Using Static-Single-Information-Form for SCAD code generation

Alexander Schneiders

Bachelor Thesis Presentation

June 28, 2018

Outline

1. Introduction / Motivation
   - the SCAD machine
   - move code
2. Control flow
3. Remove constraints per basic-block
4. Summary

Outline

1. Introduction / Motivation
   - the SCAD machine
   - move code
2. Control flow
3. Remove constraints per basic-block
4. Summary

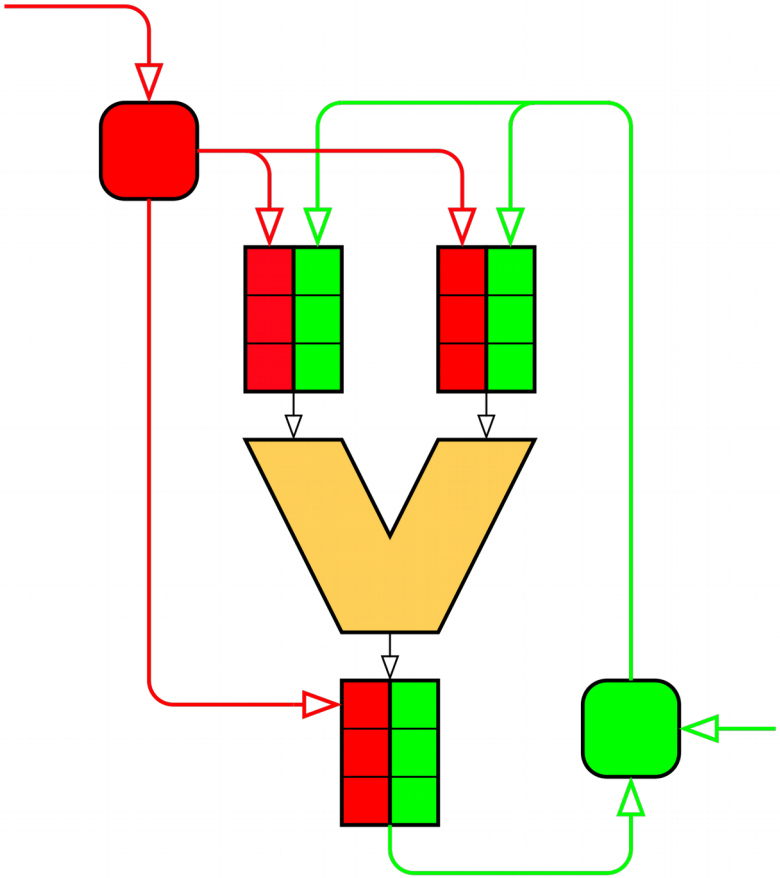
The SCAD machine

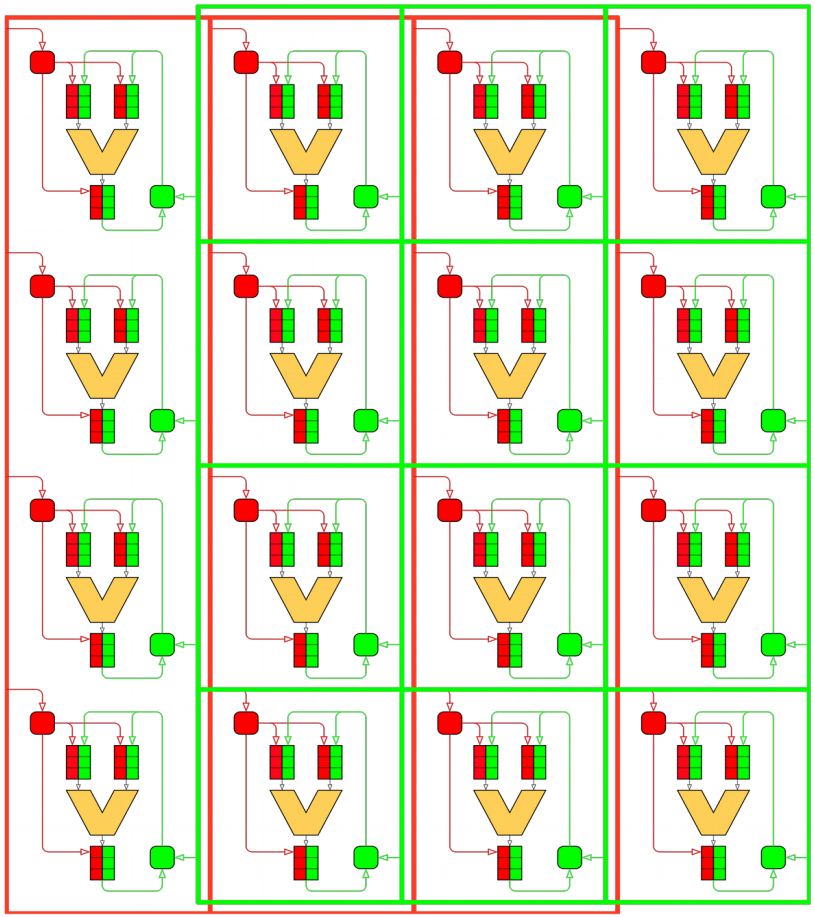
Synchronous Control Asynchronous Dataflow

- Exposed Datapath Architecture
- many processing units (PUs)
- PUs have buffered inputs/outputs (FIFO buffers)
- buffers are connected
- values are moved from output to input buffers
- PUs fire when they find operands in their input buffers

The SCAD machine

Synchronous Control Asynchronous Dataflow





The SCAD machine

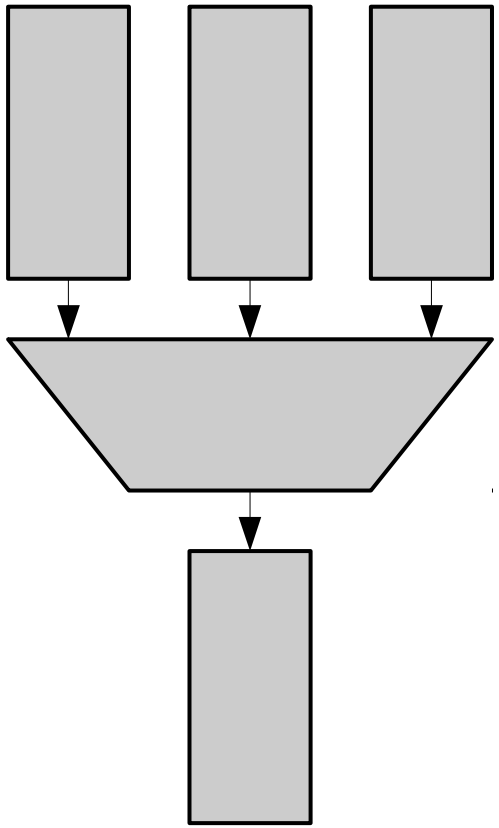
Goal: distribute instructions over pool of PUs to get Instruction Level Parallelism (ILP)

Outline

1. Introduction / Motivation
   - the SCAD machine
   - move code
2. Control flow
3. Remove constraints per basic-block
4. Summary

