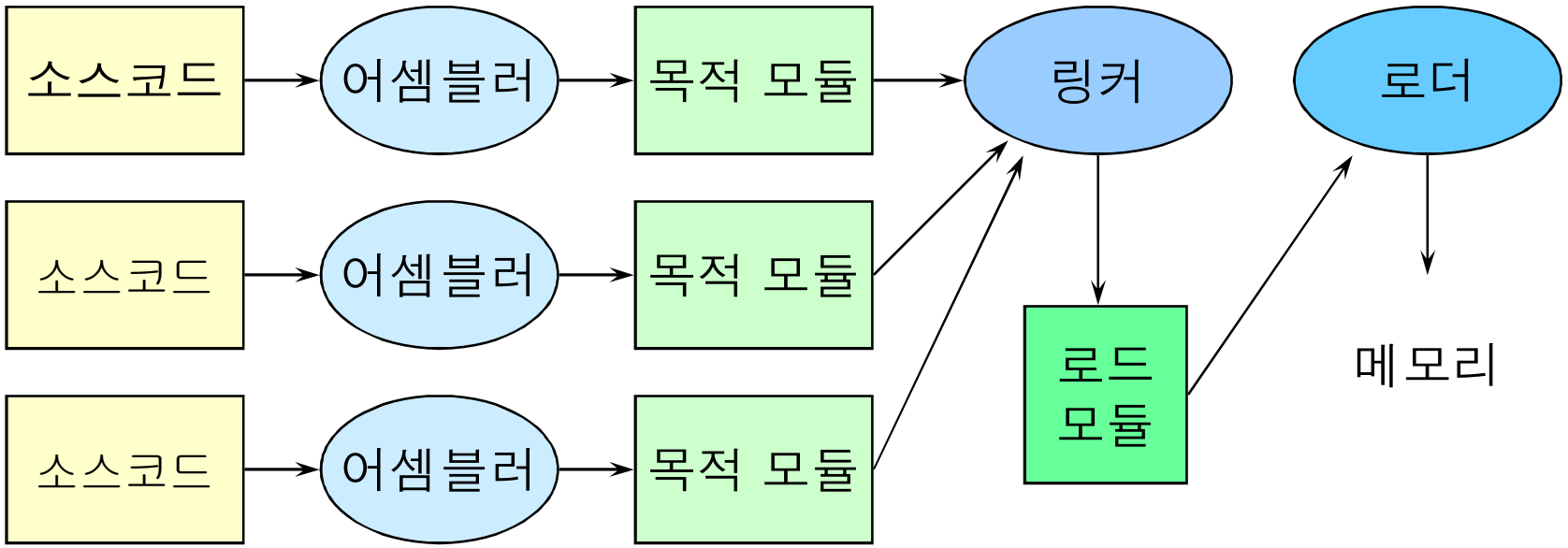
Assembler

어셈블러의 역할

- 어셈블리어로 쓰인 원시 프로그램을 입력 받아 목적 프로그램을 생성하는 일을 한다.



