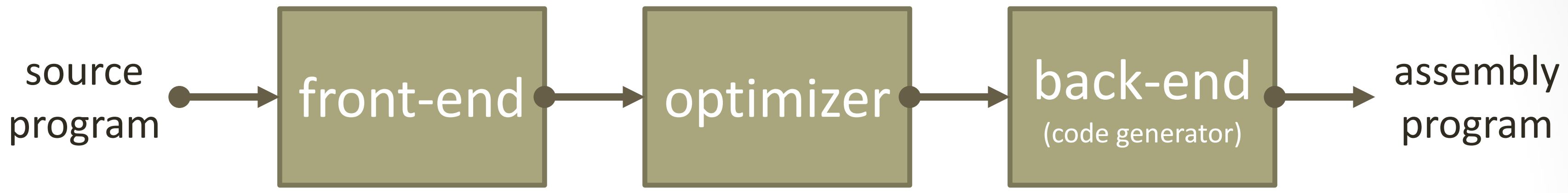
Rethinking Code Generation in Compilers

Christian Schulte

KTH & SICS

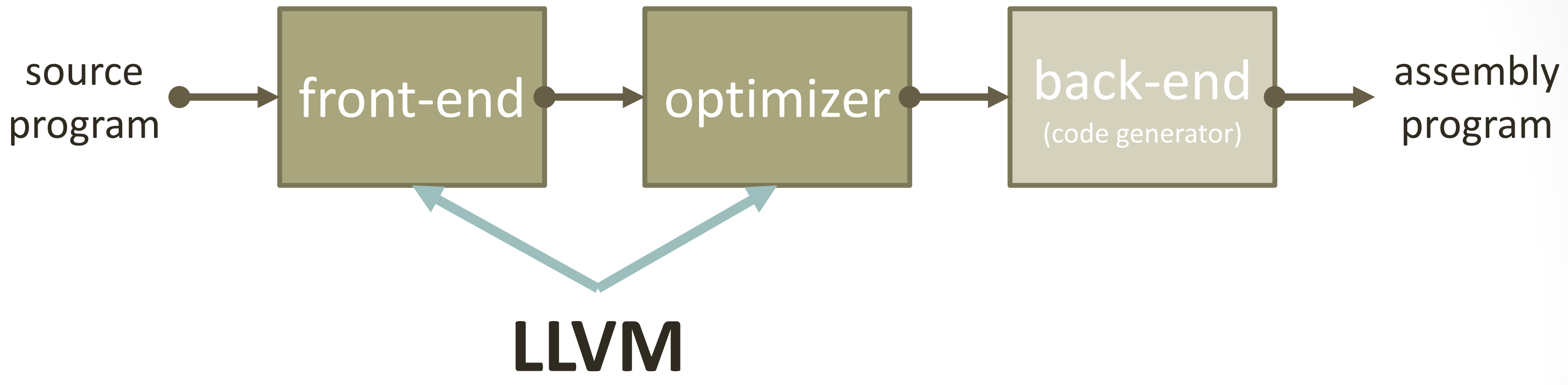
KTH Information and Communication Technology

Compilation



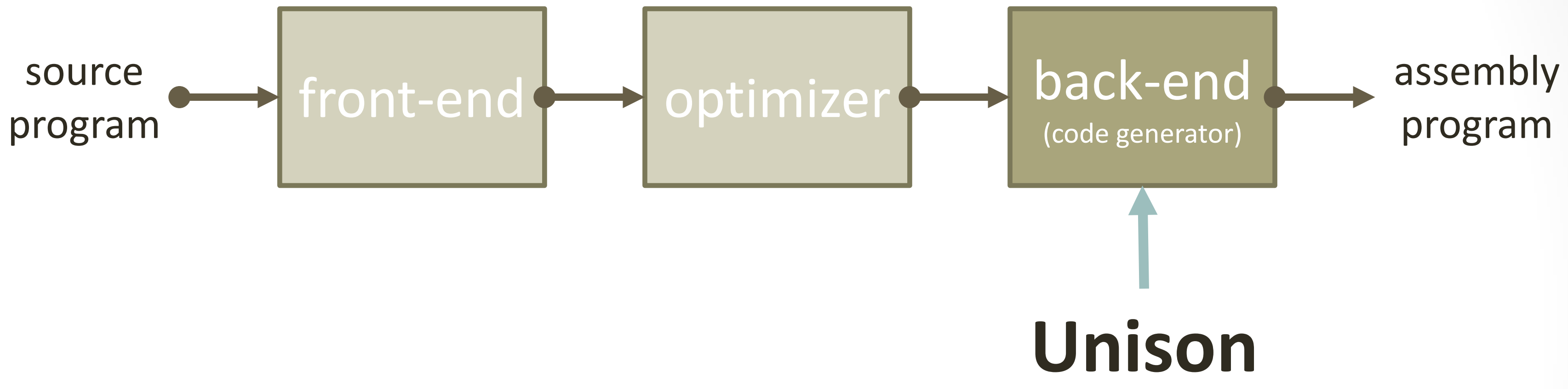
- Front-end: depends on source programming language
  - changes infrequently
- Optimizer: independent optimizations
  - changes infrequently
- Back-end: depends on processor architecture
  - changes often: new architectures, new features, …