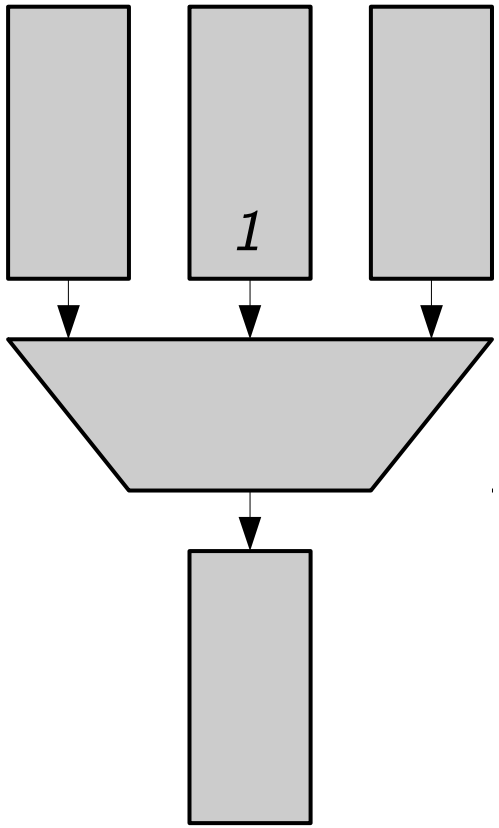
Move Code

$x \leftarrow 1 + 2$



Move Code

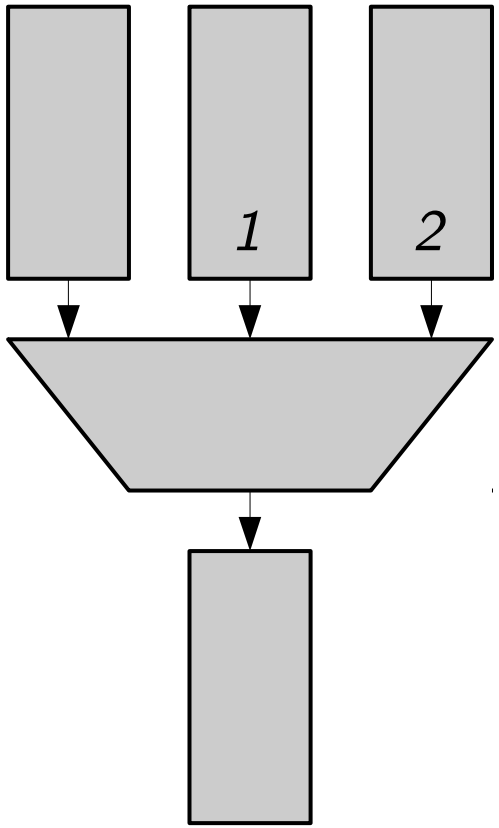
$$x \leftarrow 1 + 2$$



- move 1 to input buffer

Move Code

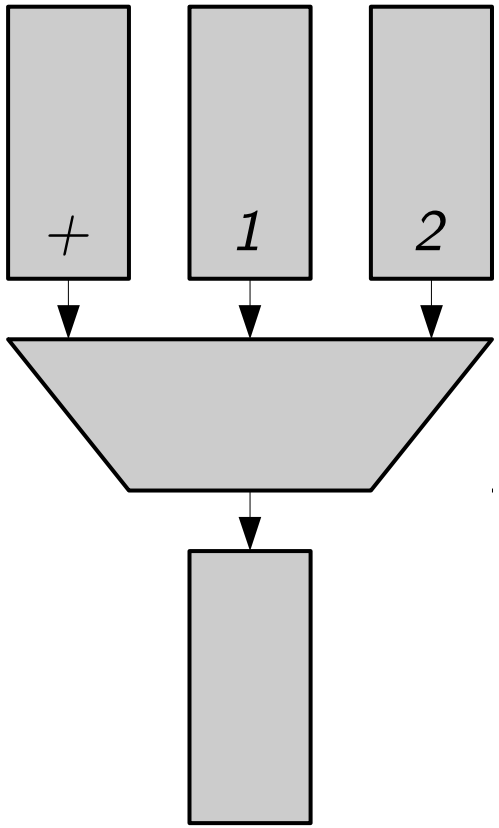
$$x \leftarrow 1 + 2$$



- move 1 to input buffer
- move 2 to input buffer

Move Code

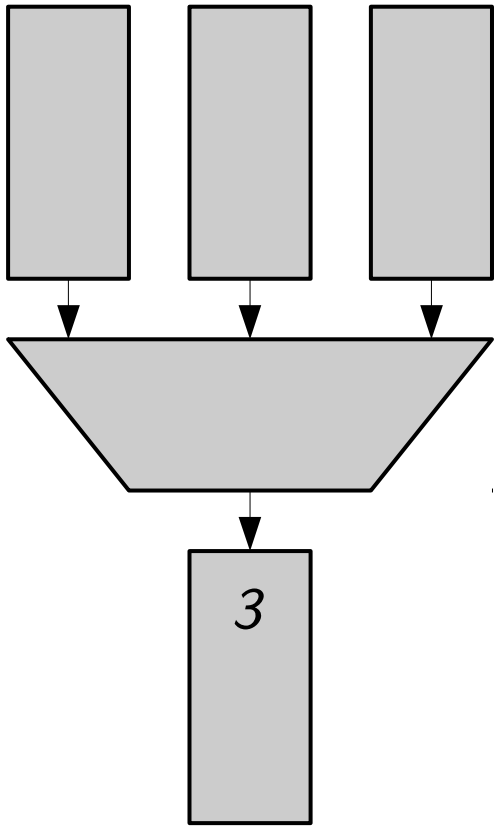
$x \leftarrow 1 + 2$



- move 1 to input buffer
- move 2 to input buffer
- move opcode to input buffer

Move Code

$x \leftarrow 1 + 2$



- move 1 to input buffer
- move 2 to input buffer
- move opcode to input buffer
- PU fires

Move Code

special move instructions

duplicate (dup)

- $x \leftarrow y$
- move from one buffer to another
- add with 0, multiply with 1 or even separate instruction

discard

- $\textsc{Discard}(x)$
- dequeue from output buffer and get rid of a variable
- dup with 0 count or special $NULL$ buffer as target

