (symbol)

**2. 프로그램 설명**

* 1. **프로그램 흐름도**



Main shell에서 help\_, type\_, symbol\_, assemble\_(make\_lst와 make\_obj도 같이) 모듈을 호출하여 실행하는 구조이다. Assemble\_ 모듈을 실행하게 되면 inst, symb\_tmp의 구조체 배열에 대한 접근이 발생하고, 내부에서 obj\_ 모듈을 호출하여 object code를 구조체에 저장시켜 준다. 그 후 구조체에 대한 저장이 된 상태에서 make\_lst, make\_obj와 symbol에서 해당 기능을 수행하게 되는 구조이다.

**3. 모듈 정의**

* 1. **정의한 모듈에 대해 간략적인 설명**
     1. **void help\_()**

프로젝트1에서 구현하였던 help와 거의 유사하지만, 프로젝트2에서 새롭게 구현된 명령어를 추가하게 되었다. Assemble filename, type filename, symbol 의 3항목이 추가되었다.

텍스트, 검은색, 앉아있는, 하얀색이(가) 표시된 사진

자동 생성된 설명

<실행 모습>

* + 1. **void type\_(FILE\* file)**

파일 포인터를 파라메터로 받아서 그 파일 포인터에 해당하는 파일을 출력하는 모듈이다.

검은색이(가) 표시된 사진

자동 생성된 설명검은색이(가) 표시된 사진

자동 생성된 설명

<실행 모습>

* + 1. **int assemble\_(FILE\* file, hash\* table[20])**

.asm 파일을 읽어들여서 각 line 별로 해당 내용(label, symbol, format, target 등)을 구조체 배열에 각각 저장해주고, sym\_tab을 형성하고, location counter와 line number를 계산해준다. 그리고 어셈블리 구동에 있어서 에러가 발생하는지 체크한다. 그리고 마지막에 obj\_함수를 호출한다.

검은색, 화면, 오렌지, 모니터이(가) 표시된 사진

자동 생성된 설명

* + 1. **void obj\_(hash\* table[20])**

assemble\_에서 호출되어지는 함수로, 각 line들의 format을 바탕으로 구분하여 object code를 계산하여 저장하는 함수이다. 현재로서 Format 1을 제외한 instruction까지 분류가 가능하도록 설계하였다.

* + 1. **void make\_obj(char\* name)**

.asm의 assemble을 통해 데이터가 저장된 구조체 배열을 이용하여 .obj 파일을 생성하는 모듈이다. Header/ text / modify/ end record로 구분되어 있다. Text record의 출력은 구조체 배열에 저장된 Object code를 출력하는 방식으로 하였다.

검은색이(가) 표시된 사진

자동 생성된 설명

<.obj파일의 출력>

* + 1. **void make\_lst(char\* name)**

.asm의 assemble을 통해 데이터가 저장된 구조체 배열을 이용하여 .lst 파일을 생성하는 모듈이다.

텍스트, 검은색, 앉아있는, 테이블이(가) 표시된 사진

자동 생성된 설명

* + 1. **void symbol\_()**

assemble\_을 통해 생성된 sym\_tab을 사전식 순서로 정렬하여 location counter와 함께 출력해주는 모듈이다.

텍스트, 검은색, 앉아있는, 녹색이(가) 표시된 사진

자동 생성된 설명

**4. 전역변수 정의**

**4.1 구조체 및 변수**

검은색, 앉아있는, 화면, 테이블이(가) 표시된 사진

자동 생성된 설명

이번 프로젝트2의 구현을 위해 선언한 전역 변수 및 구조체이다. 우선 opinst라는 구조체 변수를 만들었고, 그 안의 멤버로 label, symbol, target(2)를 문자열 배열로 지정하여 주었고, line, location counter, object code를 정수형 변수로 지정하였다. 그리고 assemble의 여러 분기 선택을 위해 label\_flag (label이 존재하면 1, 아니면 0), format(형식 : 0, 2, 3, 4, 5), modify flag(Modify record를 해야하면 1, 아니면 0)을 지정하였다. Location counter는 int형으로 저장해서 출력 시 %.4X를 통해 4자리 16진수를 표현하려고 계획하였다. Object code도 마찬가지로 int로 선언하여서 계산을 용이하게 하고, 출력시에는 해당 포맷에 맞게 16진수로 출력하게끔 계획하였다.

그리고 전체 명령어들을 포함하는 구조체 배열 inst를 선언하였고, symbol table의 역할을 해줄 symb\_tmp를 선언하였다. Line\_ctr은 모듈 간 인자 교환을 간소화 하기위해 전역변수로 선언하였다.

