A Gentle Introduction to High Level Synthesis

Jianwen Zhu

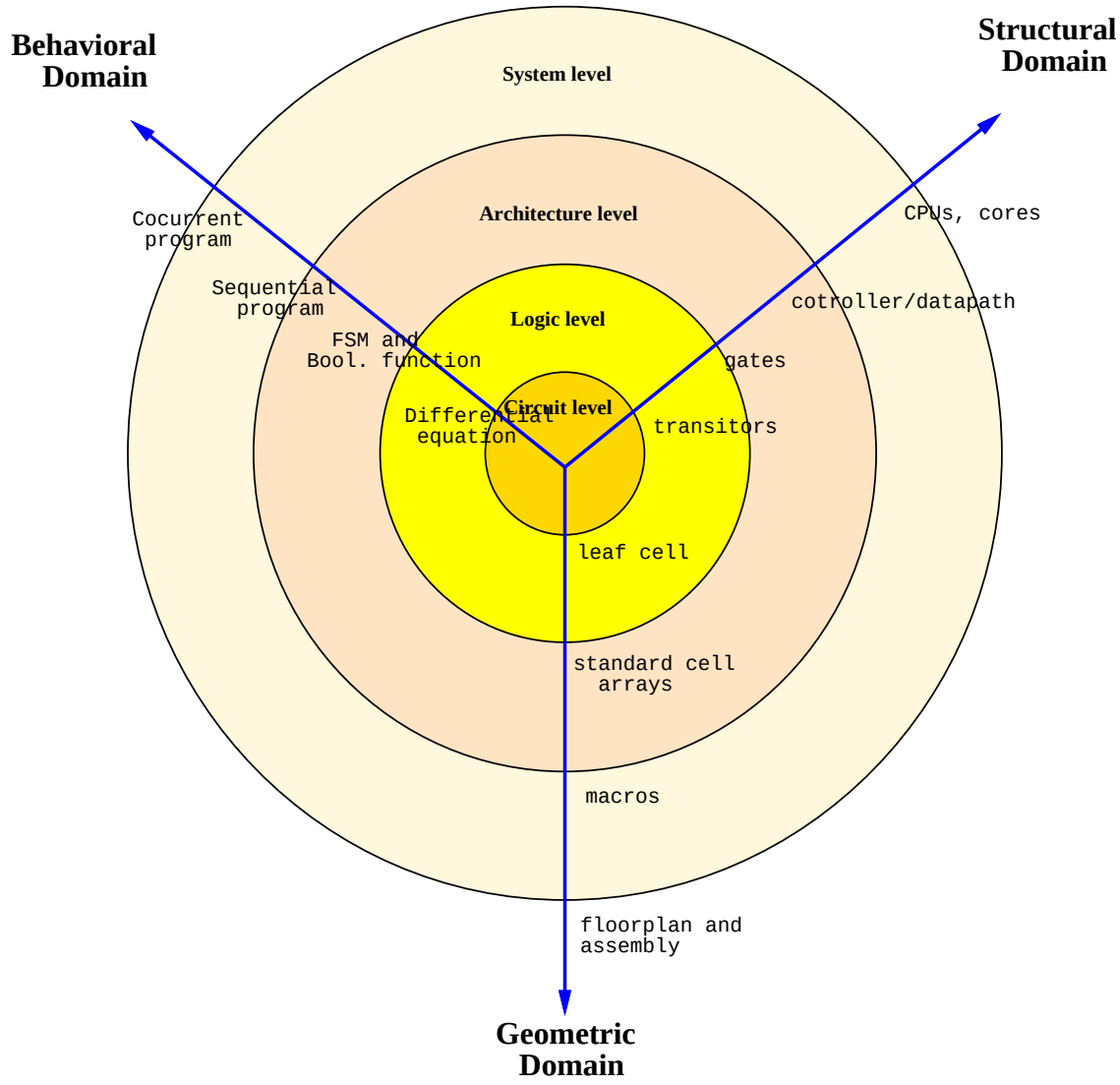
Electrical and Computer Engineering
University of Toronto

jzhu@eecg.toronto.edu
http://www.eecg.toronto.edu/~jzhu

Outline

- *Overview*
- Scheduling
- Resource Sharing
- Summary

Y-chart



High-level Synthesis

- What is Synthesis
  - Given a functional model of a design
  - Find a structural model of a design
  - Such that some figure of merit is optimized
    - Speed
    - Area
    - Power
    - Noise
  - Subject to some constraints

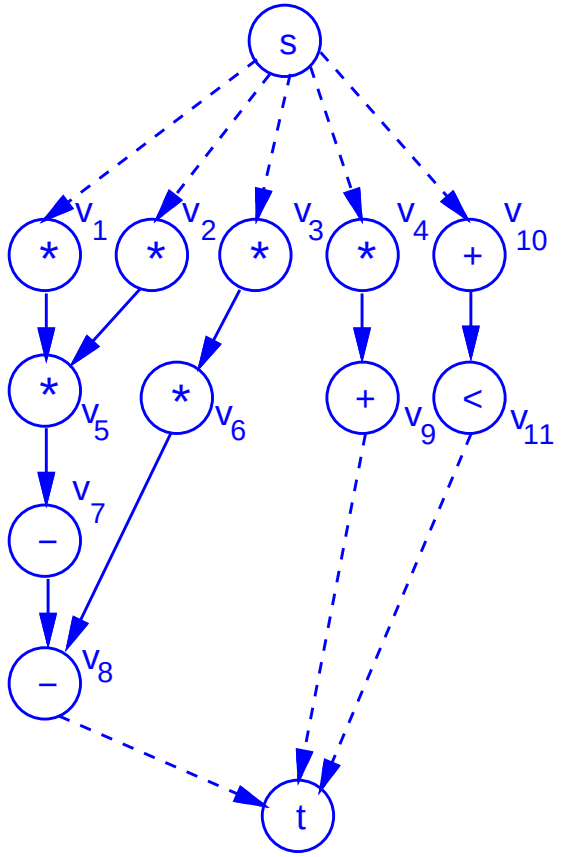
- What is High-level Synthesis
  - Given an algorithm model of a design
  - Find a micro-architecture
    - Controller: sequential random logic, ROM
    - Datapath: adder, ALU, mux, register, register file
  - Such that speed/area/power is optimized
  - Subject to some constraints

