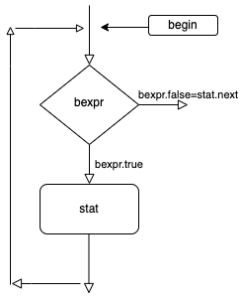
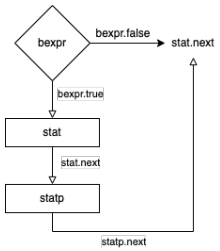
SDT

<prog> → <statlist> EOF	statlist.next = newlabel() prog.code = statlist.code || label(statlist.next) || 'stop'
<statlist> → <stat> <statlistp>	stat.next = newlabel() statlistp.next = statlist.next statlist.code= stat.code || label(stat.next) || statlistp.next
<statlistp> → ; <stat> <statlistp>	stat.next = newlabel() statlistp.next = statlistp.next statlistp.code = stat.code || label(stat.next) || statlistp.code
<statlistp> → ε	statlistp.code = " "
<stat> → assign <expr> to <idlist>	stat.code = idlist.code || dup || istore(addr(expr.lessema))
<stat> → print (<exprlist>)	stat.code = (print(exprlist.val)
<stat> → read (<idlist>)	stat.code = (read(idlist.val))
<stat> → while (<bexpr>) <stat>	begin = newlabel() bexpr.true=newlabel() bexpr.false=stat.next stat.next=begin stat.code=label(begin) || bexpr.code || label(bexpr.true) || stat.code || 'goto' stat.next



<stat> → if (<bexpr>) <stat> <statp> 	bexpr.true=newlabel() stat.next=newlabel() bexpr.false=stat.next statp.next=stat.next stat.code = bexpr.code || label(bexpr.true) || stat.code || label(stat.next || statp.code
<stat> → {<statlist>}	statlist.next=stat.next stat.code=statlist.code
<statp> → end	
<statp> → else <stat> end	stat.next=statp.next statp.code=stat.code
<idlist> → ID <idlistp>	idlistp.next=idlist.next idlist.code= iload &ID || idlistp.code
<idlistp> → , ID <idlistp1>	idlistp1.next=idlistp.next idlist.code= iload &ID || idlistp1.code
<idlistp> → ε	idlistp.code=""
<bexpr> → RELOP <expr> <expr>	bexpr=expr1.code || expr2.code || 'if_icmpRELOP' bexpr.true || 'goto' bexpr.false
<expr> → + (<exprlist>)	expr.code = exprlist.code || "iadd"
<expr> → * (<exprlist>)	expr.code = exprlist.code || "imul"
<expr> → - <expr1><expr2>	expr.code = expr1.code || "isub" || expr2.code

<expr> → / <expr1><expr2>	expr.code = expr1.code || “idiv” || expr2.code
<expr> → NUM	expr.code = ldc(NUM.val)
<expr> → ID	expr.code = iload(addr(ID.lessema))
<exprlist> → <expr> <exprlistp>	expr.next=newlabel() exprlistp.next=exprlist.next exprlist.code= expr.code || label(expr.next) || exprlistp.code
<exprlistpp> → , <expr> <exprlistp>	expr.next=newlabel() exprlistp.next=exprlist.next exprlist.code= expr.code || label(expr.next) || exprlistp.code
<exprlistp> → ε	exprlistp.code= “”

TRADUZIONE ‘ON-THE-FLY’

<prog> → {statlist.next = newlabel()} <statlist> {emitlabel(statlist.next)} EOF {emit(‘stop’)}

<statlist> → {stat.next = newlabel} <stat> {emitlabel(stat.next, statlistp.next = statlist.next}<statlistp>

<statlistp> → ; {stat.next = newlabel()} <stat> {emitlabel(stat.next), statlistp1.next = statlistp.next} <statlistp1>

<statlistp> → ε

<stat> → assign <expr> to <idlist> {emit(‘dup’), emit(istore(addr(expr.lessema))}

<stat> → print (<exprlist>) {emit(print(exprlist.val)}

<stat> → read (<idlist>) {emit(read(idlist.val))}

<stat> → while ({begin = newlabel(), emitlabel(begin), bexpr.true = newlabel(), bexpr.false = stat.next}<bexpr>){emitlabel(bexpr.true), stat.next=begin} <stat> {emit(‘goto’ stat.next)}

<stat> → if ({bexpr.true=newlabel(), bexpr.false=stat.next}<bexpr>){emitlabel(bexpr.true), stat.next=newlabel()} <stat> {emitlabel(stat.next), statp.next=stat.next} <statp>

<stat> → {{statlist.next=stat.next}<statlist>}

<statp> → end

<statp> → else {stat.next=statp.next}<stat> end

<idlist> → ID {idlistp.next=idlist.next} <idlistp> {emit(iload(addr(ID.lessema)))}

<idlistp> → , ID {idlistp1.next=idlistp.next} <idlistp1> {emit(iload(addr(ID.lessema)))}

<idlistp> → ε

<bexpr> → RELOP <expr><expr> {emit(if_icmpRELOP bexpr.true), emit(goto bexpr.false)}

<expr> → + ({<exprlist>.op-type=’plus’} <exprlist>)

```
<expr> → * {<exprlist>.op-type='mul'} (<exprlist>)

<expr> → - <expr1> <expr2> {emit(isub)}
<expr> → / <expr1> <expr2> {emit(idiv)}

<expr> → NUM {emit(ldc(NUM.val))}

<expr> → ID {emit(iload(addr(ID.lessema)))}

<exprlist> → <expr> {<exprlistp>.op-type=<exprlist>.op-type } <exprlistp>


<exprlistp> → , <expr> {if (<exprlistp>.op-type='plus') emit ('iadd') elseif (<exprlistp>.op-type='mul') emit('imul'),
<exprlistp1>.op-type=<exprlistp>.op-type}<exprlistp>
<exprlistp> → ε


// Le traduzioni di ⟨exprlist⟩ e ⟨exprlistp⟩ tengono conto solo del loro uso nelle espressioni aritmetiche.
```