컴파일러

AST를 활용한 PL0 Intermediate Langauge 생성

학번: 201711050
이름: 김정우

Contents

1. 설계 내용

◆ 전체 흐름 ◆ grammar ◆ 해시 심볼 테이블

◆ if-then-else ◆ array variable

2. Basic PL0

◆ Mnemonic Code ◆ Binary Code ◆ Result

3. If-then-else extended PL0

◆ Mnemonic Code ◆ Binary Code ◆ Result

4. Array extended PL0

◆ Mnemonic Code ◆ Binary Code ◆ Result

1. 설계 내용

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

◆ 전체 흐름

- Lex에서 token 분석

- Yacc에서 syntax 분석
  & build AST

- s_CodeGen.c에서 AST Traverse
  & Symbol table 생성
  & AST의 subscript로 구문 확인
  & Action 수행

1. 설계 내용

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

◆ Lex에서 token 분석

```
%{
#include <string.h>
#include "s_mypl0Ast.h"
//#include "s_pl0Ast.tab.h"
Leaf nodeinfo;
int clev=0;
%}
%option noyywrap
%x  CMNT
 %
[ \t] |
"(*"([^*]|"*"+[^*/])*"*)" ;
"//".*   ;
"/*"        { BEGIN CMNT; clev++; }
<CMNT>"/*"   {clev++;}
<CMNT>.      |
<CMNT>\n     ;
<CMNT>"*/"  { if(!--clev) BEGIN INITIAL; }
"odd"       {cp+=3; return(ODD);}
"const"  {cp+=5; return(TCONST); }
"var"       {cp+=3; return(TVAR); }
"procedure"     {cp+=8; return(TPROC); }
"call"      {cp+=4; return(TCALL); }
"begin" {cp+=5; return(TBEGIN); }
"end"       {cp+=3; return(TEND); }
"if"        {cp+=2; return(TIF); }
"then"      {cp+=4; return(TTHEN); }
"else"      {cp+=4; return(TELSE); }
"while"     {cp+=5; return(TWHILE); }
"do"        {cp+=2; return(TDO); }
":="        {cp+=2; return(ASSIGN); }
"!="        {cp+=2; return(NE); }
"<="        {cp+=2; return(LE); }
">="        {cp+=2; return(GE); }
[0-9]+      { nodeinfo.num=atoi(yytext); yylval.ast=buildNode(-1,nodeinfo); cp+=yyleng; return NUM; } ;
[a-zA-Z][a-zA-Z0-9]*    {cp+=yyleng; strcpy(nodeinfo.ident, yytext); yylval.ast=buildNode(-2,nodeinfo); return(ID); }
.       { cp++; return yytext[0]; } ;
[\n]        { ln++; cp=0; }
 %
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
NORMAL  master!  s_pl0Ast.l
```