High-level Function Model

- Can be captured by an imperative program
  - C/C++, Java, ...
  - Behavioral VHDL/Verilog
- Untimed state machine
- Can be transformed into control-dataflow graph (CDFG) by synthesis front-end
  - Ease for machine manipulation
  - Control flow: statement
  - Data flow: expression

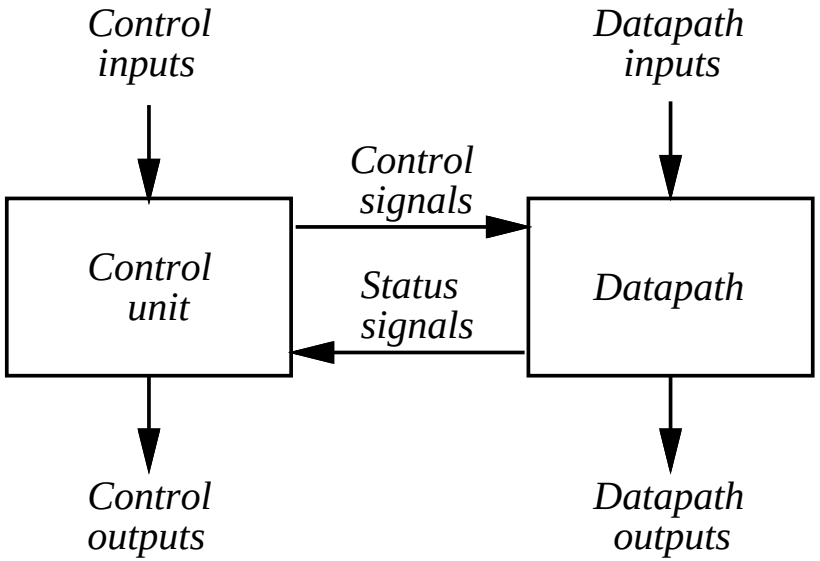
Example of Dataflow Graph

```
...                                    1
while( x ¡a ) {                        2
   x1 = x + dx;                        3
   u1 = u - (3 * x + u * dx )          4
        - (3 * y * dx);                5
   y1 = y + u * dx;                    6
   x = x1;                             7
   u = u1;                             8
   y = y1;                             9
   }                                  10
...                                   11
```



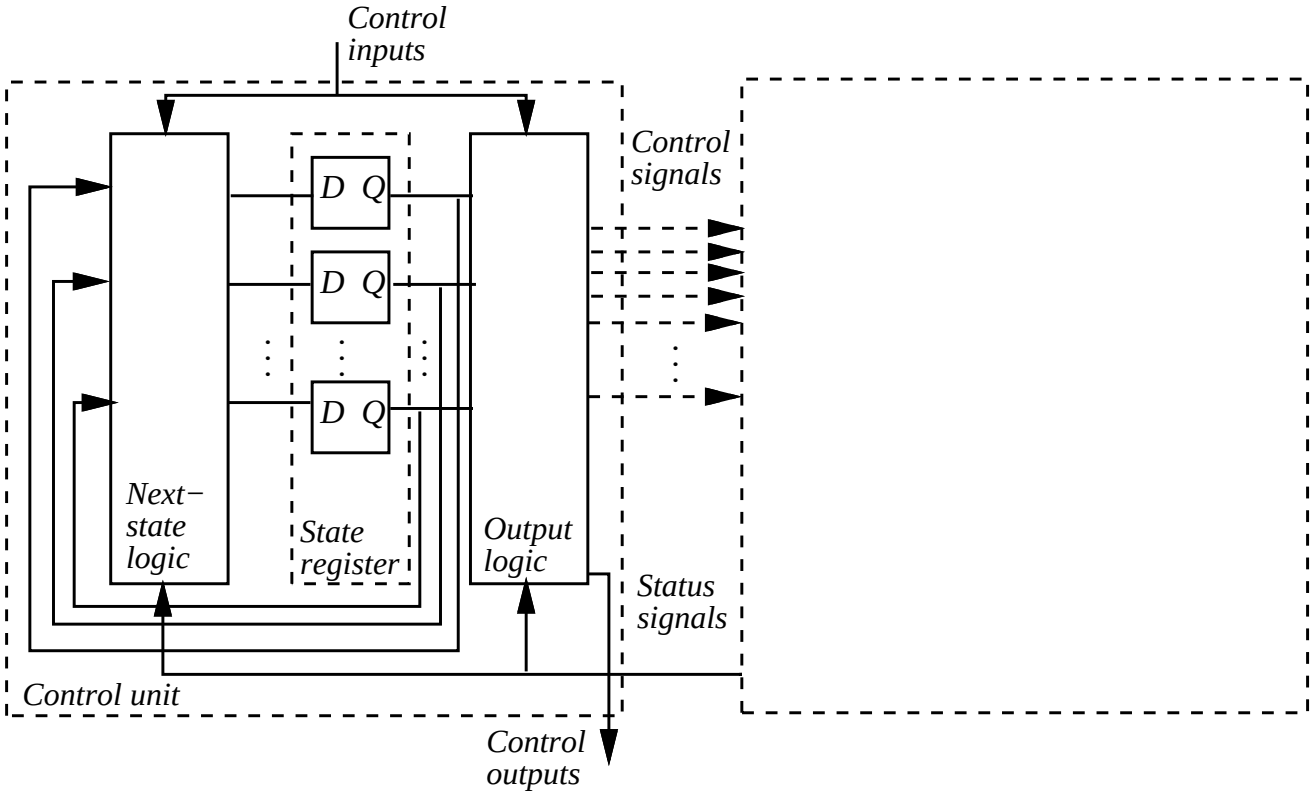
Micro-architecture

- Controller is responsible for *when* and *what* register transfer operations (assignments) to perform
- Datapath is responsible for *how* to perform the register transfer operations