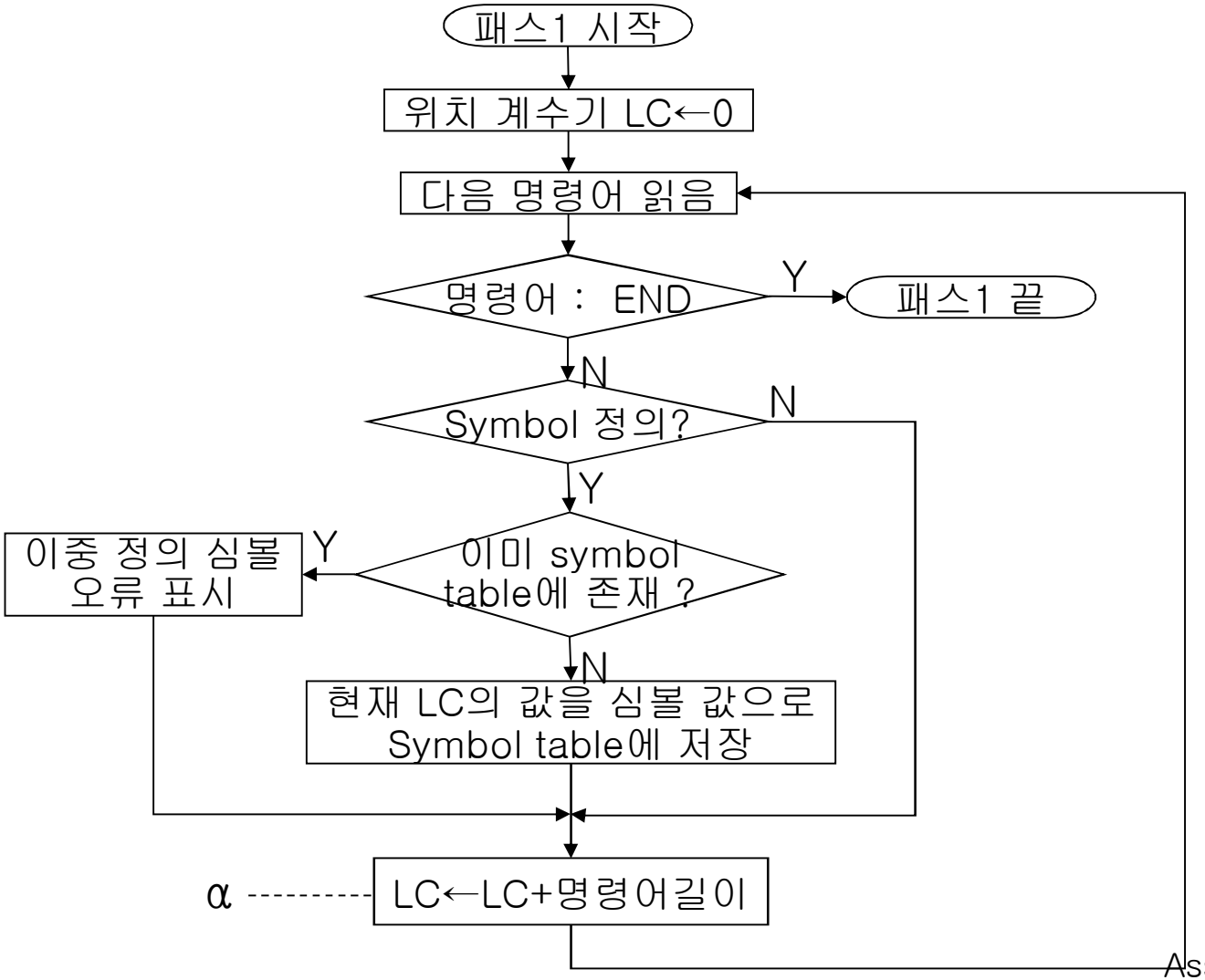
어셈블러 task 정의

- 어셈블러의 작업 - 명령어 생성
  - 연산 기호 -> 기계어 명령
  - 심볼과 리터럴에 주소 할당
- 2 pass assembler
  - 심볼이 정의되지 않은 상태에서 참조(reference)하는 경우 그 값을 알 수가 없다.
  - 패스 1:각 기호의 값만을 결정
  - 패스 2: 목적 프로그램을 만듦

