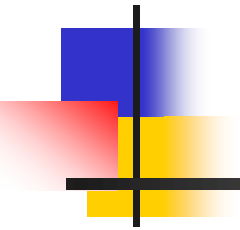


數位IC設計

Datapath

Datapath Optimization

Datapath optimization: (done by high-level synthesis tool)

1. Resource optimization

(a) storage sharing (b) functional unit sharing

(c) bus sharing (d) register merging

2. Time optimization

(a) chaining or multicycling (b) functional-unit pipelining

(c) datapath pipelining (d) controlpath pipelining

The square-root approximation (SRA) is used in the following discussion

$$\sqrt{a^2+b^2} \approx \max((0.875x+0.5y), x)$$

where

***x*=max(|a| , |b|)**

***y*=min(|a| , |b|)**

min: minimum operation
max: maximum operation

Scheduling

1. **Resource-constrained scheduling:**

   fixed hardware components

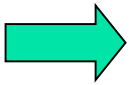
   e.g., one arithmetic unit (abs, max, min, +, -)

   two shifter

2. **Time-constrained scheduling:**

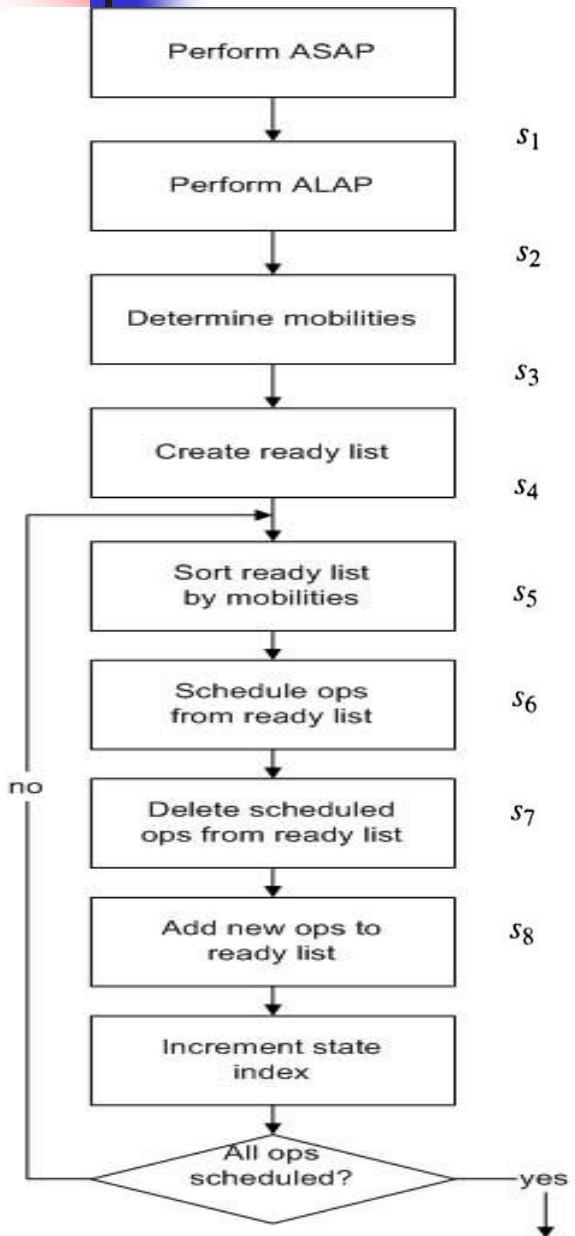
   fixed timing performance

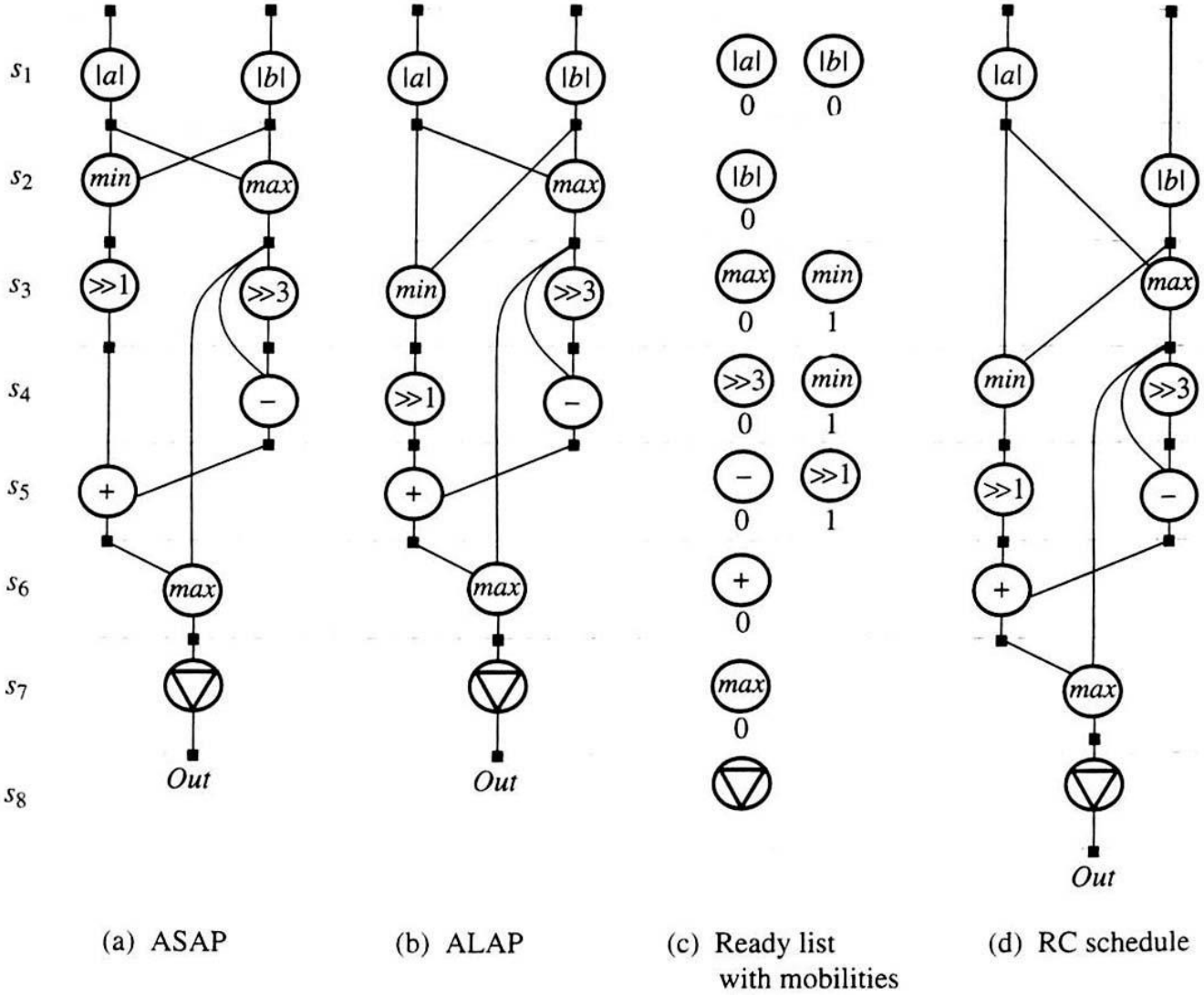
   e.g., eight states or five states



Tradeoffs problem

Resource-Constrained Scheduling

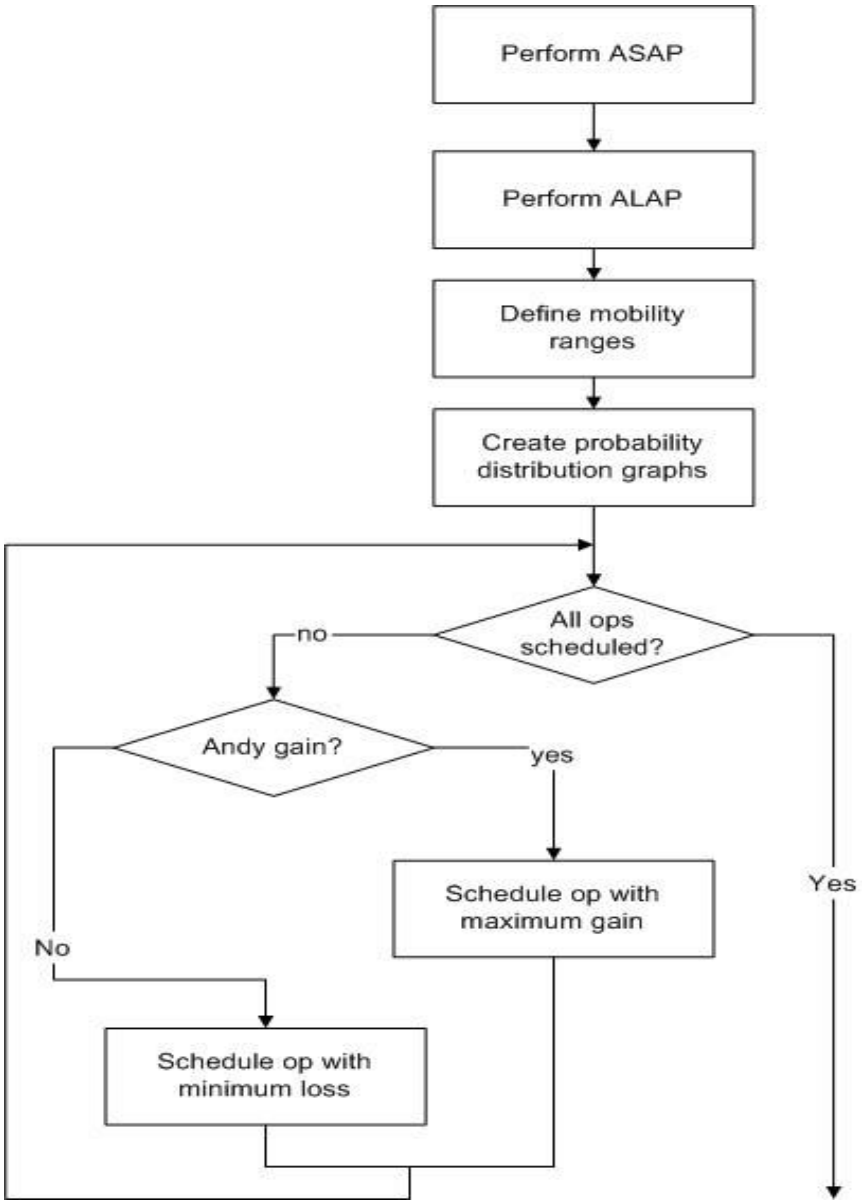




(a) ASAP (b) ALAP (c) Ready list with mobilities (d) RC schedule

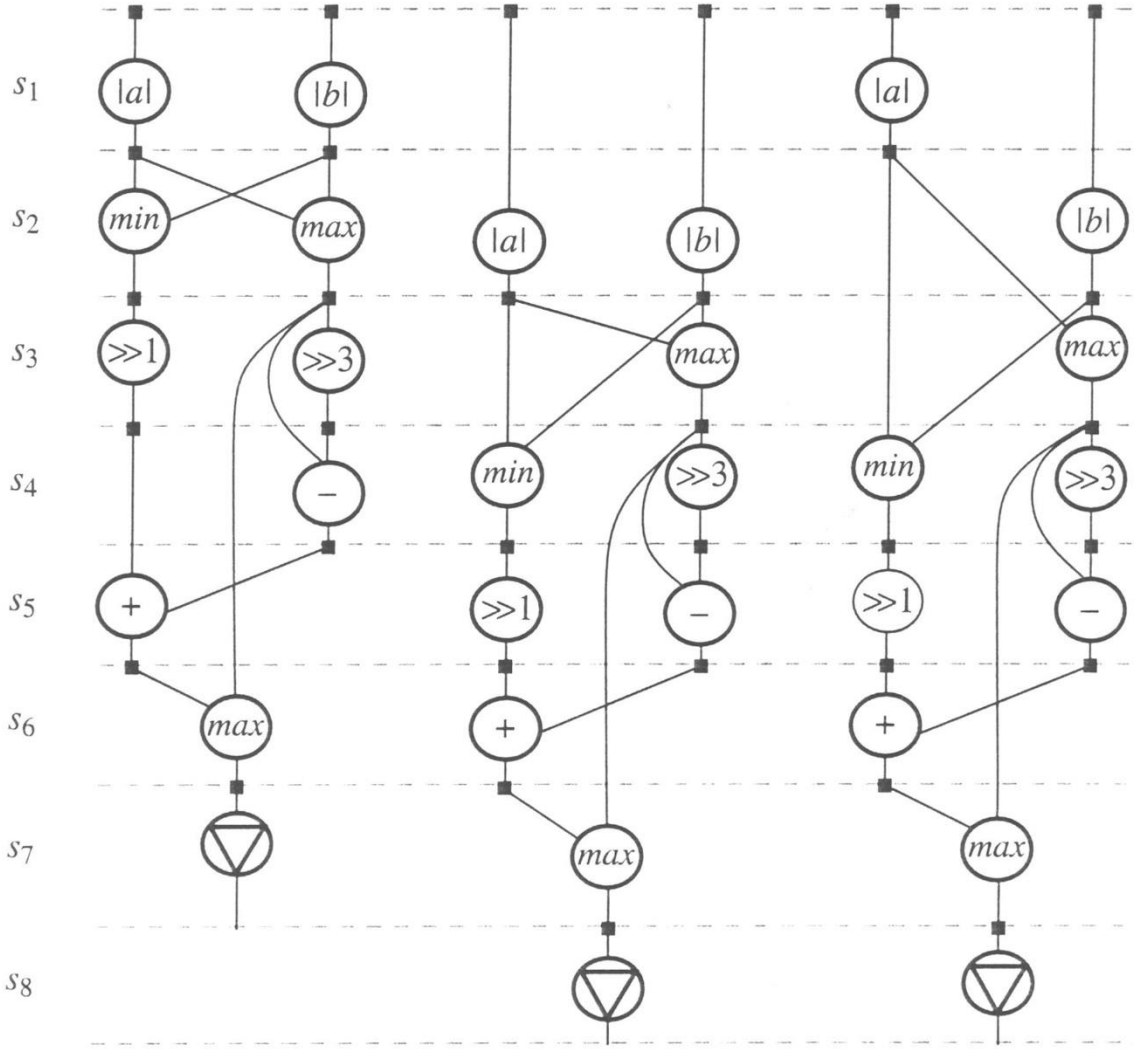
List scheduling algorithm

Time-Constrained Scheduling (1/3)



Time-Constrained Scheduling (2/3)

If the time-constrain is 8 clock cycles



Time-Constrained Scheduling (3/3)

STATES	AU UNITS	PROBABILITY SUM/STATE	SHIFT UNITS	PROBABILITY SUM/STATE
s_1	|a|	1.0		
s_2	|b|	1.83		
s_3	max	.83	≫3	.83
s_4	min	.83	≫3	.83
s_5	−	1.0		.33
s_6	+	1.0		
s_7	max	.5		

(a) Initial probability distribution graph

STATES	AU UNITS	PROBABILITY SUM/STATE	SHIFT UNITS	PROBABILITY SUM/STATE
s_1	|a|	1.0		
s_2	|b|	1.33		
s_3	max	1.33		.33
s_4	min	.33	≫3	1.33
s_5	−	1.0		.33
s_6	+	1.0		
s_7	max	1.0		

(b) Distribution graph after max, +, and − were scheduled

STATES	AU UNITS	PROBABILITY SUM/STATE	SHIFT UNITS	PROBABILITY SUM/STATE
s_1	|a|	1.0		
s_2	|b|	1.0		
s_3	max	1.0		
s_4	min	1.0	≫3	1.0
s_5	−	1.0	≫1	1.0
s_6	+	1.0		
s_7	max	1.0		

(c) Distribution graph after max, min, +, −, ≫3, and ≫1 were scheduled

STATES	AU UNITS	PROBABILITY SUM/STATE	SHIFT UNITS	PROBABILITY SUM/STATE
s_1	|a|	1.0		
s_2	|b|	1.0		
s_3	max	1.0		
s_4	min	1.0	≫3	1.0
s_5	−	1.0	≫1	1.0
s_6	+	1.0		
s_7	max	1.0		

(d) Distribution graph for final schedule

Resource Optimization

Datapath optimization:

1. **Resource optimization**

   (a) storage sharing **(b) functional unit sharing**

   (c) bus sharing **(d) register merging**

2. **Time optimization**

   (a) chaining or multicycling **(b) functional-unit pipelining**

   (c) datapath pipelining **(d) controlpath pipelining**

Register Sharing for SRA (1/5)

Resource optimization

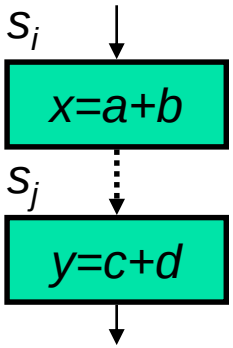
(a) storage sharing **(b) functional-unit sharing**

(c) bus sharing **(d) register merging**

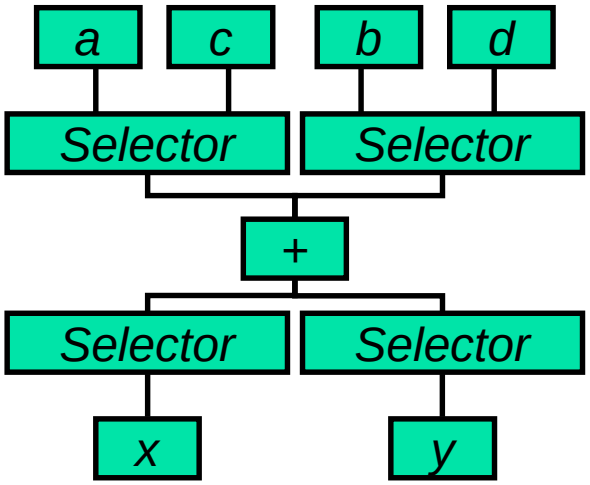
Storage sharing ⇨ Register sharing (variable merging)