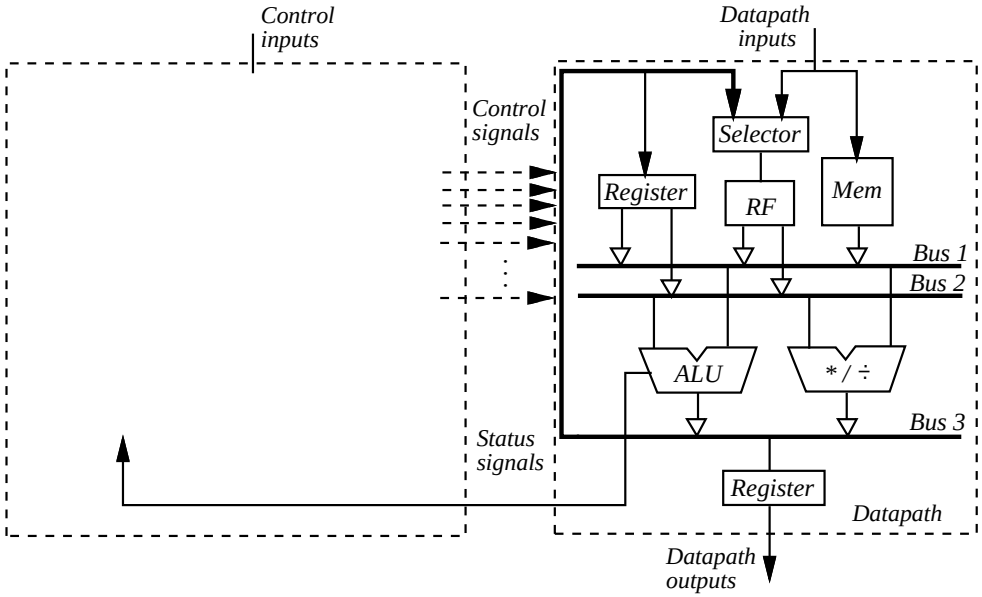
Micro-architecture: Controller

- Controller is a sequential network
- Can be implemented using logic synthesis and layout tool



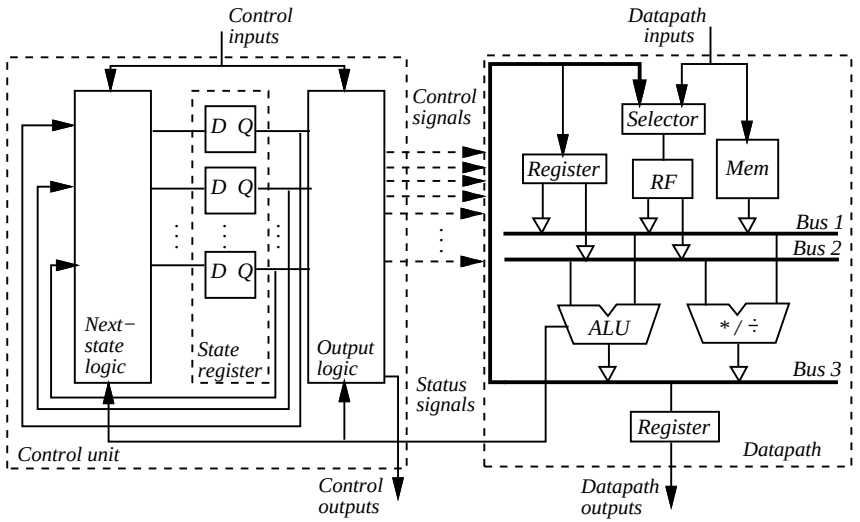
Micro-architecture: Datapath

- Datapath is a network of sequential and combinational components:
  - Registers, register files
  - Adders, Subtracters, ...
  - Steering components: buses, selectors, ...



Quality Metrics/Constraints

- Latency: the time it takes to process the data
- Throughput: the rate to process the data
- Cycle time
- Area
- Power



Outline

- Overview
- *Scheduling*
- Resource Sharing
- Summary

Scheduling

- Have to decide *when* each operation is performed

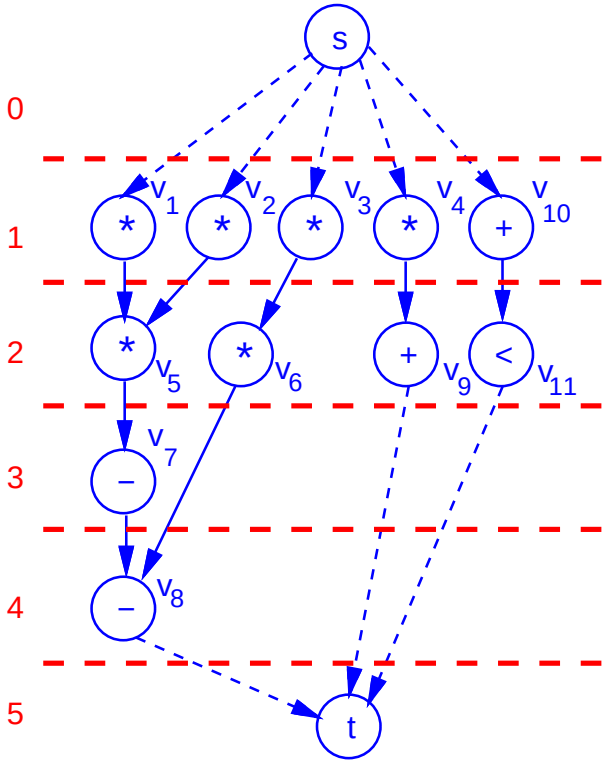
  untimed state machine $\mapsto$ timed state machine

  flow graph $\mapsto$ ASM chart

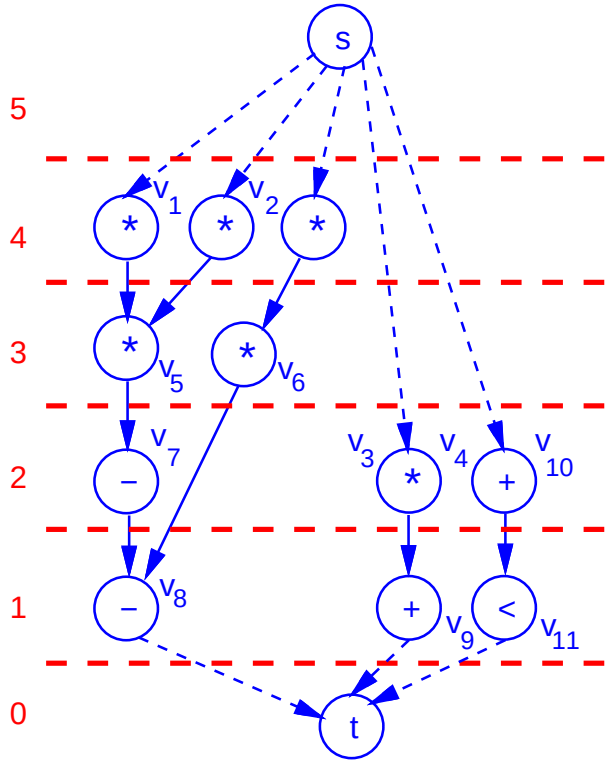
- Pretty much like the determine the class schedule
  - Dependency constraint: ECE241 is a prerequisite of ECE451
  - Resource constraint: Do not have enough #rooms to hold all classes simultaneously
- Here
  - Dependency constraint: A depends on the result of B
  - Resource constraint: there are at most 3 adders