◆ yacc에서 token 분석 및 AST 생성

```
%right TTHEN TELSE
%left '+' '-'
%left '*' '/'
%left UM
%start Program
 %
Program: Block '.' {
        printf(" ==== valid syntax ====\n");
        printf("-- Print AST --\n"); printTree($1, 0);
        printf("-- CodeGen --\n"); Traverse($1);
        } ;
Block: Decl Statement       { $$=buildTree(0, linking($1, $2)); } ;
Decl: ConstDec VarDec ProcDef_list  { $$ = linking($1, linking($2, $3)); } ;
ConstDec:           { $$ = NULL; }
    | TCONST Constdef_list ';'  { $$ = buildTree(TCONST, $2); } ;
Constdef_list: Constdef_list ',' ConstDef  { $$ = linking($1, $3); }
    | ConstDef ;
ConstDef: ID '=' NUM        { $$ = linking($1, $3); } ;
VarDec: TVAR Ident_list ';'     { $$ = buildTree(TVAR, $2); } ;
    |           { $$ = NULL; } ;
Ident_list: Ident_list ',' IndexedDec       { $$ = linking($1, $3); }
     | IndexedDec ;
ProcDef_list: ProcDef_list ProcDef  { $$ = linking($1, $2); }
    |           { $$ = NULL; } ;
ProcDef: TPROC ID ';' Block ';'     { $$ = buildTree(TPROC, linking($2, $4)); } ;
Statement: Var ASSIGN Expression    { $$ = buildTree(ASSIGN, linking($1, $3)); }
    | TCALL ID      { $$ = buildTree(TCALL, $2); }
    | TBEGIN Statement_list TEND { $$ = buildTree(TBEGIN, $2); }
    | TIF Condition TTHEN Statement TELSE Else_extended
       { $$=buildTree(TIF,linking($2, linking($4, $6))); }
    | TIF Condition TTHEN Statement { $$ = buildTree(TIF, linking($2, $4)); }
    | TWHILE Condition TDO Statement { $$ = buildTree(TWHILE, linking($2, $4)); }
    | error { $$=NULL; }
    |  { $$=NULL; }  ;

Else_extended: Statement ;

Statement_list: Statement_list ';' Statement { $$ = linking($1, $3); }
    | Statement ;
Condition: ODD Expression       { $$ = buildTree(ODD, $2); }
    | Expression '=' Expression { $$ = buildTree('=', linking($1, $3)); }
    | Expression NE Expression  { $$ = buildTree(NE, linking($1, $3)); }
    | Expression '<' Expression { $$ = buildTree('<', linking($1, $3)); }
    | Expression '>' Expression { $$ = buildTree('>', linking($1, $3)); }
    | Expression GE Expression  { $$ = buildTree(GE, linking($1, $3)); }
    | Expression LE Expression  { $$ = buildTree(LE, linking($1, $3)); }  ;
Expression: Expression '+' Term { $$ = buildTree('+', linking($1, $3)); }
    | Expression '-' Term   { $$ = buildTree('-', linking($1, $3)); }
    | '+' Term %prec UM { $$=$2; }
    | '-' Term %prec UM { $$ = buildTree(NEG, $2); }
    | Term ;
Term:   Term '*' Factor     { $$ = buildTree('*', linking($1, $3)); }
    | Term '/' Factor       { $$ = buildTree('/', linking($1, $3)); }
    | Factor ;
Factor:     Var
    | NUM
    | '(' Expression ')'        { $$=$2; } ;
Var:        ID
    | ID '[' Expression ']'     { $$ = buildTree('[', linking($1, $3)); } ;
IndexedDec: ID
    | ID '[' NUM ']'        { $$ = buildTree('[', linking($1, $3)); } ;
 %
```

1. 설계 내용 - s_CodeGen.c에서의 symbol table

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

<Lookup 함수>

```
459 bool    Lookup(char *name, int type)
460 {
461     int idx = tx;
462     LDiff = -88;
463     OFFSET = -88;
464
465     int bucketIdx = hash(name, type);
466     int tableIdx = bucket[bucketIdx];
467     while (tableIdx >= 0)
468     {
469         if (strcmp(table[tableIdx].name, name) == 0 && table[tableIdx].type == type)
470         {
471             LDiff = level - table[tableIdx].lvl;
472             OFFSET = table[tableIdx].offst;
473             ARRSIZE = table[tableIdx].size;
474             return true;
475         }
476         tableIdx = table[tableIdx].link;
477     }
478     if (strcmp(table[tableIdx].name, name) == 0 && table[tableIdx].type == type)
479     {
480         LDiff = level - table[tableIdx].lvl;
481         OFFSET = table[tableIdx].offst;
482         ARRSIZE = table[tableIdx].size;
483         return true;
484     }
485     return false;
486 }
```

- Symbol의 hash를 이용해 symbol 검색
- 이름과 type이 같지 않으면, backward link를 통해 다음 symbol로 이동
- Symbol을 찾으면 level diff와 offset 받아와 전역변수 설정

<Enter 함수>

```
488 void Enter(char *name, int type, int lvl, int offst, int size)
489 {
490     if (Lookup(name, type))
491     {
492         if (LDiff == 0)
493         {
494             printf("Redefined error : [ name: %s, type: %d, level: %d, offst: %d ]\n",
495                     name, type, lvl, offst);
496             exit(1);
497         }
498     }
499     unsigned int bucketIdx = hash(name, type);
500     if (bucket[bucketIdx] != -1)
501     {
502         // backward linking
503         int origin = bucket[bucketIdx];
504         bucket[bucketIdx] = tx++;
505         table[bucket[bucketIdx]].link = origin;
506 #if DEBUG
507         printf("<<<<<<<<<<  backward linking occur!! >>>>>>>>>>>>\n");
508 #endif
509     }
510     else
511     {
512         bucket[bucketIdx] = tx++;
513     }
514     strcpy(table[bucket[bucketIdx]].name, name);
515     table[bucket[bucketIdx]].type = type;
516     table[bucket[bucketIdx]].lvl = lvl;
517     table[bucket[bucketIdx]].offst = offst;
518     table[bucket[bucketIdx]].size = size;
519 #if DEBUG
520     printf("[ Symbol Table Enter ] name: %s, type; %d, lvl: %d, offst: %d, link: %d, size: %d, hash: %d\n",
521             name, type, lvl, offst, table[bucket[bucketIdx]].link, size, bucketIdx);
522 #endif
523 }
```

- 동일한 symbol이 같은 level에 있는지를 검사한다.
- 없으면 Name와 type을 이용하여 hash값을 생성
- Hash값이 충돌되었는지를 검사하고, 충돌되었으면 backward link를 기존 symbol로 설정해준다.
- Table에 name, type, level, offst, size 등을 할당한다.