Building a Compiler



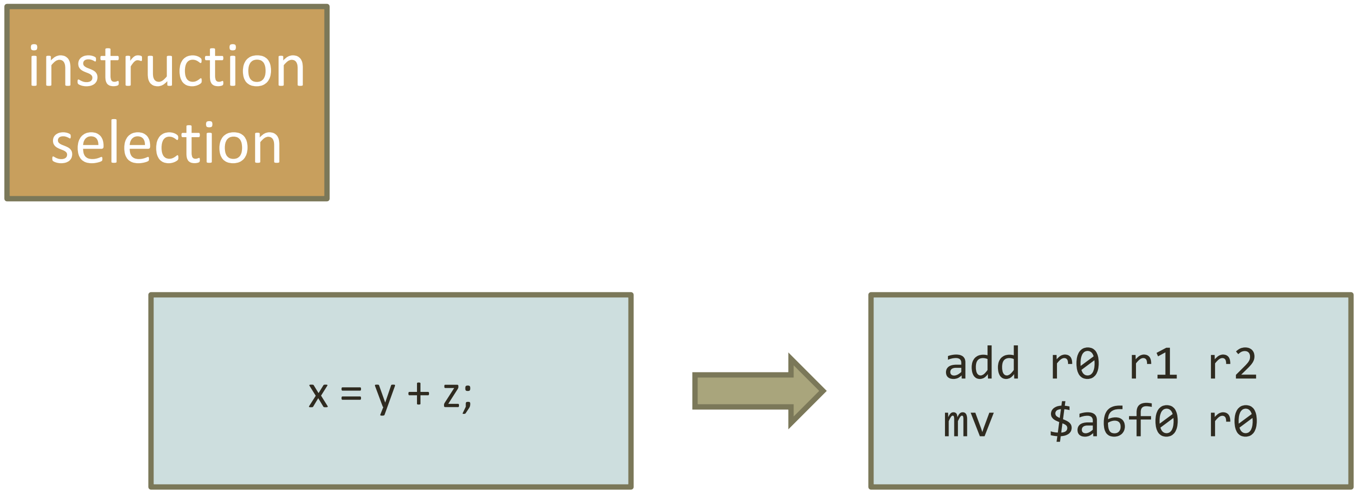
- Infrequent changes: front-end & optimizer
  - reuse state-of-the-art: LLVM, for example

Building a Compiler



- Infrequent changes: front-end & optimizer
    - reuse state-of-the-art: LLVM, for example
- Frequent changes: back-end
    - use flexible approach: **Unison (this project)**

