Binary Code Translation from Register to Stack based code

Charles Pigott

Computer Science Dept
University of York

May 2017

Introduction

Figure: The Transmeta TM5600 CPU from a Fujitsu laptop

Register machines

- Named locations

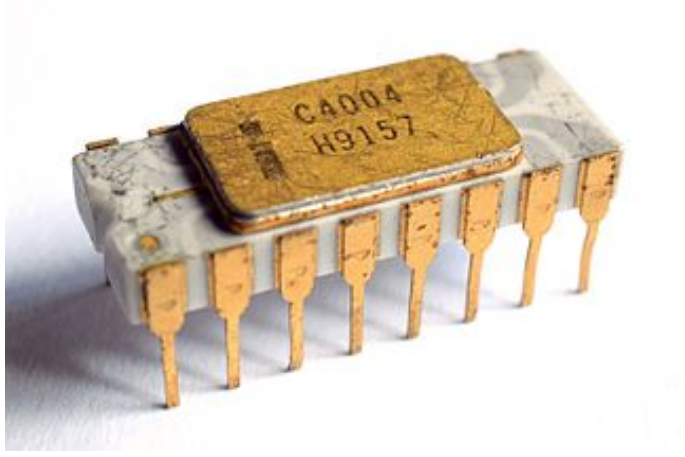

Figure: Intel's 4004 microprocessor

Register machines

- Named locations
- Almost exclusively used in practice

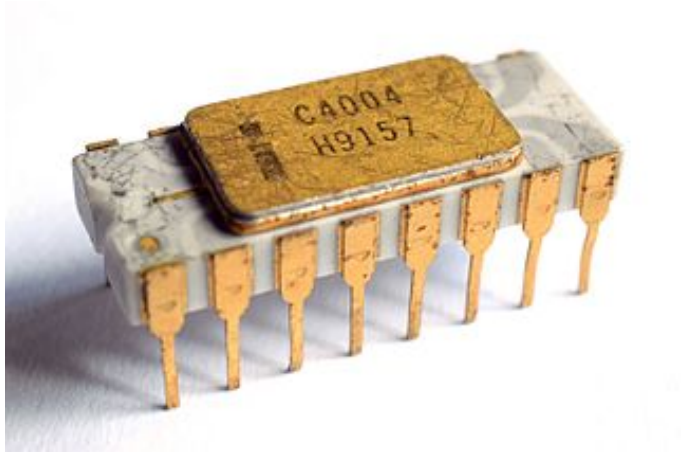

Figure: Intel's 4004 microprocessor

Register machines

- Named locations
- Almost exclusively used in practice
- RISC & CISC

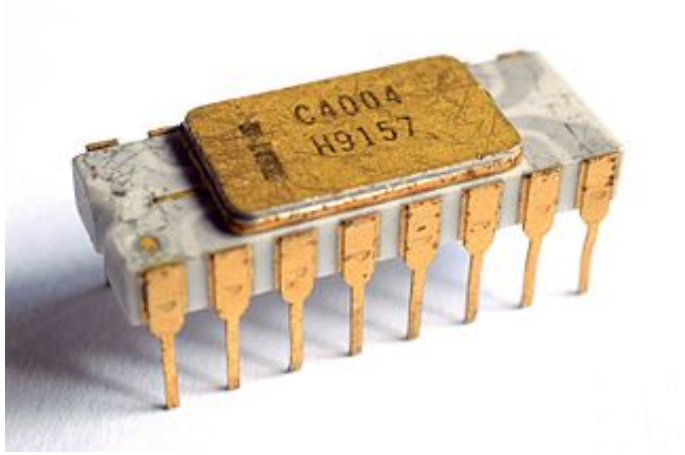

Figure: Intel's 4004 microprocessor

Stack machines

- Stack instead of registers

```
: .sol3 ( fn ... f1 x x f0 -- unchanged )
 dup N 0 do
   I 3 * 4 + pick ( fi fi+1 )
   2dup xor .rank  nip
 loop drop cr ;
: third ( a b c -- a b c a ) 2 pick ;    \ >r over r> swap ;
: poss ( a b c -- a b c a&b&c ) dup 2over and and ;
: next3 ( dl dr f Qfilebit -- dl dr f dl' dr' f' )
 invert >r third r@ and 2* 1+  third r@ and 2/  third r> and ;
\ bitmasks for unused diagonals and files
: try ( dl dr f -- )
 dup if 1 nodes +!  poss
   begin ?dup while
     dup >r lowBit next3 recurse r> lowBit-
   repeat
 else ( .sol3) 1 solutions +! then drop 2drop ;
: queens3  -1 dup Nbits try ;
```

Listing: 8 Queens problem in Forth

Stack machines

- Stack instead of registers
- Comparatively simpler than register architectures

```
: .sol3 ( fn ... f1 x x f0 -- unchanged )
 dup N 0 do
   I 3 * 4 + pick ( fi fi+1 )
   2dup xor .rank  nip
 loop drop cr ;
: third ( a b c -- a b c a ) 2 pick ;    \ >r over r> swap ;
: poss ( a b c -- a b c a&b&c ) dup 2over and and ;
: next3 ( dl dr f Qfilebit -- dl dr f dl' dr' f' )