⇨ Minimize the number of registers

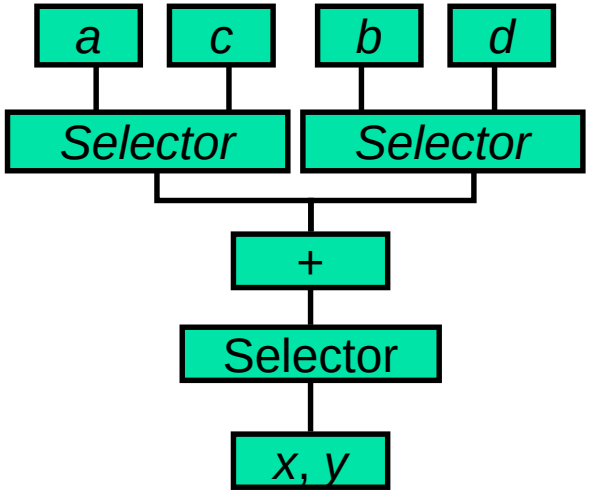
It is different with software program (think with hardware)



(a) Partial ASM chart



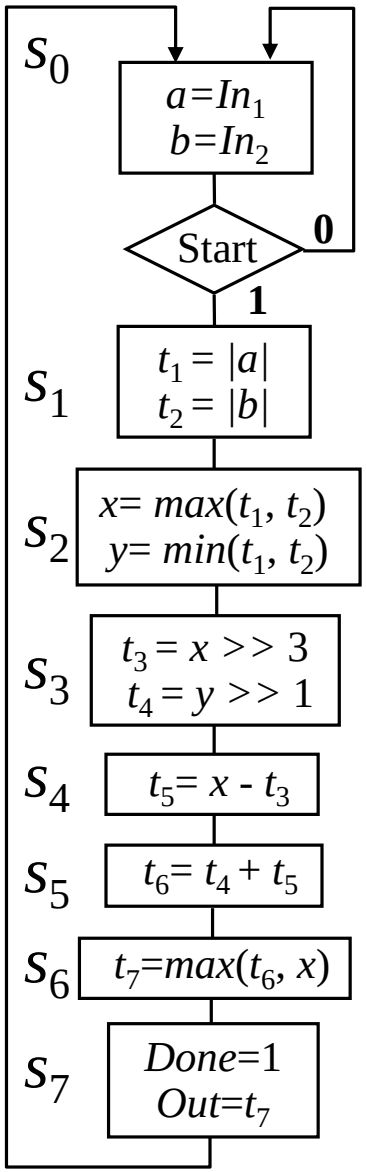
(b) Datapath without Register sharing



(c) Datapath with register sharing (possible only in the condition that lifetimes of x and y are not overlapped)

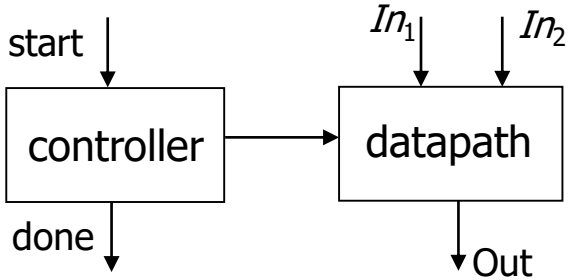
If the lifetimes of *a* and *c* are not overlapped, ..

Register Sharing for SRA (2/5)



$$\sqrt{a^2+b^2} \approx \max((0.875x+0.5y), x)$$

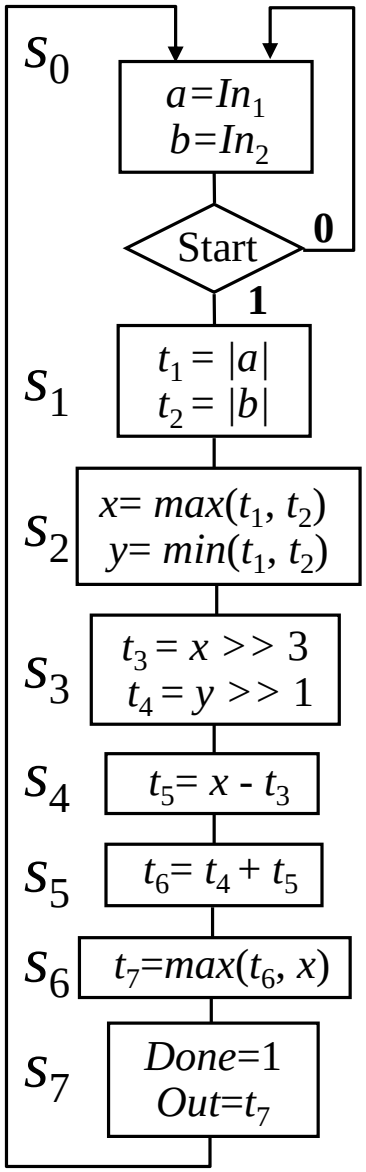
x=max(|a|, |b|) y=min(|a|, |b|)



Register Sharing:

1. **Determine the number of states (scheduling)**
2. **Determine the variables' lifetimes**
3. **Find the minimum number of necessary registers**
4. **Allocate each variable to a proper register**

Register Sharing for SRA (3/5)



Lifetime of variables

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
a	x						
b	x						
t_1		x					
t_2		x					
x			x	x	x	x	
y			x				
t_3				x			
t_4				x	x		
t_5					x		
t_6						x	
t_7							x
no. of live variable	2	2	2	3	3	2	1

at least 3 registers

Variable usage

Variable x is calculated at state s_2 and used in states s_3, s_4, s_6.

....

Two vars. are alive at s_3

Three vars. are alive at s_4

Three vars. are alive at s_5

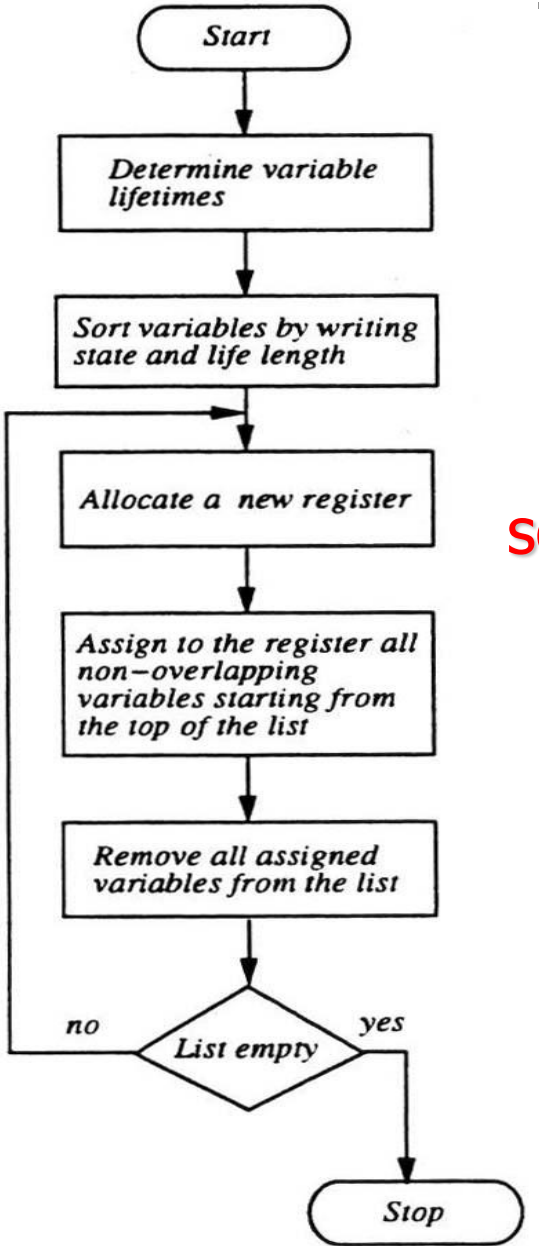
...So, the min # of regs is

(1) determine the number of states

(2) determine the variables' lifetimes

(3) find the minimum number of necessary registers

Register Sharing for SRA (4/5)



sorted

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
a		X						
b		X						
t_1			X					
t_2			X					
x				X	X	X	X	
y				X				
t_4					X	X		
t_3					X			
t_5						X		
t_6							X	
t_7								X

(a) Sorted list of variables

$R_1=[a, t_1, x, t_7]$

$R_2=[b, t_2, y, t_4, t_6]$

$R_3=[t_3, t_5]$

Register assignments

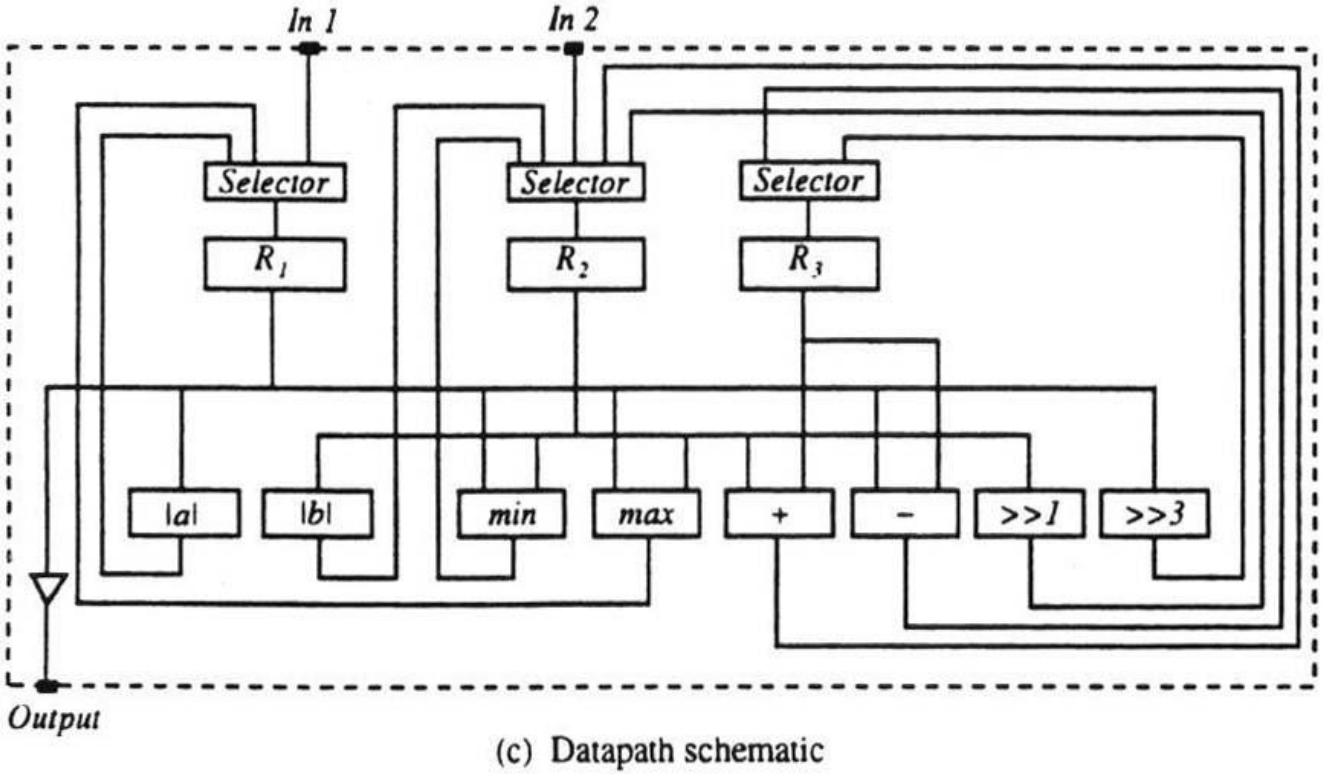
Register Sharing for SRA (5/5)

$R_1=[a, t_1, x, t_7]$

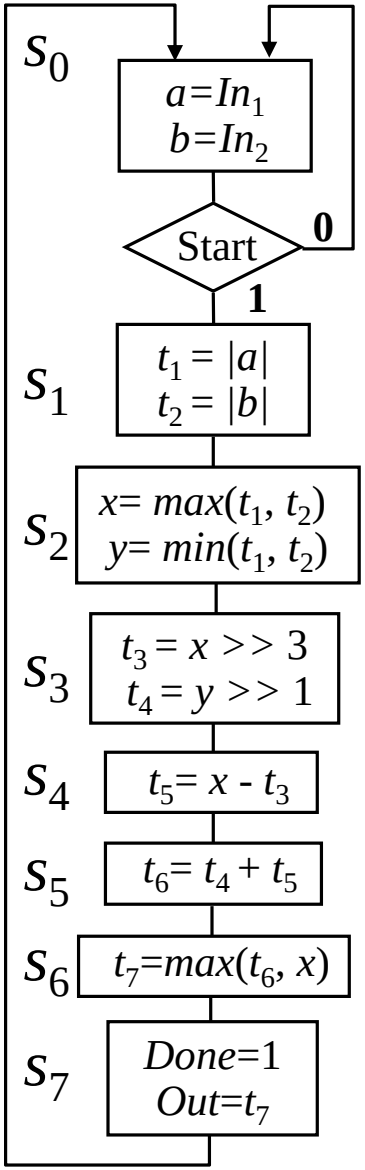
$R_2=[b, t_2, y, t_4, t_6]$

$R_3=[t_3, t_5]$

Register assignments



(c) Datapath schematic



Note: Which wire should be connected ?

Back

Resource Optimization

Datapath optimization:

1. Resource optimization

(a) storage sharing (b) functional unit sharing

(c) bus sharing (d) register merging

2. Time optimization

(a) chaining or multicycling (b) functional-unit pipelining

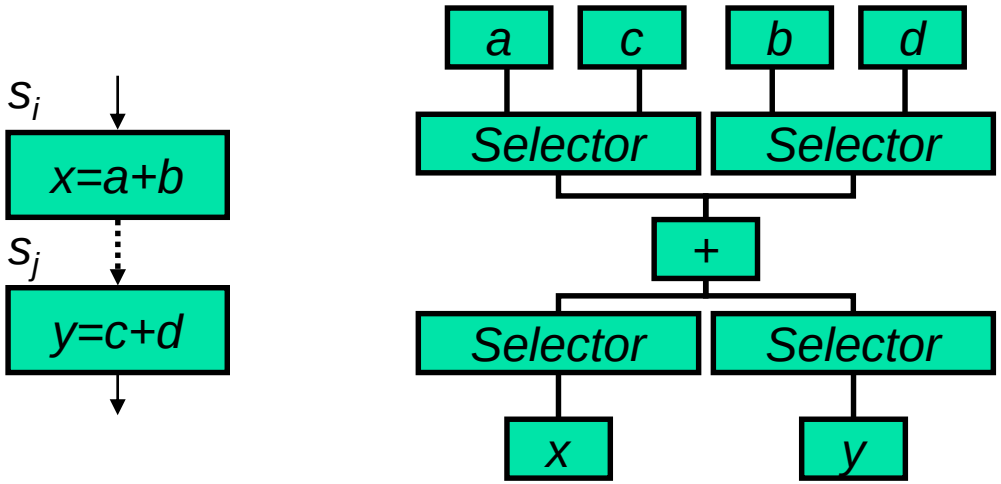
(c) datapath pipelining (d) controlpath pipelining

Functional-Unit Sharing for SRA (1/8)

Functional-unit sharing (operator merging)

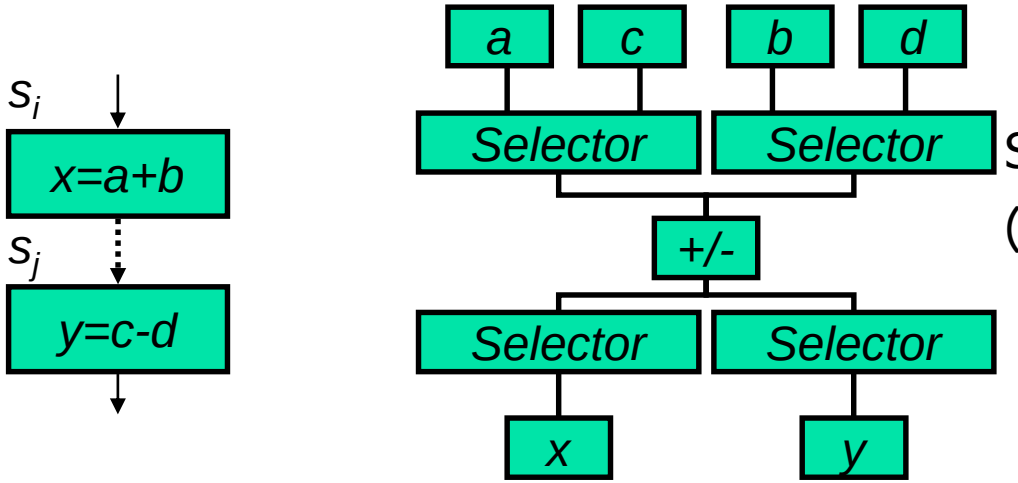
(a) The sharing of the same operation

(b) The sharing of different (but similar) operations (+/-,max/min)



(a)

Sharing of the same operation (the operations are executed at different time period)