Unconstrained Scheduling: List Scheduling

- As Soon As Possible (ASAP) schedule



- As Late As Possible (ALAP) schedule



List Scheduling Example

- Keep a list a ready operations

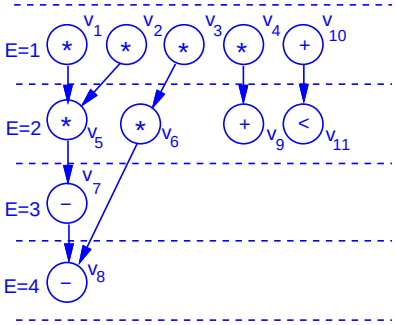
  whose predecessors are all scheduled

- Schedule an operation from the ready list
  - Has no resource conflict
  - Has higher priority

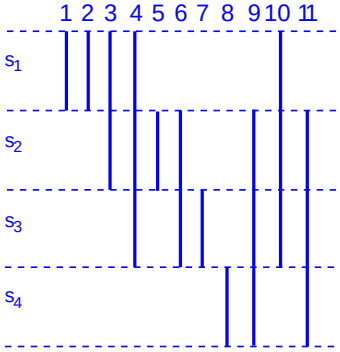
- Heuristics to determine the priority
  - Mobility
  - Out degree
  - Distance to the sink

Resource-constrained Scheduling: List Scheduling

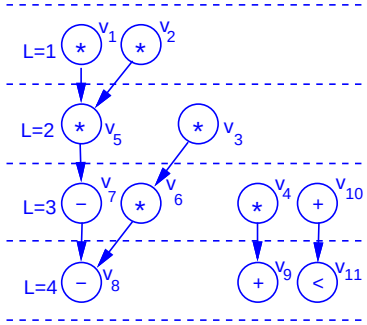
ASAP Schedule



Operator Mobility

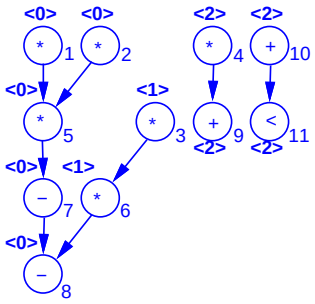


ALAP Schedule



Mobility(op) = ASAP − ALAP

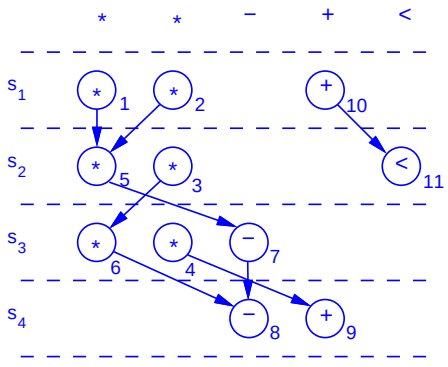
Node:	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11
Operation:	*	*	*	*	*	*	−	−	+	+	<
Mobility:	0	0	1	2	0	1	0	0	2	2	2



DFG with mobilities

$Resources_{*}$: 2
$Resources_{+}$: 1
$Resources_{-}$: 1
$Resources_{<}$: 1

Resource Constraints



Scheduled DFG

Outline

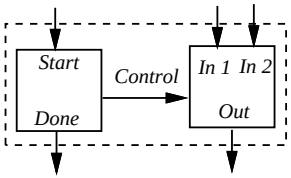
- Overview
- Scheduling
- *Resource Sharing*
- Summary

Square Root Approximation (SRA) Example

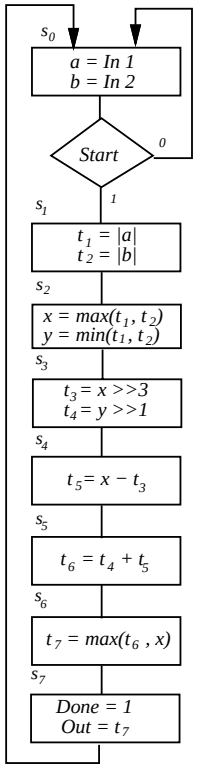
- Computes $O = \sqrt{a^2 + b^2}$
- Approximation:

  $O = max((0.875x + 0.5y), x)$ where

  $x = max(|a|, |b|), y = min(|a|, |b|)$
- A scheduled design
- ASM chart



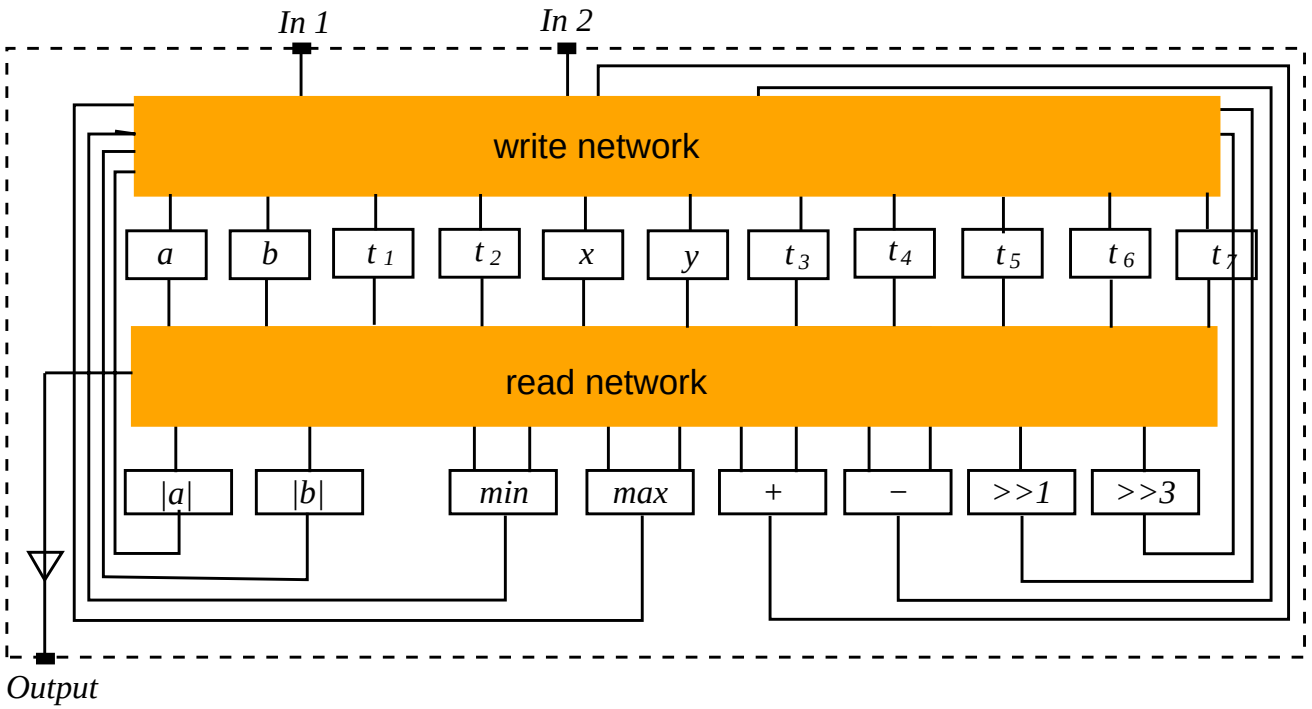
(a) Block diagram



(b) ASM Chart of square−root approximation

A Straight-forward Approach

- Map each variable to a distinct register
- Map each operations to a distinct combinational component

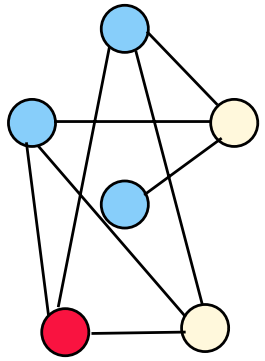


Solving Resource Sharing Problem

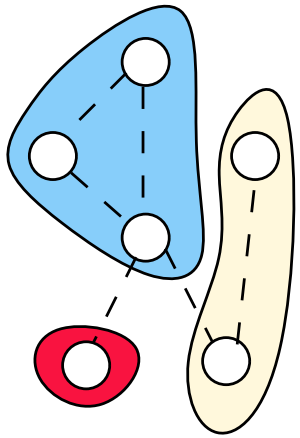
- Resources (registers, functional units) can be shared
- Cast resource sharing problem into a graph problem
  - Graph nodes: subject objects to be shared
  - Graph edges: sharing relation
  - Dual problem:
    - Coloring of conflict graph
    - Clique-partitioning of compatibility graph

Resource Sharing Algorithm

- Graph coloring
  - Color one node at a time
  - Select the color different from its neighbors



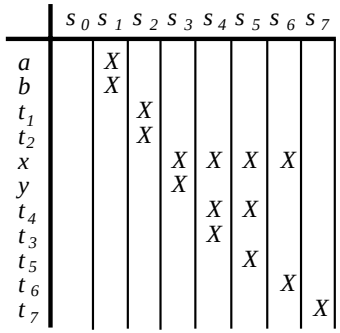
- Graph partitioning
  - Select two compatible nodes at a time
  - Merge them into a supernode
  - Update edges accordingly

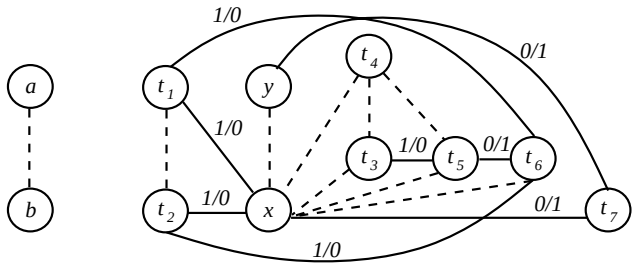


Register Sharing

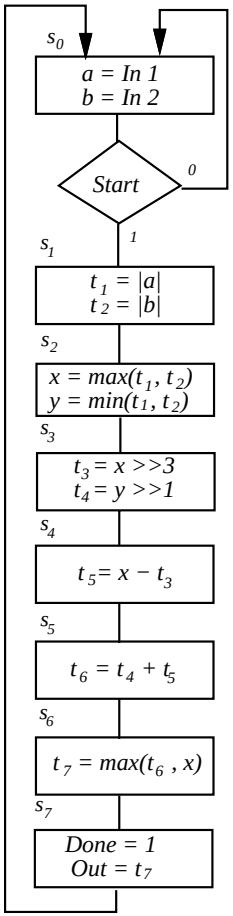
- Two variables can share the same register if and only if they have non-overlapping life time
- Lifetime = [first time write, last time read]
- Casting register sharing into Graph partitioning problem
  - Each vertex represents a variable
  - There is a *dashed edge* between two vertices if the corresponding vertices have overlapping life time
  - There is a *solid edge* between two vertices if the corresponding vertices have non-overlapping life time
  - The solid edge is annotated with #common src/#common dest

