6. 코드 생성

충북대학교

이 재 성

학습내용

- 스택 기계
- 가상 스택 기계
- 가상 스택 기계용 중간코드 생성
- 8086 어셈블리어용 코드 생성

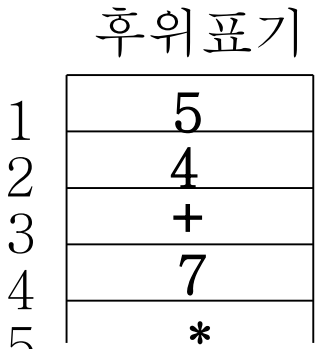
스택 기계

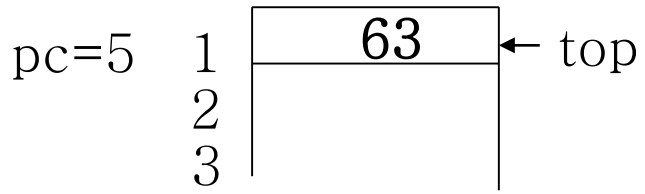
후위 표기 스택 연산

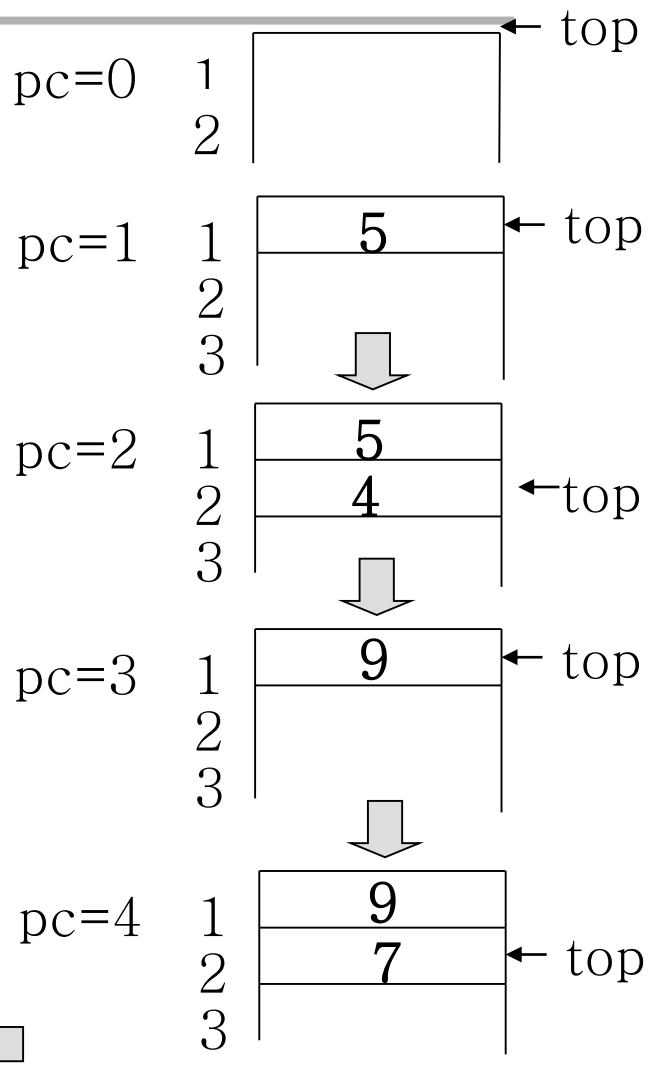
- 피연산자: 스택에 push
- 연산자(이항):
  - 스택의 맨위값과 그 밑의 값을 피연산자로 pop한 후
  - 연산을 수행하고 그 결과를 스택에 push

연산 예

- 5 4 + 7 *







가상 스택 기계

가상 스택 기계

- 중간 표현 형식을 수행할 수 있는 스택 기계

중간 표현 형식

- 컴파일러 앞부분(단계)의 결과로 중간 표현 형식을 출력
- 컴파일러 뒷부분(단계)에서는 중간 표현 형식을 목적프로그램으로 생성

가상 스택 기계의 명령어 종류

- 정수 산술 연산(+, -, *, / 등)
- 스택 관리(push, pop, rvalue, lvalue 등)
- 실행 흐름 조정(goto, gotrue, halt 등)