1. 설계 내용 - s_CodeGen.c에서의 symbol table

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

<SetBlock 함수>

```
631 void SetBlock(void)
632 {
633     block[level++] = tx;
634 }
635
```

- 현재의 table top index를 기록해둔다.
- 현재의 level을 1 증가시킨다.
- Block함수에서 재귀적으로 block으로 들어가기 전에 SetBlock을 호출시켜준다.

<ResetBlock 함수>

```
636 void ResetBlock(void)
637 {
638     int previousBlockTx = block[--level];
639
640     for (; tx > previousBlockTx; tx--)
641     {
642         int backlink = table[tx - 1].link;
643         if (backlink < 0)
644             continue;
645 #if DEBUG
646         printf("symbol name: %s -> [[[[[[  Reset Collision Chain ]]]]]]]]\n", table[tx - 1].name);
647 #endif
648         char *name = table[tx - 1].name;
649         int type = table[tx - 1].type;
650         bucket[hash(name, type)] = backlink;
651     }
652 }
```

- 현재 Table top index부터 이전 level에서의 table top index까지 빼가며 backward chain들을 원상복구 해준다.
- Block함수에서 재귀호출된 Block함수가 리턴된 후에 ResetBlock을 호출시켜준다.

<Hash 함수>

```
616 unsigned int hash(char *name, int type)
617 {
618     unsigned int h = 0;
619
620     for (int i = 0; name[i] != '\0'; i++)
621     {
622         h = (h << 4) + name[i];
623     }
624     h = (h << 4) + type;
625 #if DEBUG
626     printf("name: %s, type: %d => hash : %u\n", name, type, (h % BKSIZE));
627 #endif
628     return h % BKSIZE;
629 }
```

- Name과 type에 따라 구분되어야 한다.
  - Name이 같고 type이 다른데 hash값이 같으면 안됨.
  - 그 반대도 마찬가지
  - 최대한 이 규칙을 지키도록 구성하였다.
- 마지막에 구한 hash값을 bucket size로 모듈러 연산하여 bucket 안에 들어가도록 함

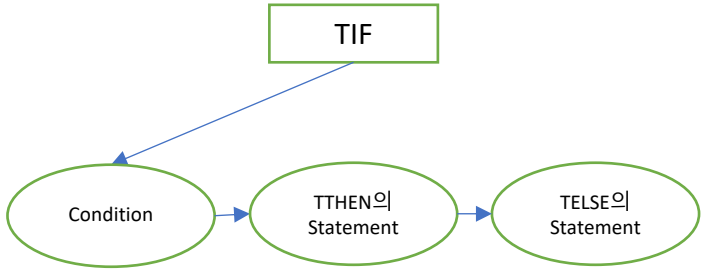
1. 설계 내용 – if-then-else

◆ syntax

Statement: TIF Condition TTHEN Statement TELSE Else_extended
{ $$=buildTree(TIF,linking($2, linking($4, $6))); }

| TIF Condition TTHEN Statement
{ $$ = buildTree(TIF, linking($2, $4)); } ;

Else_extended: Statement ;



- **주의할 점**
  - TIF Cond TTHEN St TELSE St 구문에서 **shift-reduce conflict**가 발생한다.
  - 이는 TTHEN의 St를 본 직후, LR Parser는 **TIF Cond THEN St**를 Statement로 **Reduce**해야 할지 아니면 뒤의 **TELSE를 Shift**해야 할지에 대한 두가지 결정을 할 수 있기 때문