Variable Compatibility Graph

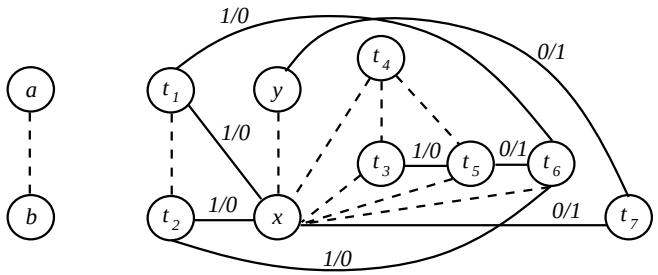




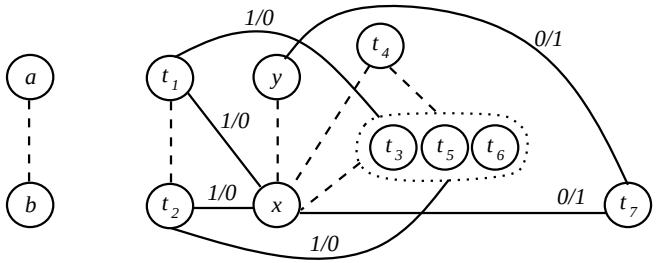
(a) Initial compatibility graph



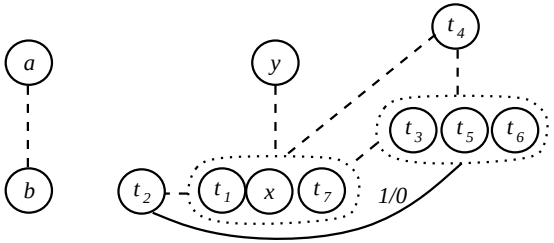
Register Sharing by Graph Partitioning



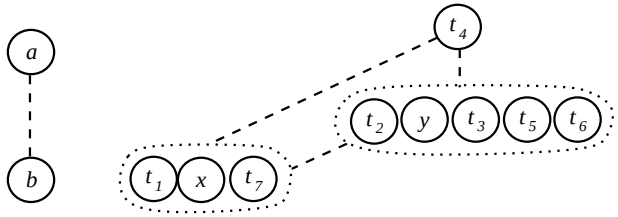
(a) Initial compatibility graph



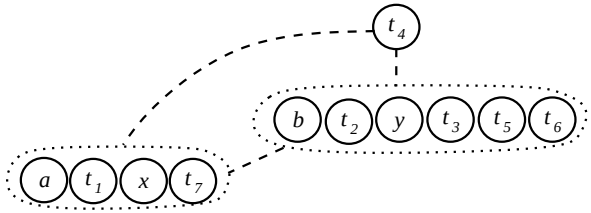
(b) Compatibility graph after merging t_3, t_5 and t_6



(c) Compatibility graph after merging t_1, x, and t_7



(d) Compatibility graph after merging t_2 and y

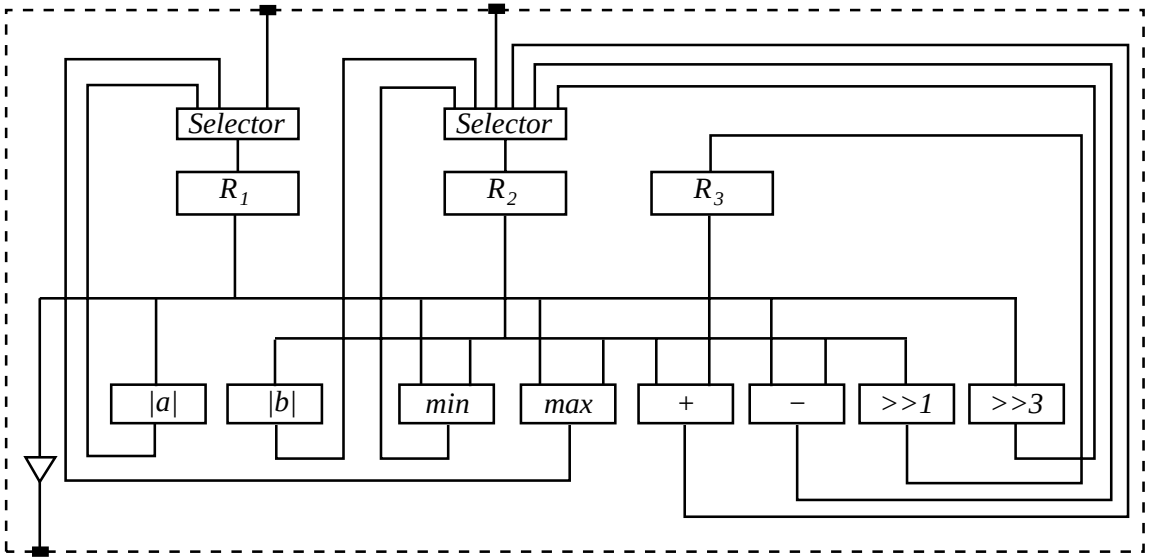


(e) Final compatibility graph

SRA Implementation after Register Sharing

$R_1 = [a, t_1, x, t_7]$
$R_2 = [b, t_2, y, t_3, t_5, t_6]$
$R_3 = [t_4]$

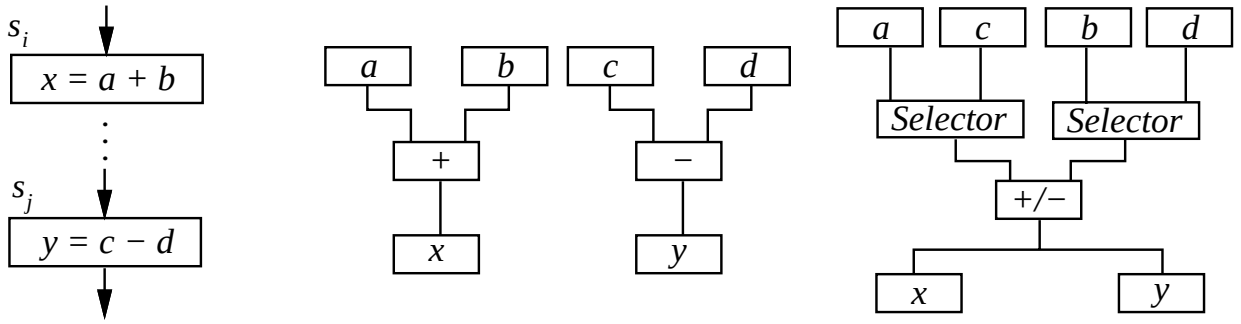
(a) Register assignments



(b) Datapath

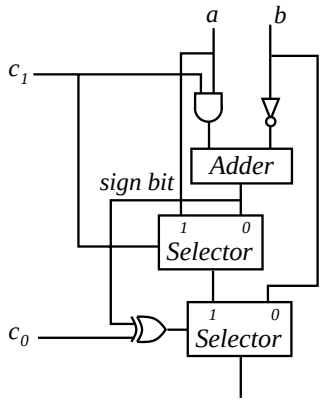
Functional Unit Sharing

- Two Operations can share the same functional unit if they are non-concurrent and has “similar” functionality
- Sharing priority: #common source, #common destinations