 invert >r third r@ and 2* 1+  third r@ and 2/  third r> and ;
\ bitmasks for unused diagonals and files
: try ( dl dr f -- )
 dup if 1 nodes +!  poss
   begin ?dup while
     dup >r lowBit next3 recurse r> lowBit-
   repeat
 else ( .sol3) 1 solutions +! then drop 2drop ;
: queens3  -1 dup Nbits try ;
```

Listing: 8 Queens problem in Forth

Stack machines

- Stack instead of registers
- Comparatively simpler than register architectures
- Hardly used in production these days

```
: .sol3 ( fn ... f1 x x f0 -- unchanged )
 dup N 0 do
   I 3 * 4 + pick ( fi fi+1 )
   2dup xor .rank  nip
 loop drop cr ;
: third ( a b c -- a b c a ) 2 pick ;    \ >r over r> swap ;
: poss ( a b c -- a b c a&b&c ) dup 2over and and ;
: next3 ( dl dr f Qfilebit -- dl dr f dl' dr' f' )
 invert >r third r@ and 2* 1+  third r@ and 2/  third r> and ;
\ bitmasks for unused diagonals and files
: try ( dl dr f -- )
 dup if 1 nodes +!  poss
   begin ?dup while
     dup >r lowBit next3 recurse r> lowBit-
   repeat
 else ( .sol3) 1 solutions +! then drop 2drop ;
: queens3  -1 dup Nbits try ;
```

Listing: 8 Queens problem in Forth

Stack optimisations

- Peephole (McKeeman, 1965)

Original stack code	Optimised stack code
`SET 1` `ADD`	`INC`

`SET 0`
`TEQ`
`DROP`
	`TSZ`

`DUP` `SWAP`	`DUP`
`SET x` `DROP`	`<NOP>`

Table: Peephole optimisation examples

Stack optimisations

- Peephole (McKeeman, 1965)
- Stack scheduling (Koopman, 1995)

Stack optimisations

- Peephole (McKeeman, 1965)
- Stack scheduling (Koopman, 1995)
- Inter-block scheduling (Bailey, 2000)

Stack optimisations

- Peephole (McKeeman, 1965)
- Stack scheduling (Koopman, 1995)
- Inter-block scheduling (Bailey, 2000)
- Global scheduling (Shannon, 2006)