**5. 코드 설명**

* 1. **모듈**
     1. **Int main()**

Main 모듈에서는 프로젝트1의 틀에 그대로 프로젝트2에서 구현해야 하는 기능을 추가 조건문으로 실행될 수 있게하였다.

시계, 개체, 오렌지, 도시이(가) 표시된 사진

자동 생성된 설명

파일포인터 tp와 as는 각각 type과 assemble을 수행할 때 가져다 쓰는 파일을 지정하기 위해 선언하였고, buf는 type과정에서 해당 파일이 디렉토리인지 검출하기 위해 선언하였다.

검은색, 오렌지, 앉아있는, 어두운이(가) 표시된 사진

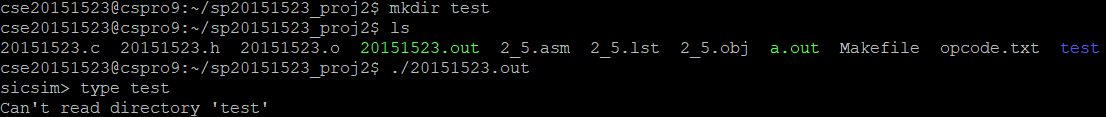
자동 생성된 설명

위는 symbol 기능을 수행하는 조건문이다. Symbol은 출력 유무와 관계없이 history node에 추가시켜주므로 add\_node를 선행하도록 작성하였다.

앉아있는, 노트북, 테이블, 검은색이(가) 표시된 사진

자동 생성된 설명

위는 type 기능을 수행하는 조건문이다. Filename을 argument로 받아야 하므로 argv1이 NULL이라면 명령어가 없다는 에러코드를 출력, 그리고 argv1을 인식했다면 그 파일에 대한 stat을 buf에 저장해준다. 그리고 S\_ISDIR 모듈을 통하여 directory인지 구분하여서 디렉토리라면 디렉토리는 읽을 수 없다는 에러코드를 출력, 만일 tp가 NULL이라면 해당 argv1의 파일이 폴더 내에 없다는 의미이므로 찾지못했다는 에러코드를 출력한다. 이 이외에는 정상적으로 수행되는 case이므로 add\_node를 해준 뒤 type에 해당 파일포인터를 넘겨주면서 실행하게 된다.



<type directory의 경우>

화면, 모니터, 텔레비전, 테이블이(가) 표시된 사진

자동 생성된 설명

위는 assemble 기능을 수행하는 조건문이다. Filename을 argument로 받아야 하므로 argv1이 NULL이라면 명령어가 없다는 에러코드를 출력, 만일 as가 NULL이라면 해당 argv1의 파일이 폴더 내에 없다는 의미이므로 찾지못했다는 에러코드를 출력한다. 그 외의 경우는 정상 수행되는 case이므로 add\_node를 해준 뒤, assemble 내에서 에러가 발생하지 않은 경우(assemble() == 0)는 성공적으로 어셈블하였다는 메시지와 make\_lst, make\_obj를 실행하도록 하게 설계하였다.

* + 1. **void help\_()**

테이블, 컴퓨터, 앉아있는, 화면이(가) 표시된 사진

자동 생성된 설명

단순히 추가된 3가지의 기능에 대해서 추가적으로 printf하도록 하였다.

* + 1. **void type\_(FILE\* file)**

사진, 검은색, 앉아있는, 어두운이(가) 표시된 사진

자동 생성된 설명

Type 모듈에서는 파일 포인터를 인자로 받아서 단순하게 파일의 끝(EOF)가 나올 때 까지 1글자씩 letter에 받아서 출력하도록 설계하였다.

* + 1. **int assemble\_(FILE\* file, hash\* table[20])**

모니터, 화면, 검은색, 텔레비전이(가) 표시된 사진

자동 생성된 설명

반환형은 int로 assemble 중 에러가 발생하면 1을 반환하여 main에서 make\_obj, make\_lst를 실행하지 않게끔 하였고 asm 파일 포인터와 format을 참조하기 위한 해시 테이블을 파라메터로 받아왔다.

스크린샷, 모니터, 앉아있는, 화면이(가) 표시된 사진

자동 생성된 설명

Asm 파일이 끝날 때까지 반복하도록 하였고 second\_argu flag를 통해 STCH처럼 2개의 레지스터를 인자로 하는 instruction을 구분하도록 하였다. fgets으로 문자열 한 줄을 읽어와서 ins\_tmp라는 문자열 변수에 복사하였다. 그리고 strtok을 통해 instruction을 label, symbol, target 등으로 구분하여 n1~3에 저장해 두었다. 만일 아무것도 없는 경우(n1 == NULL)는 바로 break를 하게 하고, asm 파일 내에 .으로 주석 처리 된 부분은 line만 증가하도록 하였다. 만일 RSUB이 포함된 instruction을 제외한 n2가 first target이고, 쉼표로 구분을 하여 second target이 있는 경우 second\_argu의 flag를 올려주었다.