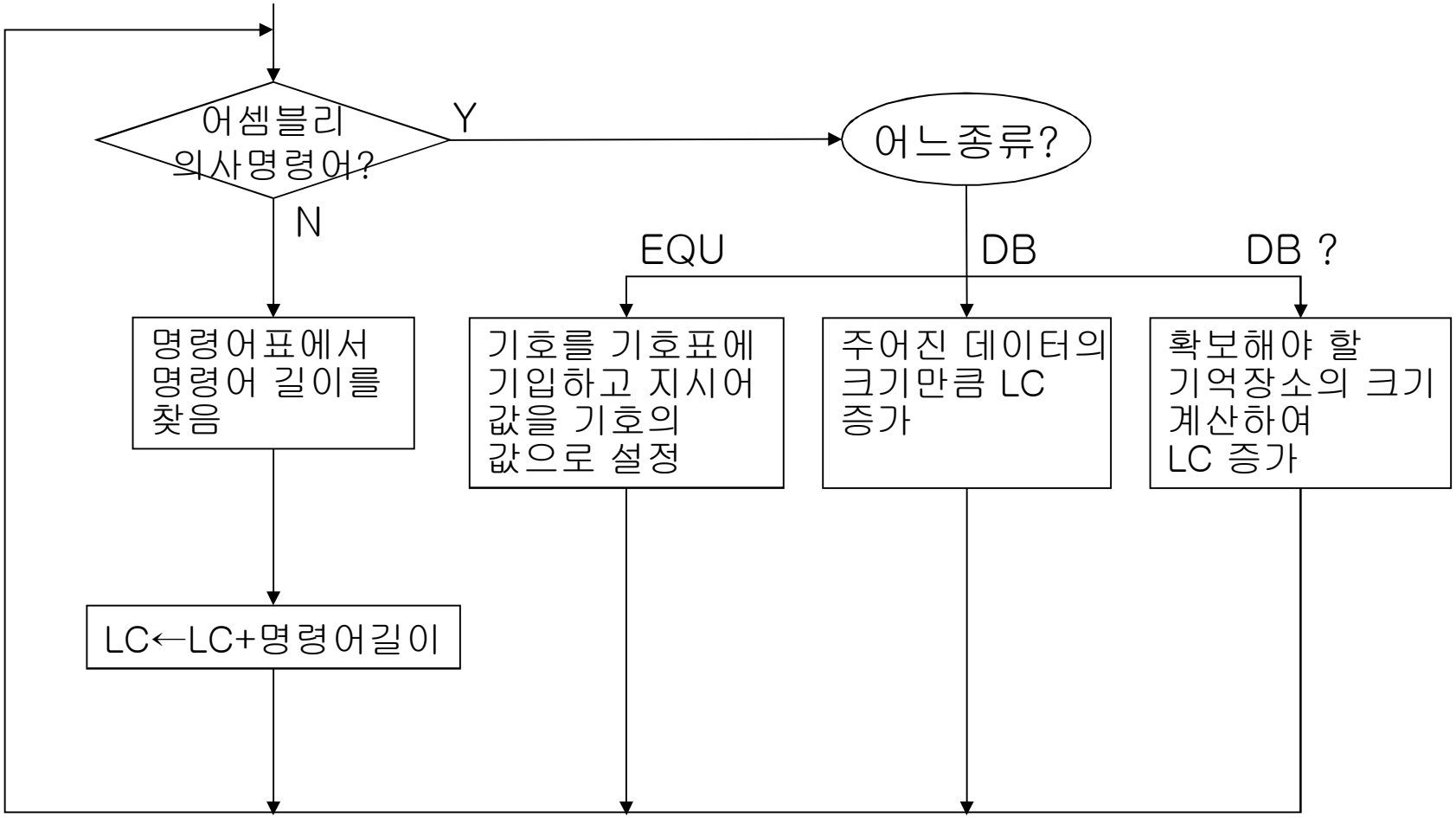
어셈블러 설계

- Pass 1 (심볼을 정의)
  - 각 기계어의 길이 결정
  - 위치 카운터(Location Counter)값 증가
  - 심볼의 값 기억

Pass 1


패스1 시작
위치 계수기 LC←0
다음 명령어 읽음
명령어 : END
Y
패스1 끝
N
Symbol 정의?
N
Y
이미 symbol table에 존재 ?
Y
이중 정의 심볼 오류 표시
N
현재 LC의 값을 심볼 값으로 Symbol table에 저장
α
LC←LC+명령어길이