Binary translation

- Transputer
- IBM

Binary translation

Transmeta

- Published 2000

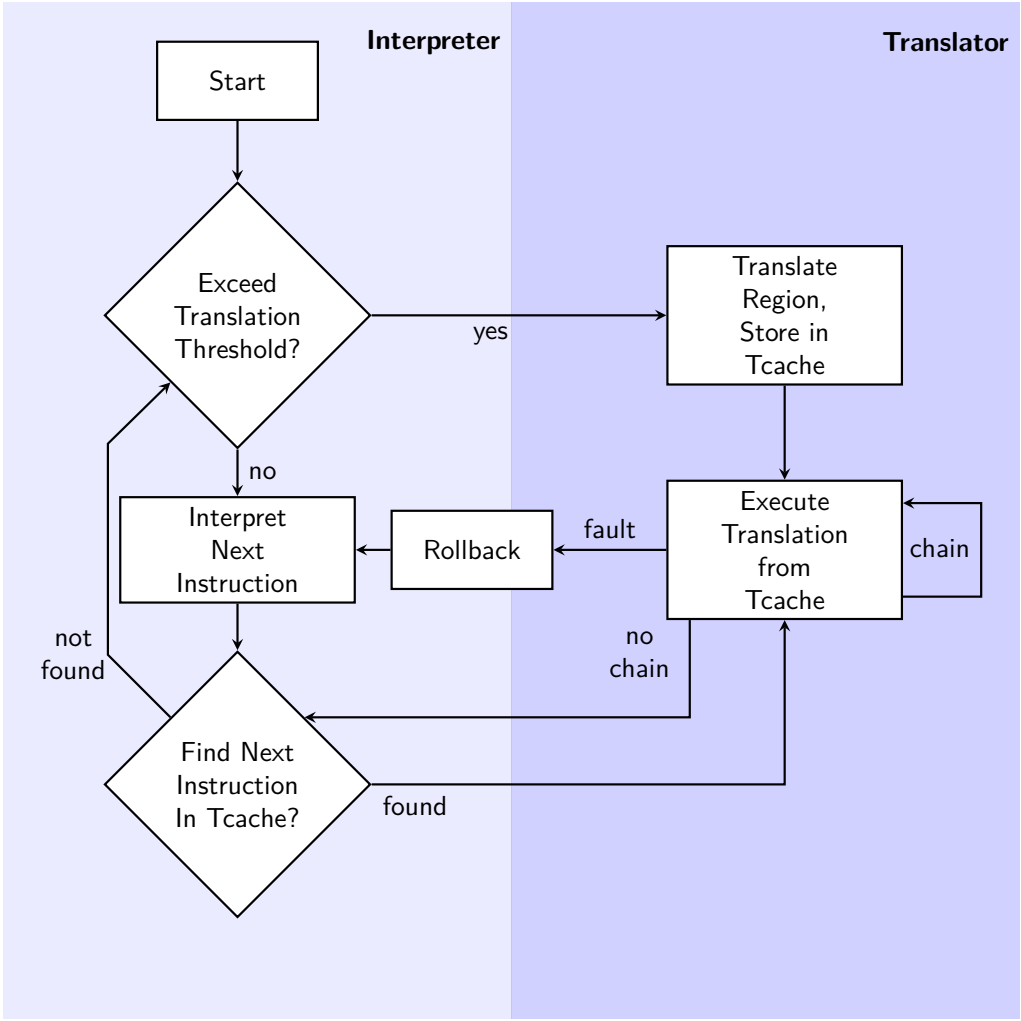

Figure: Typical CMS Control Flow

Binary translation

Transmeta

- Published 2000
- Sophisticated x86 translation and emulator

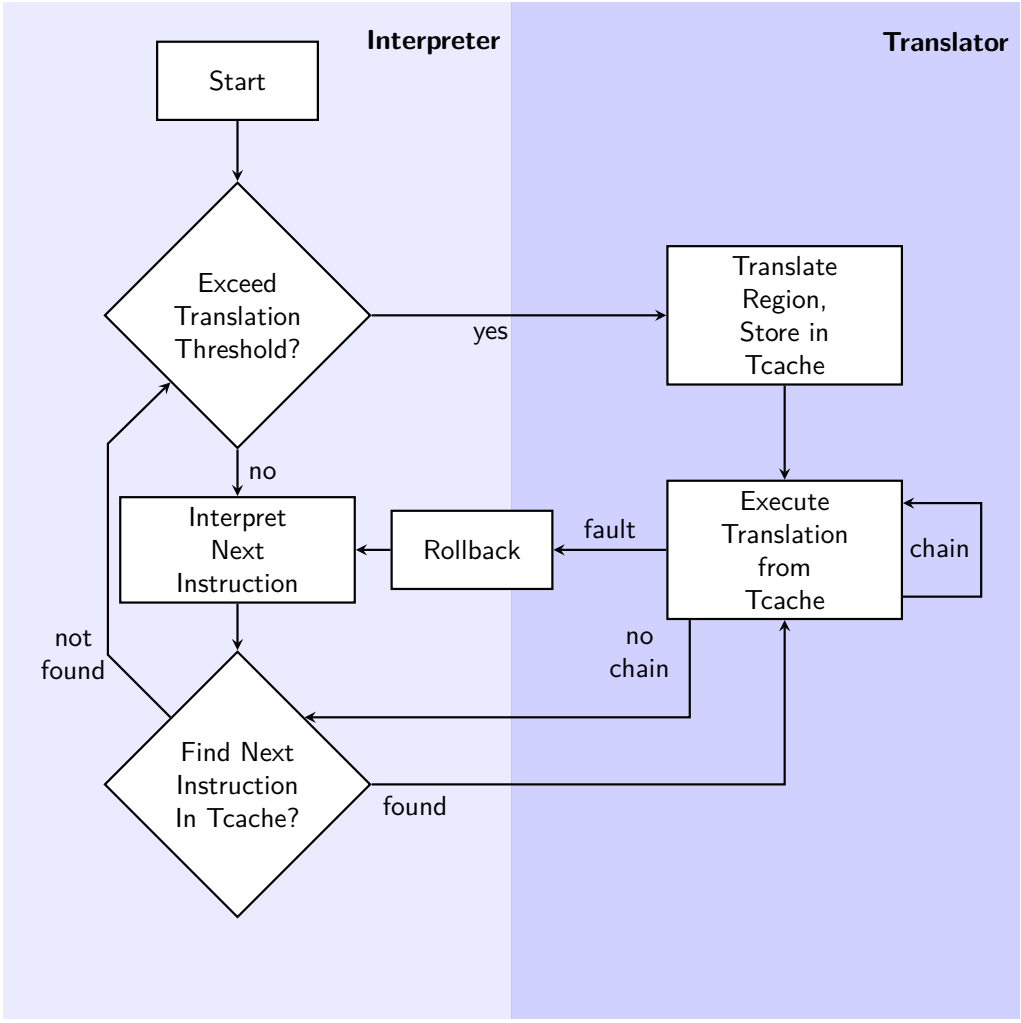

Figure: Typical CMS Control Flow

Implementation

- Implemented in C++

Implementation

- Implemented in C++
- Source register architecture DCPU-16

Implementation

- Implemented in C++
- Source register architecture DCPU-16
- Target stack architecture J5

Implementation

- Implemented in C++
- Source register architecture DCPU-16
- Target stack architecture J5
- Implemented Peephole & Koopman-style optimisations

Implementation

- Implemented in C++
- Source register architecture DCPU-16
- Target stack architecture J5
- Implemented Peephole & Koopman-style optimisations
- Implemented basic translation cache

Implementation

- Implemented in C++
- Source register architecture DCPU-16
- Target stack architecture J5
- Implemented Peephole & Koopman-style optimisations
- Implemented basic translation cache
- Tested using 4 different register test programs

Implementation

- Implemented in C++
- Source register architecture DCPU-16
- Target stack architecture J5
- Implemented Peephole & Koopman-style optimisations
- Implemented basic translation cache
- Tested using 4 different register test programs