State-of-the-art

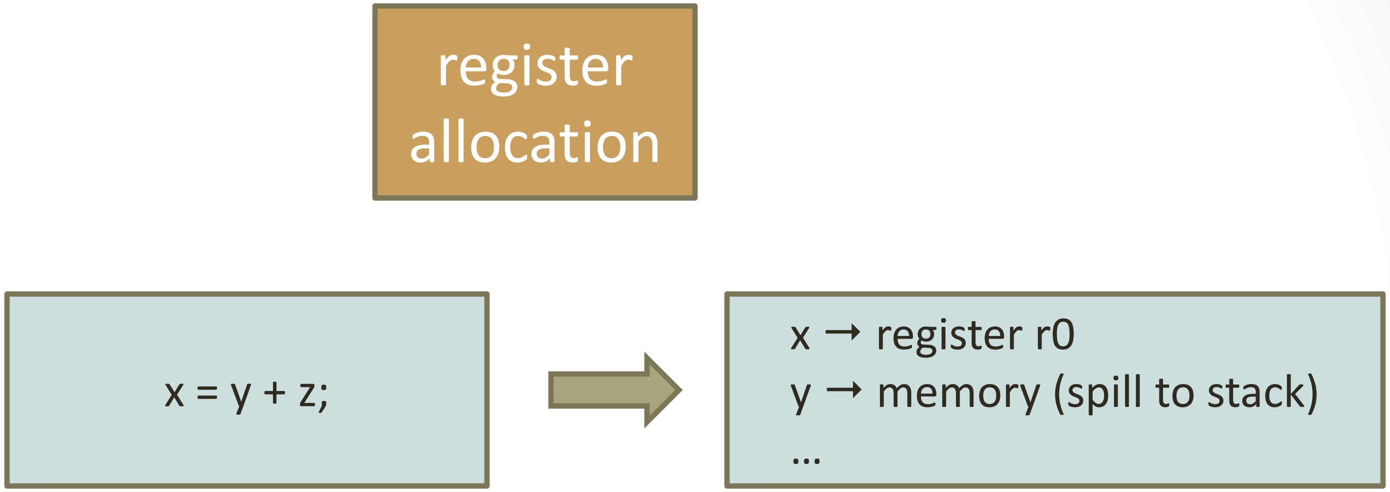


x = y + z;

```
add r0 r1 r2
mv  $a6f0 r0
```

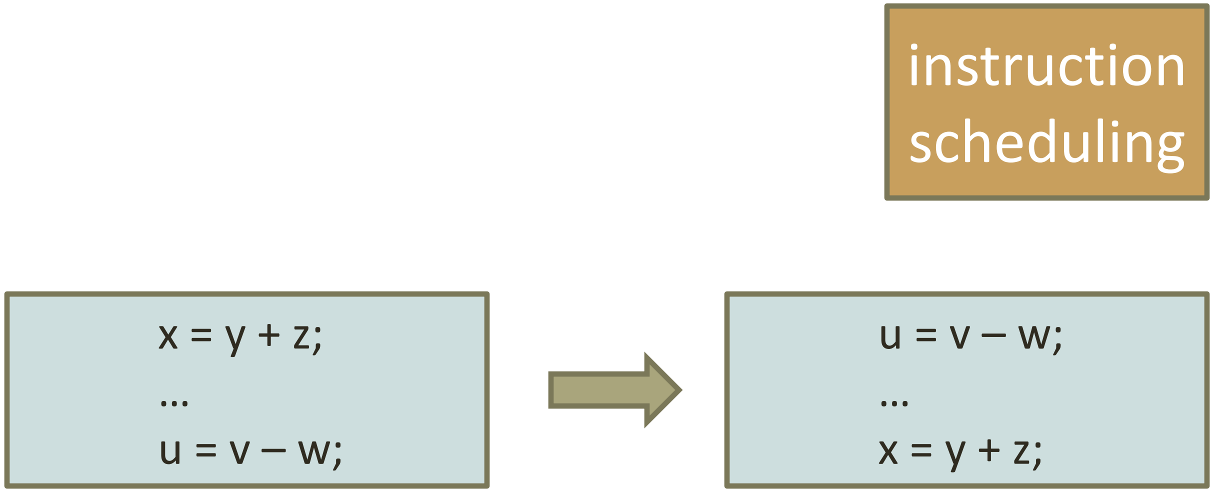
- Code generation organized into stages
    - instruction selection,

State-of-the-art



- Code generation organized into stages
  - instruction selection, register allocation,

State-of-the-art

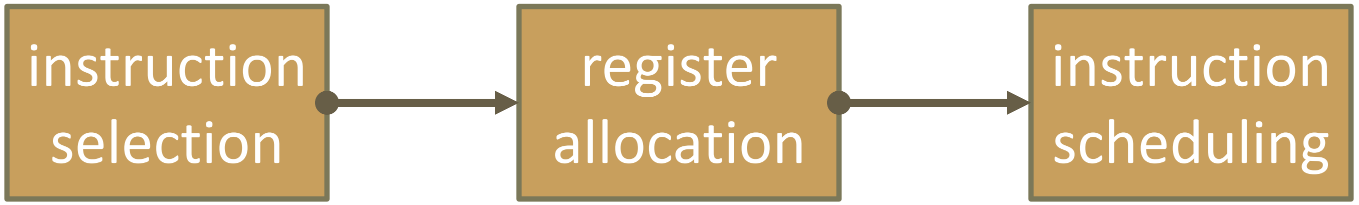


```
x = y + z;
…
u = v – w;
```

```
u = v – w;
…
x = y + z;
```