Move Code

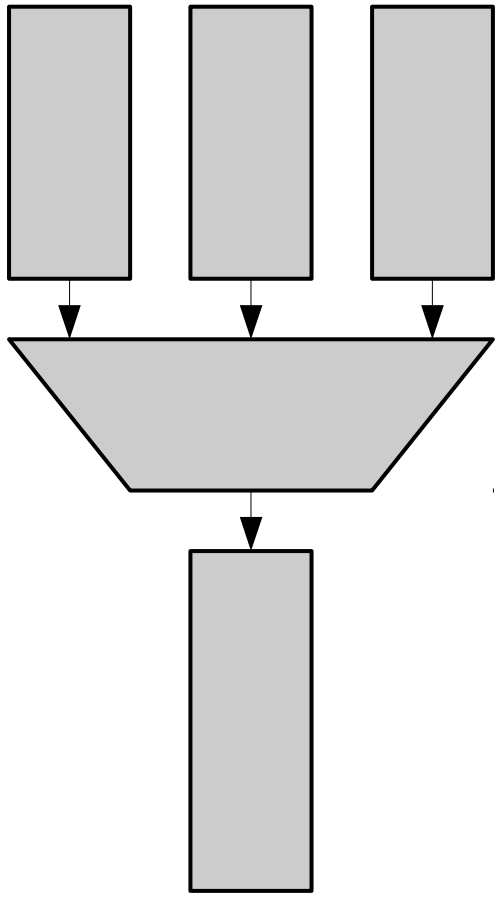
$x \leftarrow \ldots$

$y \leftarrow \ldots$

$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$



Move Code

dequeue all copies of a variable consecutively

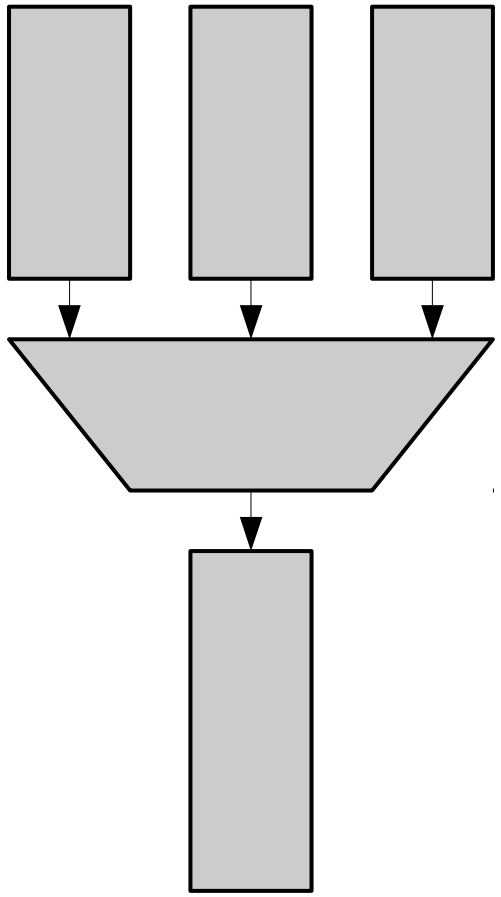
$x \leftarrow \ldots$

$y \leftarrow \ldots$

$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$



Move Code

dequeue all copies of a variable consecutively

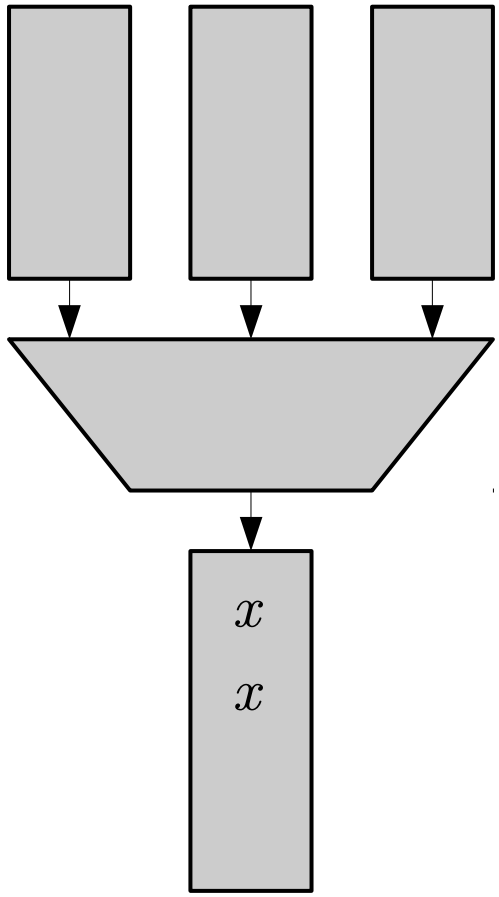
$x \leftarrow \ldots$

$y \leftarrow \ldots$

$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$



Move Code

dequeue all copies of a variable consecutively

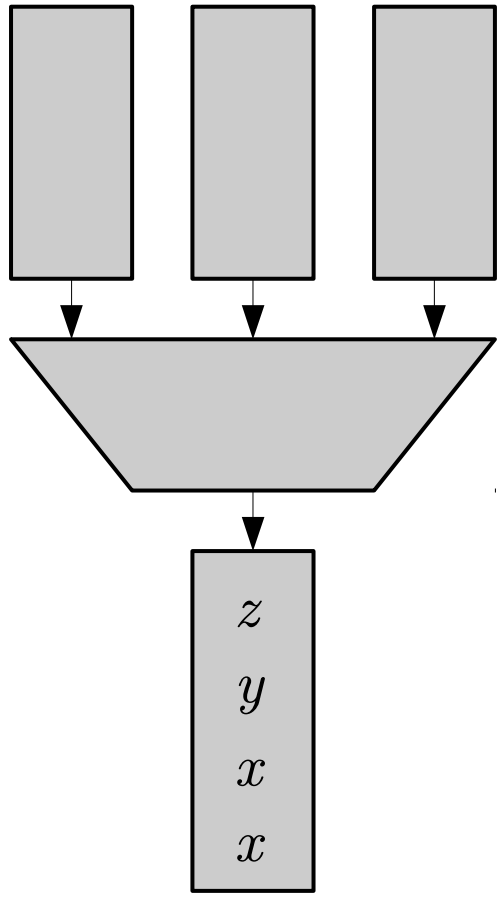
$x \leftarrow \ldots$

$y \leftarrow \ldots$

$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$



Move Code

dequeue all copies of a variable consecutively

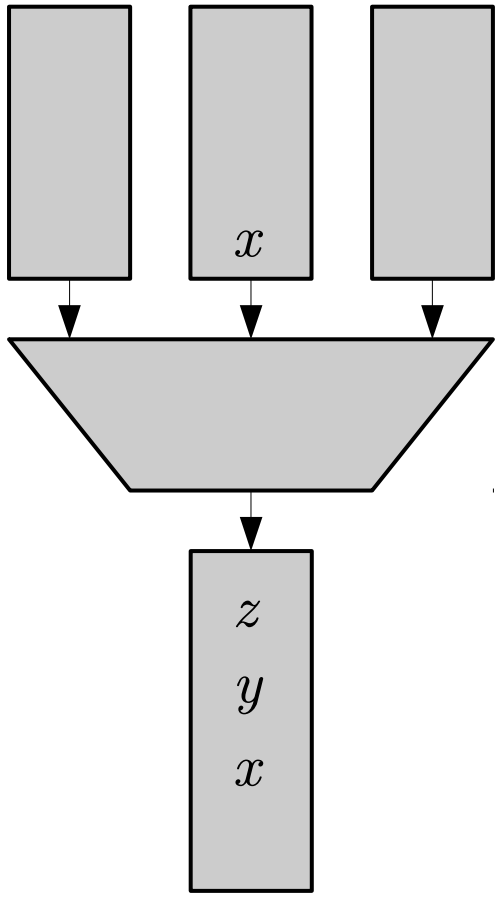
$x \leftarrow \ldots$

$y \leftarrow \ldots$

$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$



Move Code

dequeue all copies of a variable consecutively

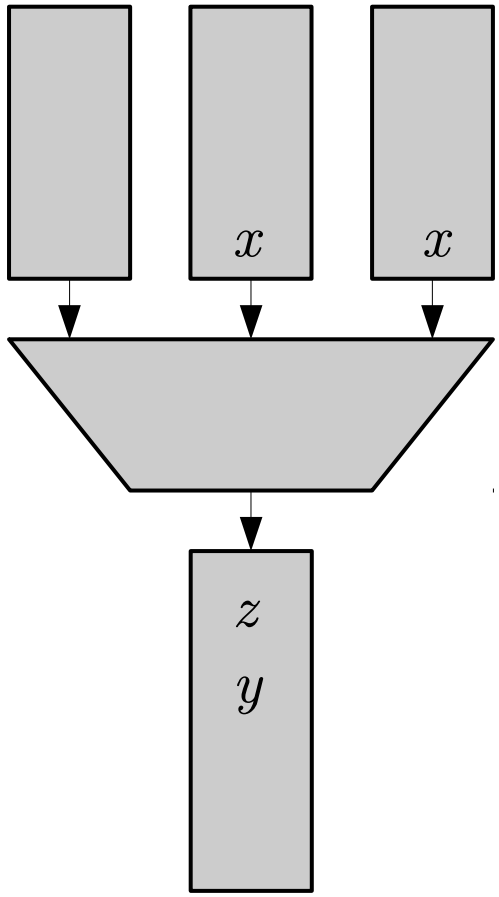
$x \leftarrow \ldots$

$y \leftarrow \ldots$

$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$



Move Code

dequeue all copies of a variable consecutively

$x \leftarrow \ldots$

$y \leftarrow \ldots$

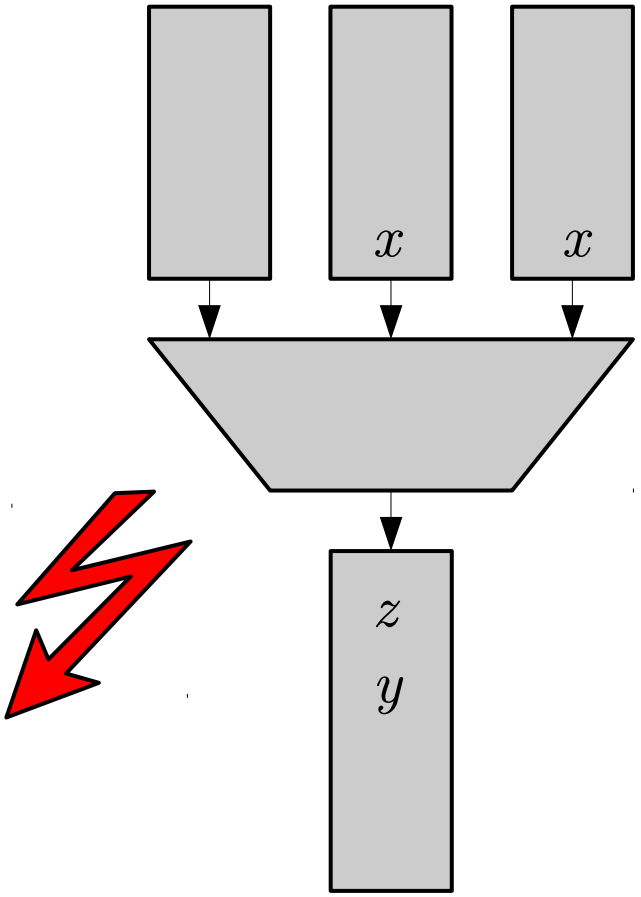
$z \leftarrow \ldots$

$\ldots \leftarrow x + y$

$\ldots \leftarrow z + x$

wrong order of enqueing

must be mapped to different PUs



Move Code

previous work: optimal move code generation

(making due with least number of PUs)