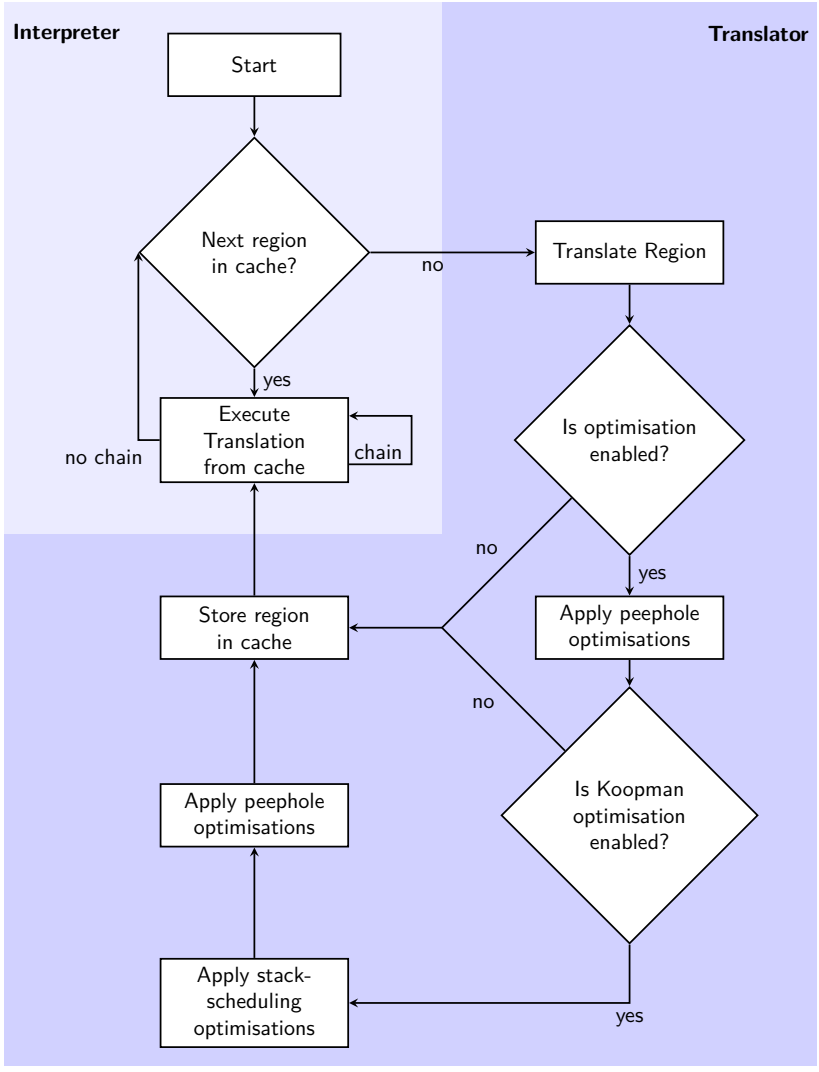

Figure: Implementation structure

Results

Traces

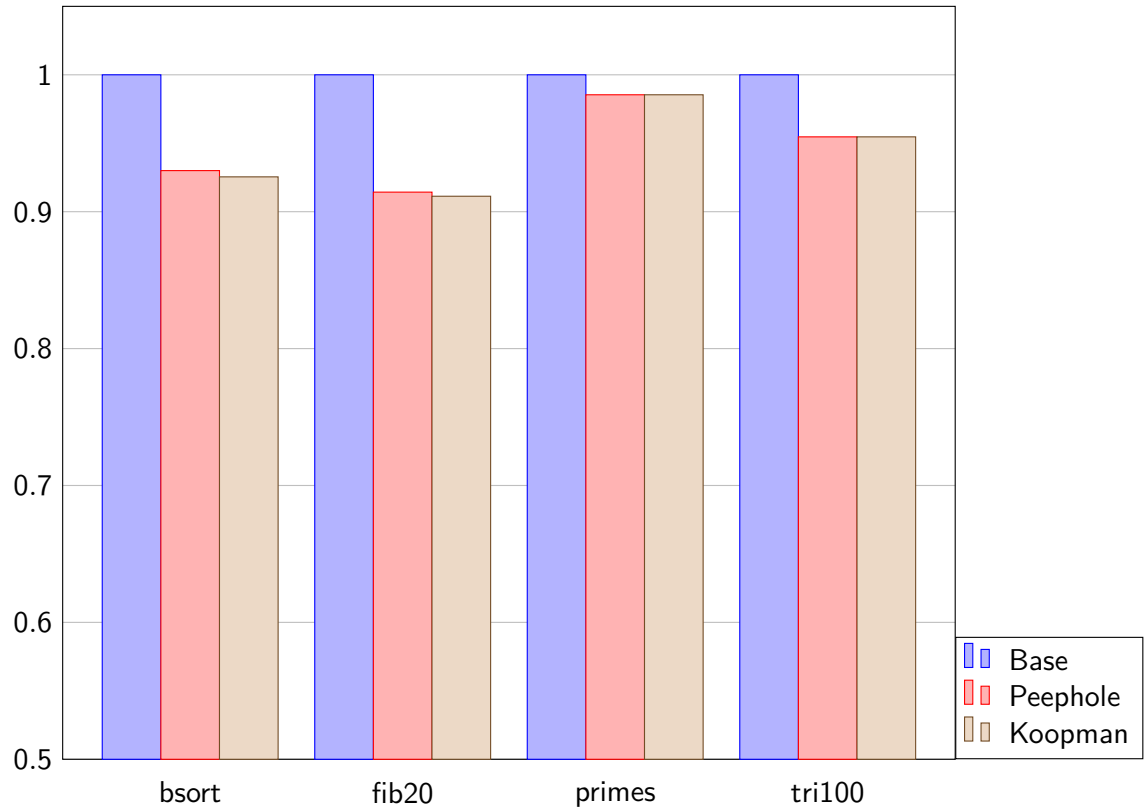

Figure: Relative no. of instructions executed

Results

Traces

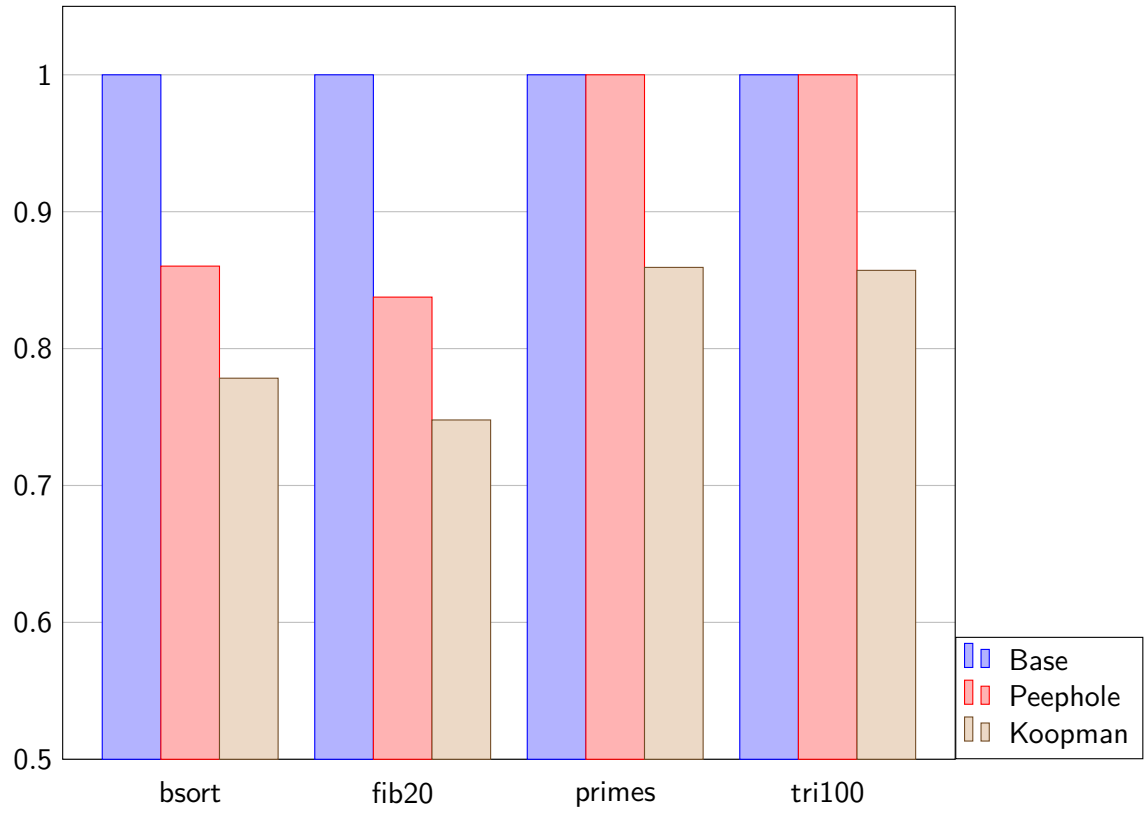