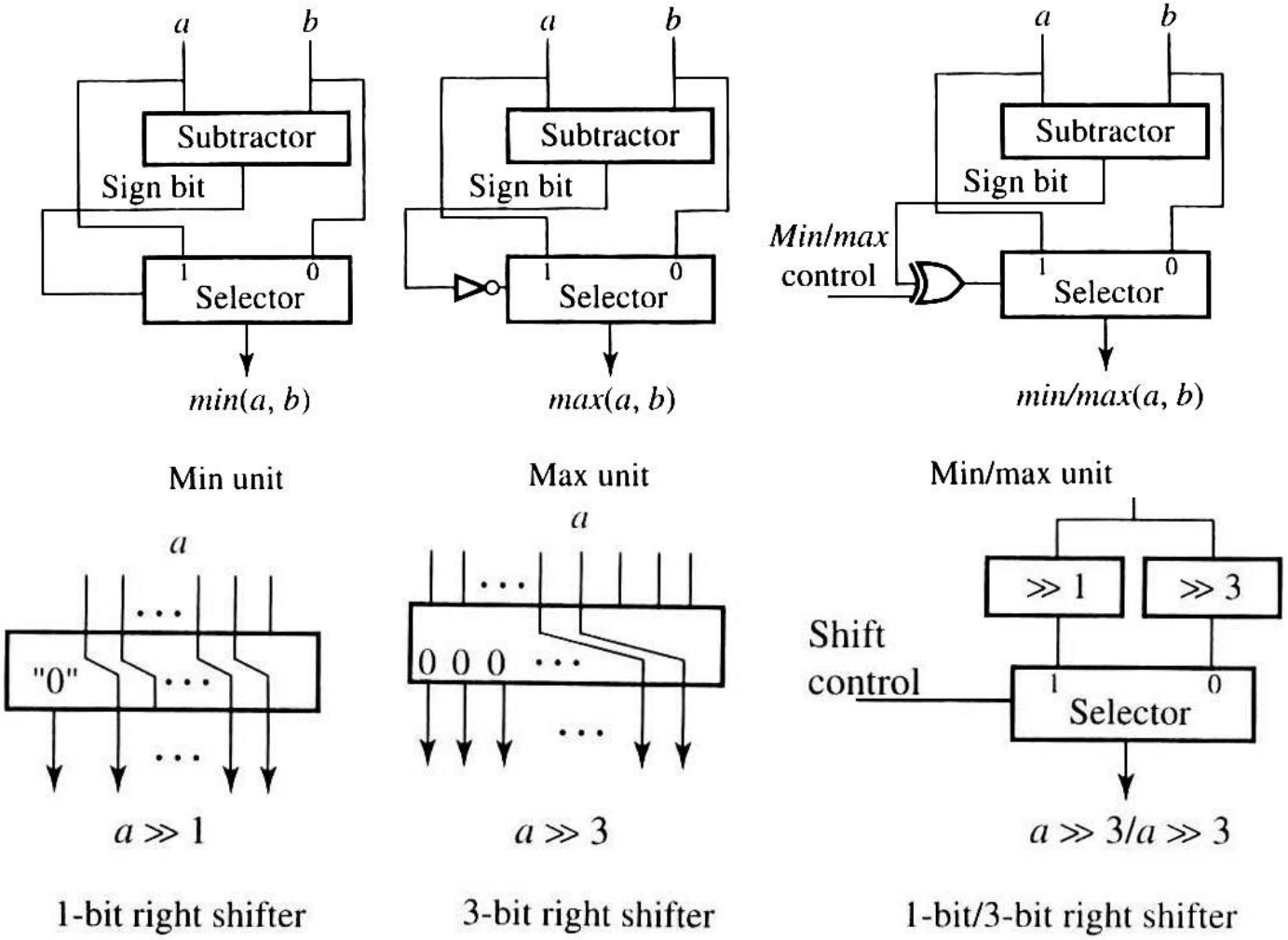


(b)

Sharing of the similar operations (the operations are executed at different time period)

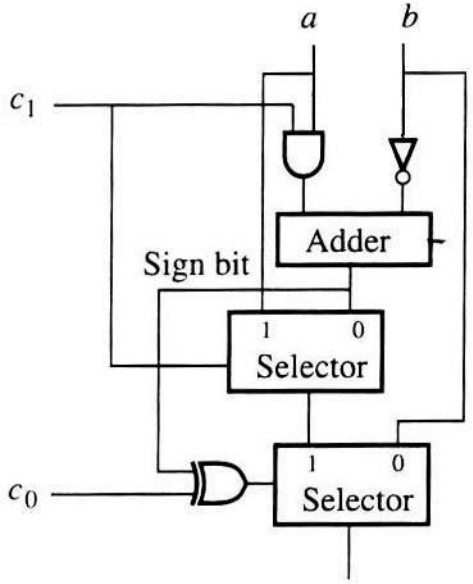
Functional-Unit Sharing for SRA (2/8)

Similar operations are grouped into a single multifunction unit



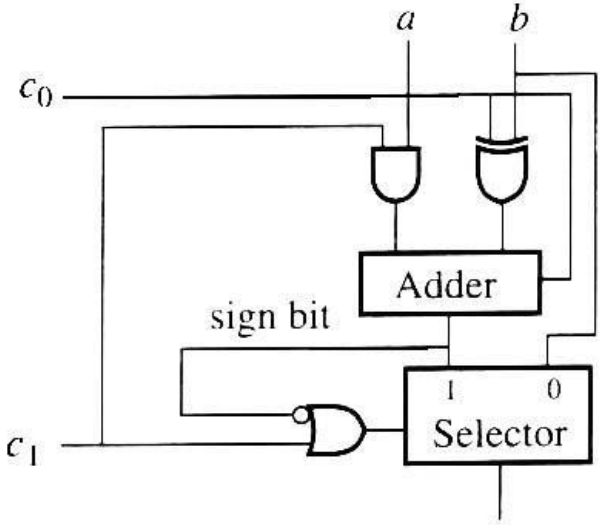
The cost of shift operator is zero

Functional-Unit Sharing for SRA (3/8)



c_1	c_0	OPERATION
0	1	Absolute
1	0	Minimum
1	1	Maximum

Possible multifunction unit



c_1	c_0	OPERATION
1	0	Addition
0	1	Absolute
1	1	Subtraction

Functional-Unit Sharing for SRA (4/8)

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	Max. no. of units
abs	2							2
min		1						1
max		1				1		1
>>			2					2
-				1				1
+					1			1
NO. of operations	2	2	2	1	1	1		

Operation Usage

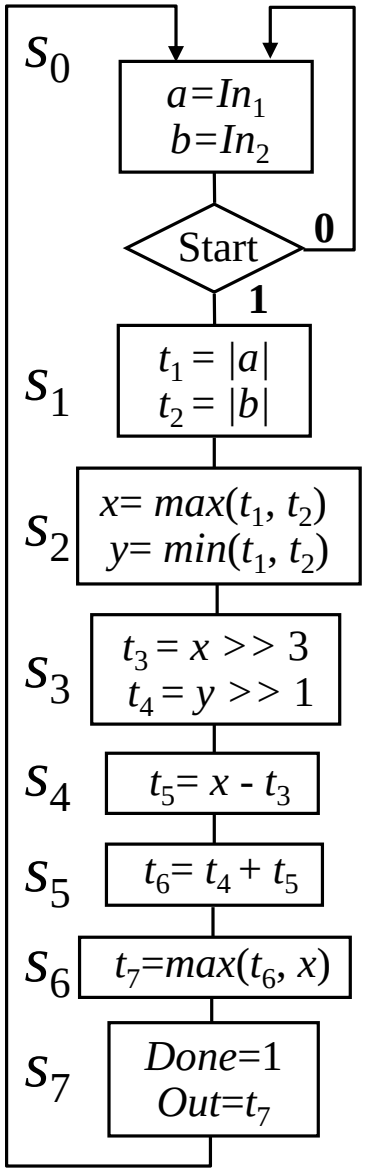
At least two operations are necessary

Functional-unit Sharing:

(1) determine the number of states (scheduling)

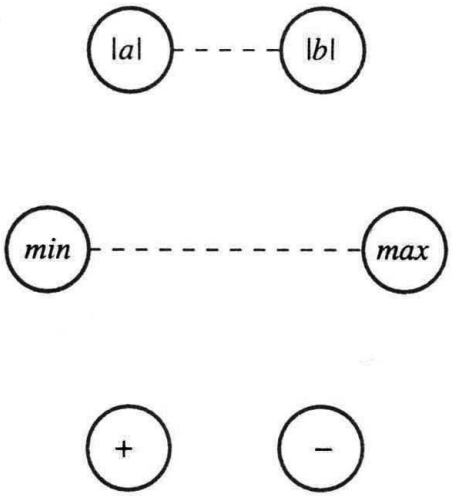
(2) determine the operations required at each state

(3) minimize the number of functional units (construct a compatibility graph, merge operations and minimize the cost)



Functional-Unit Sharing for SRA (5/8)

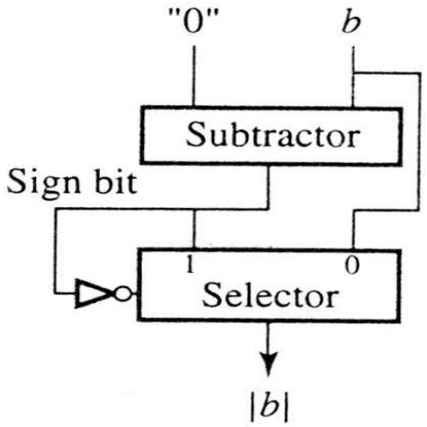
The shift operator is ignored here because its cost is zero



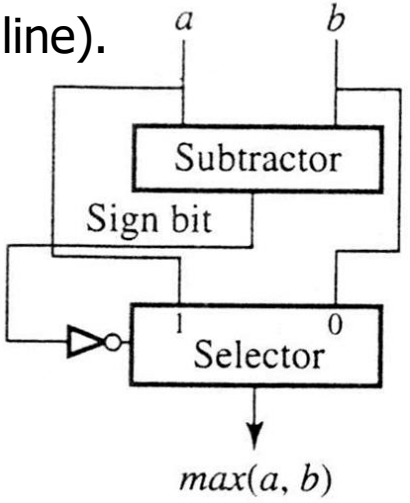
UNIT \ COMPONENT	AND LOGIC	INVERT LOGIC	EX-OR LOGIC	ADDER	SELECTOR
|a|		1		1	1
|b|		1		1	1
min		1		1	1
max		1		1	1
+				1	
−		1		1	
Total		5		6	4

Note: |a| and |b| (min and max) cannot be merged since they are executed concurrently at the same state (denoted by a dish line).

Absolute unit

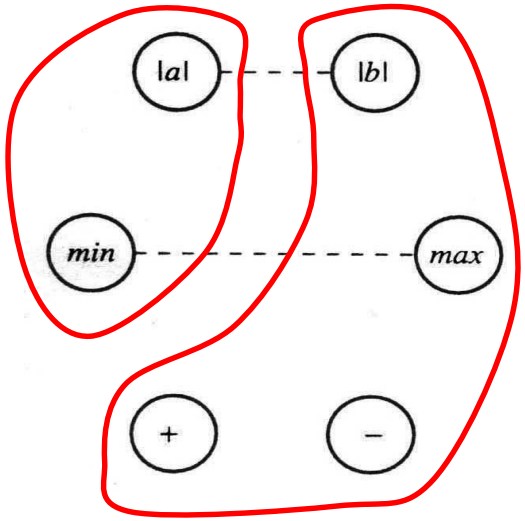


Max unit



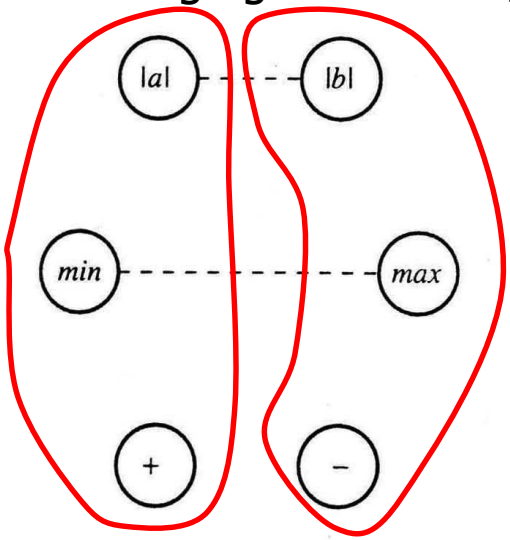
Functional-Unit Sharing for SRA (6/8)

Merging Alternative_1 (Remember, at least two operations are necessary)



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

Merging Alternative_2



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

Functional-Unit Sharing for SRA (7/8)

Merging Alternative_1

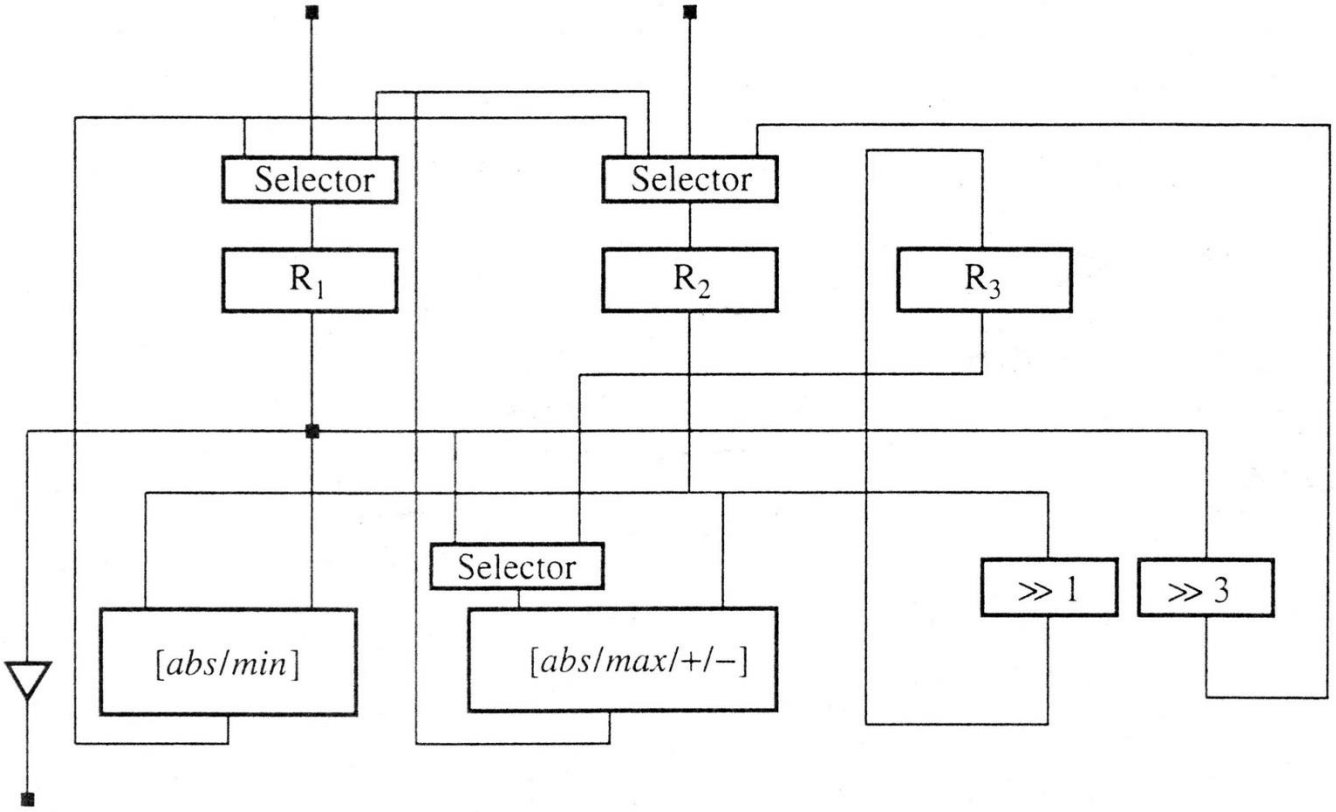
After sharing, 3 registers and 2 operations

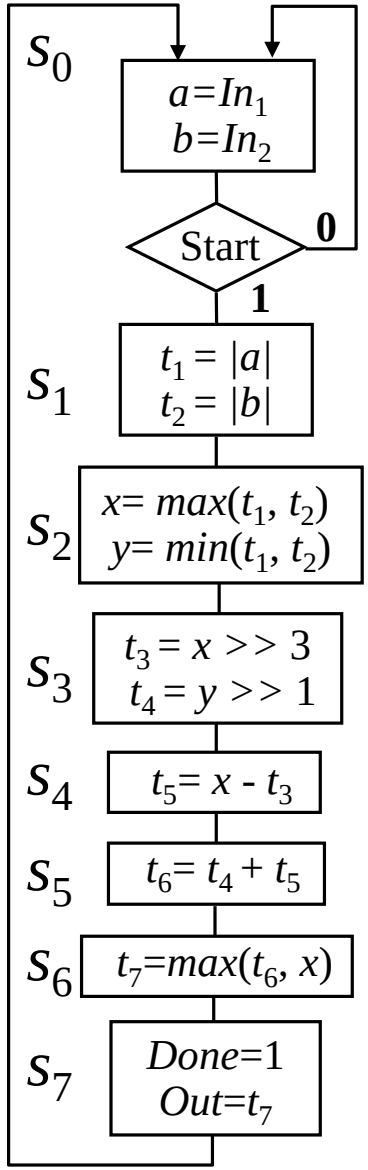
$R_1=[a, t_1, x, t_7]$ $R_2=[b, t_2, y, t_3, t_5, t_6]$ $R_3=[t_4]$

register assignments

[|*a*| / min]

[|*b*| / max / + / -]





Compare with the circuit without functional-unit sharing

Functional-Unit Sharing for SRA (8/8)