- **해결방법**

1. TTHEN의 **Precedence**를 TELSE보다 작게 해야 함
   - Reduce보다 **TELSE에 대한 Shift를 우선시** 하게 됨.

2. Shift/Reduce를 먼저 판단하지 않고, **Associativity**를 보고 판단하도록 함
   - **Right-associativity**를 주어 **TELSE부터 결합**하도록 한다.

1. 설계 내용 – if-then-else

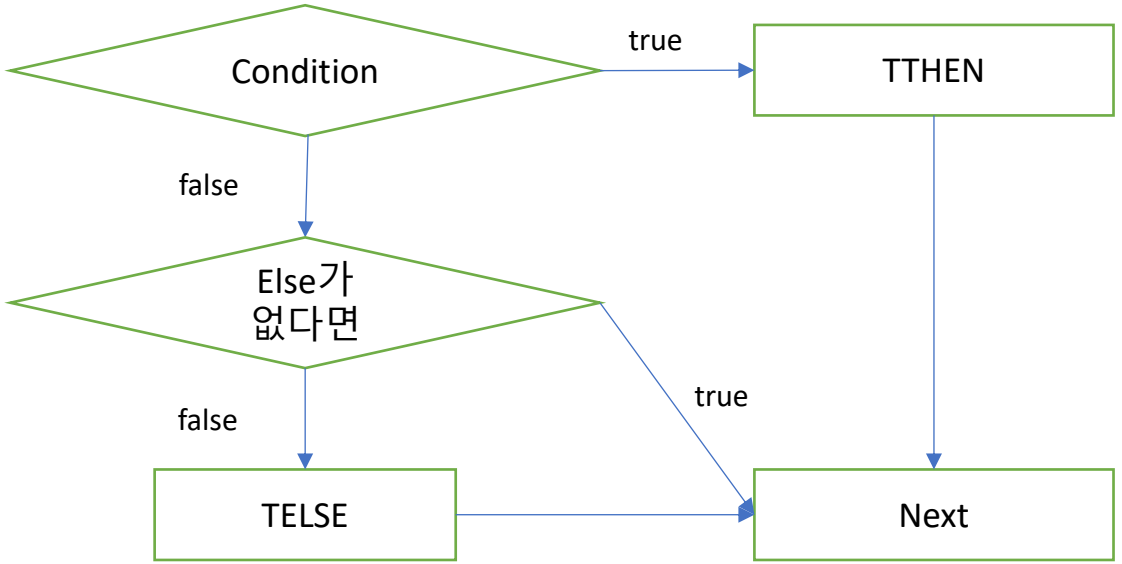
◆ if-then-else (in s_CodeGen.c)

```
case TIF:
{
    /**
     * TIF Condition TTHEN Statement TELSE Statement
     * temp : Condition
     * temp->right : Statement (then의 statement)
     * temp->right->right : Statement (else의 statement)
     */
    Condition(temp); //condition of if
    lab1 = GenLab(Lname1);
    Emit3("JPC", Jpc, lab1);
    Statement(temp->right);
    if (temp->right->right)
    {
        // after then
        lab2 = GenLab(Lname2);
        Emit3("JMP", Jmp, lab2);
    }
    EmitLab(lab1); // else or after if-then-else block
    if (temp->right->right)
    {
        Statement(temp->right->right);
        EmitLab(lab2);
    }
    break;
}
```

```
begin
   if f < g
   then g:=g-f
   else f:=f-g;
end;
```

```
        LOD     0       3
        LOD     0       4
        LT
        JPC     LAB4
        LOD     0       4
        LOD     0       3
        SUB
        STO     0       4
        JMP     LAB5
LAB4
        LOD     0       3
        LOD     0       4
        SUB
        STO     0       3
LAB5
```

◆ 로직



◆ 결과

1. f와 g를 Lod하고 LT로 Condition을 수행
2. **False라면**, LAB4로 Jmp 수행
   - if-then-else 구문 다음에 오는 code 또는 else 구문으로 Jmp함
   - f와 g를 Lod하여 f에 Sto.
3. **True라면**, g와 f를 Lod하여 뺄셈 수행 후 g에 Sto.
4. 맨 밑에서 LAB5로 Jmp를 수행, 다음 코드로 진행한다.

1. 설계 내용 – array variable

01 설계내용

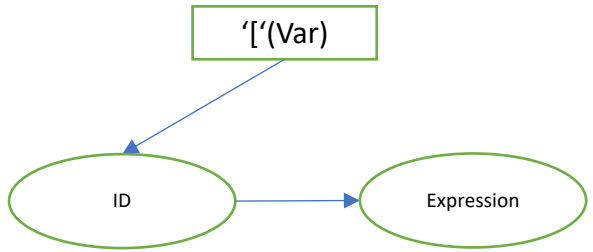
02 Basic PL0

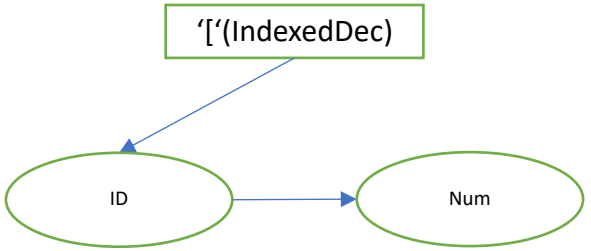
03 If-then-else extended

04 array extended

◆ syntax

```
Ident_list: Ident_list ',' IndexedDec      { $$ = linking($1, $3); }
     | IndexedDec ;

Statement: Var ASSIGN Expression      { $$ = buildTree(ASSIGN, linking($1, $3)); }

Factor:    Var
     | NUM
     | '(' Expression ')'                                  { $$=$2; } ;

Var:       ID
     | ID '[' Expression ']'          { $$ = buildTree('[', linking($1, $3)); } ;

IndexedDec: ID
     | ID '[' NUM ']'                                      { $$ = buildTree('[', linking($1, $3)); } ;
```