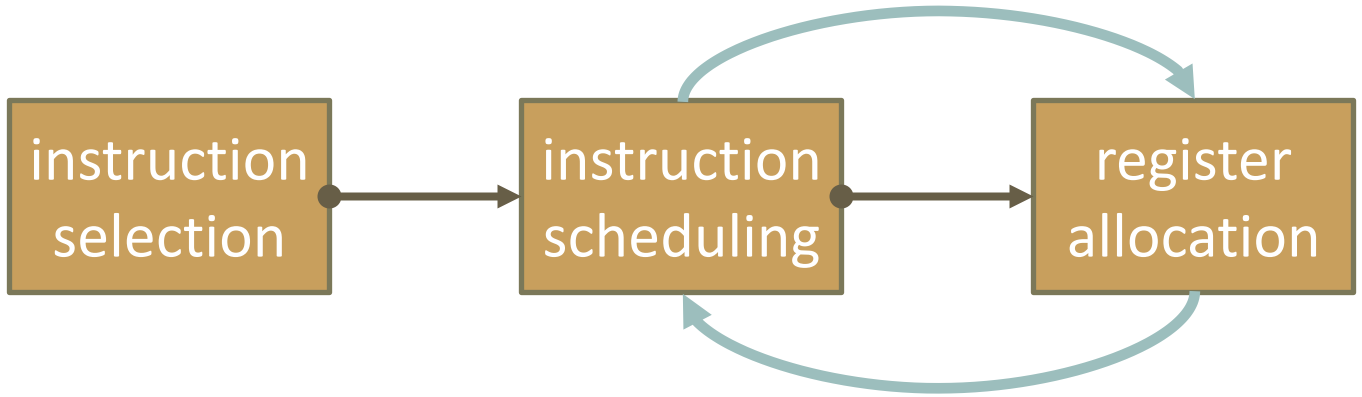
- Code generation organized into stages
    - instruction selection, register allocation, instruction scheduling

State-of-the-art



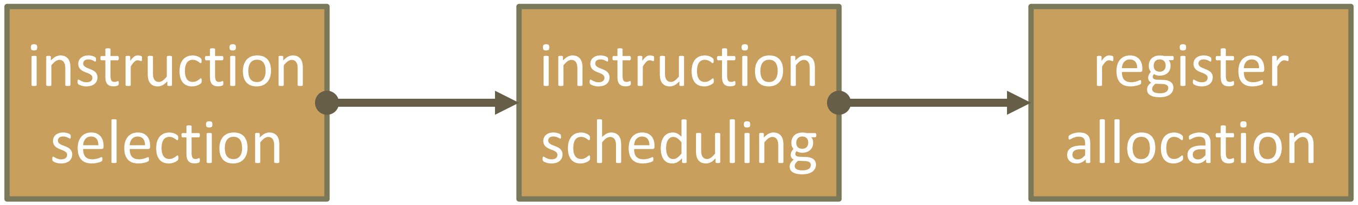
- Code generation organized into stages
  - instruction selection, register allocation, instruction scheduling
  - stages are interdependent: no optimal order possible

State-of-the-art



- Code generation organized into stages
    - instruction selection, register allocation, instruction scheduling
    - stages are interdependent: no optimal order possible

State-of-the-art



- Code generation organized into stages
  - instruction selection, register allocation, instruction scheduling
  - stages are interdependent: no optimal order possible

- Stages use heuristic algorithms
  - for hard combinatorial problems
  - assumption: optimal solutions not possible anyway
  - difficult to take advantage of processor features
  - error-prone when adapting to change

State-of-the-art

$$\begin{tikzcd}\boxed{\text{instruction selection}} \arrow[r] & \boxed{\text{instruction scheduling}} \arrow[r] & \boxed{\text{register allocation}}\end{tikzcd}$$