Figure: Relative no. of LOAD/STORE instructions executed

Results

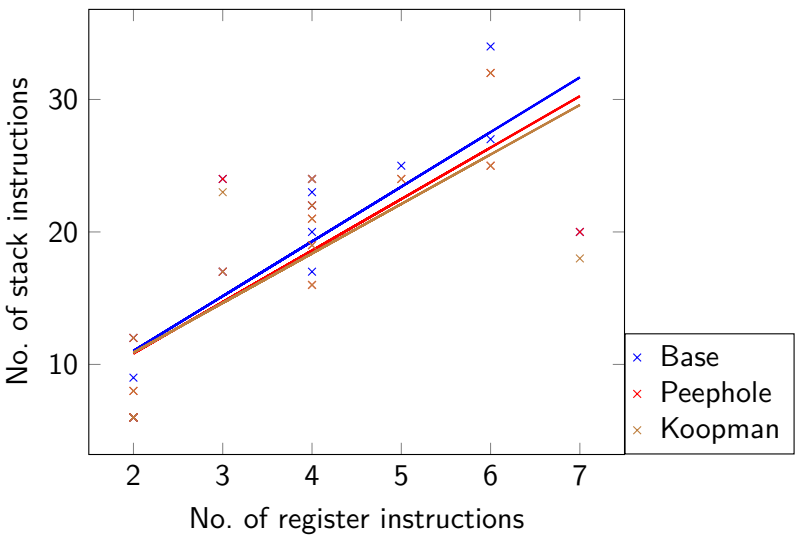

Figure: Register blocks to stack equivalents

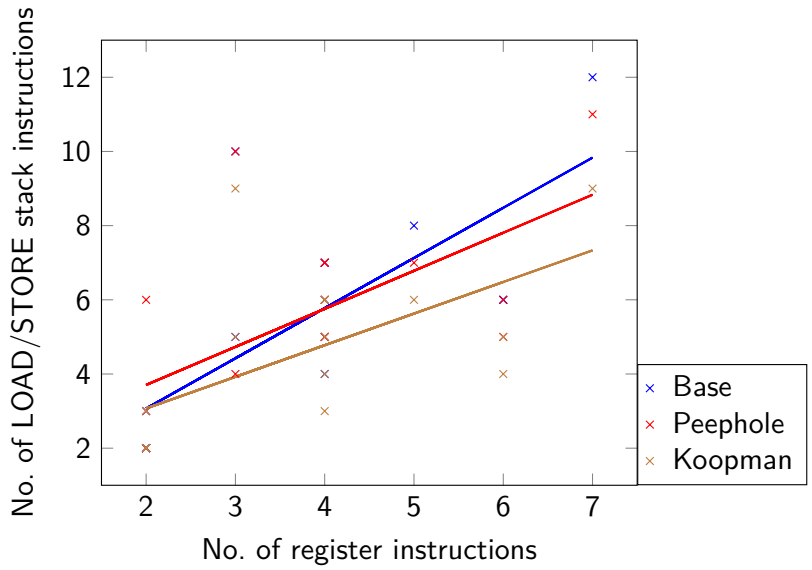

