# COMPILERS Basic Blocks and Traces

hussein suleman uct csc3003s 2008

### **Evaluation Order**

- Its useful to evaluate the subexpressions of an expression in any order.
- Some IR trees can contain side effects.
- ESEQ and CALL can contain side effects
  - assignment
  - I/O
- It there were no side effects in these statements then the order of evaluation would not matter.

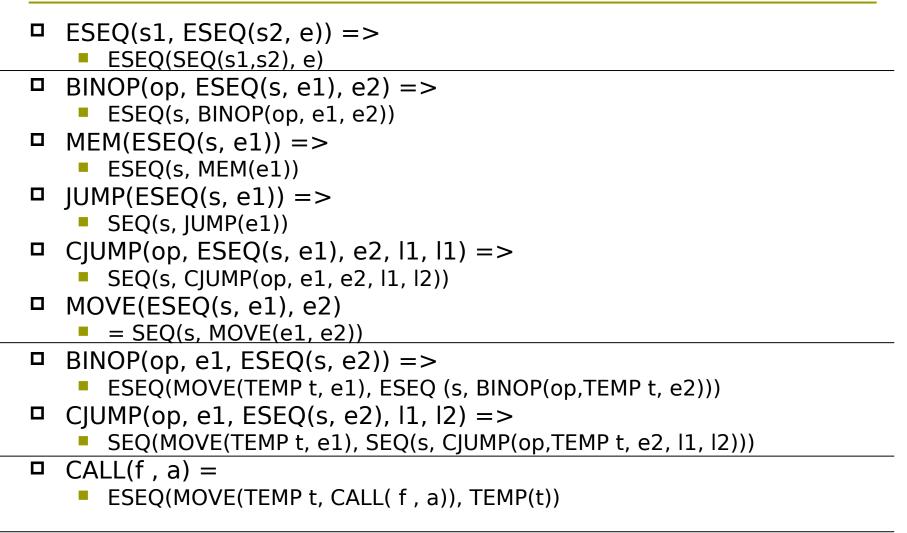
### **IR/MC** mismatches

- CJUMP jumps to one of two labels not one label and next instruction.
- ESEQ nodes within expressions make order of evaluation significant.
- CALL nodes within expressions make order of evaluation of parameters significant.
- CALL nodes within the argument of other CALL nodes make allocation of formalparameter registers difficult.

### **Canonical Trees**

- □ 1: No SEQ or ESEQ
- Description 2: CALL can only be subtree of EXP(..) or MOVE(TEMP t,..)
- Transformations:
  - Iift ESEQs up tree until they can become SEQs
  - turn SEQs into linear list

## **Simplification Rules**



## **General Technique**

#### For subexpressions of a node, e1..en,

- [e1, e2, ... ESEQ(s,ei), ..., en-1, en]
  - if s commutes with e1..ei-1 (independent),
    - (s; [e1, e2, ... ei, ..., en-1, en]
  - otherwise,
    - SEQ(MOVE(TEMP t1, e1),
    - SEQ(MOVE(TEMP t2, e2),
    - ... SEQ(MOVE(TEMP ti-1, ei-1)...))
    - [TEMP t1, TEMP t2, ... TEMP ti-1, ei, ..., en-1, en]

In general, extract children, reorder and then reinsert children

### **Basic Blocks**

- Divide linear sequence of nodes in each subprogram into basic blocks, where:
  - execution always starts at top and stops at bottom
  - first statement is a LABEL
  - Iast statement is a JUMP or CJUMP
  - no intervening LABELs, JUMPs or CJUMPs

Basic blocks are easier to work with for future optimisations since they can be rearranged, while maintaining logic.

# **Basic Blocks Algorithm**

- Scan sequence of statements from start to end
  - If LABEL, start new block
  - If JUMP or CJUMP, end block
- If a block does not start with a LABEL
  - Create new LABEL
- If a block does not end with JUMP/CJUMP
  - Create new JUMP to next LABEL
- Add terminal "JUMP done" for end of subprogram.

### Traces

- We want to rearrange basic blocks to optimise the number and nature of jumps.
- A trace is a sequence of statements that can be consecutively executed during the program execution (e.g., b1, b3, b6 below)
  - block b1: LABEL a ... JUMP b
  - block b3: LABEL b ... JUMP c
  - block b6: LABEL c ... CJUMP ?,a
- Every program has many overlapping traces – we want a single set that covers all the instructions.

### **Trace Generation**

- Put all basic blocks into a list Q
- while Q is not empty
  - Start a new (empty) trace T
  - Remove an element b from Q
    - while b is not marked
      - Mark b
      - Append b to T
      - Check succesors if b for unmarked node and make this the new b
    - End the trace T

# JUMP considerations

- We prefer CJUMP followed by its false label, since this translates to MC conditional jump.
- If CJUMP followed by its true label,
  - switch true and false labels, and negate conditional
- If CJUMP (cond, a, b, lt, lf) followed by some other label, replace with:
  - CJUMP (cond, a, b, lt, lfprime)
  - LABEL Ifprime
  - JUMP (NAME If)
- Remove all JUMPs followed by their target LABELs.