PROG → STATLIST EOF { PROG.CODE = STAT.CODE }

STATLIST → STAT { STAT.NEXT = NEWLABEL() }
STATLISTP { STATLISTP.NEXT = STATLIST.NEXT }
{ STATLIST.CODE = STAT.CODE || STAT.CODE : STATLISTP.CODE }

STATLISTP → ; STAT { STAT.NEXT = NEWLABEL() }
STATLISTP₁ { STATLISTP₁.NEXT = STATLISTP.NEXT }
{ STATLIST.CODE = STAT.CODE || STAT.CODE : STATLISTP₁.CODE }
| ε { STATLISTP.CODE = GOTO STATLISTP.NEXT }

STAT → ASSIGN ASSINGLIST { STAT.CODE = ASSIGNLIST.CODE || STAT.NEXT }
| READ ({ IDLIST.CONTEX = READ }
IDLIST) { STAT.CODE = IDLIST.CODE || STAT.NEXT }
| PRINT ({ EXPRLIST.CONTEX = PRINT }
EXPRLIST) { STAT.CODE = EXPRLIST.CODE || STAT.NEXT }
| { { STATLIST.NEXT = STAT.NEXT }
STATLIST } { STAT.CODE = STATLIST.CODE }
| { LTRUE = NEWLABEL(), LFALSE = NEWLABEL() }
IF ({ BEXPR.TRUE = LTRUE, BEXPR.FALSE = LFALSE }
BEXPR) { STAT₁.NEXT = STAT.NEXT }
STAT₁ { PELSE.TRUE = LTRUE, PELSE.FALSE = LFALSE, PELSE.NEXT = STAT.NEXT }
PELSE { STAT.CODE = BEXPR.CODE || LTRUE : STAT₁.CODE || PELSE.CODE }

{ LTRUE = NEWLABEL(), LFALSE = STAT.NEXT }
| FOR ({ BFOR.TRUE = LTRUE, BFOR.FALSE = LFALSE }
BFOR { STAT₁.NEXT = LTRUE }
) DO STAT₁
{ STAT.CODE = LTRUE : BFOR.CODE || STAT₁.CODE || LFALSE : }

BFOR → ID := EXPR ; BEXPR
| BEXPR
{ BEXPR.TRUE = BFOR.TRUE, BEXPR.FALSE = BFOR.FALSE }

ASSEGNA O NO IL EXPR + CODE IN ORDINE.

PELSE → ELSE { STAT.NEXT = PELSE.NEXT }
STAT END { PELSE.CODE = PELSE.FALSE : STAT.CODE }
| END " " " " " " " " "

BEXPR → RELOP EXPR₁ EXPR₂ { BEXPR.CODE = EXPR₁.CODE || EXPR₂.CODE || IF_ICMP RELOP TRUE || GOTO FALSE }
| OR BEXPR₁ BEXPR₂
| AND BEXPR₁ BEXPR₂
| NOT BEXPR

FARE JUMP IN BASE AL VALORE NELLO STACK
E MODIFICARE BEXPR PER VRAI ZE
BBEXPR → BEXPR
COSI BEXPR FA JUMP E OEXPR FA LOAD SULLO STACK

ASSIGNLIST → [EXPR TO { IDLIST.CONTEX = ASSIGN }
IDLIST] ASSIGNLISTP { ASSIGNLIST.CODE = EXPR.CODE || IDLIST.CODE || ASSIGNLIST.CODE }
ASSIGNLISTP → [EXPR TO IDLIST] ASSIGNLISTP " " " " " " " ↗
| ε " " " " NULLA " " " " " " "

IDLIST → ID { IDLISTP.CONTEX = IDLIST.CONTEX }
IDLISTP { IDLIST.CODE = IF (IDLIST.CONTEX = ASSIGN) DUP || ISTORE &ID || IDLISTP.CODE }
↳ IF (IDLIST.CONTEX = READ) INVOKESTATIC READ || ... || ... }

IDLISTP → , ID IDLISTP " " " " " " "
| ε { IDLIST.CODE = IF (IDLIST.CONTEX = ASSIGN) POP }

EXPR → + ({ EXPRLIST.CONTEX = EXPR, EXPRLIST.OPERATION = + }
EXPRLIST) { EXPR.CODE = LDC 0 || EXPRLIST.CODE }
| * (" " " OPERATION = *, LDC = 1 " " "
EXPRLIST)
| - EXPR₁ EXPR₂ { EXPR.CODE = EXPR₁.CODE || EXPR₂.CODE || ISUB }
| / EXPR₁ EXPR₂ { EXPR.CODE = EXPR₁.CODE || EXPR₂.CODE || IDIV }
| NUM { EXPR.CODE = LDC NUM.V }
| ID { EXPR.CODE = ILOAD &ID }

EXPRLIST → EXPR { EXPRLISTP.OPERATION = EXPRLIST.OPERATION, EXPRLISTP.CONTEX = ... }
EXPRLISTP { EXPRLIST.CODE = EXPR.CODE ||
IF (EXPRLIST.CONTEX = PRINT) INVOKESTATIC PRINT || EXPRLISTP.CODE }
IF (EXPRLIST.CONTEX = EXPR) EXPRLIST.OPERATION ↗

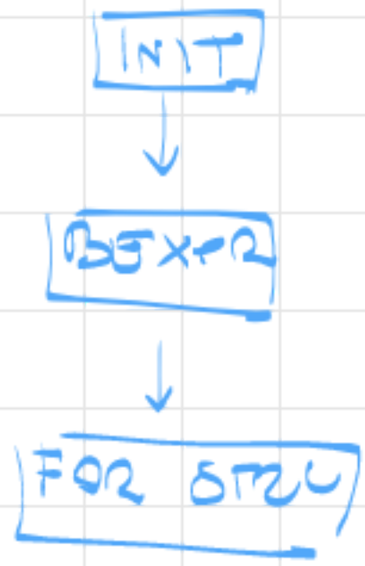
EXPRLISTP → , EXPR EXPRLISTP₁ " " " " " " " " " " "
| ε " " " " "

Struttura ASM IF/ELSE

```
CJUMP !COND LELSE

    S1
    GOTO NEXT
LELSE:

    S2
NEXT:
```

Struttura ASM FOR

```
LTRUE:
  CJUMP !COND LFALSE

    GOTO LTRUE
LFALSE:
```



PROG → STATLIST EOF

STATLIST → STAT STATLISTP

STATLISTP → ; STAT STATLISTP

STAT → ASSIGN ASSIGNLIST
| PRINT (EXPRLIST)
| READ (IDLIST)
| FOR (BFOR) DO STAT
| IF (BEXPR) STAT PELSE
| { STATLIST }

PELSE → ELSE STAT END
| END

BFOR → ID := EXPR ; BEXPR
| BEXPR

ASSIGNLIST → [EXPR TO IDLIST] ASSIGNLISTP

ASSIGNLISTP → [EXPR TO IDLIST] ASSIGNLISTP
| ε

IDLIST → ID IDLISTP

IDLISTP → , ID IDLISTP
| ε

BEXPR → RELOP EXPR EXPR

EXPR → + (EXPRLIST)
| * (EXPRLIST)
| - EXPR EXPR
| / EXPR EXPR
| NUM
| ID

EXPRLIST → EXPR EXPRLISTP

EXPRLISTP → , EXPR EXPRLISTP
| ε