앉아있는, 모니터, 테이블, 검은색이(가) 표시된 사진

자동 생성된 설명

이 경우는 n3 == NULL인 경우는 label이 없는 경우로 판단하고, 그 중에서 n2 == NULL인 경우는 RSUB과 같이 지시어만 존재하는 경우로 지시어만 symbol 항목에 저장해 주었다. N2 != NULL이라면 label이 없는 경우로 보아서**(해당 프로젝트에서는 명세서에서도 명시되어 있듯이, 완벽한 assembler를 구현하는 것이 아니므로 모든 예외처리를 해주지 않았다. 이는 이후에 설정한 case 구분에도 적당히 해당되는 내용이다.)** 처음 들어온 n1을 symbol에, 그 다음 들어온 n2를 target에 저장해주었다. 그리고 모든 경우는 locator counter와 line을 증가시켜 저장해준다. 그리고 label이 없으므로 label\_flag는 0으로 지정해주었다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

N3 != NULL인 경우는 second\_argu flag를 통해 구분하였다. 만일 second\_argu가 0이라면 n1~3는 각각 label, 지시어, target으로 보고 각각 저장해주었으며, 이 경우 label이 중복되면 그 즉시 해당 라인에 에러가 있다는 에러코드를 출력한 후 1을 반환하게 하였다. Second\_argu가 1이라면 target2가 존재한다는 것 이므로 n2의 ‘,’를 제거한 후 각각 symb, target, target2 멤버에 저장해주었다. 그리고 여기까지 하게되면 모든 case를 분류하였으므로 i를 증가시킨다.

앉아있는, 작은, 테이블, 전화이(가) 표시된 사진

자동 생성된 설명

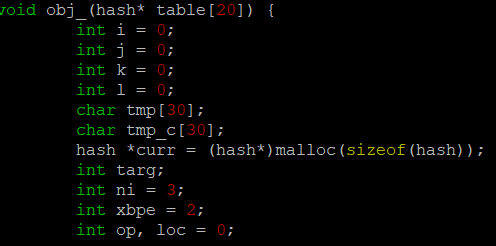
위의 문자열 분류가 끝나면 각각 지시어에 따라서 증가하는 location counter와 format을 지정해주는 과정을 실행하였다. 우선 기존의 1,2,3,4 포맷 외에 변수 선언 및 start, end, base는 포맷 0, 상수 선언은 포맷 5로 임의로 설정해 주었다. 그리고 상수 선언의 경우, char인지 16진수인지에 따라서 locctr의 증가를 다르게 해주었다. 그리고 symb 멤버의 첫 문자가 ‘+’인 경우는 포맷 4로 저장하게끔 하였다.

스크린샷, 모니터, 앉아있는, 화면이(가) 표시된 사진

자동 생성된 설명

위의 특수 case를 제외한 포맷들의 지정은 hash table의 idk 멤버를 통해 실시하였는데, 만일 해당 지시어를 opcode table에서 찾을 수 없다면 해당 라인에 대해 에러가 있다고 에러 코드를 출력하고, 1을 반환하게 하였다. 그리고 찾았다면 해당 opcode table의 idk가 2이면 포맷 2, 나머지는 3이라고 해주었다. (포맷 1은 아직 쓰지 않아서 딱히 case를 구분하지않고 3로 포함시켜주었다.) 여기까지 문자열 구분하여 구조체 배열에 저장, 포맷과 location counter 저장이 끝나면 전역 변수 line\_ctr과 sym\_cnt에 구조체 배열의 개수와 symbol table의 개수를 각각 저장해주고, obj 모듈을 호출한다.

* + 1. **void obj\_(hash\* table[20])**



우선 object code를 계산하는 데에 있어서 ni와 xbpe 변수를 사용하였다. 하나하나 구분하기에는 너무 많은 경우의 수가 존재하고, 비효율적이라는 생각이 들어서 위와 같이 2개의 변수를 정수형으로 선언하였고, 초기값으로 가장 많이 등장하는 (n,i) = (1,1)과 (x,b,p,e) = (0,0,1,0)을 넣어주었다. 파라메터로는 opcode table을 받아왔는데, 해당 지시어의 opcode를 검색하기 위함이었다.