Move Code

previous work: optimal move code generation
(making due with least number of PUs)

- employing SAT and SMT solvers
- bad running time for larger programs
- least number of PUs
- cannot map instructions to PUs freely

Move Code

contribution: remove constraints

- arbitrary number of PUs (even 1)
- choose mapping freely

Move Code

generate move code instruction-wise

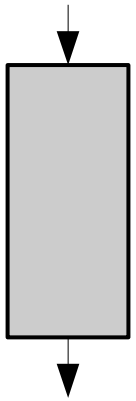
$x \leftarrow 1 + 2$	enqueue right operand enqueue left operand enqueue opcode
$\ldots \leftarrow x + 4$	enqueue right operand enqueue left operand enqueue opcode
$\ldots \leftarrow 5 + x$	enqueue right operand enqueue left operand enqueue opcode

Move Code

$$x \leftarrow 1 + 2$$
$$y \leftarrow 3 + 4$$
$$\ldots \leftarrow x$$
$$\ldots \leftarrow y$$
$$\ldots \leftarrow x$$

Move Code

$$x \leftarrow 1 + 2$$
$$y \leftarrow 3 + 4$$
$$\ldots \leftarrow x$$
$$\ldots \leftarrow y$$
$$\ldots \leftarrow x$$



Move Code

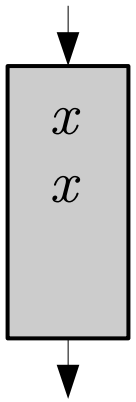
$x \leftarrow 1 + 2$ ⟵

$y \leftarrow 3 + 4$

$\ldots \leftarrow x$

$\ldots \leftarrow y$

$\ldots \leftarrow x$



Move Code

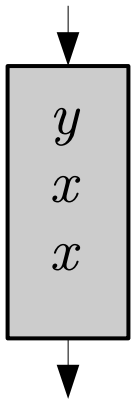
$x \leftarrow 1 + 2$

$y \leftarrow 3 + 4$ ⟵

$\ldots \leftarrow x$

$\ldots \leftarrow y$

$\ldots \leftarrow x$



Move Code

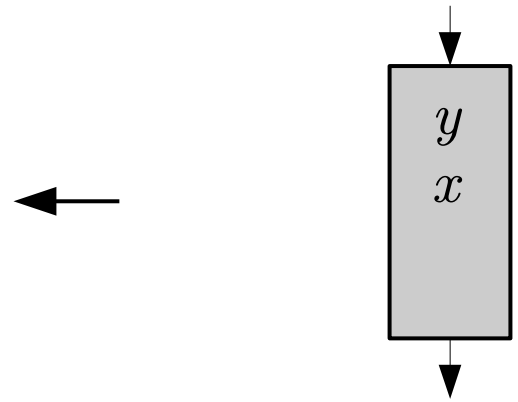
$x \leftarrow 1 + 2$

$y \leftarrow 3 + 4$

$\ldots \leftarrow x$

$\ldots \leftarrow y$

$\ldots \leftarrow x$



Move Code

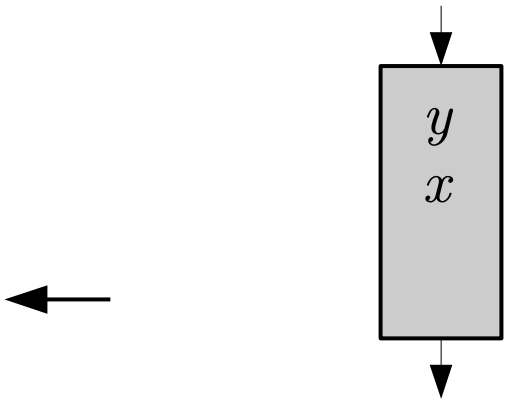
$x \leftarrow 1 + 2$

$y \leftarrow 3 + 4$

$\ldots \leftarrow x$

$\ldots \leftarrow y$

$\ldots \leftarrow x$



Move Code

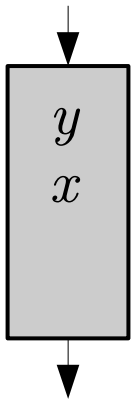
$x \leftarrow 1 + 2$

$y \leftarrow 3 + 4$

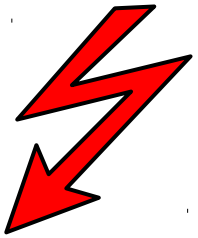
$\ldots \leftarrow x$

$\ldots \leftarrow y$

$\ldots \leftarrow x$



attempt to dequeue y but find x instead



Move Code

$x \leftarrow 1 + 2$

$y \leftarrow 3 + 4$

$\ldots \leftarrow x$

$x' \leftarrow x$

$\ldots \leftarrow y$

$\ldots \leftarrow x'$

resolve buffer interference by inserting a dup instruction

Move Code

rest of this talk:

- deal with control flow
  - produce correct number of copies
  - restrict interferences to basic blocks
- resolve buffer interferences
  - inserting dup instructions

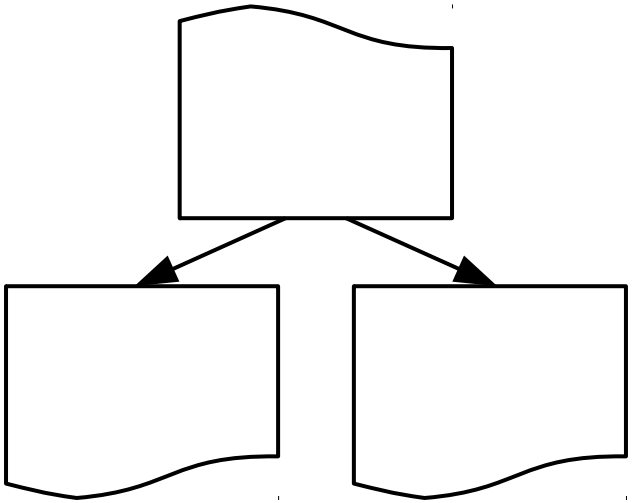
Outline

1. Introduction / Motivation
   - the SCAD machine
   - move code
2. Control flow
3. Remove constraints per basic-block
4. Summary