- 구문
  - Var : array 변수를 사용할 때의 구문
  - IndexedDec : array 변수를 선언할 때의 구문
    - Symbol table에 추가할 때 size를 알아야 하므로 따로 Rule을 추가함
- Subscript Variable '['
  - Symbol의 type을 지정해주기 위한 문자를 '['로 지정함
  - Symbol에 저장할 때 type을 4번으로 하여 저장한다.

1. 설계 내용 – array variable

01 설계내용

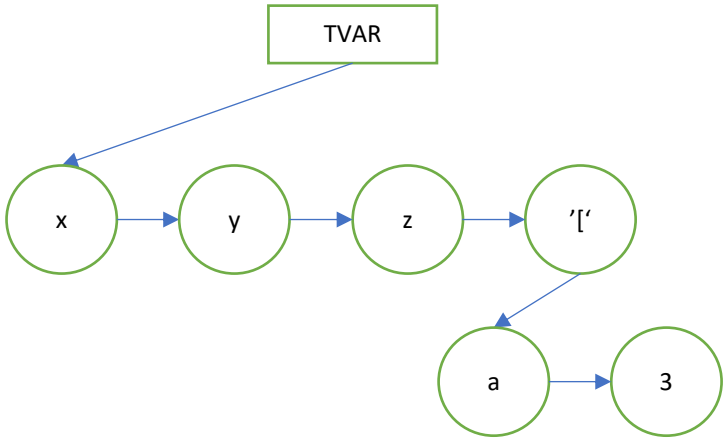
02 Basic PL0

03 If-then-else extended

04 array extended

◆ array variable (in s_CodeGen.c)

```
if (node && node->op == TVAR)
{
    link = node->left;
    while (link)
    {
        if (link->left)
        {
            myAstNode *tmp = link->left;
            int size = tmp->right->value.num;
            Enter(tmp->value.ident, INDEXED, level, offset, size);
            offset += size;
        }
        else
        {
            Enter(link->value.ident, VAR, level, offset++, -1);
        }
        link = link->right;
    }
    node = node->right;
}
```



- TVAR 토큰에서 array variable은 **한 level 더 내려와** 있기 때문에 일반 variable과 구분 가능하다.
- **Array의 Size**를 같이 **symbol table에 저장**한다.
- 현재 **Block의 크기**를 해당 **Size만큼 늘려준다**.

```
TVAR    259
    x
    y
    z
    q
    r
       91
        a
        3
```