스크린샷, 앉아있는, 전화이(가) 표시된 사진

자동 생성된 설명

Assemble 모듈을 통해 저장된 line\_ctr만큼의 object code 계산을 시행해주었고, 포맷 별로 구분하여 object code 계산을 해주었다. 첫번째로 포맷 4의 지시어에서 ‘+’를 없애야 opcode table에서 검출이 가능하므로 ‘+’를 없애는 과정을 실시하였다. 포맷 0,4,5가 아닌 경우에는 그냥 opcode table에 symb에 저장된 문자열로 검출이 가능하고, 4인 경우에는 +를 없애준 뒤 검출을 해주었다.

모니터, 텔레비전, 앉아있는, 화면이(가) 표시된 사진

자동 생성된 설명

포맷 0인 경우는 object code가 없으므로 0, 포맷 5인 경우에는 상수 값에 따라서 달라지게 되므로 C인 경우와 X인 경우로 구분하였다. C인 경우는 tmp\_c[2]를 설정해 3bytes의 상수를 저장하게끔 해주었다. 예를 들어 C’EOF’의 경우, tmp\_c[0]에 ‘E’, tmp\_c[1]에 ‘O’, tmp\_c[2]에 ‘F’를 넣어주고 int형의 obj에 넣기 위해 tmp\_c[0]는 16^4를 곱해주고, tmp\_c[1]에는 16^2를 곱해준 뒤 3를 더해주면 16진수로 보았을 때 6자리의 object code에서 각각의 자리에 표현되게 된다. X의 경우에는 그 상수의 크기에 따라 표시되는 object code 자리수가 달라지게 된다 만일 그냥 strtoul을 통해 X’05’를 변환하면 000005로 출력이 되므로 050000로 표현되게 하는 작업을 해주었다.

모니터, 스크린샷, 앉아있는, 제어이(가) 표시된 사진

자동 생성된 설명

모니터, 텔레비전, 화면, 원격이(가) 표시된 사진

자동 생성된 설명

포맷 2의 경우는 레지스터를 이용한 연산이고, target register가 1개인 경우와 2개인 경우를 구분해주었다. switch문을 통해 각각의 register number를 지정하여서 해당 숫자를 targ 정수형 변수에 넣어서 저장했는데, 4자리의 object code 중 target1은 뒤에서 2번째 자리, target2는 뒤에서 1번째 자리에 해당되므로 target1의 targ연산에는 16을 추가로 곱해주었고, target2의 연산에는 target1이 저장된 targ에 추가적으로 더해주는 방식으로 해주었다. 그리고 마지막에는 opcode를 앞 1,2자리에 넣었다.

스크린샷, 앉아있는이(가) 표시된 사진

자동 생성된 설명

포맷 4에서는 target에 #, @가 있는 경우로 구분했고 이에 따라 ni와 xbpe를 바꿔주는 것이 필요했다. #인 경우는 immediate address이므로 n,i가 0,1이므로 1을 넣어주었고, opcode의 binary에서 뒤의 2개를 ni로 교체 해야하므로 4를 나눈 나머지를 빼고 op + ni를 해주어서 opcode의 첫 2자리를 만들어 주었다. 그리고 target이 상수인 경우 xbpe는 0001이므로 1, 아닌 경우에는 0011이므로 3으로 해주었다.

@가 있는 경우는 indirect address로 n,i가 1,0이므로 2를 넣어주었고, 마찬가지로 xbpe도 0011로 3으로 해주어서 object code를 계산해주었다.

마지막으로 absolute address를 참조하는 경우는 n,i가 1,1이므로 3, xbpe는 1으로 해주어서 계산해주었다.

잔디, 스크린샷, 앉아있는, 모니터이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 노트북, 잔디이(가) 표시된 사진

자동 생성된 설명

포맷 3의 경우도 위와 비슷한데 아래에 보면 xbpe에서 1000이어서 8인 경우를 추가해주었다.