Merging Alternative_2

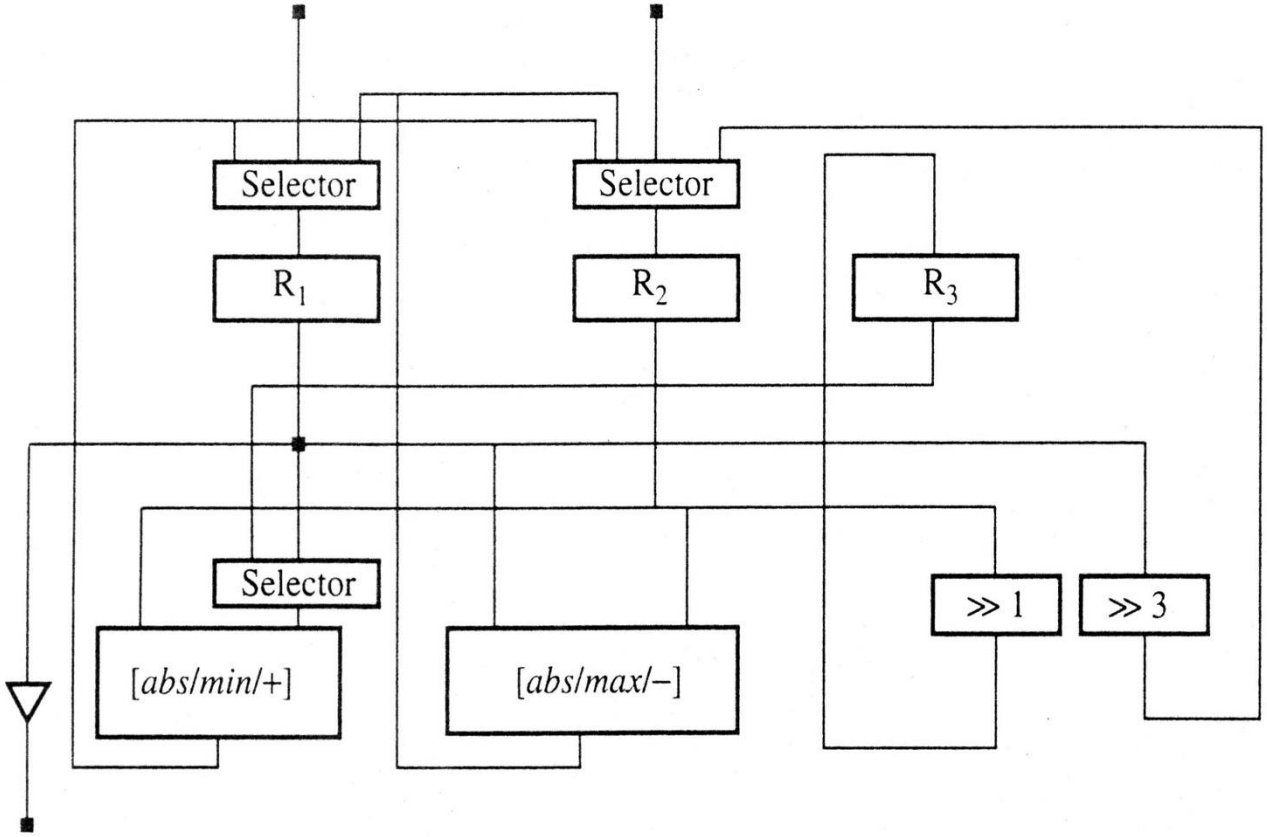
After sharing, 3 registers and 2 operations

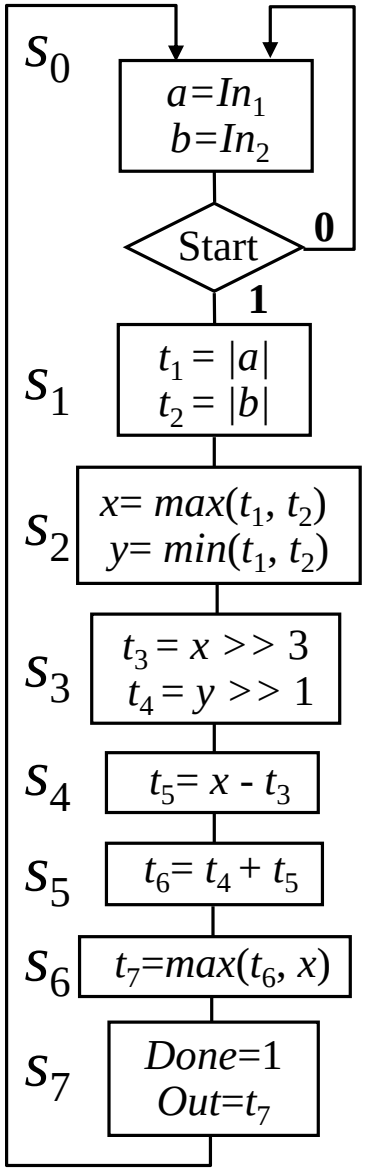
$R_1=[a, t_1, x, t_7]$ $R_2=[b, t_2, y, t_3, t_5, t_6]$ $R_3=[t_4]$

register assignments

[|*a*| / min / +]

[|*b*| / max / -]





Resource Optimization

Datapath optimization:

1. Resource optimization

(a) storage sharing (b) functional unit sharing

(c) bus sharing (d) register merging

2. Time optimization

(a) chaining or multicycling (b) functional-unit pipelining

(c) datapath pipelining (d) controlpath pipelining

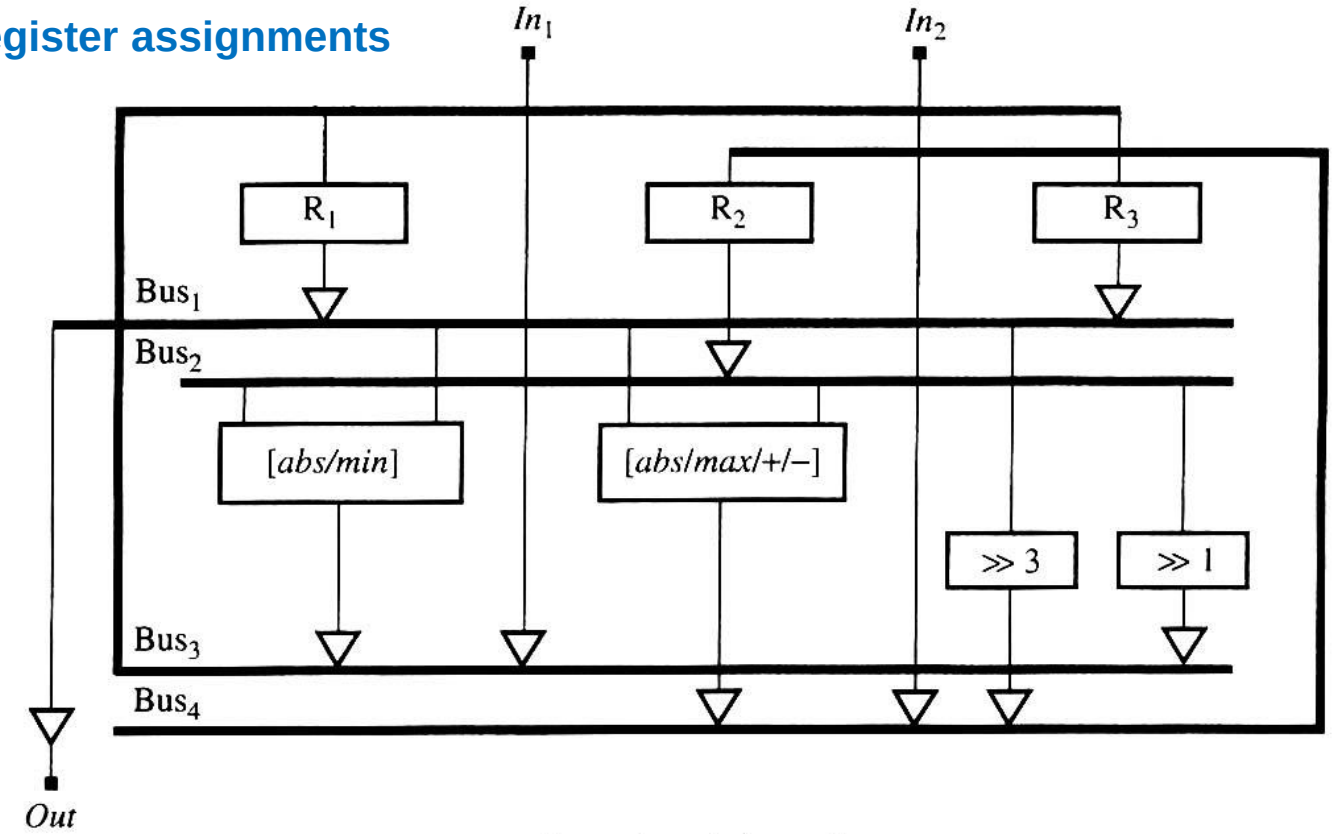
Bus Sharing for SRA (1/2)

Bus sharing (connection merging)

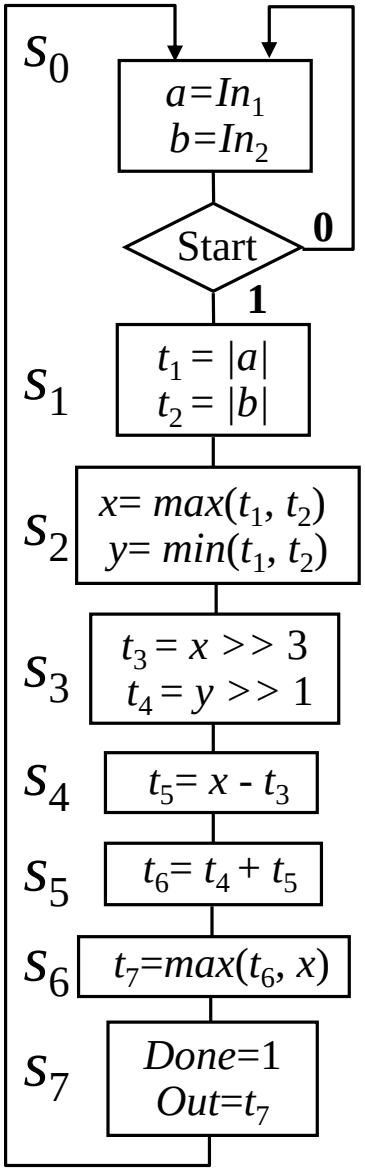
Wires (connecting each register output to the input of a functional unit and connecting each functional-unit output to the input of a register) can be merged into bus.

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

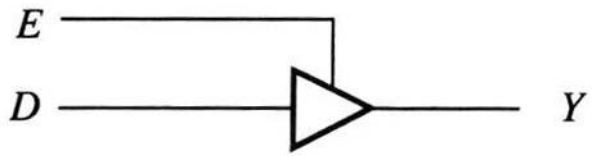
register assignments



Bus-oriented datapath



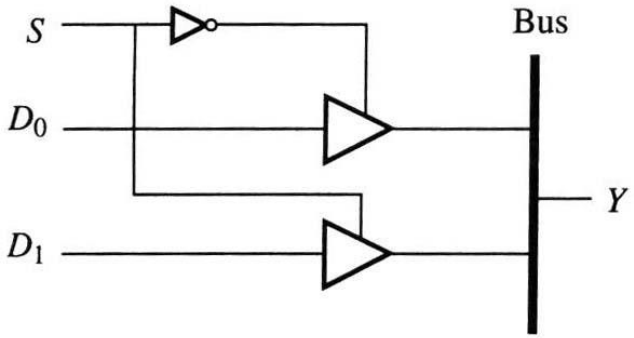
Bus Sharing for SRA (2/2)



(a) Tristate driver symbol

E	Y
0	Z
1	D

(b) Truth table for tristate driver



(c) 2-input bus

S	Y
0	D_0
1	D_1

(d) Truth table for 2-input bus

Resource Optimization

Datapath optimization:

1. **Resource optimization**

   (a) storage sharing (b) functional unit sharing

   (c) bus sharing (d) register merging

2. **Time optimization**

   (a) chaining or multicycling (b) functional-unit pipelining

   (c) datapath pipelining (d) controlpath pipelining

Register Merging for SRA (1/2)

Variables with non-overlapping lifetime can share the same register (Register sharing).

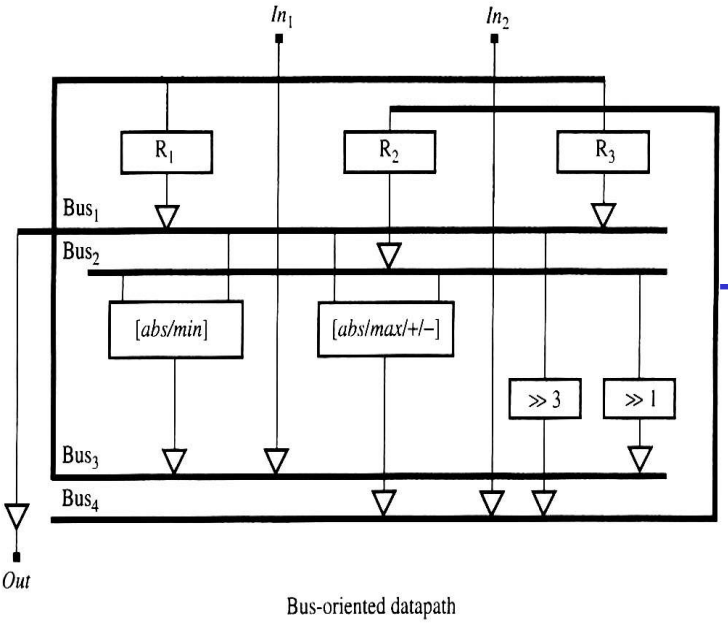
Registers with non-overlapping access time can be merged into register files to share register input and output port (Register merging).

$R_1=[a, t_1, x, t_7]$

$R_2=[b, t_2, y, t_3, t_5, t_6]$

$R_3=[t_4]$

Register assignments

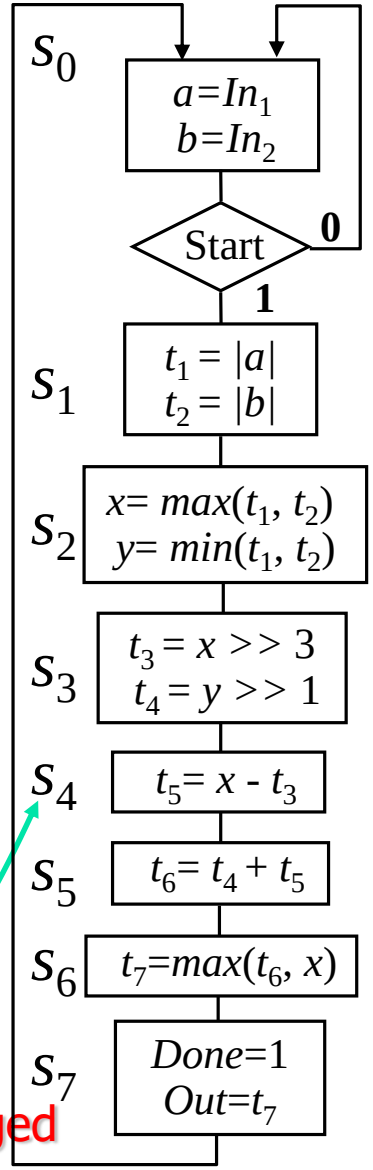


Bus-oriented datapath

	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
R_1	w	rw	rw	r	r		rw	r
R_2	w	rw	rw	rw	rw	rw	r	
R_3				w		r		

2 read
2 write

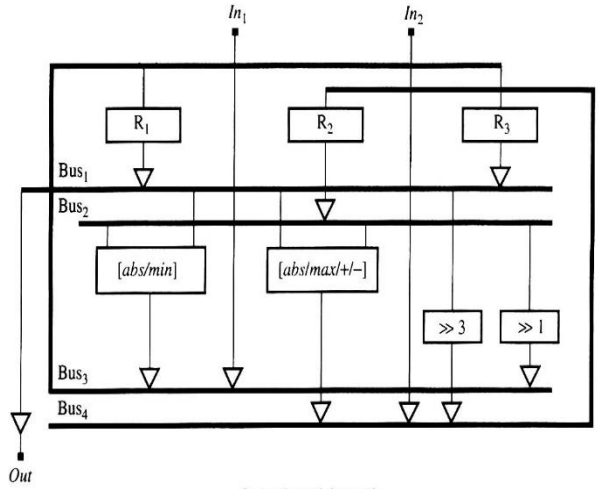
2 read
1 write



Since R1 and R3 are never used at the same state, they can be merged

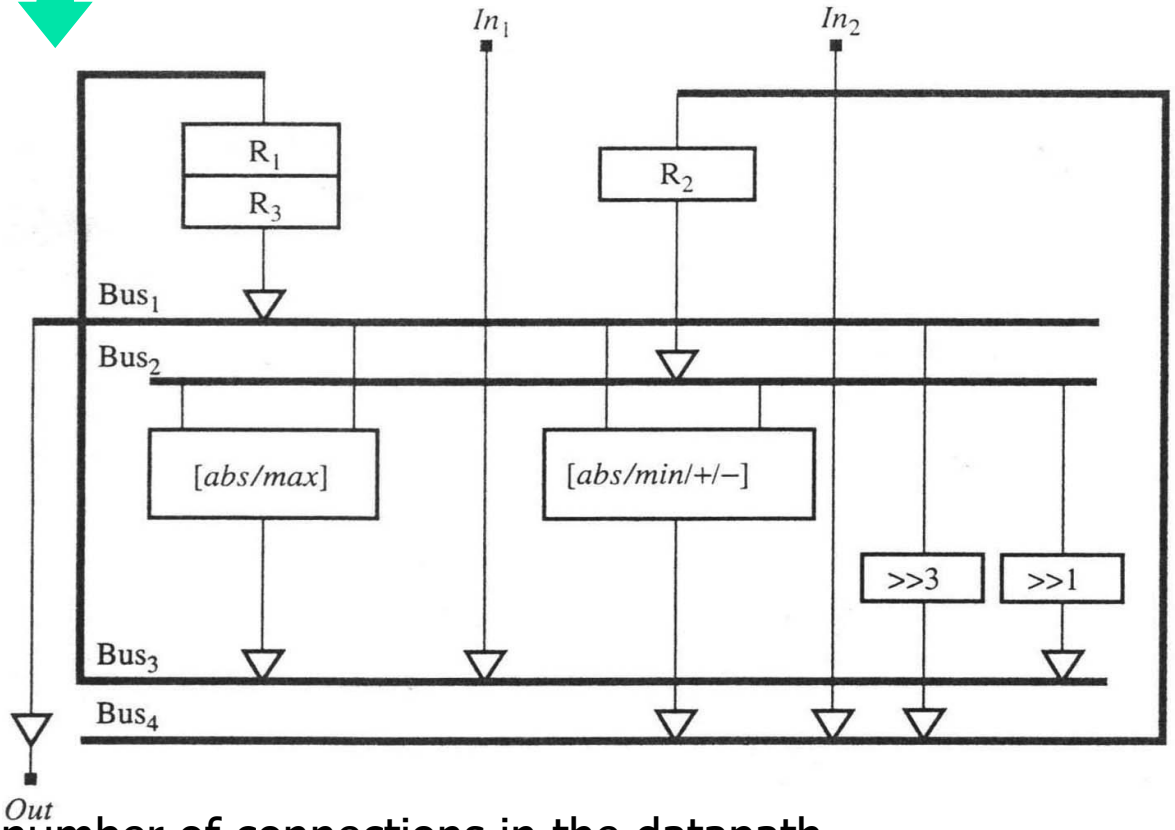
Register Merging for SRA (2/2)

