

## Relational Expressions

- parsed with same grammar rules as arithmetic expressions
- have to keep track of what type of expression we are parsing (relational or arithmetic)
  - note that arithmetic expressions can be embedded in relational expressions
    - $(a + b) < c$
- relational embedded in control flow structures (IF and WHILE)
- code generated involves **jumps** to different locations depending on truth or falsity of the expression

## Generating TVI Code

- Generate branch statements
  - **goto, blt, beq, bne, bgt, blt, bge, ble**
- Usually, we have not yet generated the code that is the target of jumps

*Example:* `if (a<b) then c:=d;`

*When quadruple 24 is generated, 26 and 27 do not yet exist*

```
[24] blt a,b,26
[25] goto 27
[26] move d,c
[27] ..
```

when we parse  $a < b$  need to generate code that will branch to the statement  $c := d$  if  $a < b$  is true, branch to the statement after  $c := d$  otherwise

## Solution

- Generate the quadruples for relational expressions with targets left empty
  - same strategy as with ALLOC
- Keep a **list** of the quads that need to be filled in with the target quad for the TRUE case, another list of quads to be filled in for FALSE case
  - **These lists will be called E.TRUE and E.FALSE**
    - “expression true” and “expression false”

## Simple Relational Expressions

- Handled by productions for **<expression>** and **<expression-tail>**
  - note grammar does not allow e.g.  $a < b < c$
- **SA #38 : push RELOP**

## E.TRUE and E.FALSE

- Also in SA #39, make note of the quadruples just generated so their targets can be filled in later

- **SA #39 : generate code**

**E.TRUE := makelist(NEXTQUAD)**

**E.FALSE := makelist(NEXTQUAD+1)**

**GEN(if ID op ID goto \_\_\_\_\_)**

**GEN(goto \_\_\_\_\_)**

*empty targets*

- Need to hang on to E.TRUE and E.FALSE so push them on the stack