Figure: Register blocks to stack memory read/writes

Results

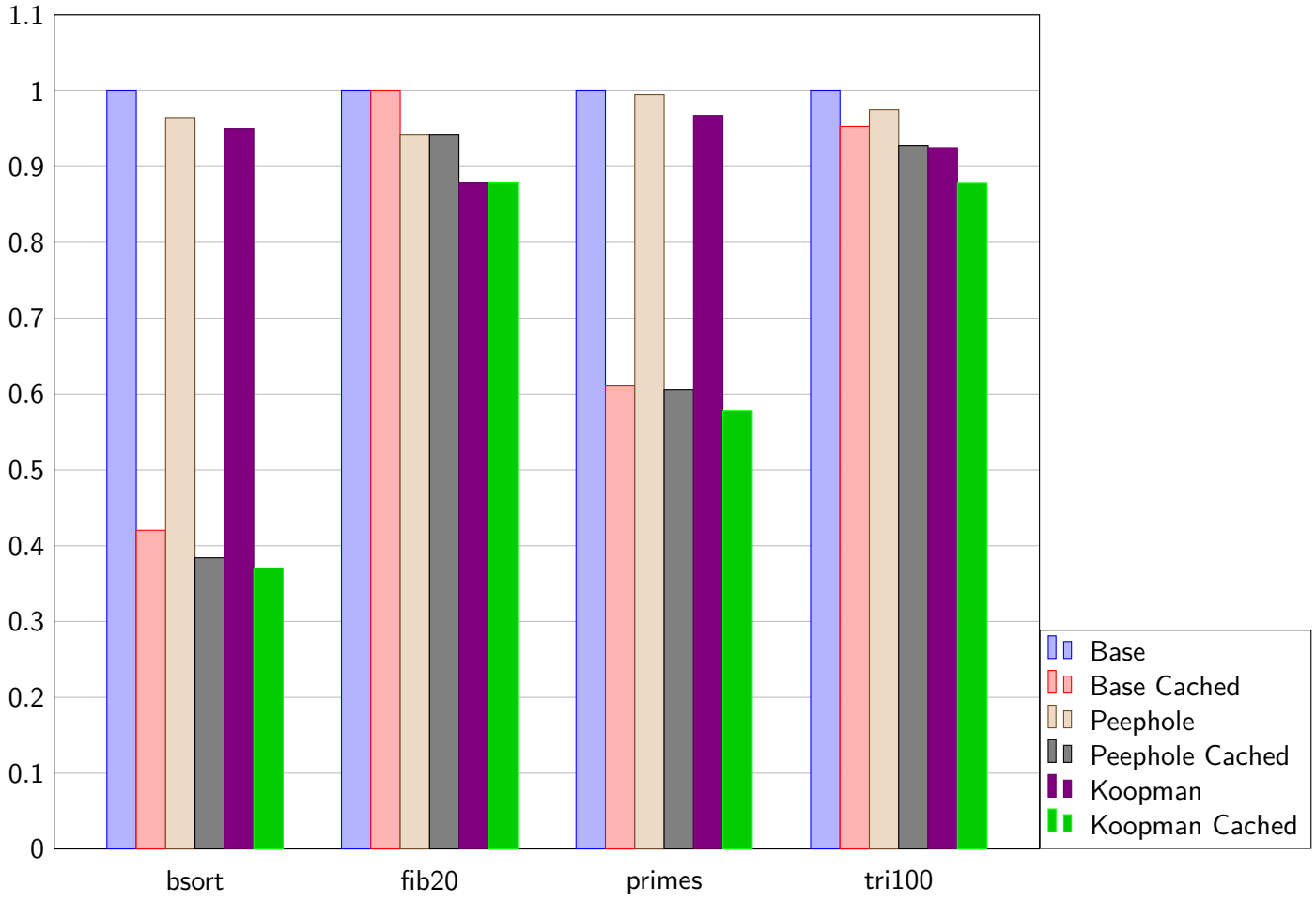

Figure: Relative program cost with and without caching

Conclusions

Further work

- Further optimisation algorithms

Conclusions

Further work

- Further optimisation algorithms
- More sophisticated caching system

Conclusions

Further work

- Further optimisation algorithms
- More sophisticated caching system
- More formal testing

Conclusions

Any questions?