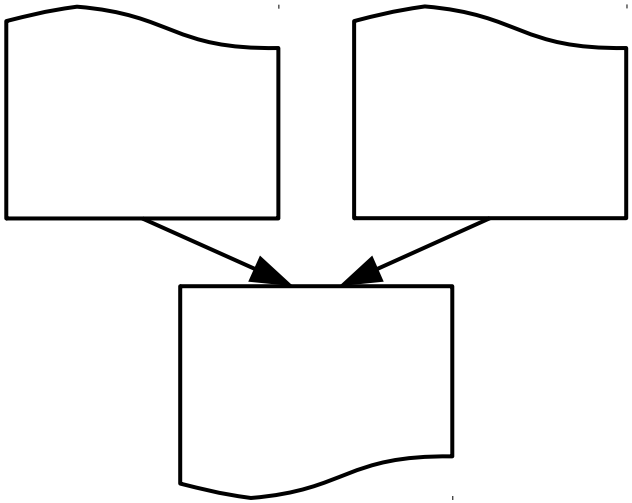
Controlflow

control flow in general:

split points

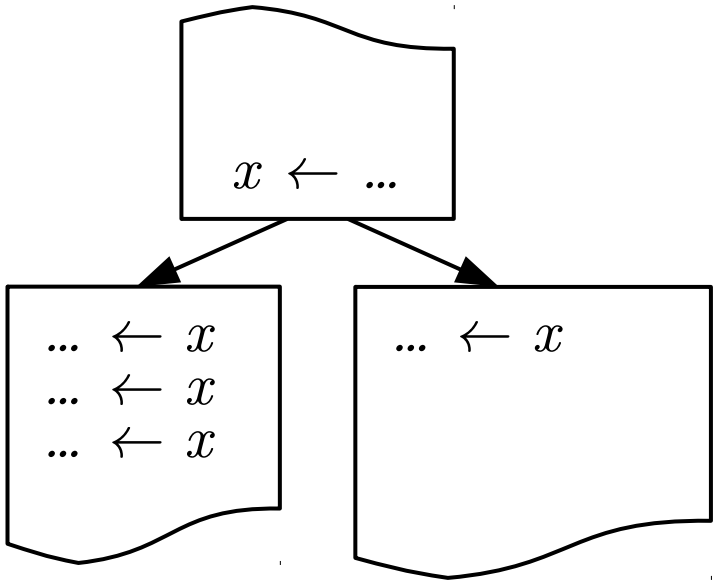


join points



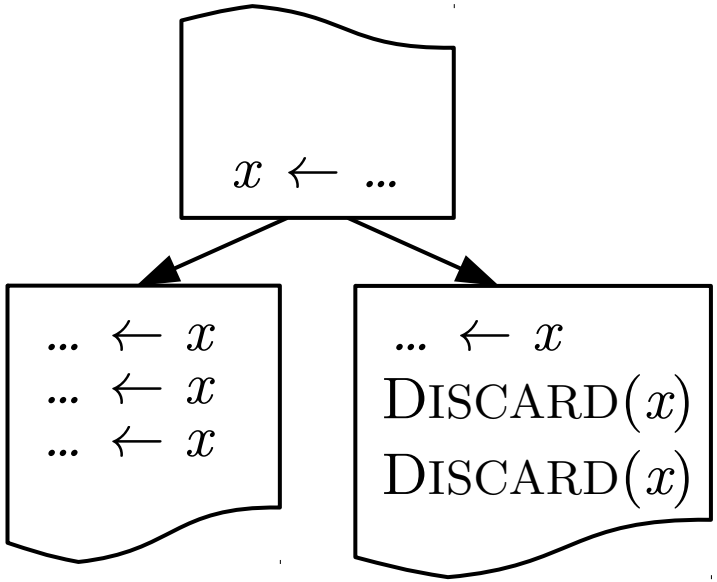
Controlflow

balancing:



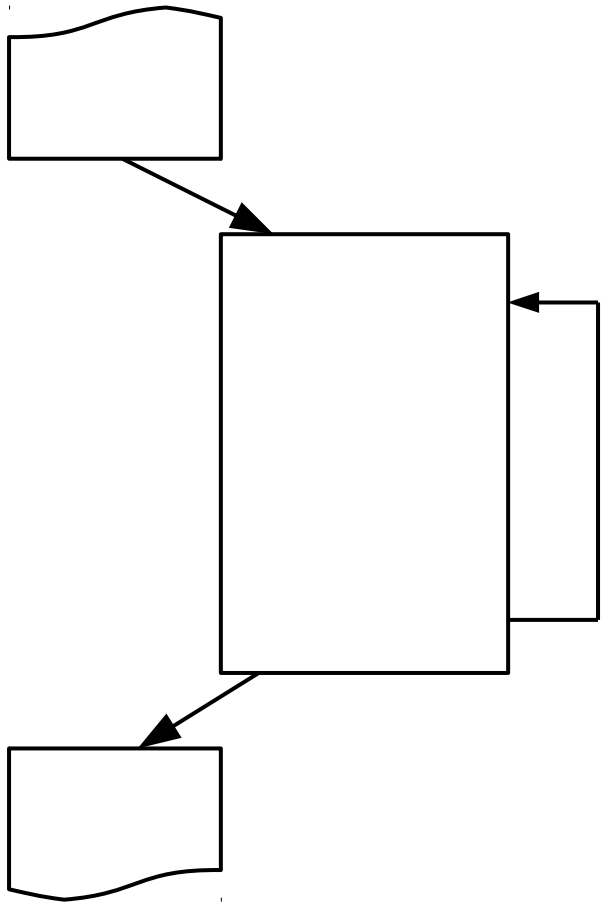
Controlflow

balancing:



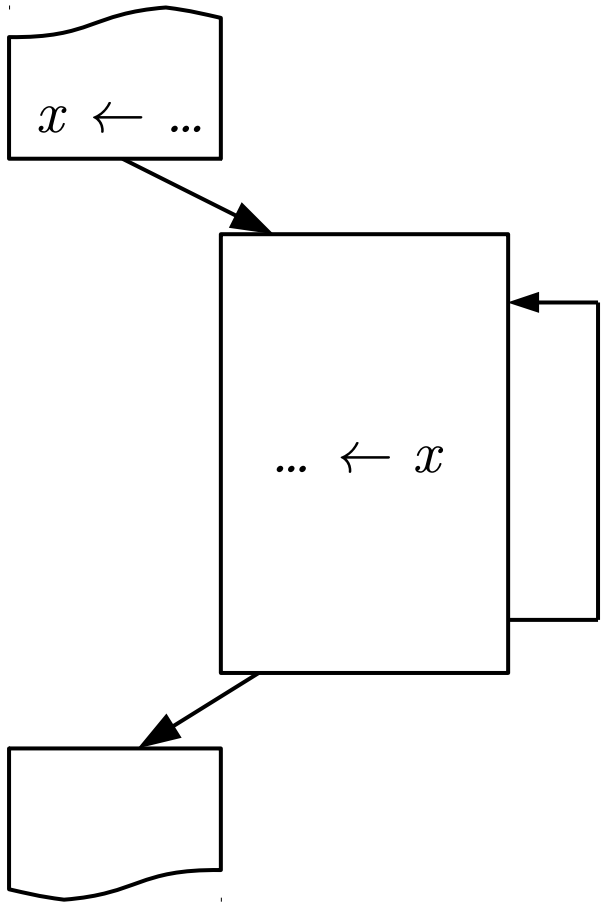
Controlflow

loops:



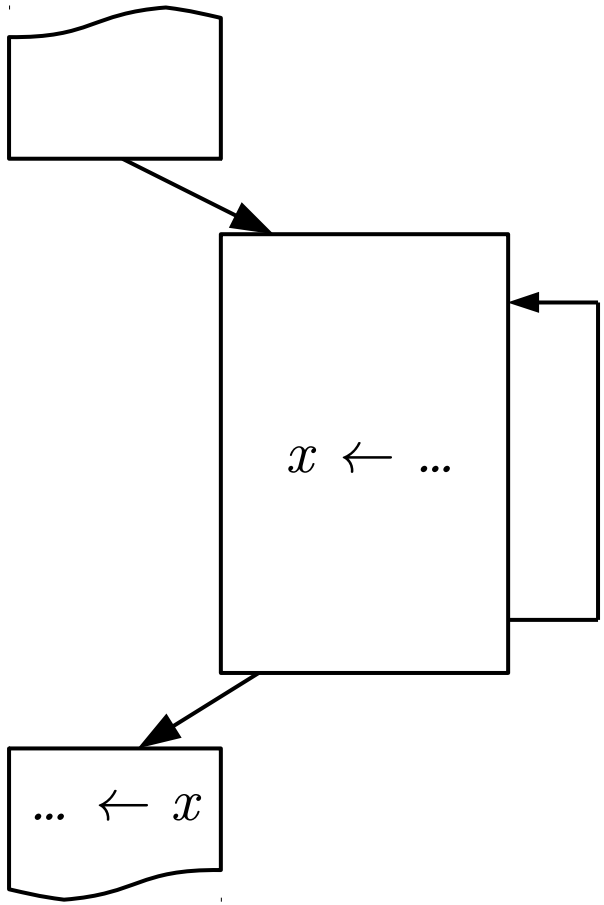
Controlflow

loops:



Controlflow

loops:



Static Single Information (SSI)

- program intermediate representation
- related to Static Single Assignment (SSA)
- properties (simplified)
  - variables are defined only once
  - and then used along one chain of uses
  - each branch gets it‘s own copy of a variable

Static Single Information (SSI)

σ (sigma) and ϕ (phi) functions

- pseudo assignment
- σ placed at split points
- ϕ placed at join points

Static Single Information (SSI)

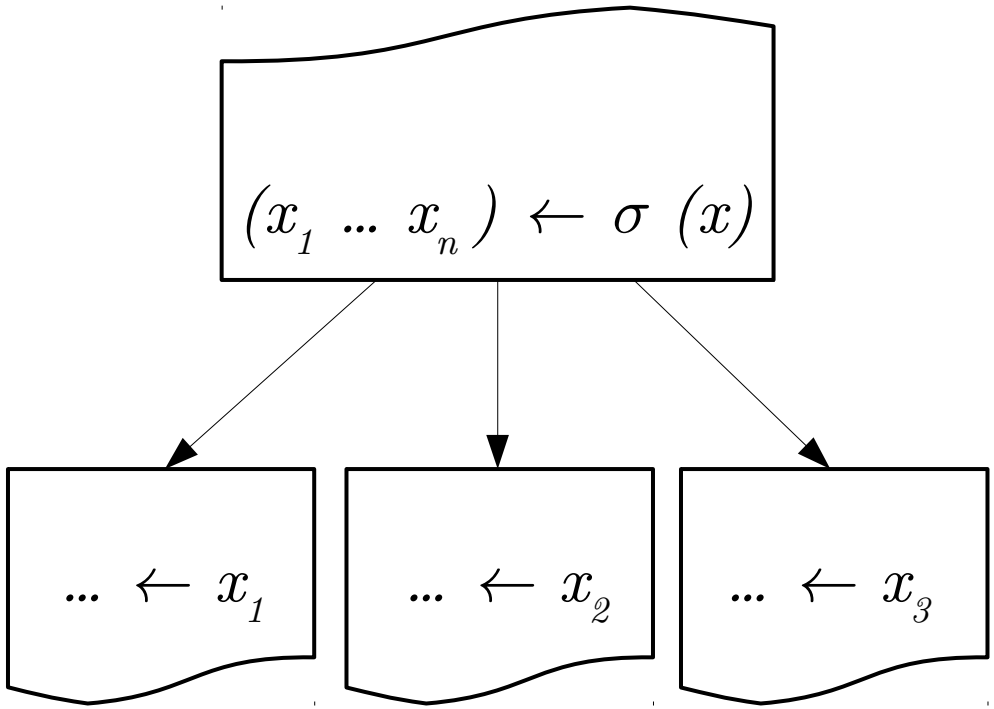
$$(x_1 \ldots x_n) \leftarrow \sigma\ (x)$$

Static Single Information (SSI)

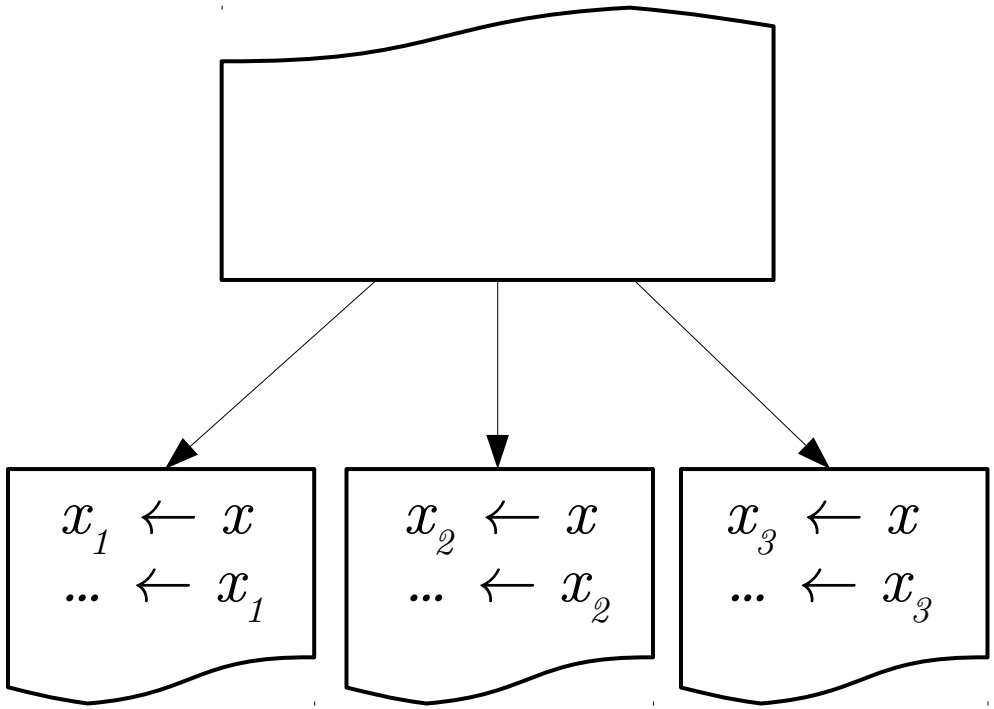
$$(x_1 \ldots x_n) \leftarrow \sigma\ (x)$$

$$x_1 \leftarrow x \qquad x_2 \leftarrow x \qquad x_3 \leftarrow x$$

Static Single Information (SSI)



Static Single Information (SSI)



Static Single Information (SSI)

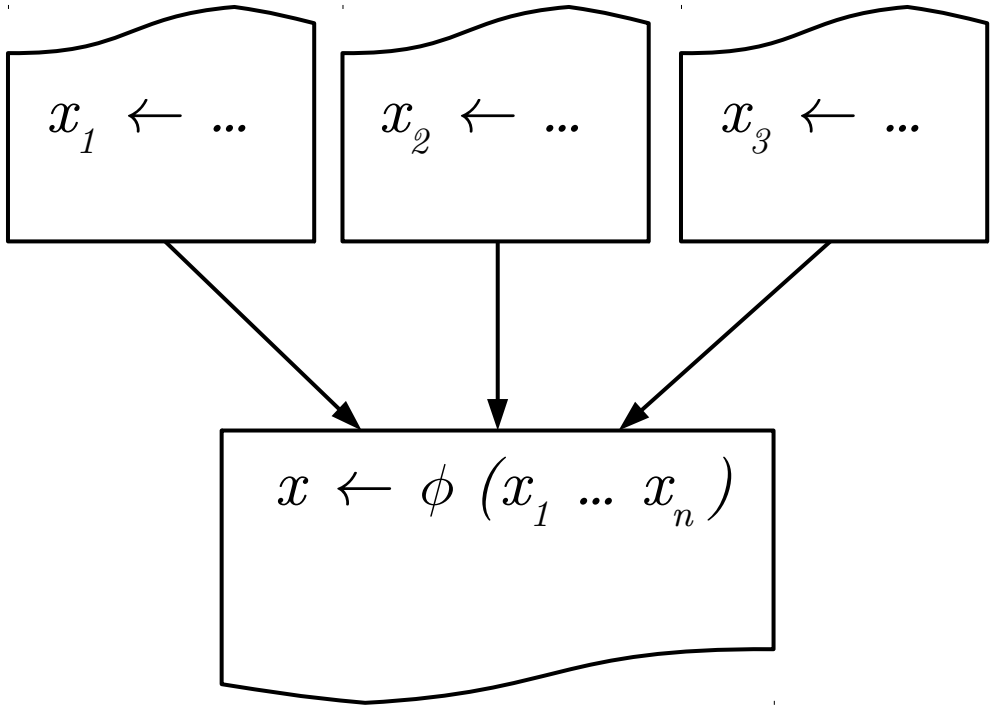
$$x \leftarrow \phi\,(x_1 \ldots x_n)$$

Static Single Information (SSI)