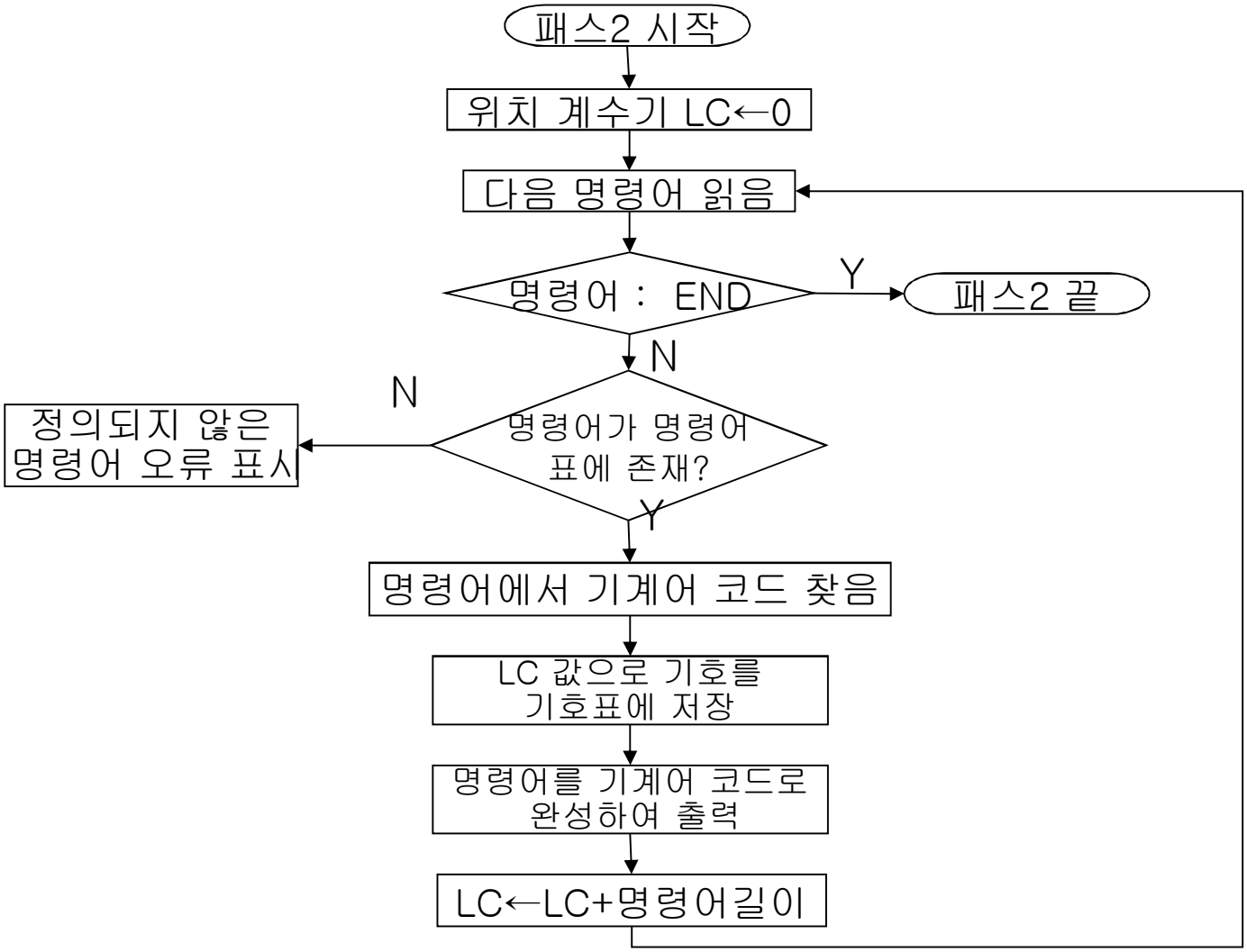
Pass 1 (계속)


어셈블리
의사명령어?
Y
N
어느종류?
EQU
DB
DB ?
명령어표에서
명령어 길이를
찾음
기호를 기호표에
기입하고 지시어
값을 기호의
값으로 설정
주어진 데이터의
크기만큼 LC
증가
확보해야 할
기억장소의 크기
계산하여
LC 증가
LC←LC+명령어길이