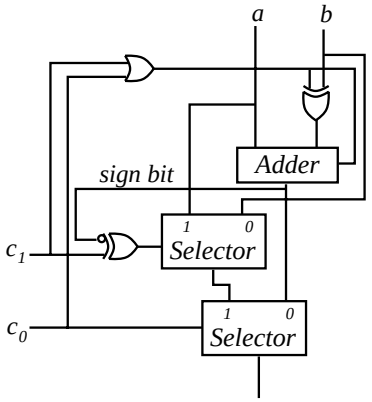
(a) Partial ASM Chart (b) Non−shared design (c) Shared design

More Components in the Library



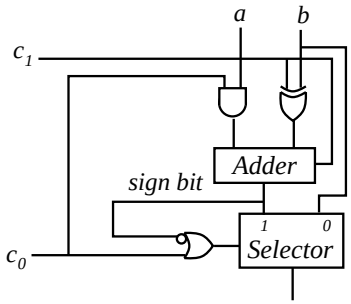
c_1 c_0	Operation
0 1	absolute
1 0	minimum
1 1	maximum

(a) Unit for computing minimum, maximum and absolute value



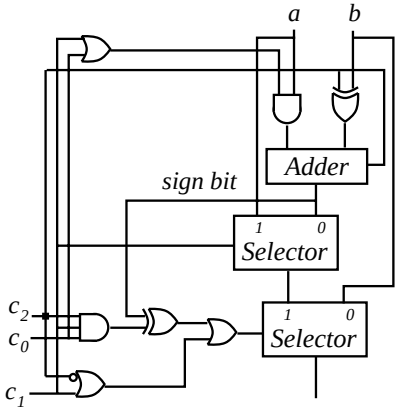
c_1 c_0	Operation
0 0	addition
0 1	minimum
1 0	subtraction
1 1	maximum

(b) Unit for computing addition, subtraction, minimum and maximum



c_1 c_0	Operation
1 0	addition
0 1	absolute
1 1	subtraction

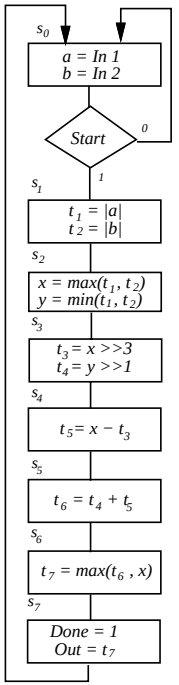
(c) Unit for computing addition, subtraction, and absolute value



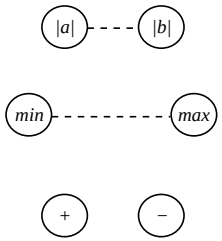
c_2 c_1 c_0	Operation
0 0 1	addition
1 0 0	absolute
1 0 1	subtraction
1 1 0	minimum
1 1 1	maximum

(d) Unit for computing addition, subtraction, minimum, maximum and absolute value

Functional Unit Sharing by Graph Partitioning



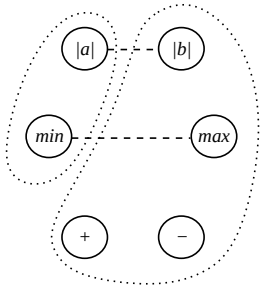
ASM Chart



(a) Compatibility graph

Unit \ Component	AND logic	Invert logic	EX−OR logic	Adder	Selector
|a|		1		1	1
|b|		1		1	1
min		1		1	1
max		1		1	1
+				1	
−		1		1	
Total		5		6	4

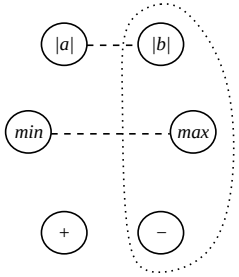
(b) Cost table



(c) Merging alternative

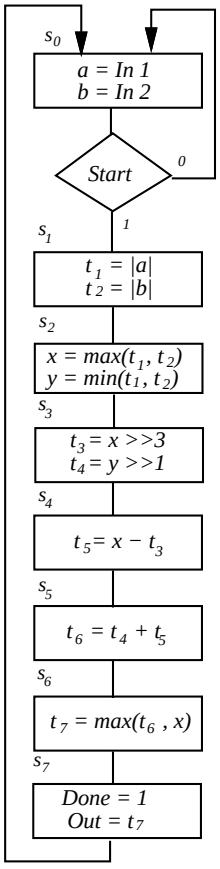
Unit \ Component	AND logic	Invert logic	EX−OR logic	Adder	Selector
[|a|/min]	1	1		1	2
[|b|/max/+/−]	1		1	1	2
Total	2	1	1	2	2

(d) Cost table

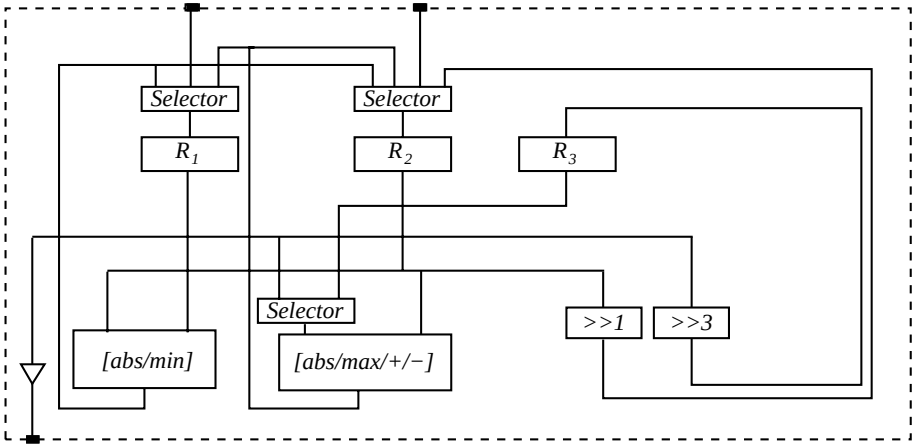


Unit \ Component	AND logic	Invert logic	EX−OR logic	Adder	Selector
[|a|/min/+]	1		1	1	2
[|b|/max/−]	1	1		1	2
Total	2	1	1	2	2