◆ 결과

- AST를 출력할 경우 이와 같이 한 단계 더 들어가 있게 됨

1. 설계 내용 – array variable

01 설계내용

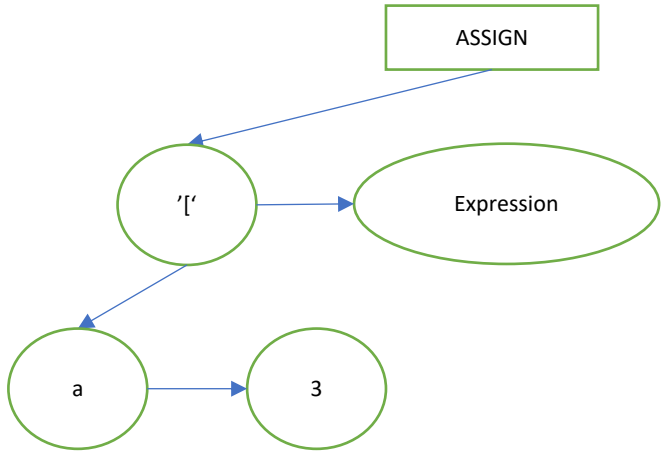
02 Basic PL0

03 If-then-else extended

04 array extended

◆ array variable (in s_CodeGen.c)

```
case ASSIGN:
{
    /**
     * Var ASSIGN Expression
     * temp : Var
     * temp->right : Expression
     */
    if (temp->op == '[')
    {
        if (!Lookup(temp->left->value.ident, 4))
        {
            printf("reference error: variable : %s is not an array\n", temp->left->value.ident);
            exit(1);
        }
        if (ARRSIZE <= temp->left->right->value.num)
        {
            sprintf(warning[warningIdx++], "warning: array index %d is past the end of the array (which c
ontains %d elements)\n",
                    temp->left->right->value.num, ARRSIZE);
        }
        Expression(temp->left);
        Expression(temp->left->right);
        Emit("ADD", 2);
        Expression(temp->right);
        Emit1("STI", Sti, LDiff, OFFSET);
        break;
    }
    else
    {
        Expression(temp->right);
        if (Lookup(temp->value.ident, 1))
        {
            Emit1("STO", Sto, LDiff, OFFSET);
            break;
        }
    }
    printf("undefined variable symbol error : %s\n", temp->value.ident);
    exit(1);
    break;
}
case TCALL:
```



- ASSIGN의 자식이 '[' 문자인지를 확인하여 array variable임을 구분한다.
- A[3]의 연산은 다음과 같다.
  - A의 주소 + 3
- 즉, array variable symbol인 A를 발견하면, **Lod 대신 Lda를 수행**해야 한다.
- Lda로 A의 주소를 push, Lti로 3을 push 한 후, **ADD 연산**을 수행하면, **A[3]의 주소가 Stack에 Push**된다.

◆ **고려사항**

- A[3] = b와 b = A[3]과는 동작이 다르다.
- A[3]을 보면 A와 3에 대해 ADD연산을 수행한다.
- A[3]이 LHS에 나오는 구문이라면, **Ldi를 수행한 후 Sti**로 배열의 Content를 저장해야 한다.
- A[3]이 RHS에 나오는 구문이라면, 위 Add연산의 결과를 끝으로 끝낸다.

1. 설계 내용 – array variable

◆ array variable (in s_interpreter.c)

<Lda Instruction>

```
case Lda: s[++sp] = addr;
```

- Stack Top에 Array variable의 Symbol의 주소를 push한다.

<Ldi Instruction>

```
case Ldi: s[sp] = s[s[sp]];
```

- Stack Top에 있는 것은 '**주소**'값임
- 이 주소 값에 해당하는 공간에 들어 있는 '**값**'을 Stack Top에 Push한다.

<Sti Instruction>

```
case Sti:
        s[s[sp-1]] = s[sp];
        if (s[sp-1] != sp)
                sp -=2;
        else
                --sp;
```

- Stack Top에 있는 **값을** Stack Top-1에 들어있는 **주소 값에 해당하는 공간에** 할당한다.
- Sti를 수행하기 위해 사용했던 **Stack 공간을 다시 원상복귀** 해야 한다.
  - 만약 Stack Top에 Sti를 한것이라면, sp를 1감소
  - 그 외에는 2감소해야 한다.

2. Mnemonic Code – Basic

◆ PL0 Code

```
❯ cat ../sample/sample_basic.pl0
const m=84, n=36;
var x,y,z,q,r;

procedure gcd;
        var f,g;
        begin f:=x; g:=y;
        while f != g do
             begin if f < g then g:=g-f;
                  if g < f then f:=f-g;
             end;
        z:=f
        end;
begin
        x:=m; y:=n; call gcd;
end.
```

2. Mnemonic Code & Binary Code – Basic

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

◆ Mnemonic Code

```
        JMP     LAB0
gcd
        JMP     LAB1
LAB1
        INT     0       5
        LOD     1       3
        STO     0       3
        LOD     1       4
        STO     0       4
LAB2
        LOD     0       3
        LOD     0       4
        NE
        JPC     LAB3
        LOD     0       3
        LOD     0       4
        LT
        JPC     LAB4
        LOD     0       4
        LOD     0       3
        SUB
        STO     0       4
LAB4
        LOD     0       4
        LOD     0       3
        LT
        JPC     LAB5
        LOD     0       3
        LOD     0       4
        SUB
        STO     0       3
LAB5
        JMP     LAB2
LAB3
        LOD     0       3
        STO     1       5
        RET
LAB0
        INT     0       8
        LIT     0       84
        STO     0       3
        LIT     0       36
        STO     0       4
        CAL     0       gcd
        END
```

◆ Binary Code

0	6	0	31
1	6	0	2
2	5	0	5
3	2	1	3
4	3	0	3
5	2	1	4
6	3	0	4
7	2	0	3
8	2	0	4
9	1	0	9
10	7	0	28
11	2	0	3
12	2	0	4
13	1	0	10
14	7	0	19
15	2	0	4
16	2	0	3
17	1	0	3
18	3	0	4
19	2	0	4
20	2	0	3
21	1	0	10
22	7	0	27
23	2	0	3
24	2	0	4
25	1	0	3
26	3	0	3
27	6	0	7
28	2	0	3
29	3	1	5
30	1	0	0
31	5	0	8
32	0	0	84
33	3	0	3
34	0	0	36
35	3	0	4
36	4	0	1
37	1	0	7
38	0	0	0