Since R1 and R3 are never used at the same state, they can be merged



Bus-oriented datapath

Register merging



Advantage: reduce the number of connections in the datapath

Disadvantage: cause the extra delay for address decoding in the register file

Time Optimization

Datapath optimization:

1. Resource optimization

(a) storage sharing

(b) functional-unit sharing

(c) bus sharing

(d) register merging

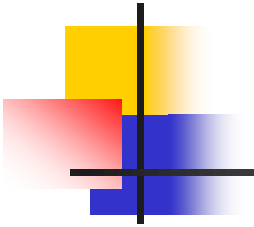
2. Time optimization (timing performance optimization)

(a) chaining or multicycling

(b) functional-unit pipelining

(c) datapath pipelining

(d) controlpath pipelining



Chaining and Multicycling

- Chaining allows serial execution of two or more operations in each state
- Chaining reduces number of states and increases performance

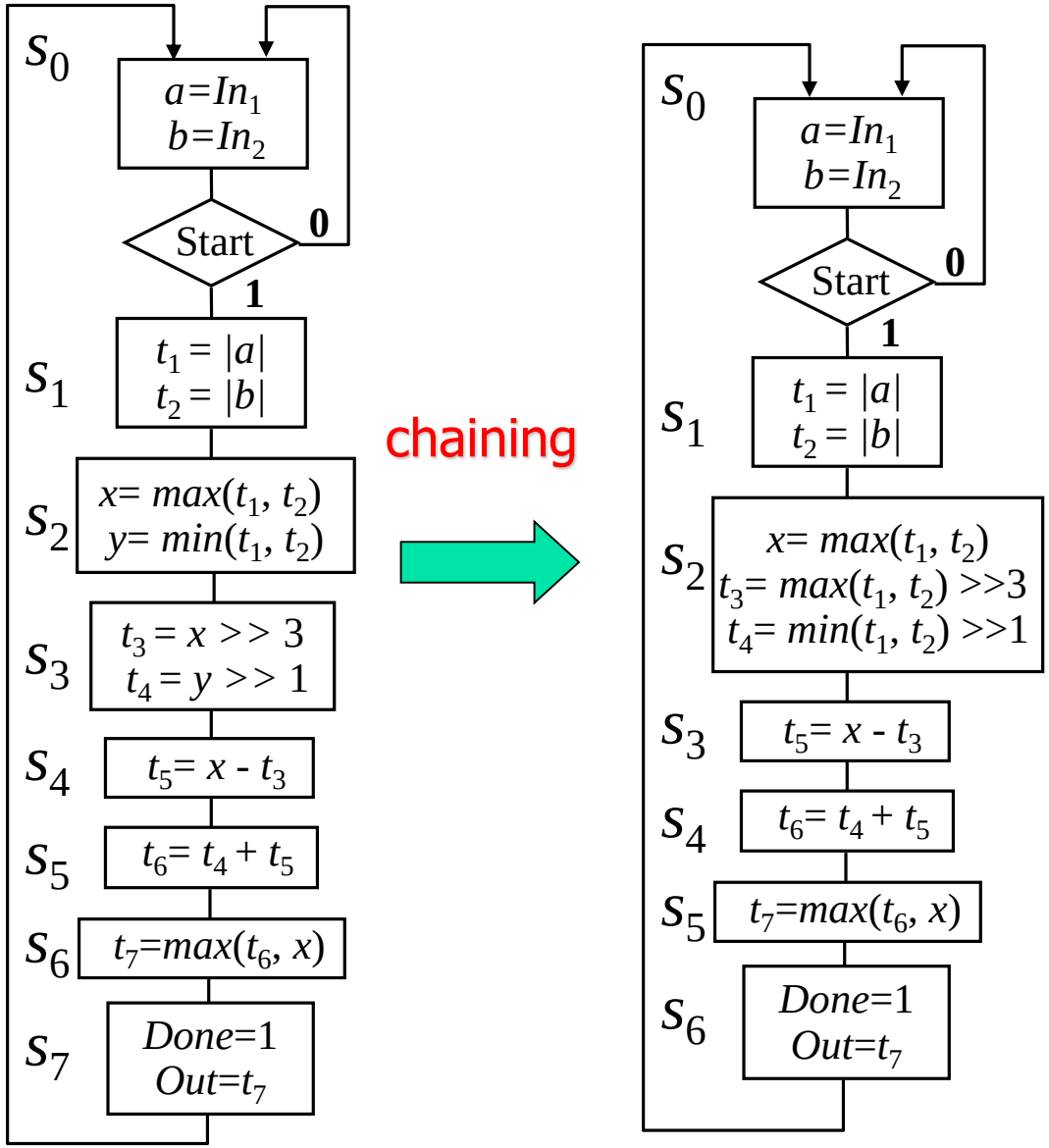
- Multicycling allows one operation to be executed over two or more clock cycles.
- Multicycling reduces size of functional units.

- Chaining and multicycling are used on noncritical paths to improve resource utilization and performance.

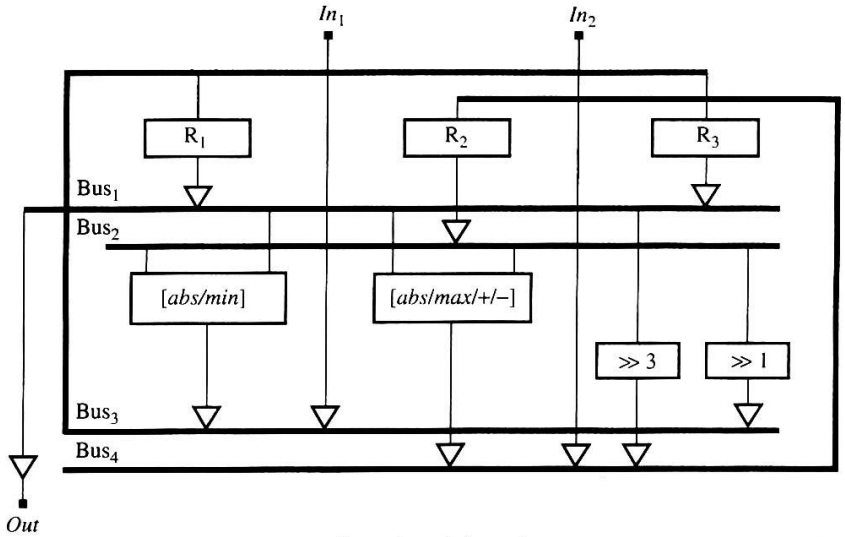
SRA Datapath with Chaining (1/2)

Since shifting to the right by three or one positions incurs almost no delay, the clock cycle for this chained datapath would be no longer than the original one.

The chained datapath (7 states) performs SRA algorithm 12.5% faster than the original one (8 states)

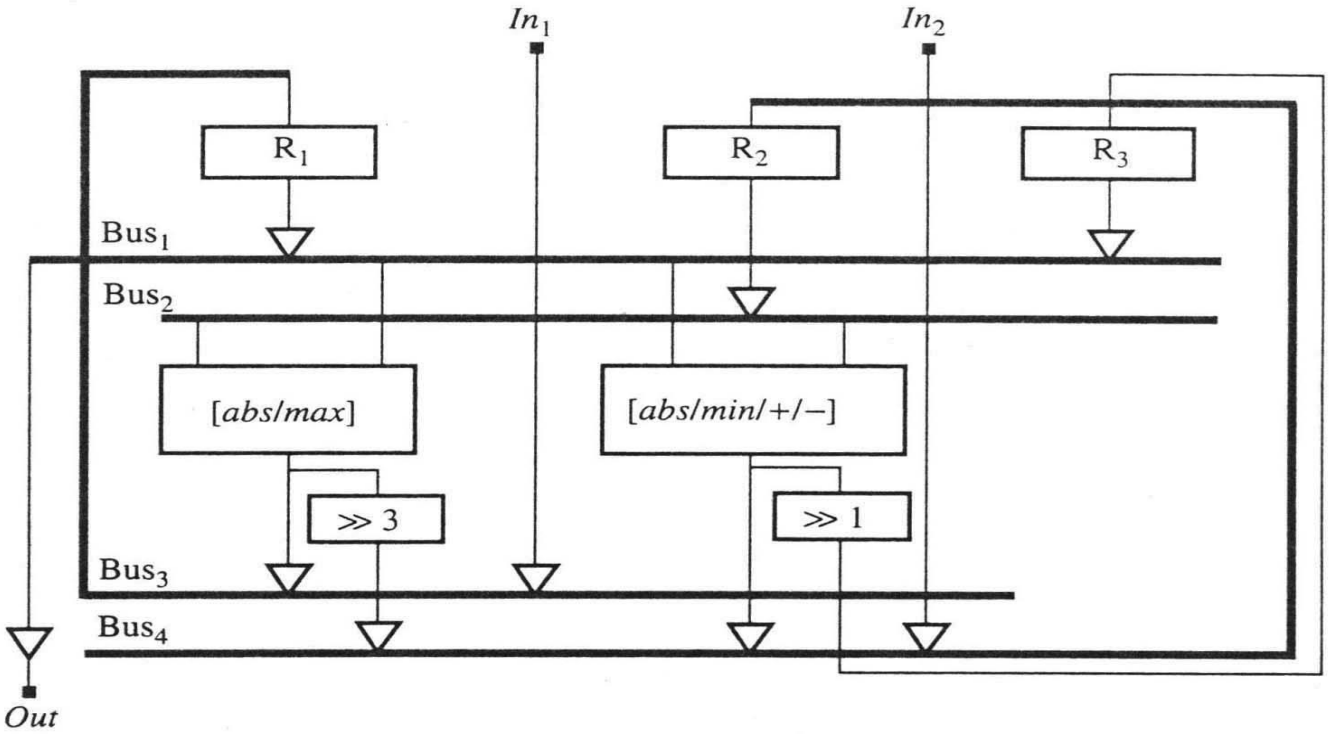


SRA Datapath with Chaining (2/2)



Bus-oriented datapatl

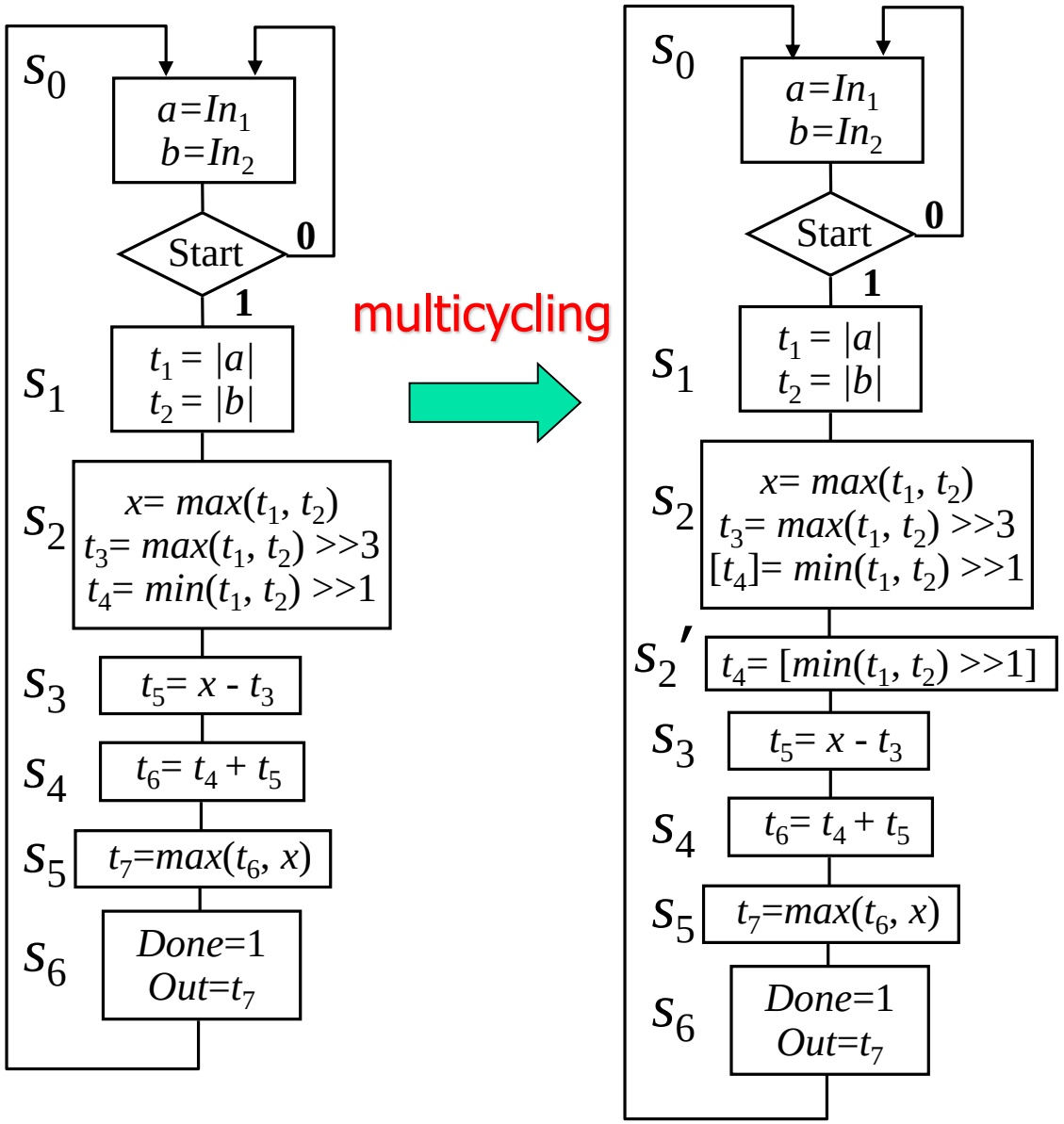
chaining



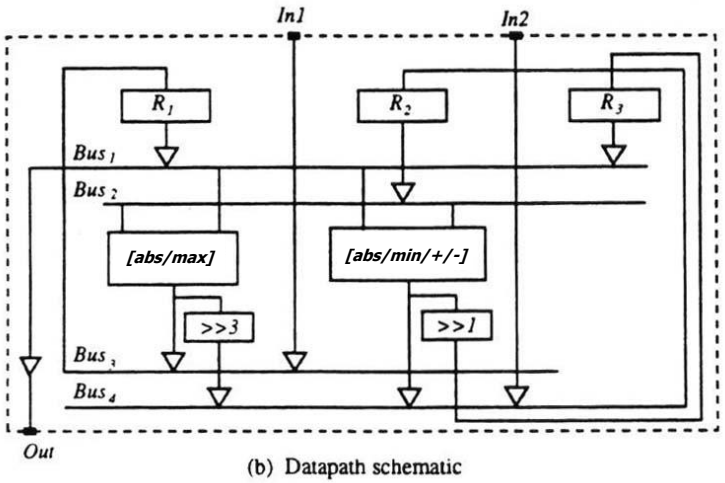
SRA Datapath with Multicycling (1/2)

Variable t_4 will be assigned a new value in state s_2, but this new value will not be used until state s_4. We could use a unit (cheaper and slower than the original one) that takes two clock cycles to compute the minimum value (for s_4).