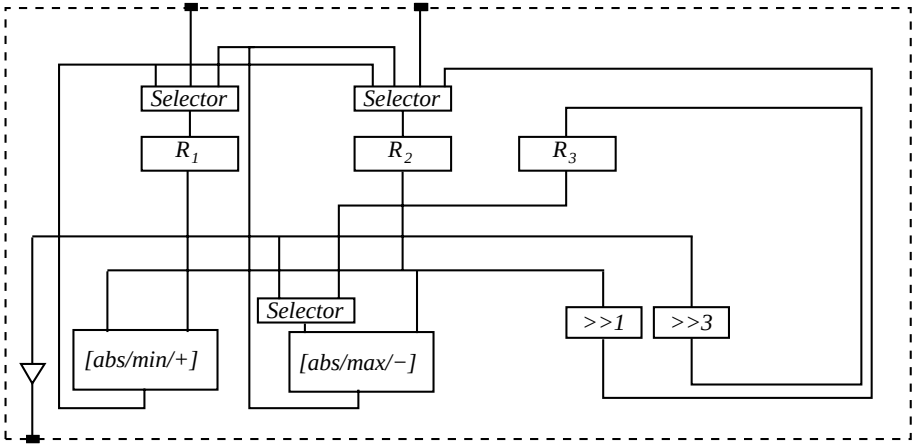
SRA Implementation after Functional Unit Sharing



ASM Chart



(a) Datapath schematic for unit allocation from Figure 8.22(c)

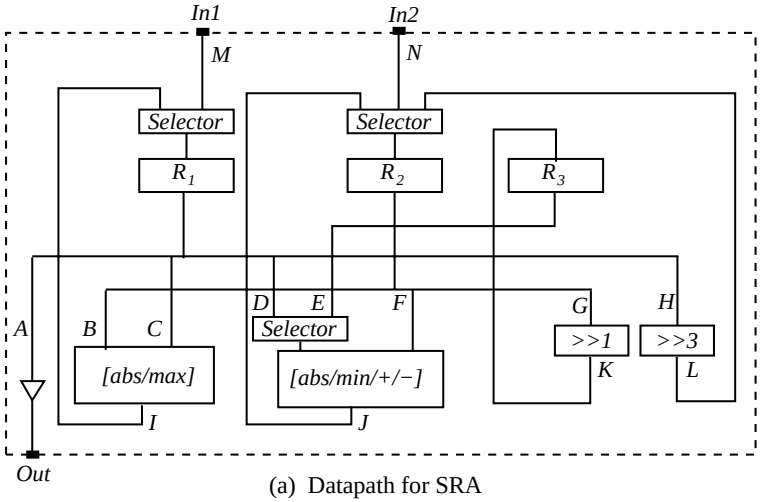


(b) Datapath schematic for unit allocation from Figure 8.22(e)

Bus Sharing

- Wires can be expensive
- Different interconnections can be shared if they are not used at the same cycle
- Pitfall: may end up longer wires

Bus Sharing by Graph Partitioning



(a) Datapath for SRA

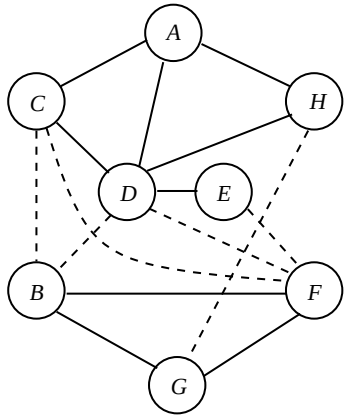
Bus1 = [A, C, D, E, H]
Bus2 = [B, F, G]

Bus3 = [I, K, M]
Bus4 = [J, L, N]

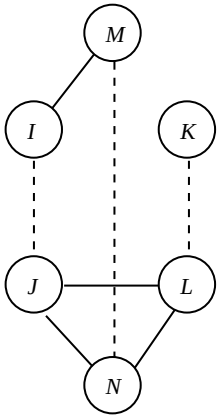
(e) Bus assignment

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
A								X
B			X				X	
C		X	X				X	
D			X		X			
E						X		
F		X	X		X	X		
G				X				
H				X				
I		X	X				X	
J		X	X		X	X		
K				X				
L				X				
M	X							
N	X							

(b) Connectivity usage table

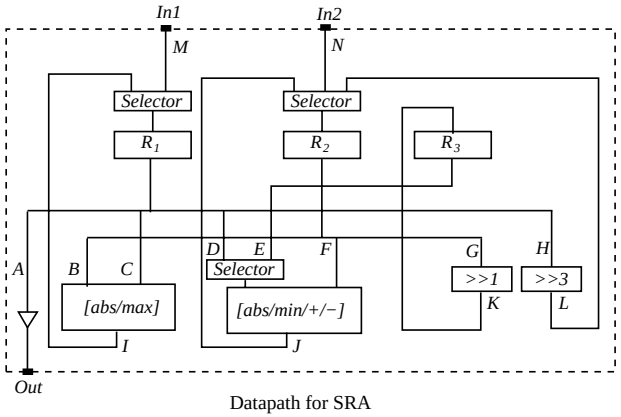


(c) Compatibility graph for input buses



(d) Compatibility graph for output buses

SRA Implementation after Bus Sharing

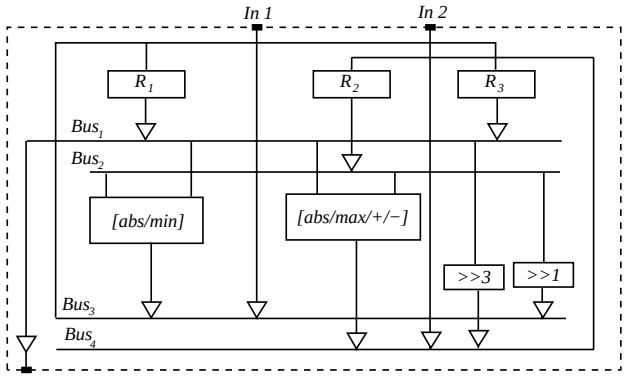


Datapath for SRA

Bus1 = [A, C, D, E, H]
Bus2 = [B, F, G]

Bus3 = [I, K, M]
Bus4 = [J, L, N]

Bus assignment



(f) Bus oriented datapath

Summary

- High-level synthesis maps an imperative program into a micro-architecture, which can be further synthesized by lower-level tools
- Scheduling determines the control step at which each operation is performed
- Binding determines how variables, operations, data transfered are mapped into shared registers, functional units and buses.