잔디, 스크린샷, 앉아있는, 전화이(가) 표시된 사진

자동 생성된 설명모니터, 노트북, 실내, 화면이(가) 표시된 사진

자동 생성된 설명

포맷 3에서는 target이 없는 지시어에 대한 예외처리를 해주어서 따로 object code를 작성하게 하였고, 중간에 포맷3의 자리수 표현을 넘어가는 (참조 label의 주소가 1000이 넘는) 경우와 위의 label을 참조하는지, 아래의 label을 참조하는지에 대해 구분을 해주었다. 우선 아래의 label을 참조하면서 주소의 차이가 1000을 넘어가지 않는다면 단순히 주소의 차이에 pc값 +3(포맷3이므로)을 해주었고, 넘어가게 된다면 BASE instruction을 찾아서 그 주소와 참조 주소의 차이를 구하여서 loc에 저장해 주었다. 역으로 label을 참조할 경우에는 주소 차이를 구할 때 단순히 빼는 순서를 바꾸는 것이 아니라 4096을 더해주어야 했다. 왜냐하면 int형으로 16진수를 표현하다 보니 4096을 더하지 않으면 음수표현으로 출력되기 때문에 앞의 빈자리에 F같은 수가 들어오게 된다. 위와 같은 일련의 과정을 통해 object code에 대한 case들을 다루어 구조체 배열에 저장한 뒤, i를 증가시켜 다시 반복하게 된다.

* + 1. **void make\_obj(char\* name)**

모니터, 화면, 텔레비전, 제어이(가) 표시된 사진

자동 생성된 설명

Name 문자열 변수를 인자로 받아서 name.obj를 생성하게 하였고, 헤더 레코드 부분에서는 처음에 1st column에서 H, 2~7 column에 시작 label을 출력하고 j < 6의 반복문을 통해 빈 공간에는 공백을 넣도록 하였다. 그리고 그 이후의 column에는 시작 location counter과 길이를 출력하였는데 길이는 마지막 location counter와 시작 location counter의 차에 1을 더하여서 구해주었다.

앉아있는, 잔디, 검은색, 전화이(가) 표시된 사진

자동 생성된 설명

그 다음으로 text record를 출력하여 주었는데 start와 end line을 제외하므로 line\_ctr-3 동안 반복하게 하였다. 그리고 cnt < 10이라는 조건 반복문을 통해 한 줄에 최대 10개 까지 출력하도록 검사를 먼저 하였고, 그 과정에 format = 0인 instruction(변수 선언문을 format 0으로 임의로 설정해둠)을 만나면 종료하도록 하였다. 그리고 format 4인 instruction 중 target이 immediate address가 아닌 경우는 m\_flag를 올려서 나중에 modify record 수행에 사용하도록 하였다. 그렇게 cnt를 적절하게 끊어준 뒤, line의 사이즈를 출력하고, object code를 출력하도록 하였다. Format 2라면 4문자로 표현해야 하므로 %.4X로 출력하도록 하였다. 만일 변수 선언문이 나오는 경우, 끊어주고 j에 cnt를 누적시킨 뒤 다시 처음으로 돌아가서 연속적으로 변수 선언이 되어있는 부분을 다 빼주었다.

노트북, 화면, 방, 앉아있는이(가) 표시된 사진

자동 생성된 설명

Modify record는 단순히 위의 과정에서 설정된 m\_flag를 확인하여 해당되는 경우만 출력하도록 하였다.

End record는 end 문의 target에 해당하는 label을 처음부터 찾은 뒤, 그 위치의 location counter를 출력하도록 하였다.

* + 1. **void make\_lst(char\* name)**

실내, 컴퓨터, 노트북, 화면이(가) 표시된 사진

자동 생성된 설명

Name 문자열 변수를 인자로 받아서 name.lst를 생성하게 하였고, fprintf 함수를 통해 양식을 맞추어서 inst 구조체 배열의 멤버를 출력하도록 하였다. Instruction에 따라서 출력되는 양식이 다르므로 위와 같이 4가지 경우로 나누어 출력하도록 하였다.

* + 1. **void symbol\_()**

모니터, 화면, 노트북, 스크린샷이(가) 표시된 사진

자동 생성된 설명

우선 assemble이 정상적으로 수행되었는지 확인하기위해 line\_ctr이 0인지 아닌지 검사를 한다. Line\_ctr이 0이라면 assemble이 이전에 수행되었지 않았거나, 정상적으로 종료되지 않은 것 이기 때문에 아무것도 출력하지 않고 종료시킨다. 그 외의 경우는 우선 사전식 정렬을 수행하는데 2중 반복문에서 strcmp의 기능을 이용하여 정렬을 수행하였다. Bubble sort의 알고리즘을 채용하였다. Strcmp(A,B)의 경우 A와 B의 알파벳 순서에 따라 더 큰지 작은지를 비교하는 기능이 있으므로 만약 정렬이 필요한 순번이라면 tmp와 tmp2를 통해 label과 location counter를 swap해준다. 그리고 정렬이 끝나면 반복문을 통해 정렬된 symbol table을 출력해준다. (Symb\_cnt는 symbol table의 원소 개수이다.)