가상 스택 기계의 수행 예-1

명령

1	push 5
2	rvalue 2
3	+
4	rvalue 3
5	*

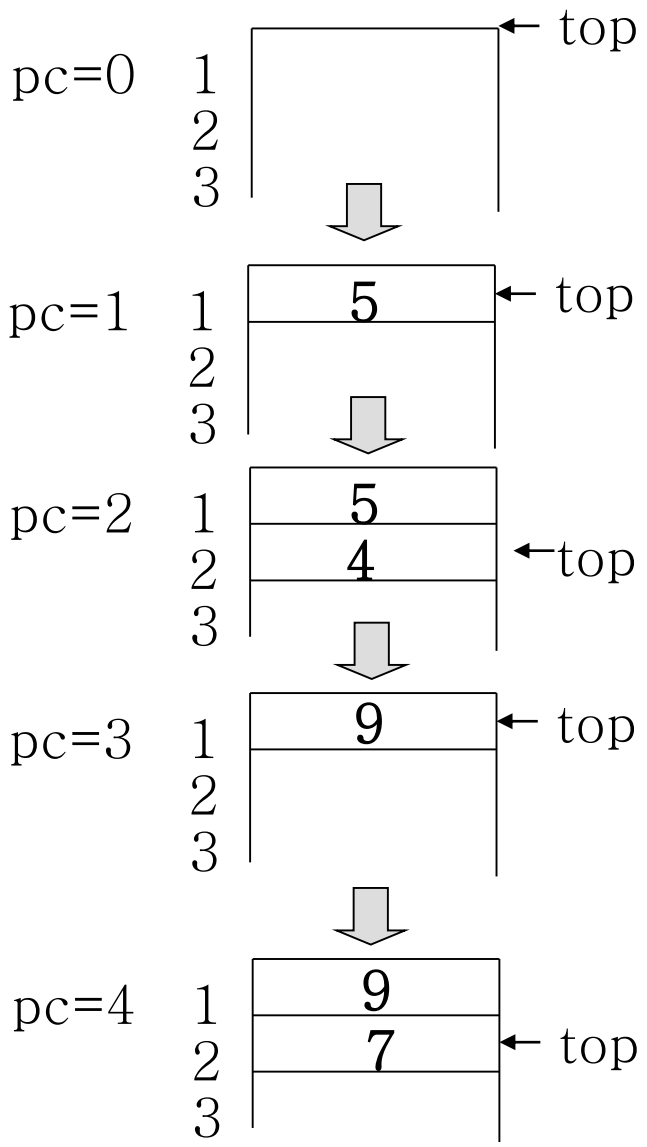
데이터

1	0
2	4
3	7
4	...

