Javaプログラミングのtip集

第6回

2007年9月30日
アイティエス

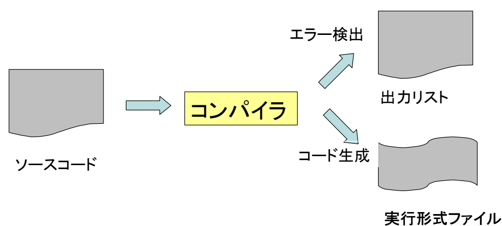
プログラム言語の処理系

プログラム言語・・・900種類余り存在
例　C、Java、Ruby、PHP、アセンブリ言語など

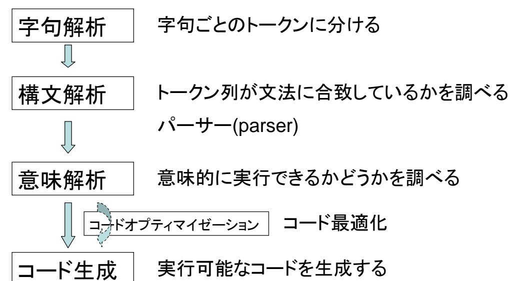
■主な処理系
➢コンパイラ、インタプリタ、アセンブラ

■主な機能
➢エラー検出、実行形式コード生成

処理系のイメージ



処理系の流れ



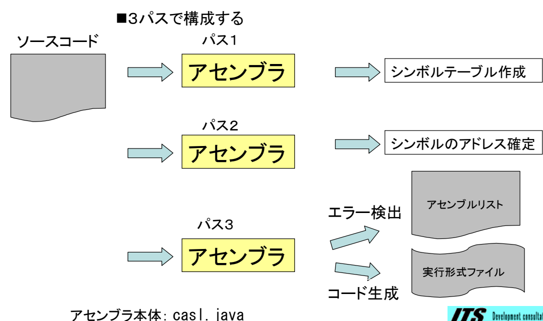
CASLアセンブラを作ろう

CASLは、仮想計算機COMETのためのアセンブリ言語であり、ソースプログラムを以下のように記述する

ラベル欄	ニーモニック欄	オペランド欄	注釈欄
label	mnemonic (op-code)	operand	comment

命令は1行に書き、次の行に継続はできない
注釈(コメント)行を1行のみで構成してもよい
オペコード:operation code

アセンブラの組み立て



アセンブラ本体: casl.java

字句解析で行うトークン振り分け

```
for(int m =0;m<cnt;m++){//トークン振り分け
        if(spflg == true){
            opcode = results[0];
            if(cnt == 3){
                operand = results[1];
                comment = results[2];
            }else if(cnt==2){
                operand = results[1];
            }else if(cnt == 1){
                operand = "";
            }
            label = "";
        }else{
            label = results[0];
            if(cnt == 4){
                opcode = results[1];
                operand = results[2];
                comment = results[3];
            }else if(cnt==3){
                opcode = results[1];
                operand = results[2];
            }else if(cnt==2){
                opcode = results[1];
            }
        }
}//for トークン
```

result[]に入っているトークンを
ニーモニック、オペランド、コメントに振り分ける

オペコード検出

```
LexicalCheck lexchk;
OpeCode code;
   ¦
lexchk = new LexicalCheck(code.valueOf(opcode));
codeinf = opeSelection(lexchk.getCode());
   ¦
```

codeinfにはニーモニック,オペコード,命令長の情報が入る

パスの構成

```
for(pass=1;pass<=3;pass++){
    ¦
    switch(pass){
        case 1://パス1
            if(semi == 0){
                pass1(label,spflg,ncode,incCode,indx);
            }
            break;
        case 2://パス2
            if(semi == 0){
                pc = pass2(crntPC,label,spflg,codeinf,operand);
            }
            break;
        case 3://パス3
            pc = pass3(semi,crntPC,label,spflg,codeinf,operand,comment);
            break;
    }
    ¦
}//pass loop
```

semi:ソースコードの先頭がセミコロンの場合コメント扱いにするフラグ

OpeCode.java

```
public enum OpeCode{
//Pseudo Instruction
    NOP,    // No Operation
    START,  // START
    DC,     // Define Constant
    DS,     // Define Storage
    END,    // END
    BREAK,  // BREAK
    STOP,   // STOP
     ¦
//General Instruction
    LD,     // Load
    ST,     // Store
    LEA,    // Load Effective Address
    ADD,    // ADD arithmetic
     ¦
}
```

ニーモニックの字句をenumで定義しておく

LexicalCheck.java

```
public class LexicalCheck{
    OpeCode opcode;
    public LexicalCheck(OpeCode code){
        opcode = code;
    }
    public String getCode(){
        switch(opcode){
//Pseudo Instruction
        case NOP:
            return ("NOP 01 4");
        case START:
            return ("START B0 0");
        case DC:
            return ("DC A0 0");
             ¦
//General Instruction
        case LD:
            return ("LD 10 4");
        case ST:
            return ("ST 11 4");
        case ADD:
            return ("ADD 20 4");
             ¦
        }
        return "";
    }
}
```

構成:
ニーモニック,オペコード,命令長の情報を持たせる

擬似命令

NOP 01 4

ニーモニック　オペコード　命令長

一般命令

パス3の実行

```
EX1   START
      LD GR0,WRK1
      SUBA GR0,WRK2
      ST GR0,ANS
      RET
WRK1  DC #0234
WRK2  DC #0011
ANS   DS 1
      END
```

ソースプログラム
EX1.cas

アセンブル

アセンブルリスト

```
>java casl EX1.cas
                      EX1     START
0000  10000010                LD     GR0,WRK1
0004  23000012                SUBA   GR0,WRK2
0008  11000014                ST     GR0,ANS
000C  81000000                RET
0010  0234            WRK1    DC     #0234
0012  0011            WRK2    DC     #0011
0014                  ANS     DS     1
                              END

>
```

現在アドレスは1バイト
構成で表示している