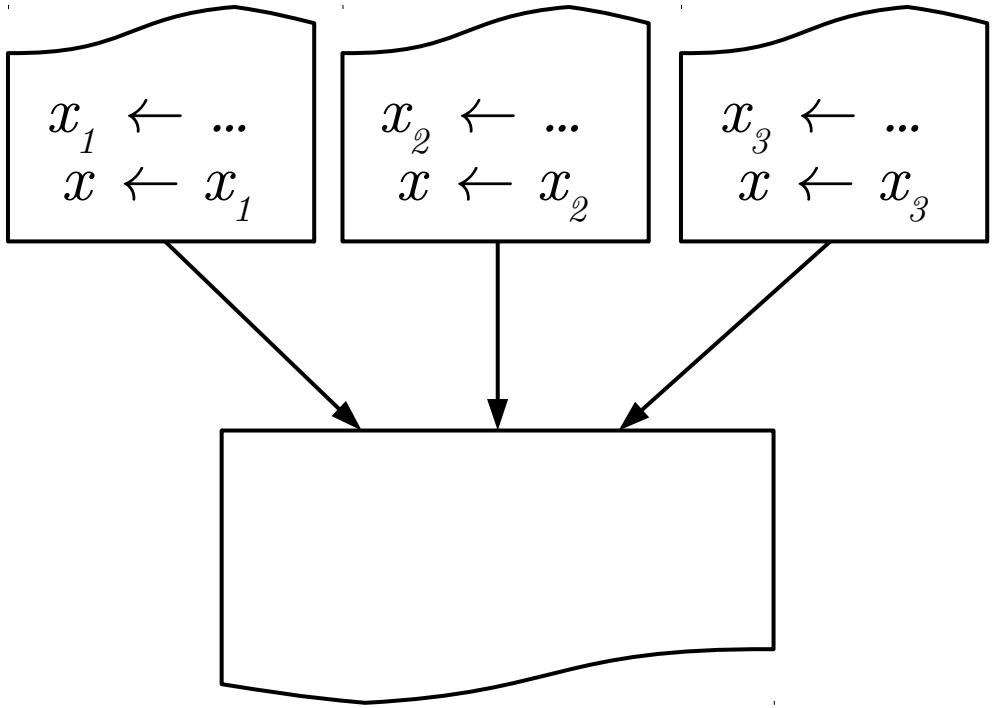
$$x \leftarrow x_1 \qquad x \leftarrow x_2 \qquad x \leftarrow x_3$$

$$x \leftarrow \phi\,(x_1 \ldots x_n)$$

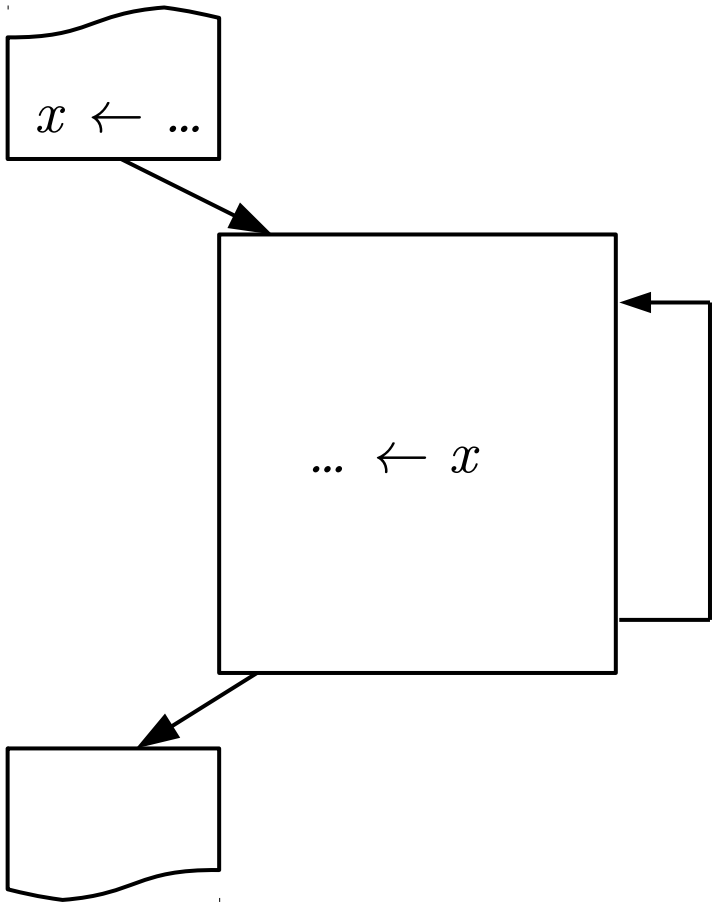
Static Single Information (SSI)



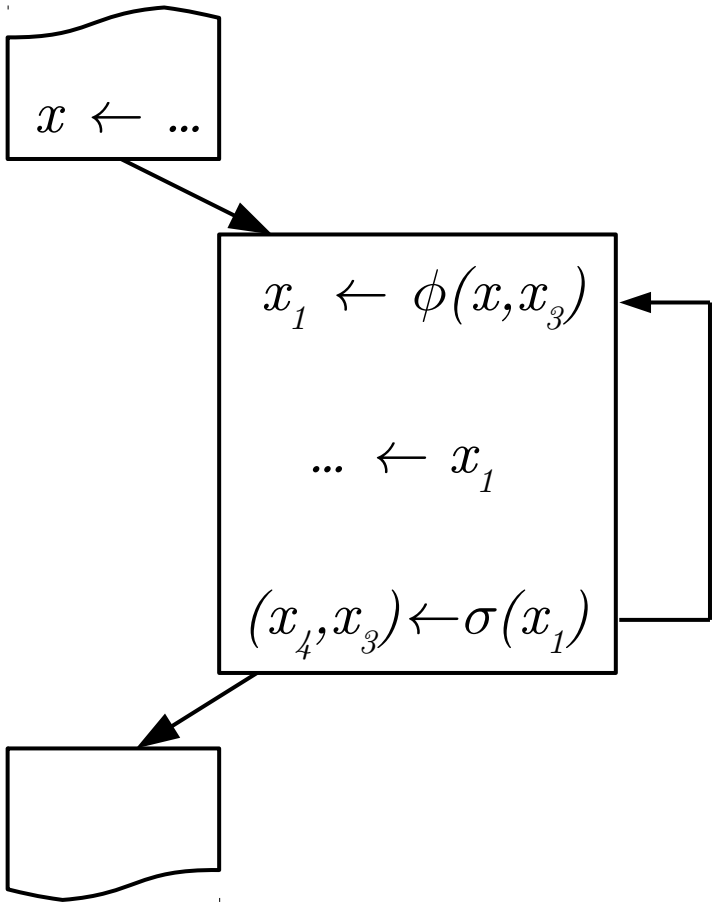
Static Single Information (SSI)