No extra delay

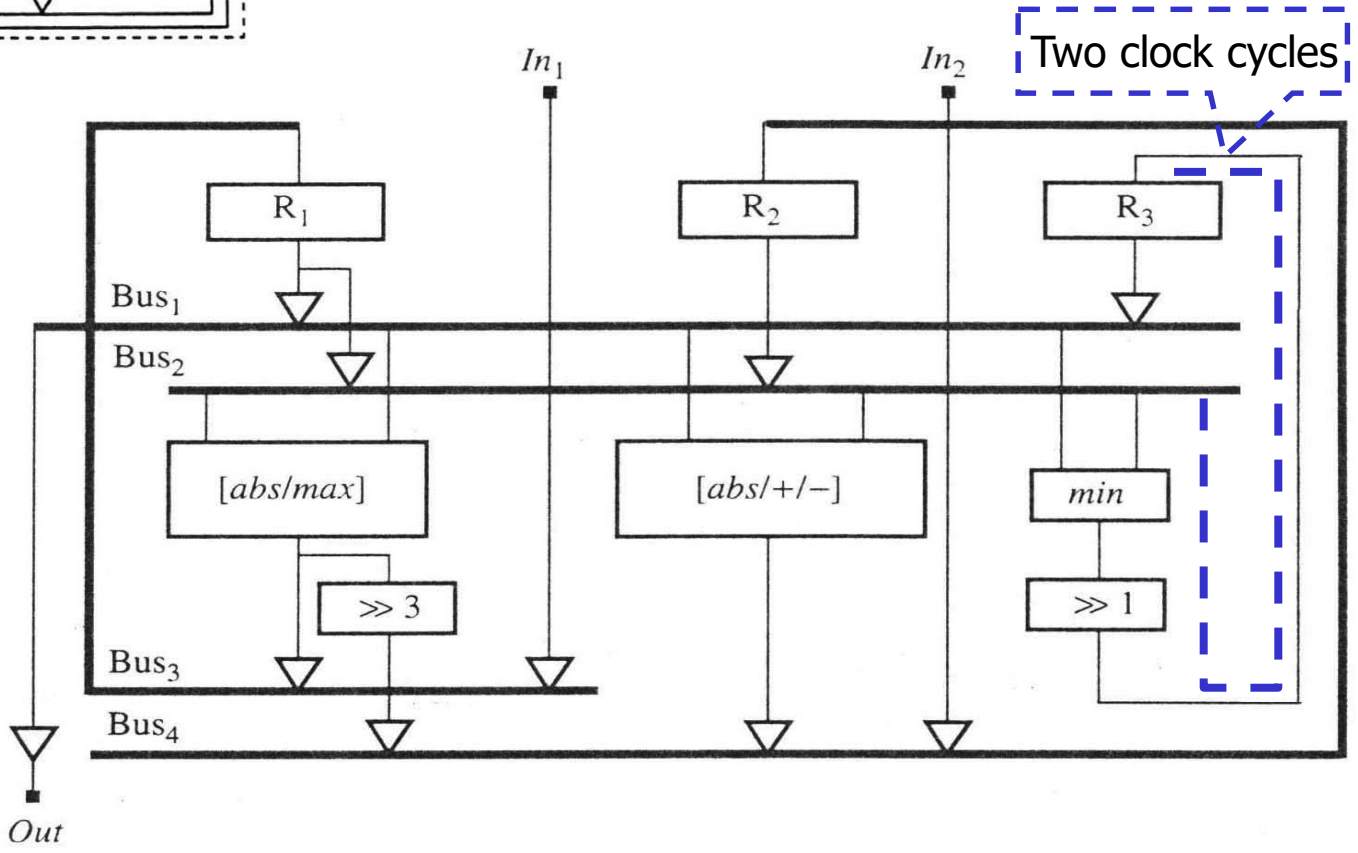


SRA Datapath with Multicycling (2/2)



(b) Datapath schematic

multicycling



Time Optimization

Datapath optimization:

1. Resource optimization

(a) storage sharing

(b) functional-unit sharing

(c) bus sharing

(d) register merging

2. Time optimization (timing performance optimization)

(a) chaining or multicycling

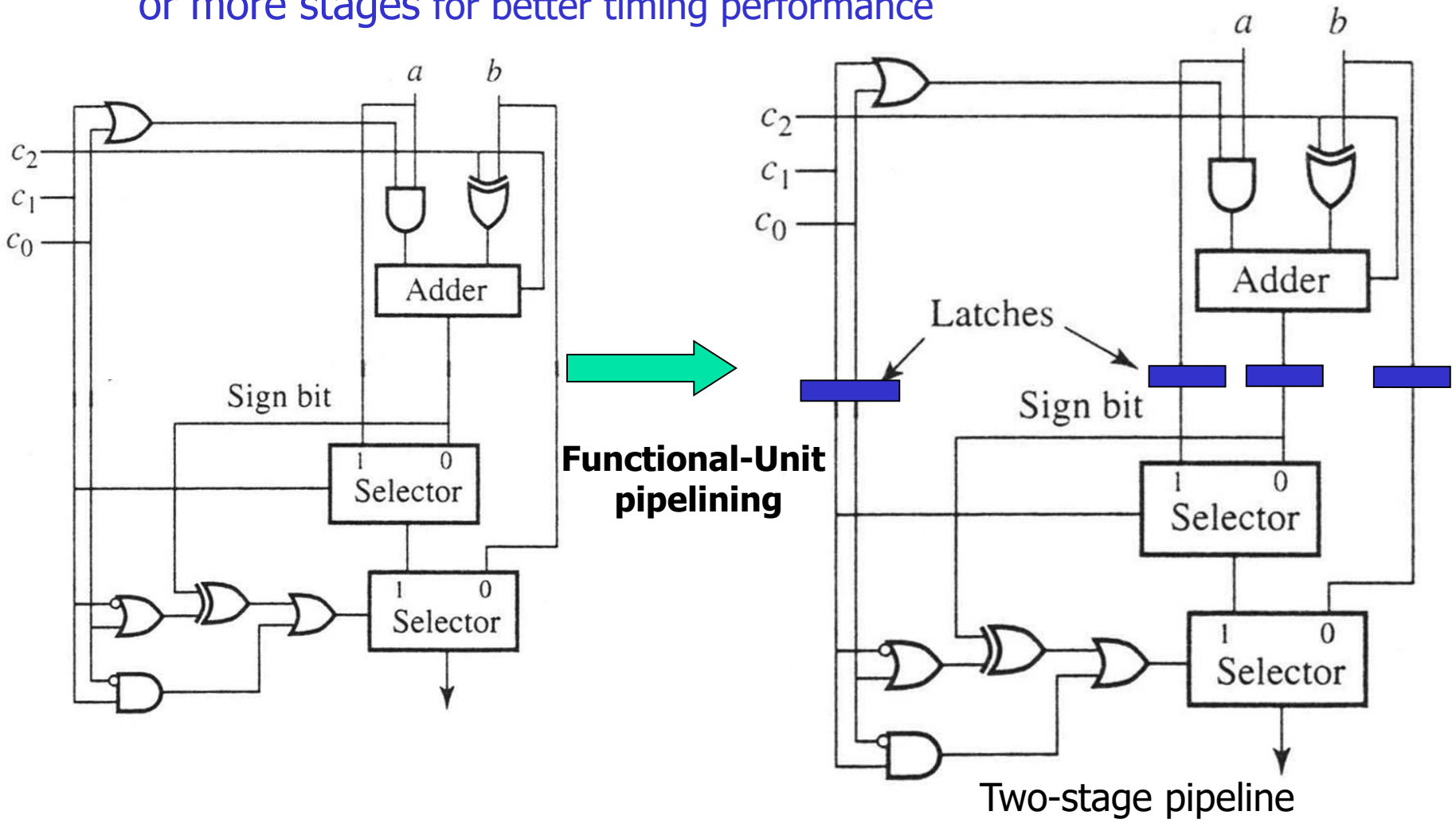
(b) functional-unit pipelining

(c) datapath pipelining

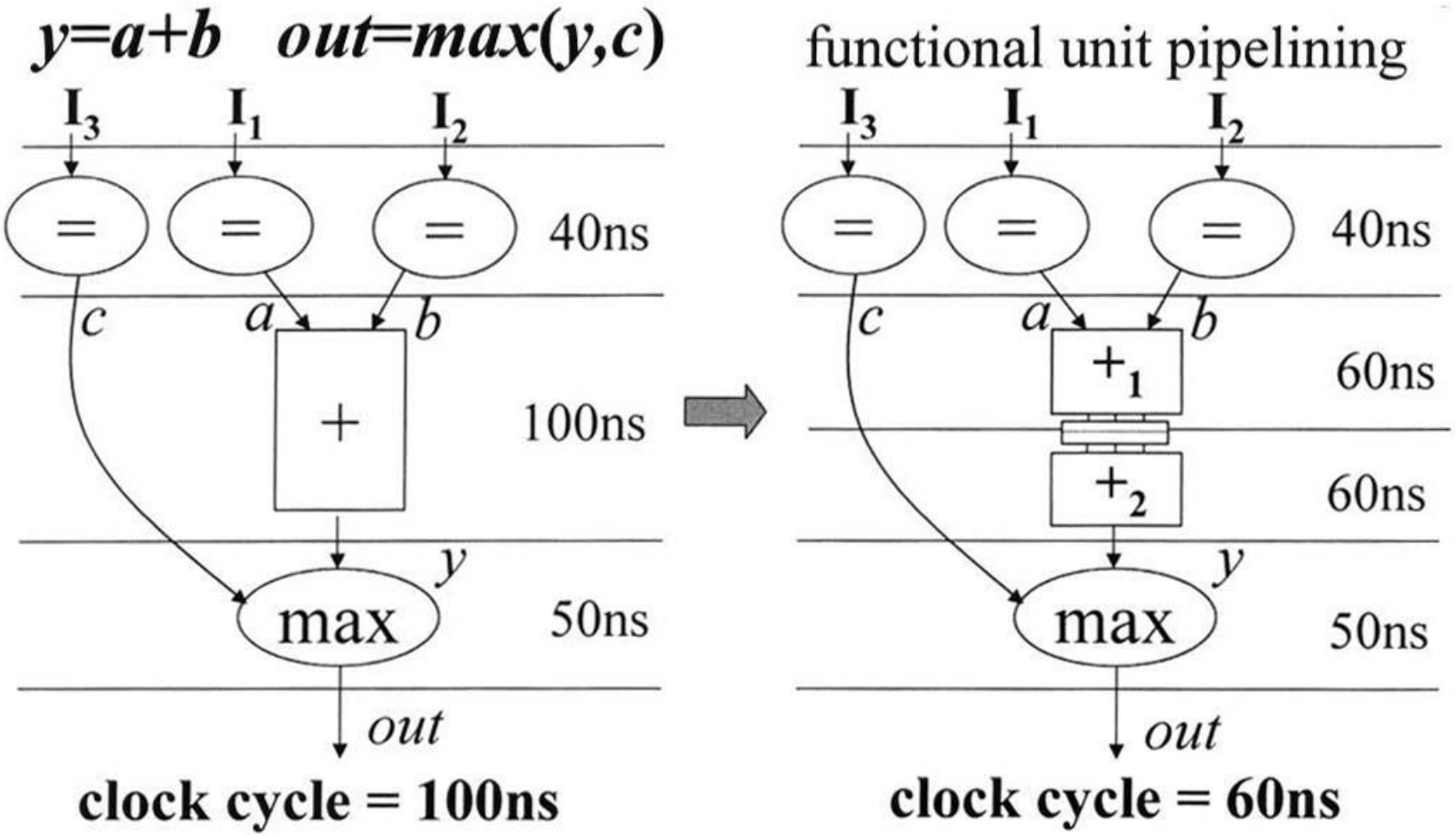
(d) controlpath pipelining

Functional-Unit pipelining (1/4)

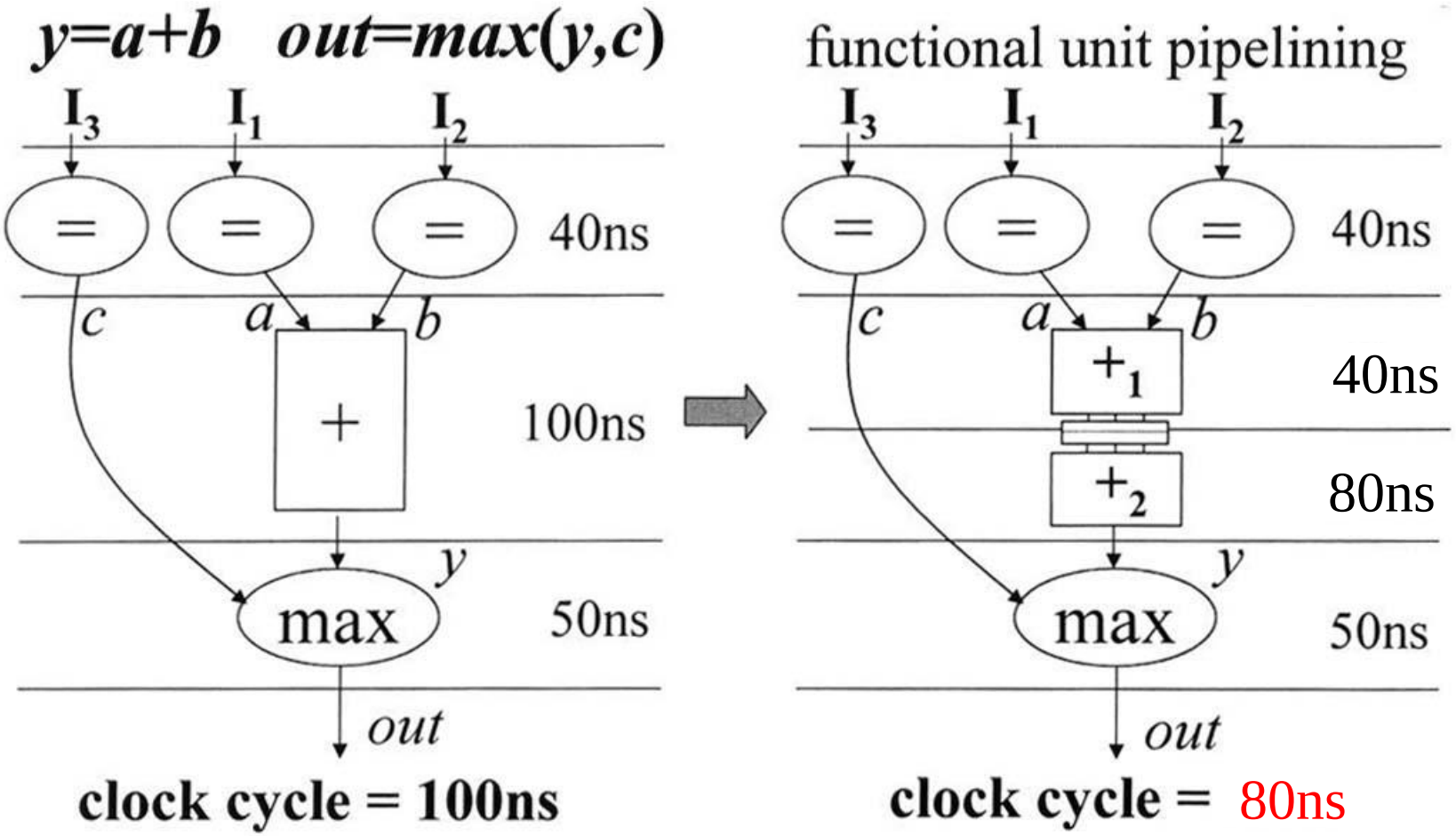
Insert a series of registers into the functional unit to divide it into two or more stages for better timing performance



Functional-Unit pipelining (2/4)

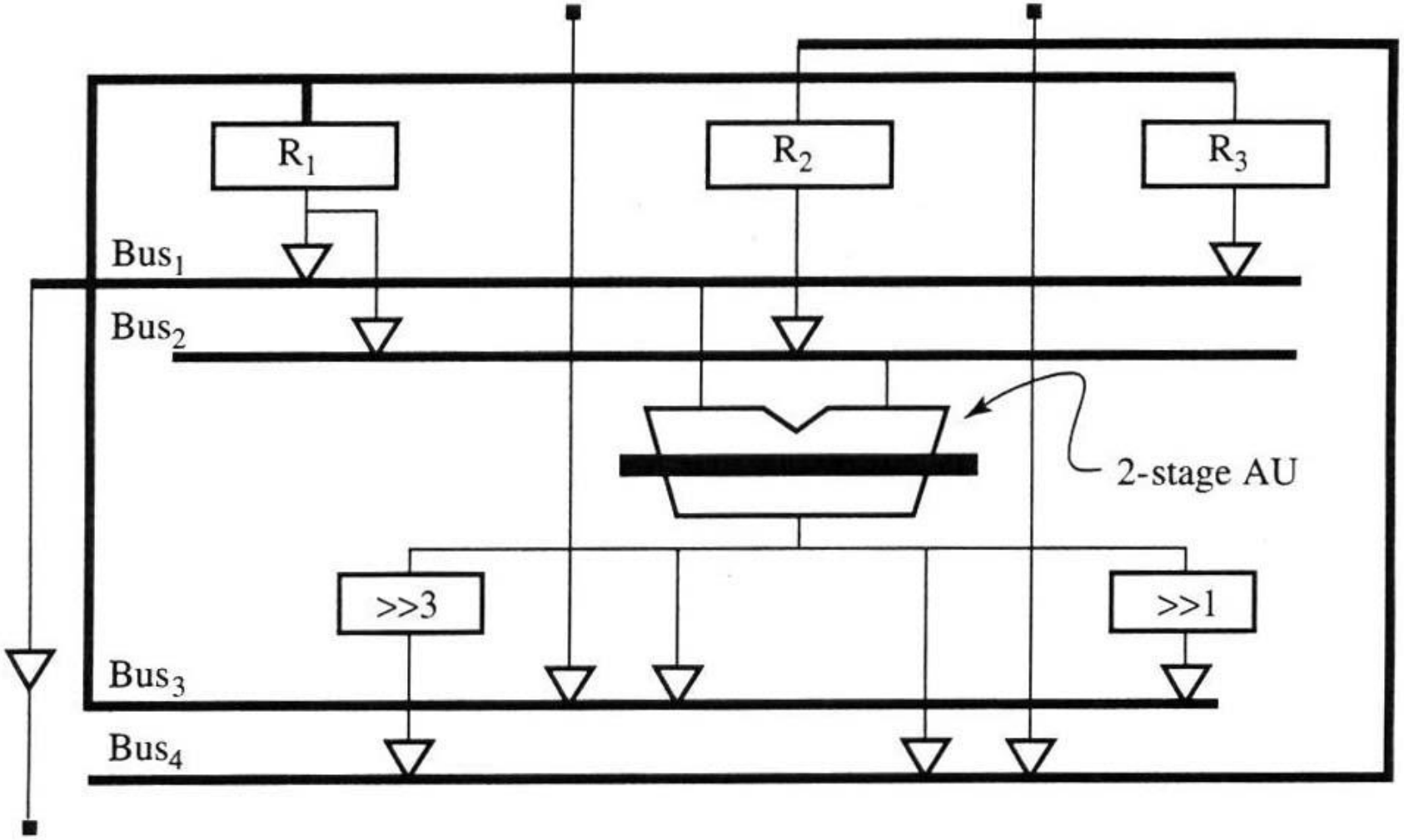


Functional-Unit pipelining (3/4)



