

# Compiler Construction

⌞ Basic Blocks ⌟

# Problem statement (1/2)

How to translate two way conditional jumps from HIR to LIR?

## Problem statement (2/2)

This does not exist in ASM

```
cjump xxx  $\alpha$   $\beta$ 
```

Since cjump (in assembly language) has a label, rather than an expression AND only one label...

```
cjump xxx Label_alpha  
 $\beta$   
...  
Label_alpha:  
 $\alpha$ 
```

# Two Way Jumps

Obviously, to enable the translation of a *cjump* into actual assembly instructions, the “false” label must follow the *cjump*.

## How?

# Basic Blocks

We must analyze the flow of the program

## Basic Block

A basic block is a sequence of statements that is always entered at the beginning and exited at the end.

- The first statement is a Label
- The last statement is a jump or a cjump
- There are no other Labels, jumps or cjump in the block

# Algorithm for building Basic Blocks

- 1 Scan the sequence from the beginning to the end
- 2 When a label is found, start a new block (and end the previous block)
- 3 Whenever a cjump/jump is found the current block is ended (and the next block is started)
- 4 If this leaves a block ending without a cjump/jump, then append a jump to the next block
- 5 If a block has no Label at the beginning, invent one, and add it

# Rearranging Basic Blocks

How basic blocks can help to solve two ways conditional jumps?

⇒ They will be used to build a correct trace for our program!

# Traces

## A trace

is a sequence of statements that could be consecutively executed during the execution of the program.

It can include conditional branches.



# Remarks on Traces

A program has many, different, overlapping traces.

For our purpose (arranging cjump) we want to make a set of traces that exactly covers the program: **each block must be in exactly one trace**

# Algorithm to build Traces (1/2)

## Main Idea

Start from the initial block, and “sew” each remaining basic block to this growing “trace”.

# Algorithm to build Traces (2/2)

- ① If the last instruction is a *jump*
  - ▶ if the “destination block” is available, add it
  - ▶ otherwise, fetch any other remaining block.
  
- ② If the last instruction is a *cjump*
  - ▶ If the false destination is available, push it
  - ▶ If the true destination is available, flip the *cjump* and push it,
  - ▶ otherwise, change the *cjump* to go to a fresh label, attach this label, and finally *jump* to the initial false destination.

# Two Way Jumps: Optimizing Traces

```
label prologue  
  Prologue.  
jump name test
```

```
label test  
cjump i <= N, body, done
```

```
label body  
  Body.  
jump name test
```

```
label done  
  Epilogue  
jump name end
```

# Two Way Jumps: Optimizing Traces

# Two Way Jumps: Optimizing Traces

```
label prologue  
  Prologue  
jump name test
```

```
label test  
cjump i > N,  
  done, body
```

```
label body  
  Body  
jump name test
```

```
label done  
  Epilogue  
jump name end
```

# Two Way Jumps: Optimizing Traces

```
label prologue  
  Prologue  
jump name test
```

```
label prologue  
  Prologue  
jump name test
```

```
label test  
cjump i > N,  
  done, body
```

```
label test  
cjump i <= N,  
  body, done
```

```
label body  
  Body  
jump name test
```

```
label done  
  Epilogue  
jump name end
```

```
label done  
  Epilogue  
jump name end
```

```
label body  
  Body  
jump name test
```

# Two Way Jumps: Optimizing Traces

```
label prologue  
  Prologue  
jump name test
```

```
label prologue  
  Prologue  
jump name test
```

```
label prologue  
  Prologue  
jump name test
```

```
label test  
cjump i > N,  
  done, body
```

```
label test  
cjump i <= N,  
  body, done
```

```
label body  
  Body  
jump name test
```

```
label body  
  Body  
jump name test
```

```
label done  
  Epilogue  
jump name end
```

```
label test  
cjump i <= N,  
  body, done
```

```
label done  
  Epilogue  
jump name end
```

```
label body  
  Body  
jump name test
```

```
label done  
  Epilogue  
jump name end
```



# Summary