◆ Result (time: 18.857s)

```
~/Documents/Univ/22년 2학 기 /compiler/Final-project/source master*
❯ time ./a.out < ../sample/sample_basic.pl0
```

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

2. Mnemonic Code – if-then-else

◆ PL0 Code

```
❯ cat sample/sample_else.pl0
const m=84, n=36;
var x,y,z,q,r;

procedure gcd;
        var f,g;
        begin f:=x; g:=y;
        while f != g do
             begin
                        if f < g
                        then g:=g-f
                        else f:=f-g;
             end;
        z:=f
        end;
begin
        x:=m; y:=n; call gcd;
end.
```

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

2. Mnemonic Code & Binary Code – if-then-else

◆ Mnemonic Code

```
        JMP     LAB0
gcd
        JMP     LAB1
LAB1
        INT     0       5
        LOD     1       3
        STO     0       3
        LOD     1       4
        STO     0       4
LAB2
        LOD     0       3
        LOD     0       4
        NE
        JPC     LAB3
        LOD     0       3
        LOD     0       4
        LT
        JPC     LAB4
        LOD     0       4
        LOD     0       3
        SUB
        STO     0       4
        JMP     LAB5
LAB4
        LOD     0       3
        LOD     0       4
        SUB
        STO     0       3
LAB5
        JMP     LAB2
LAB3
        LOD     0       3
        STO     1       5
        RET
LAB0
        INT     0       8
        LIT     0       84
        STO     0       3
        LIT     0       36
        STO     0       4
        CAL     0       gcd
        END
```

◆ Binary Code

0	6	0	28
1	6	0	2
2	5	0	5
3	2	1	3
4	3	0	3
5	2	1	4
6	3	0	4
7	2	0	3
8	2	0	4
9	1	0	9
10	7	0	25
11	2	0	3
12	2	0	4
13	1	0	10
14	7	0	20
15	2	0	4
16	2	0	3
17	1	0	3
18	3	0	4
19	6	0	24
20	2	0	3
21	2	0	4
22	1	0	3
23	3	0	3
24	6	0	7
25	2	0	3
26	3	1	5
27	1	0	0
28	5	0	8
29	0	0	84
30	3	0	3
31	0	0	36
32	3	0	4
33	4	0	1
34	1	0	7
35	0	0	0

◆ Result (time: 15.317s)

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

2. Mnemonic Code – array

◆ PL0 Code

```
❯ cat ../sample/sample_array.pl0
const m=84, n=36;
var a[5];

procedure gcd;
        var f[2];
        begin f[0]:=a[0]; f[1]:=a[1];
        while f[0] != f[1] do
             begin
                        if f[0] < f[1]
                        then f[1]:=f[1]-f[0]
                        else f[0]:=f[0]-f[1];
             end;
        a[2]:=f[0]
        end;
begin
        a[0]:=m; a[1]:=n; call gcd;
end.
```

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

2. Mnemonic Code

```
        JMP     LAB0
gcd
        JMP     LAB1
LAB1
        INT     0       5
        LDA     0       3
        LIT     0       0
        ADD
        LDA     1       3
        LIT     0       0
        ADD
        LDI     0       0
        STI     1       3
        LDA     0       3
        LIT     0       1
        ADD
        LDA     1       3
        LIT     0       1
        ADD
        LDI     0       0
        STI     1       3
```

```
LAB2
        LDA     0       3
        LIT     0       0
        ADD
        LDI     0       0
        LDA     0       3
        LIT     0       1
        ADD
        LDI     0       0
        NE
        JPC     LAB3
        LDA     0       3
        LIT     0       0
        ADD
        LDI     0       0
        LDA     0       3
        LIT     0       1
        ADD
        LDI     0       0
        LT
        JPC     LAB4
        LDA     0       3
        LIT     0       1
        ADD
        LDA     0       3
        LIT     0       1
        ADD
        LDI     0       0
        LDA     0       3
        LIT     0       0
        ADD
        LDI     0       0
        SUB
        STI     0       3
        JMP     LAB5
```

```
LAB4
        LDA     0       3
        LIT     0       0
        ADD
        LDA     0       3
        LIT     0       0
        ADD
        LDI     0       0
        LDA     0       3
        LIT     0       1
        ADD
        LDI     0       0
        SUB
        STI     0       3
LAB5
        JMP     LAB2
LAB3
        LDA     1       3
        LIT     0       2
        ADD
        LDA     0       3
        LIT     0       0
        ADD
        LDI     0       0
        STI     0       3
        RET
LAB0
        INT     0       8
        LDA     0       3
        LIT     0       0
        ADD
        LIT     0       84
        STI     0       84
        LDA     0       3
        LIT     0       1
        ADD
        LIT     0       36
        STI     0       36
        CAL     0       gcd
        END
```