★ Divide the adder as equal as possible

Functional-Unit pipelining (4/4)



Time Optimization

Datapath optimization:

1. Resource optimization

(a) storage sharing

(b) functional-unit sharing

(c) bus sharing

(d) register merging

2. Time optimization (timing performance optimization)

(a) chaining or multicycling

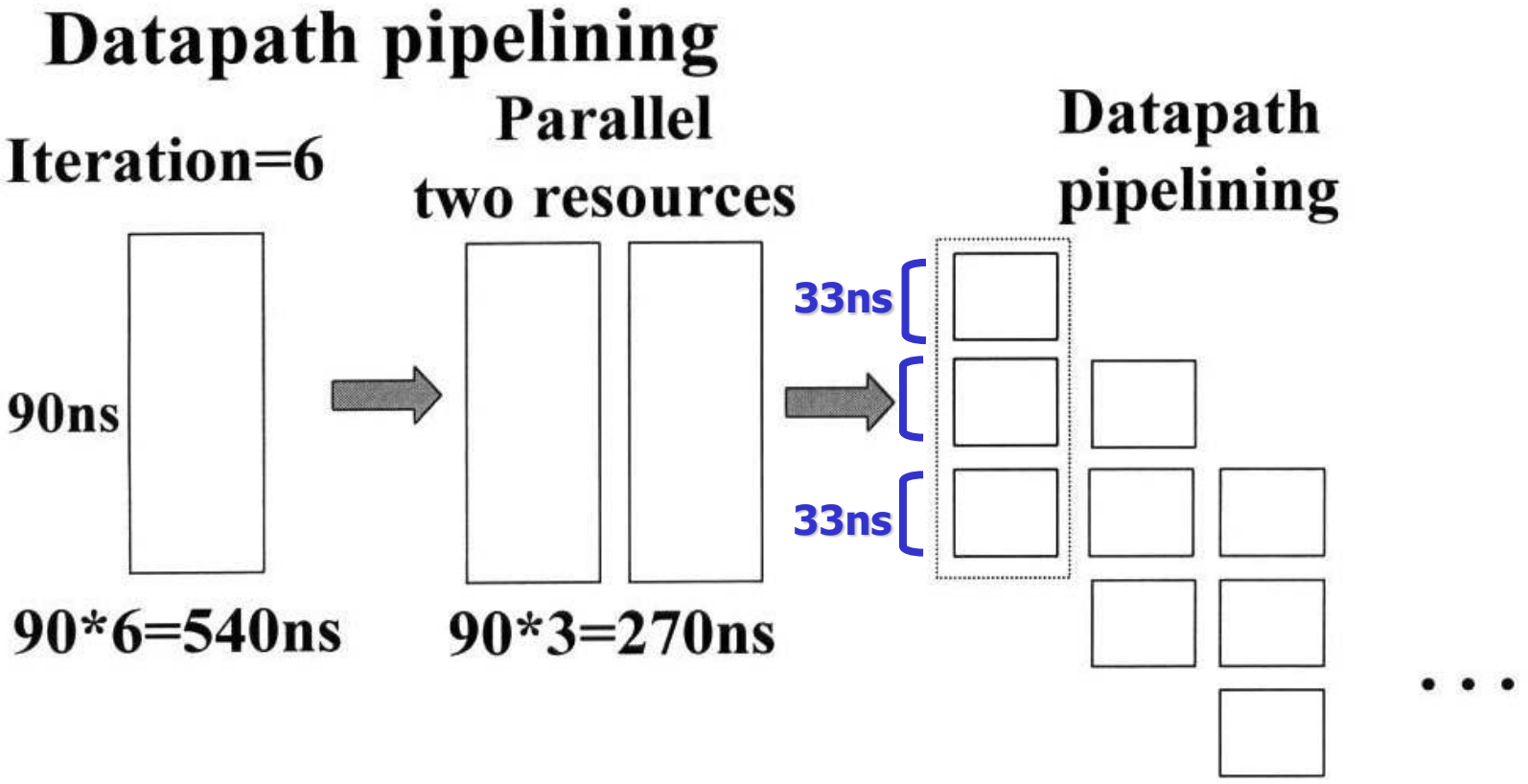
(b) functional-unit pipelining

(c) datapath pipelining

(d) controlpath pipelining

Datapath Pipelining (1/5)

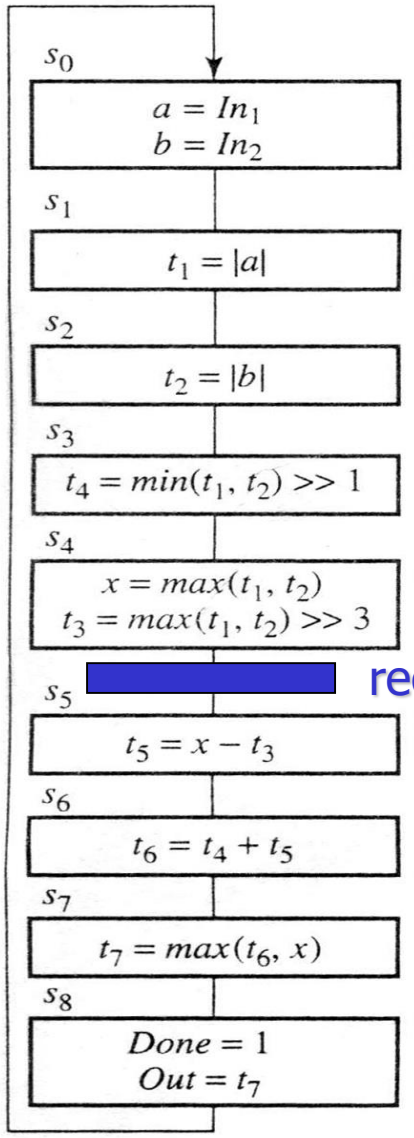
Insert a series of registers into the datapath to divide it into two or more stages for better timing performance

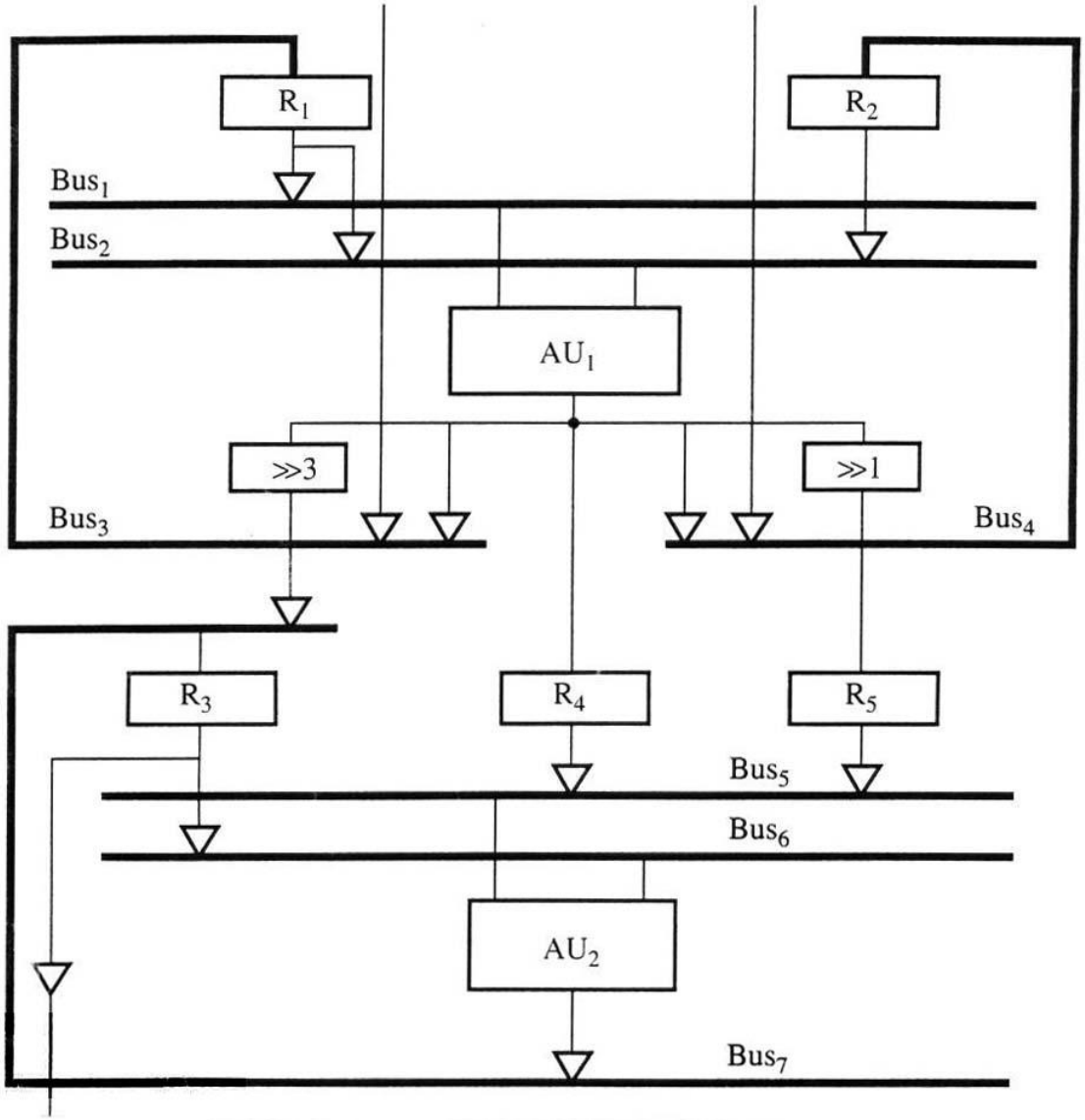


The delay for 6 events is 33*3+33*5=264ns

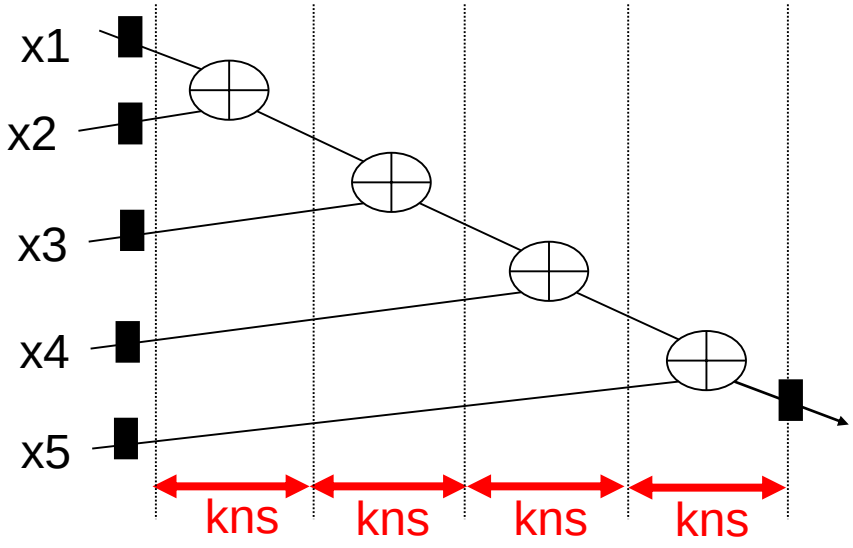
Divide the datapath as equal as possible

Datapath Pipelining (2/5)





Datapath Pipelining (3/5)

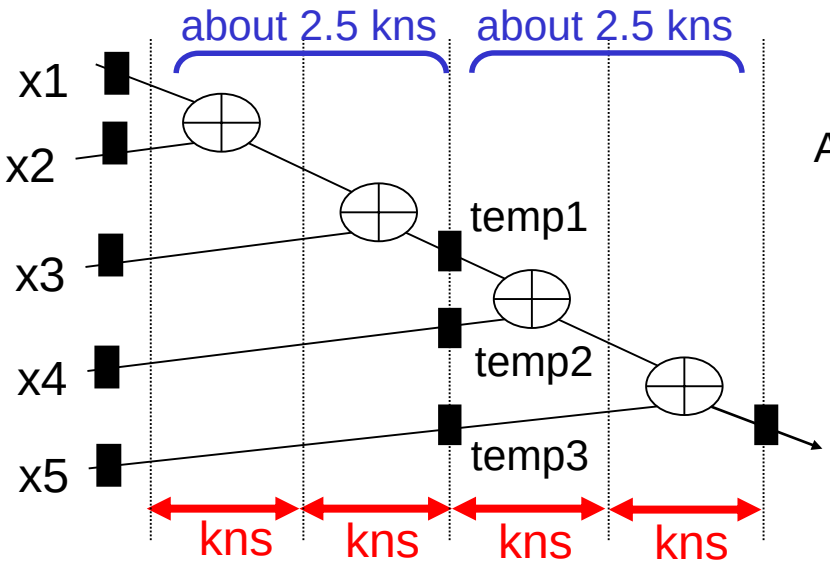


1. Wire delay
2. Register assignment delay

Critical path is about 4kns

A correct output is generated every clock cycle

Event	1	2	3	4	5	6
Completed time	4k	8k	12k	16k	20k	24k



faster clock rate

Critical path is about 2.5 kns, why?

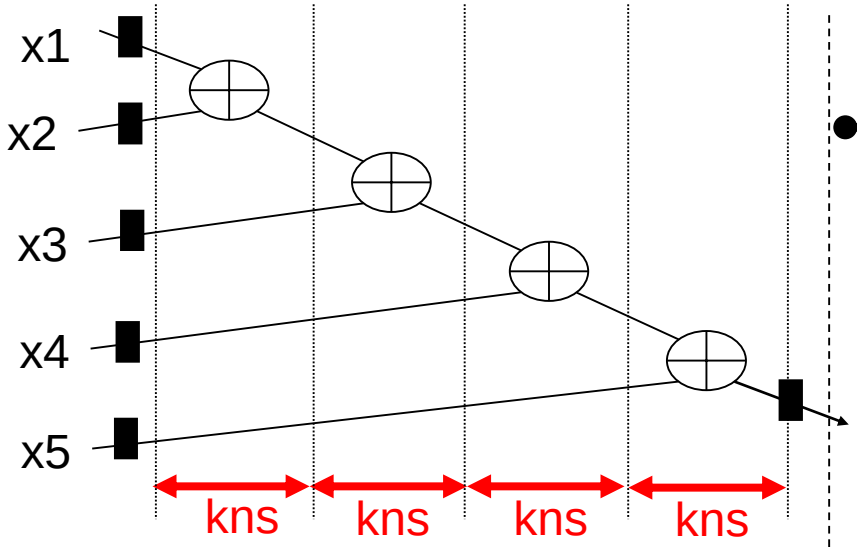
A correct output is generated after two clock cycles

Event	1	2	3	4	5	6
Completed time	5k	7.5k	10k	12.5k	15k	17.5k

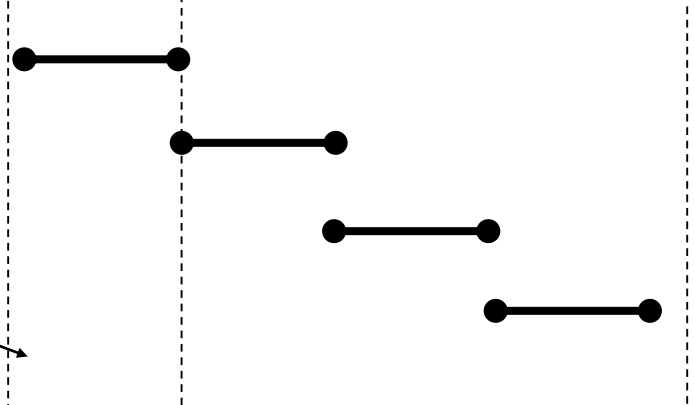
Two events are parallel processed in the unit.