- Code generation organized into stages
    - instruction selection, register allocation, instruction scheduling
    - stages are interdeper lent: no optimal order possible
- Stages use simple algorith
    - for hard combinato
    - assumption: optimal
    - difficult to take advantage
    - error-prone when adapting to change

preclude optimal code, make development complex

Rethinking: Unison Idea

- No more staging and heuristic algorithms!
  - many assumptions are several decades old...
- Use state-of-the-art technology for solving combinatorial optimization problems: **constraint programming**
  - tremendous progress in last two decades...
- Generate and solve single model
  - captures all code generation tasks in unison
  - high-level of abstraction: based on processor description
  - flexible: ideally, just change processor description
  - potentially optimal: tradeoff between decisions accurately reflected

Constraint Programming

- Model problem

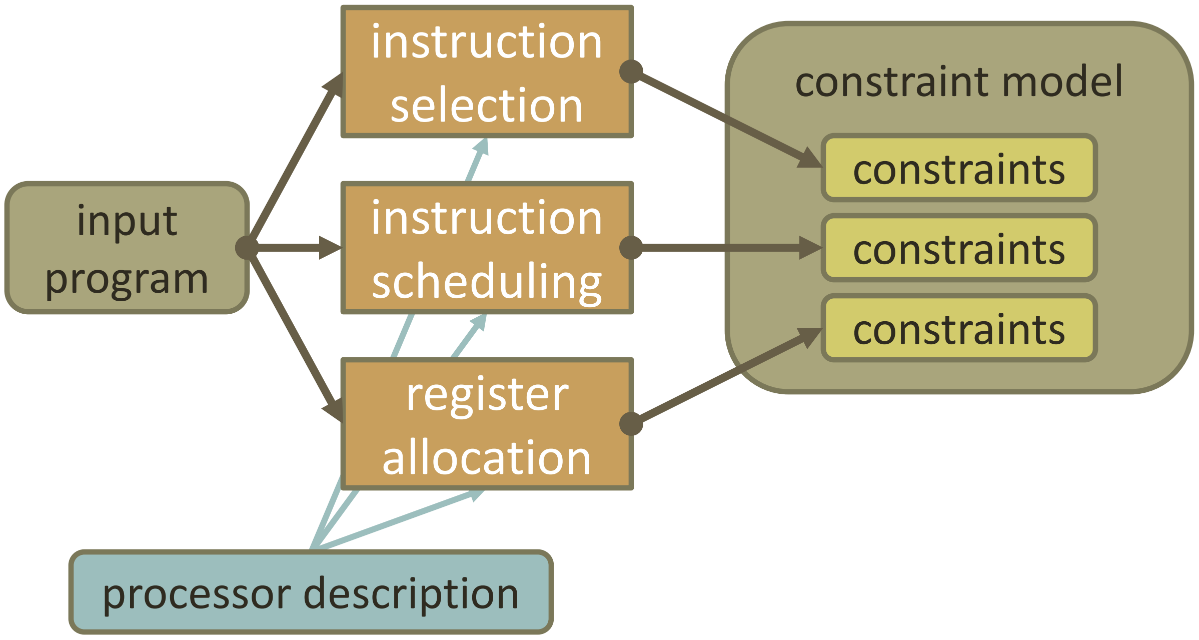
variables and possible values	problem parameters
constraints	legal value combinations
objective function	solution cost or quality

- Modeling: turn problem into constraint model
  - high-level of abstraction
  - expressive and array of advanced modeling techniques available