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

3. Binary Code

0	6	0	76
1	6	0	2
2	5	0	5
3	8	0	3
4	0	0	0
5	1	0	2
6	8	1	3
7	0	0	0
8	1	0	2
9	9	0	0
10	10	1	3
11	8	0	3
12	0	0	1
13	1	0	2
14	8	1	3
15	0	0	1
16	1	0	2
17	9	0	0
18	10	1	3
19	8	0	3
20	0	0	0
21	1	0	2
22	9	0	0
23	8	0	3
24	0	0	1
25	1	0	2
26	9	0	0
27	1	0	9
28	7	0	67
29	8	0	3
30	0	0	0
31	1	0	2
32	9	0	0
33	8	0	3
34	0	0	1
35	1	0	2
36	9	0	0
37	1	0	10
38	7	0	53
39	8	0	3
40	0	0	1
41	1	0	2
42	8	0	3
43	0	0	1
44	1	0	2
45	9	0	0
46	8	0	3
47	0	0	0
48	1	0	2
49	9	0	0
50	1	0	3
51	10	0	3
52	6	0	66
53	8	0	3
54	0	0	0
55	1	0	2
56	8	0	3
57	0	0	0
58	1	0	2
59	9	0	0
60	8	0	3
61	0	0	1
62	1	0	2
63	9	0	0
64	1	0	3
65	10	0	3
66	6	0	19
67	8	1	3
68	0	0	2
69	1	0	2
70	8	0	3
71	0	0	0
72	1	0	2
73	9	0	0
74	10	0	3
75	1	0	0
76	5	0	8
77	8	0	3
78	0	0	0
79	1	0	2
80	0	0	84
81	10	0	84
82	8	0	3
83	0	0	1
84	1	0	2
85	0	0	36
86	10	0	36
87	4	0	1
88	1	0	7
89	0	0	0

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

4. Interpreter Result

◆ Result (time: 39.174s)

01 설계내용

02 Basic PL0

03 If-then-else extended

04 array extended

Appendix

◆ error 처리

```
const m=84, n=36;
var x,y,z,q,r,a[3];

procedure gcd;
//      var f,g;
        var f[2];
        begin f[0]:=x; f[1]:=y;
        while f[0] != f[1] do
             begin if f[0] < f[1] then f[1]:=f[1]-f[0] else f[0]:=f[0]-f[1];
             end;
        a[1]:=f[0]
        end;
begin
        x[1]:=m; y:=n; call gcd;
        z:=a[1];
        a[a[a[2]]]:=a[4+3*3-a[1]];
        a[a[2]+2]:=a[1];
        a[7]:=1;
end.
```

- 결과

```
LAB3
        LDA     1       8
        LIT     0       1
        ADD
        LDA     0       3
        LIT     0       0
        ADD
        LDI     0       0
        STI     0       3
        RET
LAB0
        INT     0       11
reference error: variable : x is not an array

~/Documents/Univ/22년 2학기 /compiler/Final-project/source master*
```

◆ warning 처리

```
❯ cat sample/sample_warning.pl0
const m=84, n=36;
var x,y,z,q,r,a[3];

procedure gcd;
//      var f,g;
        var f[2];
        begin f[0]:=x; f[1]:=y;
        while f[0] != f[1] do
             begin if f[0] < f[1] then f[1]:=f[1]-f[0] else f[0]:=f[0]-f[1];
             end;
        a[1]:=f[0]
        end;
begin
        x:=m; y:=n; call gcd;
        z:=a[1];
        a[a[a[2]]]:=a[4+3*3-a[1]];
        a[a[2]+2]:=a[1];
        a[7]:=1;
end.
```

- 결과

```
        ADD
        LIT     0       1
        STI     0       8
        END
warning: array index 7 is past the end of the array (which contains 3 elements)

------------------------------
=== start PL0 ===
=== execution result(global var. contents) ===
stack:  10      12
stack:  9       12
stack:  8       12
stack:  7       0
stack:  6       0
stack:  5       12
stack:  4       36
stack:  3       84
```

감사합니다.