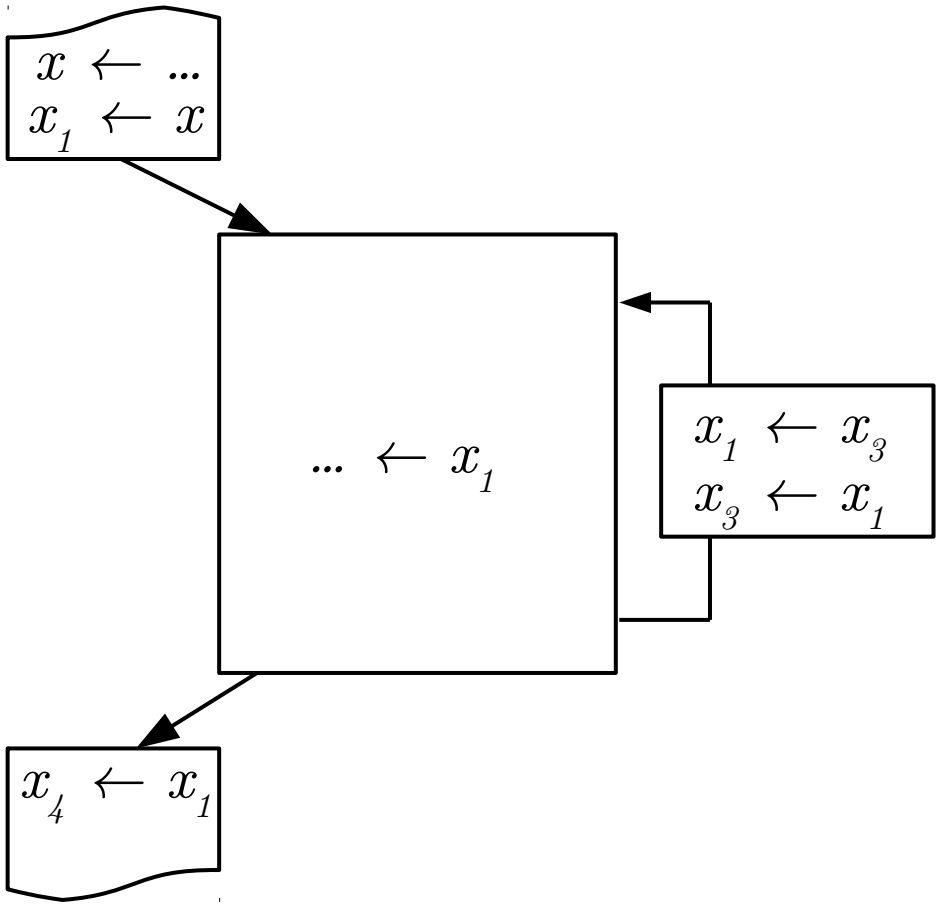
Static Single Information (SSI)



Static Single Information (SSI)



Static Single Information (SSI)



Outline

1. Introduction / Motivation
   - the SCAD machine
   - move code
2. Control flow
3. Remove constraints per basic-block
4. Summary

SSI+

- transformation on top of SSI
- unique use sites
- reduce buffer sizes

SSI+

- 2 copies of a variable waiting in an output buffer
- one copy to be used / consumed
- one to be duplicated for further use

SSI+

$$x \leftarrow \ldots$$

$$\ldots \leftarrow x$$

$$\ldots \leftarrow x$$

$$\ldots \leftarrow x$$

SSI+

$$x \leftarrow \ldots$$

$$x_1 \leftarrow x$$
$$\ldots \leftarrow x$$

$$\ldots \leftarrow x_1$$

$$\ldots \leftarrow x_1$$

SSI+

$$x \leftarrow \ldots$$

$$\begin{array}{l} x_1 \leftarrow x \\ \ldots \leftarrow x \end{array}$$

$$\ldots \leftarrow x_1$$

$$\ldots \leftarrow x_1$$

SSI+

$$x \leftarrow \ldots$$

$$x_1 \leftarrow x$$
$$\ldots \leftarrow x$$

$$\ldots \leftarrow x_1$$

$$\ldots \leftarrow x_1$$

SSI+

$$x \leftarrow \ldots$$

$$\begin{aligned} x_1 &\leftarrow x \\ \ldots &\leftarrow x \end{aligned}$$

$$\begin{aligned} x_2 &\leftarrow x_1 \\ \ldots &\leftarrow x_1 \end{aligned}$$

$$\ldots \leftarrow x_2$$

SSI+

$$x \leftarrow \ldots$$

$$x_1 \leftarrow x$$
$$\ldots \leftarrow x$$

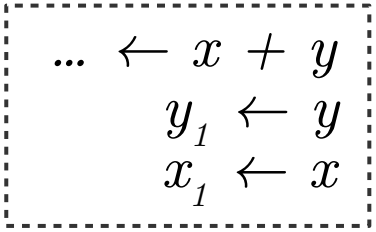
$$x_2 \leftarrow x_1$$
$$\ldots \leftarrow x_1$$

$$\ldots \leftarrow x_2$$

SSI+

$$\ldots \leftarrow x + y$$

SSI+



SSI+

$$
\begin{aligned}
x_1 &\leftarrow x \\
\ldots &\leftarrow x + y \\
y_1 &\leftarrow y
\end{aligned}
$$

Buffer Interferences

now after SSI+

- unique definition
- unique use-site (maybe several uses)

Buffer Interferences

now after SSI+

- unique definition
- unique use-site (maybe several uses)

detect buffer interferences for two variables x and y

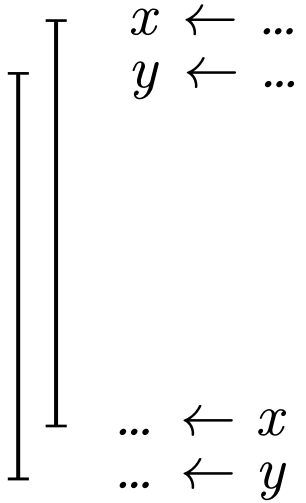
- x and y enqueued and dequeued in same order (ok)
- else: x and y interfere

Buffer Interferences

$$x \leftarrow \ldots$$
$$y \leftarrow \ldots$$

$$\ldots \leftarrow x$$
$$\ldots \leftarrow y$$

Buffer Interferences

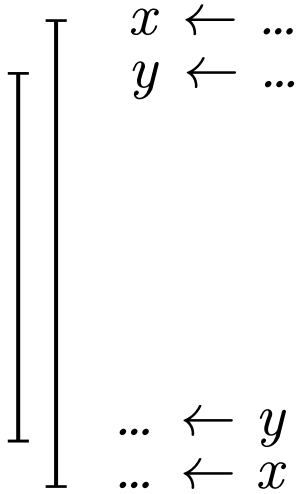


Buffer Interferences

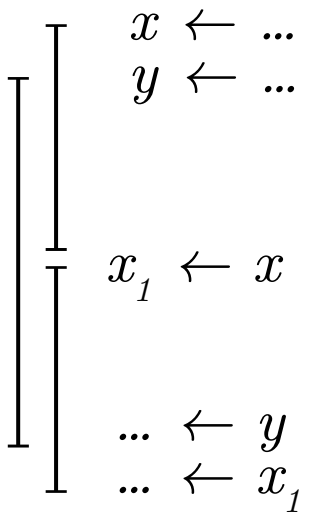
$$x \leftarrow \ldots$$
$$y \leftarrow \ldots$$

$$\ldots \leftarrow y$$
$$\ldots \leftarrow x$$

Buffer Interferences



Buffer Interferences



Summary

- control flow problems solved
- no buffer interferences

Summary

- control flow problems solved
- no buffer interferences

$\rightarrow$ generate move code for arbitrary mapping and arbitrary number of PUs

Summary

- SSI (control flow & def-sites unique)
- SSI+ (make use-sites unique)
- encapsulated intervals (remaining buffer interferences)
- instruction-wise move code generation

Summary

- SSI (control flow & def-sites unique)
  - code size ca. doubled
- SSI+ (make use-sites unique)

- encapsulated intervals (remaining buffer interferences)

- instruction-wise move code generation

Summary

- SSI (control flow & def-sites unique)
  - code size ca. doubled
- SSI+ (make use-sites unique)
  - small increase only
- encapsulated intervals (remaining buffer interferences)

- instruction-wise move code generation

Summary

- SSI (control flow & def-sites unique)
  - code size ca. doubled
- SSI+ (make use-sites unique)
  - small increase only
- encapsulated intervals (remaining buffer interferences)
  - almost none or heavy increase in code size depending on the program
- instruction-wise move code generation

Questions?