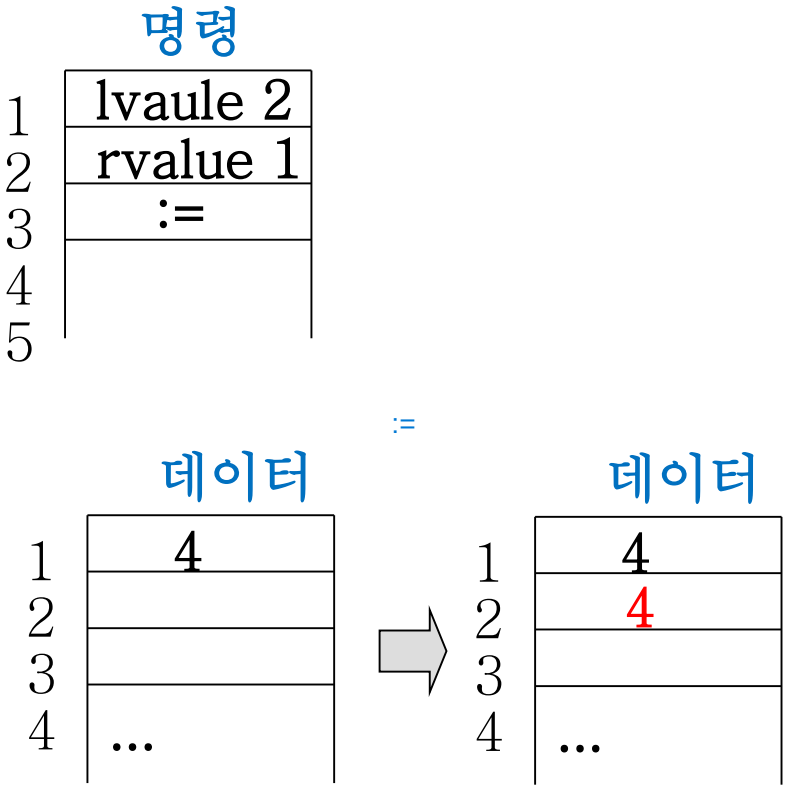
left value: 주소 값

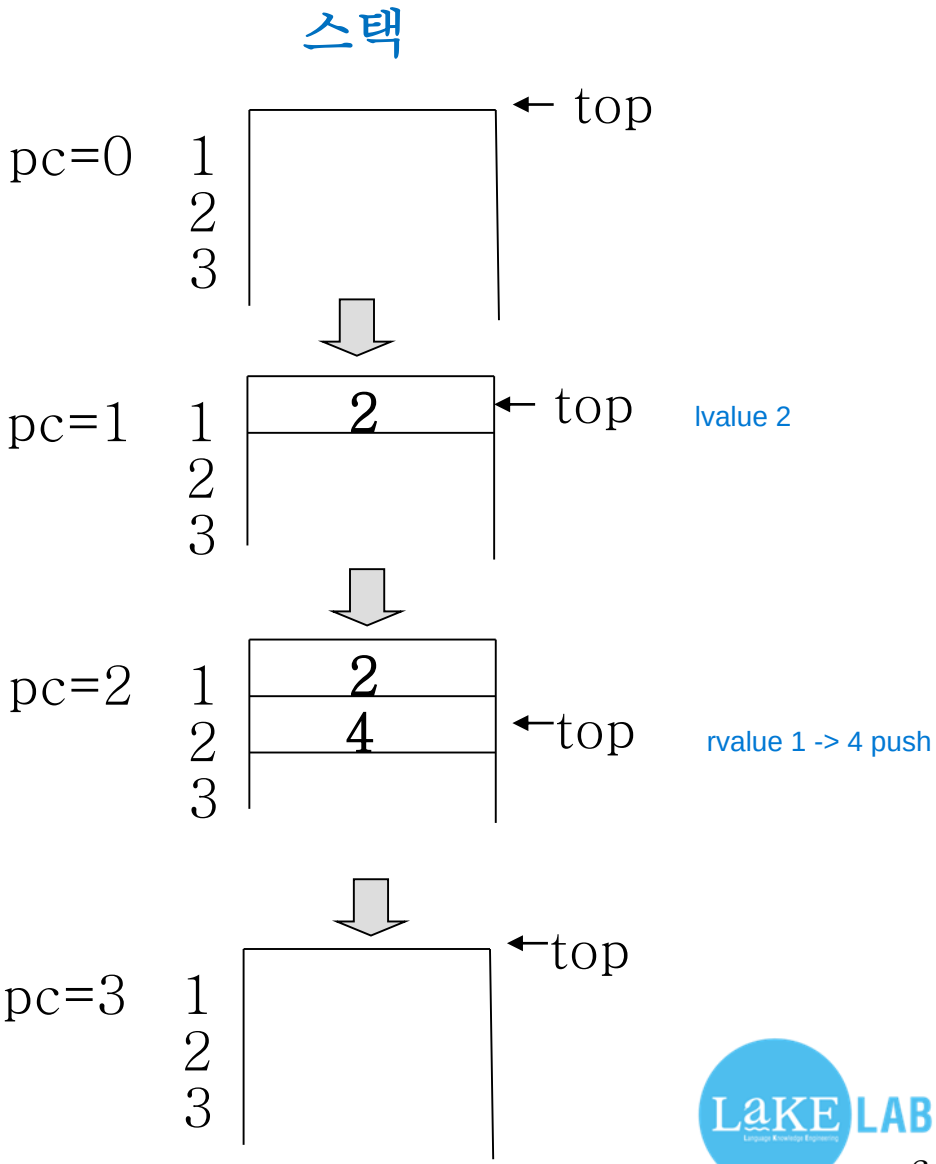
right value: 메모리 내용의 값

스택



가상 스택 기계의 수행 예-2





스택의 관리 명령

스택 관련 명령어

- push *v* : *v*를 스택에 푸시
- rvalue *l*: 데이터 저장장소 *l*의 내용을 푸시
- lvalue *l*: 데이터 저장장소 *l*의 주소를 푸시
- pop: 스택 맨위의 값을 버린다
- := : 스택 맨위에 있는 r-값을 그 아래쪽의 l-값의 장소에 넣고 양쪽값 제거
- copy: 스택 맨위의 값을 스택에 복사해서 푸시

L-값과 R-값(l-value, r-value)

- L-값: 변수의 주소(대입문의 왼쪽 변수일 때)
- R-값: 변수의 실제 값(대입문의 오른쪽 변수일 때)

실행 흐름의 제어 명령

실행 흐름 제어 명령어

- label / : 분기가 되는 위치/의 지정
- goto /: 다음부터 명령어를 label /에서 가져옴
- gofalse /: 스택의 맨위의 값을 꺼내서 그 값이 0이면 분기
- gotrue /: 스택의 맨위 값을 꺼내서 그 값이 0이 아니면 분기
- halt : 실행 종료

true 및 false

- false – 스택의 맨 위 값이 0인 경우
- true – 스택의 맨 위 값이 0이 아닌 경우

실행 흐름의 제어

분기의 종류

- 현 가상기계에서 채택
  - 목표 위치를 기호로 명시
  - 코드를 줄여도 기호화된 주소의 변경이 불필요: 코드 효율성
- 다른 처리 방법
  - 명령의 오퍼랜드로 목표 위치(절대 위치) 명시
  - 명령의 오퍼랜드로 양 또는 음의 상대 목표 위치 명시

중간코드 생성- 더하기

고급언어	중간코드	중간코드 생성
x+y	rvalue x rvalue y +	expr -> id_1 + id_2 { expr.t := 'rvalue ' || id_1.lexeme || 'rvalue ' || id_2.lexeme || '+' }

파스 트리

expr

id + id

x y

x

semantic action y

후위 표기 기법이라 '+'가 마지막으로 나옴.

rvalue x
rvalue y
+

를 결국 풀어쓴거임.