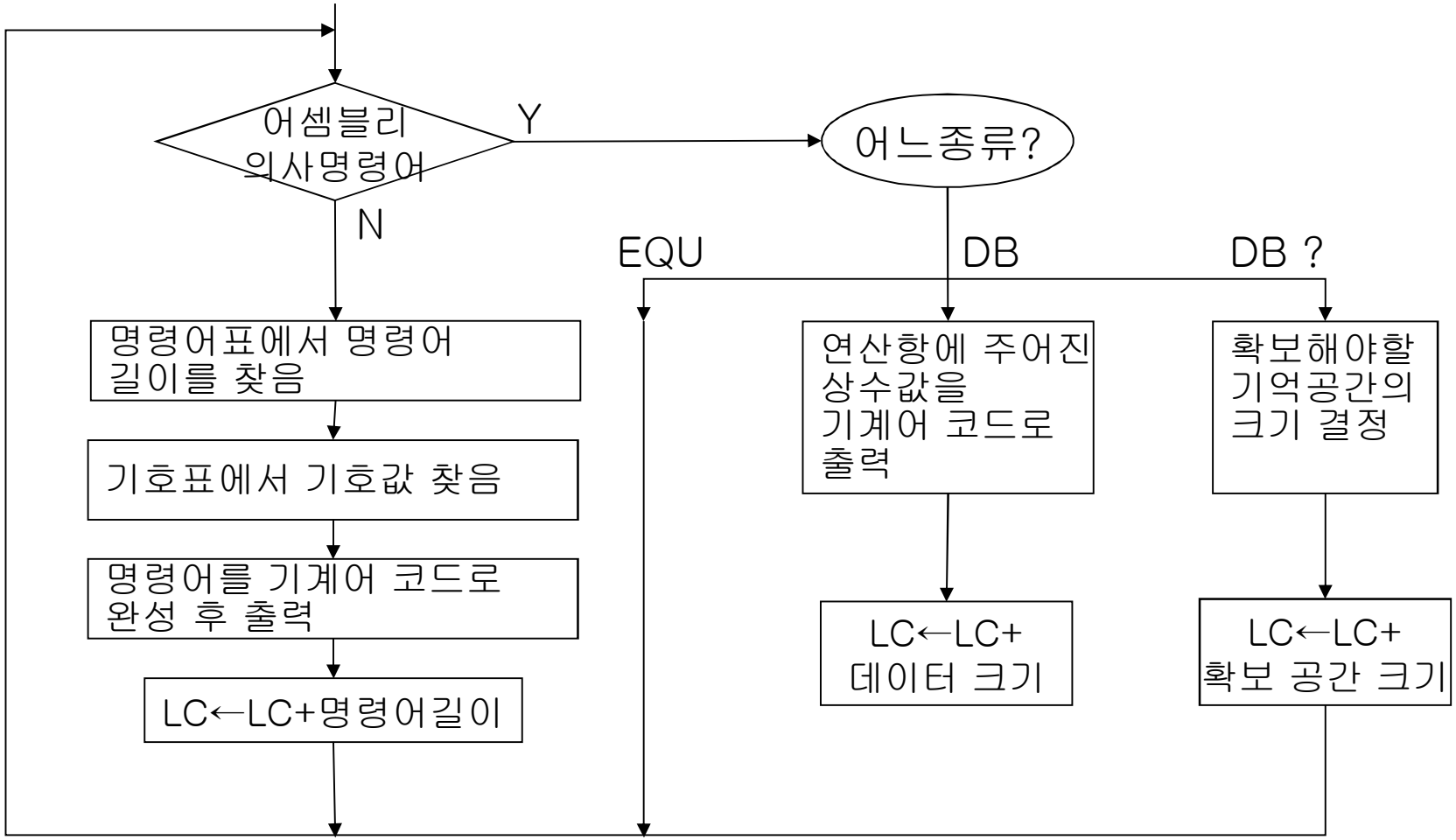
어셈블러 설계(계속)

- Pass 2 (목적 프로그램을 만들어 냄)
  - 심볼의 값을 찾음
  - 명령어를 만들어 냄
  - 상수, 심볼 등의 데이타를 만들어 냄

Pass 2


패스2 시작
위치 계수기 LC←0
다음 명령어 읽음
명령어 : END
Y
패스2 끝
N
N
명령어가 명령어 표에 존재?
정의되지 않은 명령어 오류 표시
Y
명령어에서 기계어 코드 찾음
LC 값으로 기호를 기호표에 저장
명령어를 기계어 코드로 완성하여 출력
LC←LC+명령어길이