- Solving: find solution to constraint model

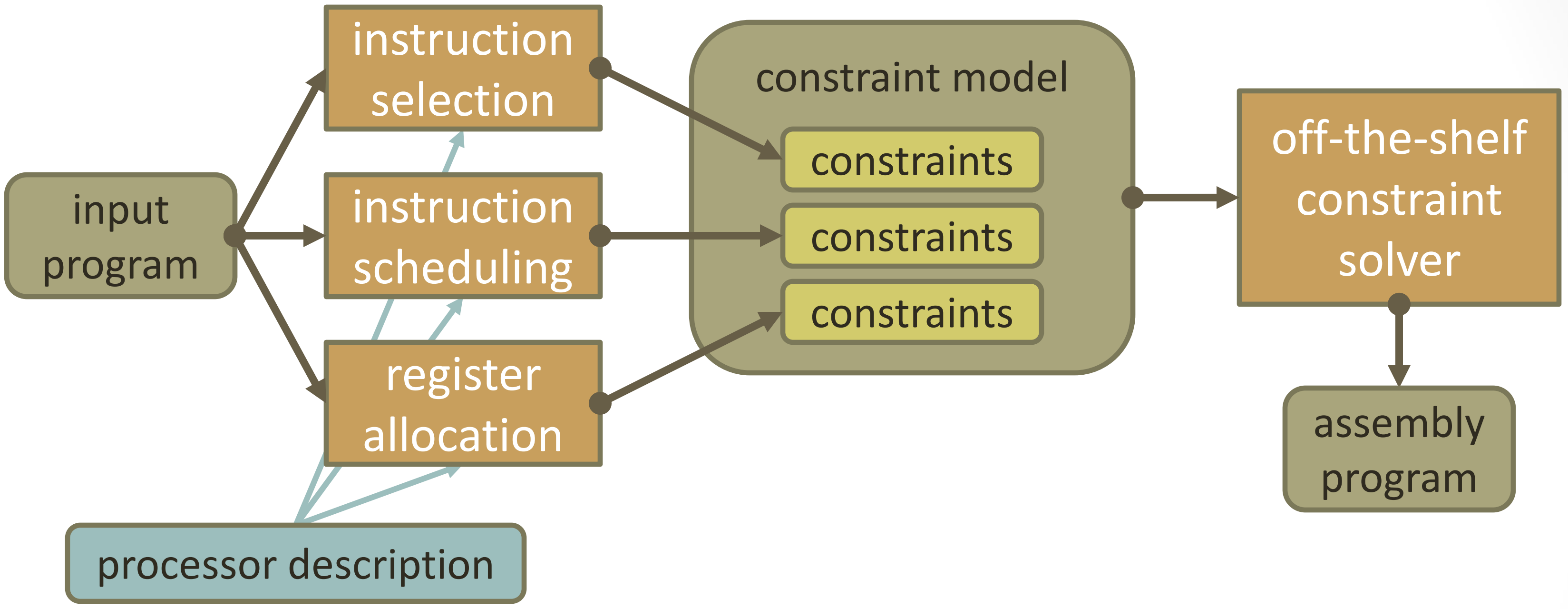
constraint propagation	remove infeasible values
heuristic search	simplify problem

Unison



- Generate constraint model
  - based on input program and processor description
  - constraints for all code generation tasks
  - **generate but not solve**: simpler and more expressive

Unison



- Off-the-shelf constraint solver solves constraint model
  - solution is assembly program
  - optimization takes inter-dependencies into account

Status

- Advanced and novel models
  - register allocation
  - instruction scheduling
- First attempts
  - instruction selection

- Register allocation
  - several register banks (generalizing spilling to memory)
  - unifying coalescing and spilling
  - register packing
- Instruction scheduling
  - full latency-based scheduling (including delay slots)
  - instruction bundling

Status

- Advanced and novel models
  - register allocation
  - instruction scheduling
- First attempts
  - instruction selection
- Register allocation
  - several register banks (g[illegible]ra[illegible]y)
  - unifying coalescing and spilling
  - register packing
- Instruction scheduling
  - full latency-based scheduling (including delay slots)
  - instruction bundling

notoriously difficult
we are proud ☺

Status

- Processor: simple RISC architecture (MIPS 32 bit)
  - compared to LLVM (state-of-the-art)
- Benchmark: `bzip2` (efficient compression tool)
  - part of SPECint 2006 benchmark suite
- Code quality: on par
- Robustness: scales to large functions (some 10^3 instructions)
  - use optimality preserving model decomposition

Status

- Processor: simple RISC architecture (MIPS 32 bit)
  - compared to LLVM (state-of-the-art)
- Benchmark: `bzip2` (effici
  - part of SPECint 2006 b
- Code quality: on par
- Robustness: scales to large functions (some 10^3 instructions)
  - use optimality preserving model decomposition

notoriously difficult
we are proud ☺

Future

- Complete model for instruction selection
    - yields complete constraint-based model
- How to describe processor architectures?
- Standard model improvements
    - tremendous scope
- Mix of basic and applied research
- Long (very long) term: establish idea as state-of-the-art
    - submit as open source to LLVM project (?)

Unison Fact Sheet

`www.sics.se/projects/unison`

- People

Joe Armstrong	Ericsson
Mats Carlsson	SICS
Roberto Castañeda Lozano	SICS
Frej Drejhammar	SICS
Gabriel Hjort Blindell (from May 1st)	KTH & SICS
Christian Schulte (lead)	KTH & SICS

- Funding

LM Ericsson AB → SICS	2010 – 2013
Swedish Research Council (VR) → KTH	2012 – 2014