중간코드 생성 - 배정문

고급언어	중간코드	중간코드 생성
sum:=x+y	lvalue sum	stmt -> id := expr
	rvalue x	{ stmt.t := 'lvalue' || id.lexeme ||
	rvalue y	expr.t ||
	+	':=' }
	:=	

sum:= 앞장의 expr
결국 다 갖다 붙이는 거임... ㅋㅋ

|| = 문자열 연결 연산자

expr.t =
rvalue x
rvalue y
+
:=

앞장 참고:

expr -> id_1 + id_2

{ expr.t :='rvalue ' ||id_1.lexeme ||
'rvalue ' ||id_2.lexeme ||
'+' }

중간코드 생성- if 문

고급언어

```
if expr then
   stmt
```

중간코드

if문

rval A
rval B
=
go false out
lvalue B
rvalue 1
:=

out1;

expr의 코드
gofalse out
stmt 코드
label out

중간코드 생성

```
stmt -> if expr then stmt1
        {out := newlabel;
         stmt.t := expr.t          ||
                   'gofalse' out   ||
                   stmt1.t         ||
                   'label' out }
```

out -> 새로운 label 출력

expr.t =
rvalue x
rvalue y
+
:=

if A = B then B = 1 파스트리

stmt

stmt.t =
lvalue sum

if expr.t then stmt.t -> lv B rv 1 :=

Id.t = A Id.t = B

A = B

Id Id

B = 1

중간코드 생성- while 문

고급언어

```
while(expr)
   stmt
```

중간코드

while문

label test
expr의 코드
gofalse out
stmt 코드
goto test
label out

A
B
>
0 A
A
1
1
:=
goto less

중간코드 생성

```
stmt -> while(expr) stmt1
        { test := newlabel;
          out := newlabel;
          stmt.t := 'label' test   ||
                    expr.t         ||
                    'gofalse' out  ||
                    stmt1.t        ||
                    'goto' test    ||
                    'label' out }
```

8086 코드 생성 - 더하기

고급언어	중간코드	중간코드 생성

x+y	mov ax, x
	mov bx, y
	add ax, bx
		expr -> id_1 + id_2
		{ expr.t := 'mov ax,' || id_1.lexeme || nl ||
		'mov bx,' || id_2.lexeme || nl ||
		'add ax, bx' }

nl: new line

8086 코드 생성 - 배정문

고급언어	코드	코드 생성

sum:=x+y	mov ax, x
	mov bx, y
	add ax, bx
	store ax, sum
		stmt -> id := expr
		{ stmt.t := expr.t || nl ||
		'store ax, ' || id.lexeme }
		
		nl: new line

앞장 참고:

expr -> id_1 + id_2

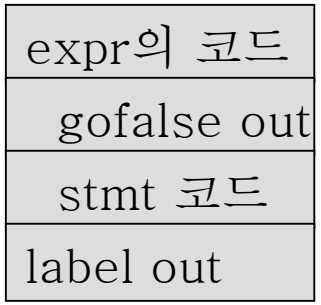
{ expr.t := 'mov ax,' || id_1.lexeme || nl ||
'mov bx,' || id_2.lexeme || nl ||
'add ax, bx' }

8086 코드 생성- if문

고급언어

```
if expr then
  stmt
```

코드



코드 생성

```
stmt -> if expr then stmt1
  { out := newlabel;
    stmt.t := expr.t   || nl   ||
    'cmp ax, 0'        || nl   ||
    'je'               || out || nl ||
    stmt1.t            || nl   ||
    out                || ':' }
```

8086 코드 생성- while문

고급언어

```
while(expr)
  stmt
```

코드

while문

label test
expr의 코드
gofalse out
stmt의 코드
goto test
label out

코드 생성

```
stmt -> while (expr) stmt1
    { test := newlabel;
      out := newlabel;
      stmt.t := test ':'          ||
                expr.t            || nl ||
                cmp ax, 0'        || nl ||
                'je '       || out|| nl ||
                stmt1.t           || nl ||
                'jmp' test        || nl ||
                out               || ':' }
```

참고 문헌

- [1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, “Compilers – Principles, Techniques, and Tools,” Bell Telephone Laboratories, Incorporated, 1986.
- [2] 오세만, “컴파일러 입문”, 정익사, 2004.