Pass 2(계속)


어셈블리
의사명령어
Y
N
어느종류?
EQU
DB
DB ?
명령어표에서 명령어
길이를 찾음
기호표에서 기호값 찾음
명령어를 기계어 코드로
완성 후 출력
LC←LC+명령어길이
연산항에 주어진
상수값을
기계어 코드로
출력
LC←LC+
데이터 크기
확보해야할
기억공간의
크기 결정
LC←LC+
확보 공간 크기

어셈블러 설계(계속)

- **패스1의 자료구조**
  - 소스 프로그램
  - 명령어 테이블
  - 심볼 테이블
  - LC

어셈블러 설계(계속)

- **패스 2의 자료구조**
  - 소스 프로그램 사본
  - 명령어 테이블
  - 심볼 테이블
  - LC
  - 목적 프로그램 출력

간단한 어셈블러 구현(계속)

- 명령어

		First Byte	Second Byte	4 More
MOV	reg, reg	89	1 1 S S S D D D	
MOV	reg, imm	1 0 1 1 1 D D D		____
ADD	reg, reg	01	1 1 S S S D D D	
ADD	reg, imm	81	1 1 0 0 0 D D D*	____
SUB	reg, reg	29	1 1 S S S D D D	
SUB	reg, imm	81	1 1 1 0 1 D D D*	____
INC	reg	0 1 0 0 0 D D D		
DEC	reg	0 1 0 0 1 D D D		
IN	EAX, [DX]	ED		
OUT	[DX], EAX	EF		
RET		C3		
JMP	imm	E9		____
JZ	imm	0F	84	____
JNZ	imm	0F	85	____
JS	imm	0F	88	____
JNS	imm	0F	89	____

간단한 어셈블러 구현

- 레지스터 코드

3-Bit Register Codes

EAX	000	ESP	100
ECX	001	EBP	101
EDX	010	ESI	110
EBX	011	EDI	111

간단한 어셈블러 구현(계속)

- 코드 생성

Label	Source Code	Address	Machine Code
	MOV EDX, 0	0	BA 00
		2	00 00
		4	00
	IN EAX, [DX]	5	ED
	MOV ECX, EAX	6	89 C1
	IN EAX, [DX]	8	ED
	MOV EDX, EAX	9	89 C2
ORD	SUB EAX, ECX	B	29 C8
	JZ GCD	D	0F 84
		F	11 00
		11	00 00
	JNS NXT	13	0F 89
		15	04 00
		17	00 00

간단한 어셈블러 구현(계속)

- 코드 생성 (계속)

```
        MOV EAX, ECX        19   89 C8
        MOV ECX, EDX        1B   89 D1
NXT     MOV EDX, EAX        1D   89 C2
        JMP ORD             1F   E9 E7
                            21   FF FF
                            23   FF
GCD     MOV EAX, EDX        24   89 D0
        MOV EDX, 1          26   BA 01
                            28   00 00
                            2A   00
        OUT [DX], EAX       2B   EF
        RET                 2C   C3
```

소스 토큰화

- 입력 받은 어셈블리 소스를 각 필드 별로 나누는 작업
- 소스 라인 규칙

레이블	명령어	오퍼랜드 1	오퍼랜드 2

  - 레이블 뒤에는 :
  - 빈칸, comma, \n, \t을 만나는 경우에 대한 처리

테이블 만들기

- 테이블의 종류
  - 명령어 테이블
  - 레지스터 테이블
  - 심볼 테이블

테이블 만들기

- 명령어 테이블

```
struct inst {
        char inst_name[4];
        char inst_code[4];
        int number_of_opnd;
}

Struct inst inst_table[8] = {
        {“BEQ”, “000”, 2}, {“SUB”, “001”, 2},
        . . .
        {“ASL”, “111”, 2}};
```

테이블 만들기

- 심볼테이블

```
struct symbol {
        char symbol_name[9];
        int value;
        int lc;
} symbol_table[100];


struct symbol *ptr;
ptr = (symbol *)malloc(sizeof(struct symbol));
```