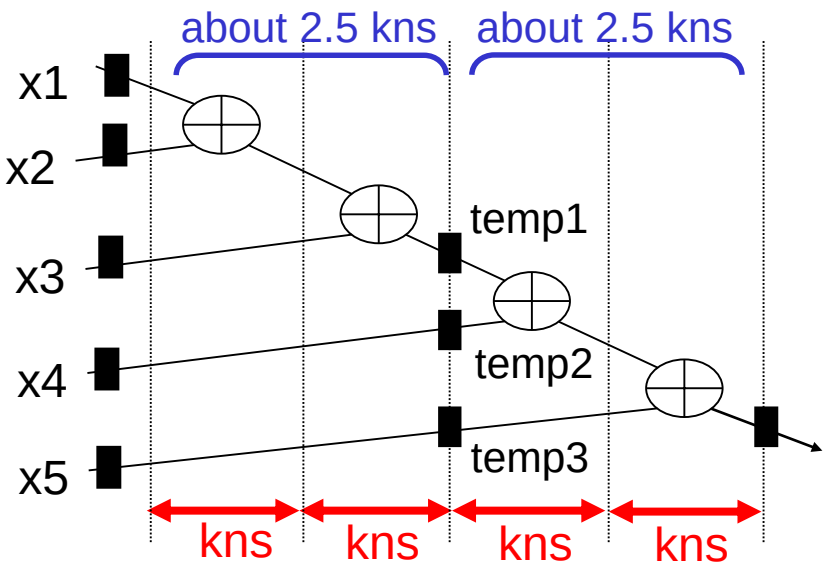
Faster clock rate but higher cost (3 extra regs)

Datapath Pipelining (4/5)

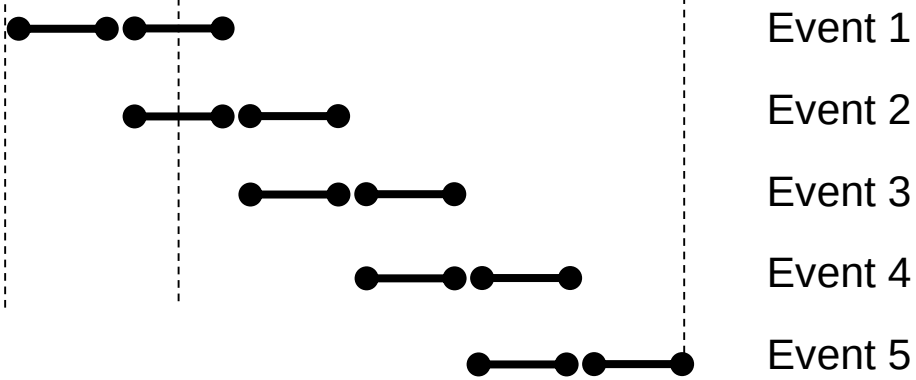


Critical path is about 4kns



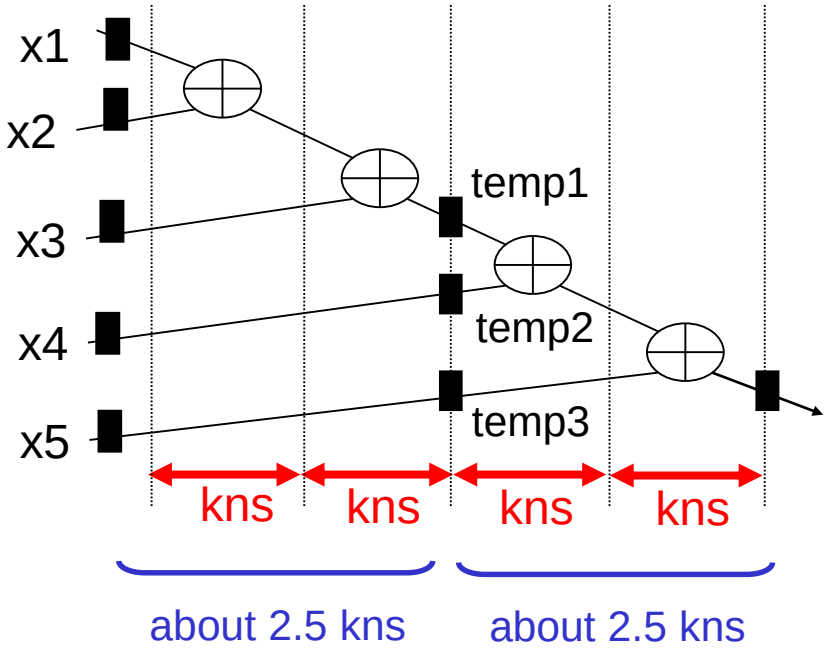


Critical path is about 2.5 kns

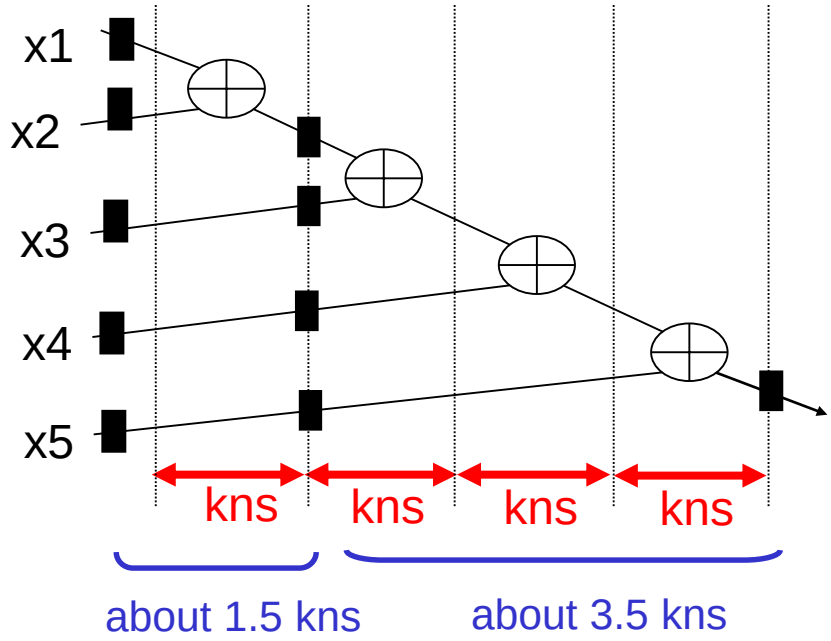


Datapath Pipelining

Datapath Pipelining (5/5)



Critical path is about 2.5 kns



Critical path is about 3.5 kns

Which one is better ? Balance is important

Time Optimization

Datapath optimization:

1. Resource optimization

(a) storage sharing

(b) functional-unit sharing

(c) bus sharing

(d) register merging

2. Time optimization (timing performance optimization)

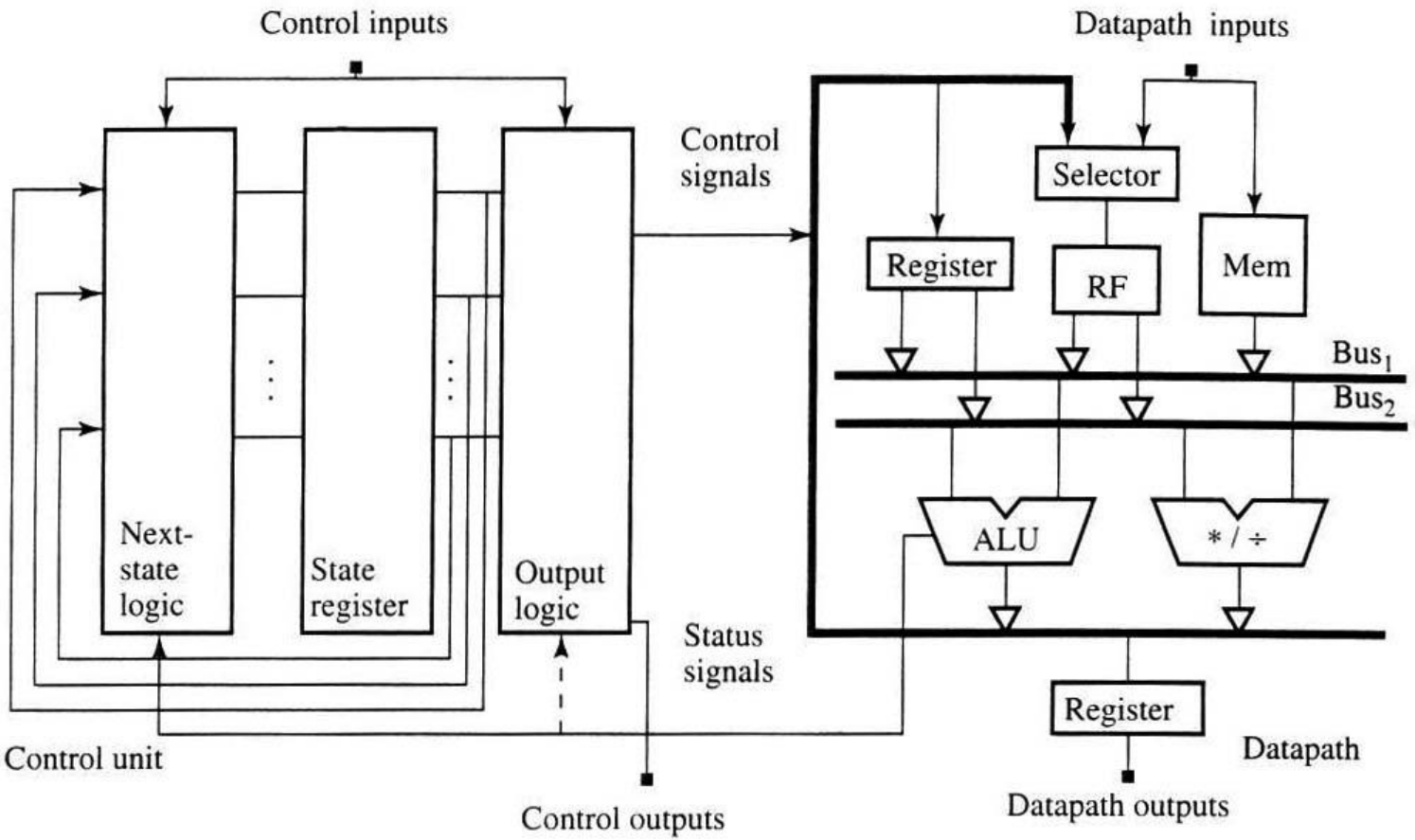
(a) chaining or multicycling

(b) functional-unit pipelining

(c) datapath pipelining

(d) controlpath pipelining

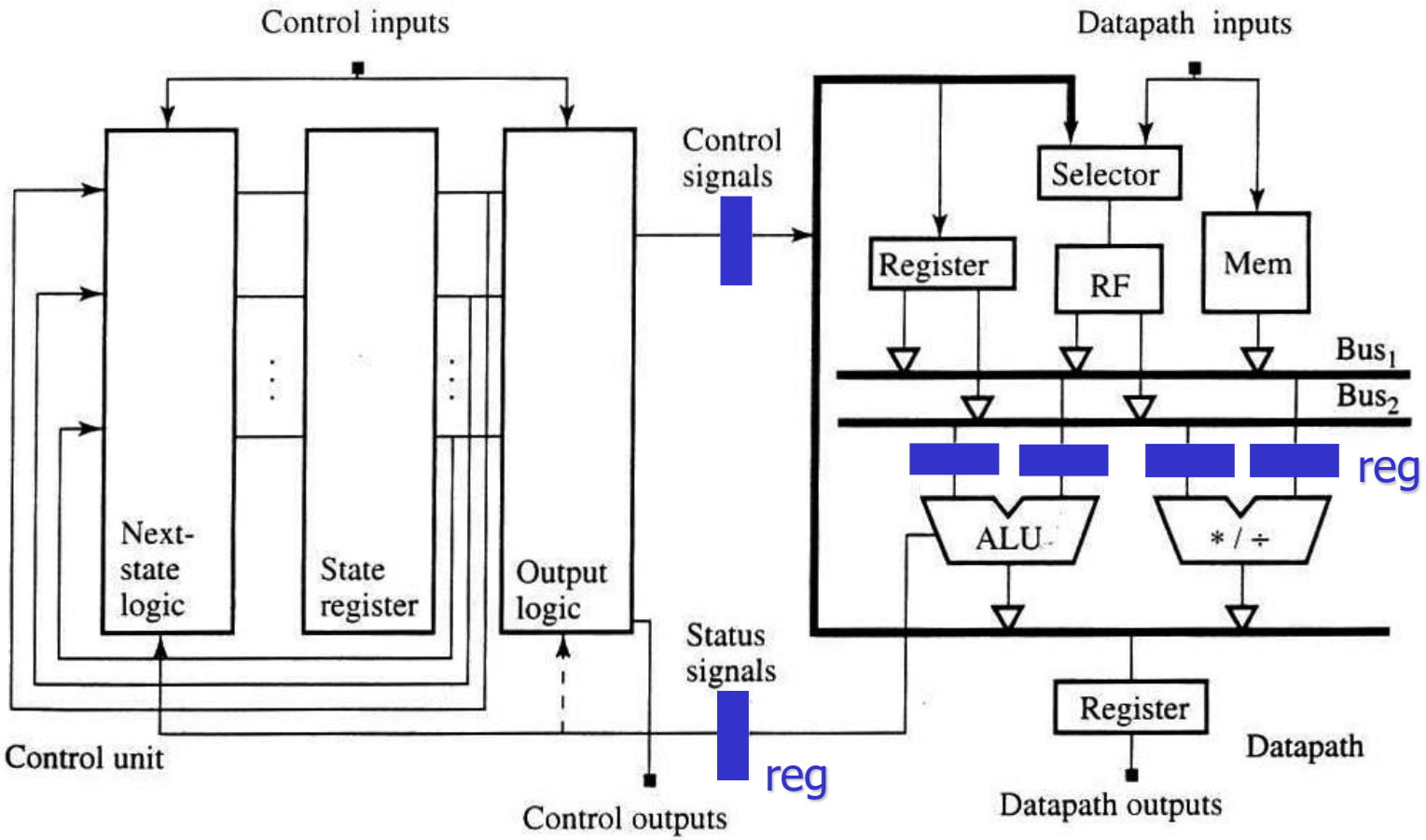
Controlpath Pipelining (1/2)



(a) Standard FSMD implementation

Controlpath Pipelining (2/2)

Insert (1) status registers, (2) control registers and (3) pipeline registers



(b) FSMD implementation with control and datapath pipelining

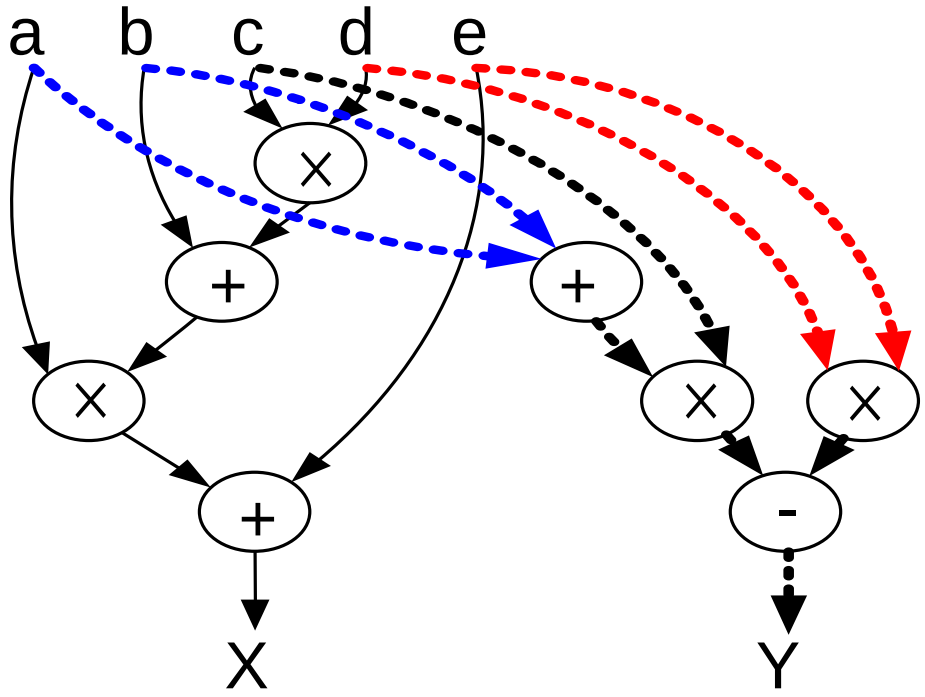
An Example for Synthesis (1/7)

$$X = a \times (b + c \times d) + e$$

$$Y = (a + b) \times c - d \times e$$

What happen if those equations are implemented directly with some parallel functional units ?

High-level synthesis：Scheduling & Allocation



An Example for Synthesis (2/7)

ASAP：1.Assume each operation needs exactly one clock cycle to execute.

2.Unlimited number of functional units or resources are available in each state

3.Schedule each operation into the earliest state in which all its operands are available

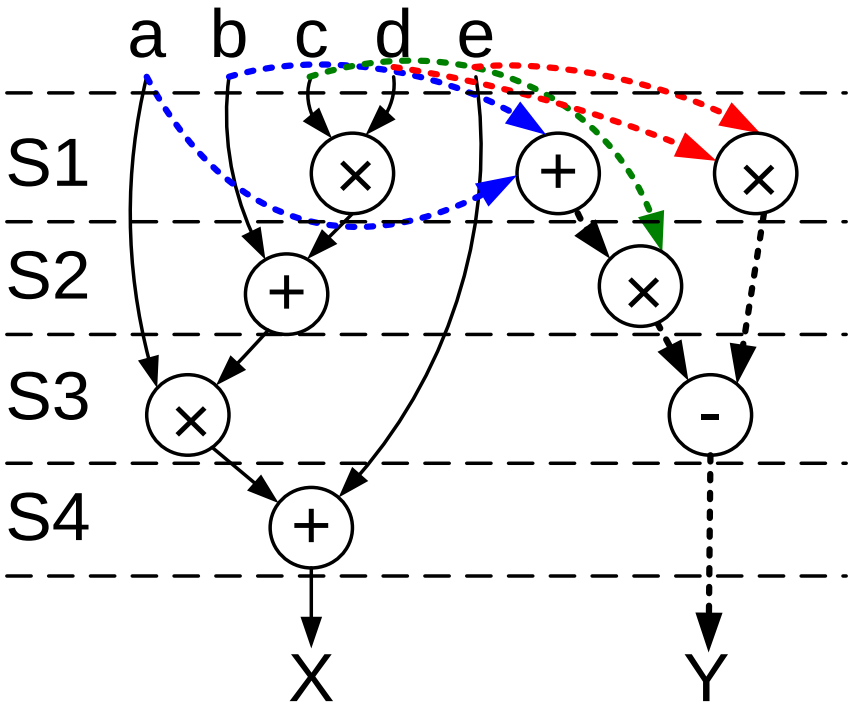
ALAP：Schedule each operation into the last possible state before its result is needed if it is given the length of the final schedule in the number of states as a constraint

An Example for Synthesis (3/7)

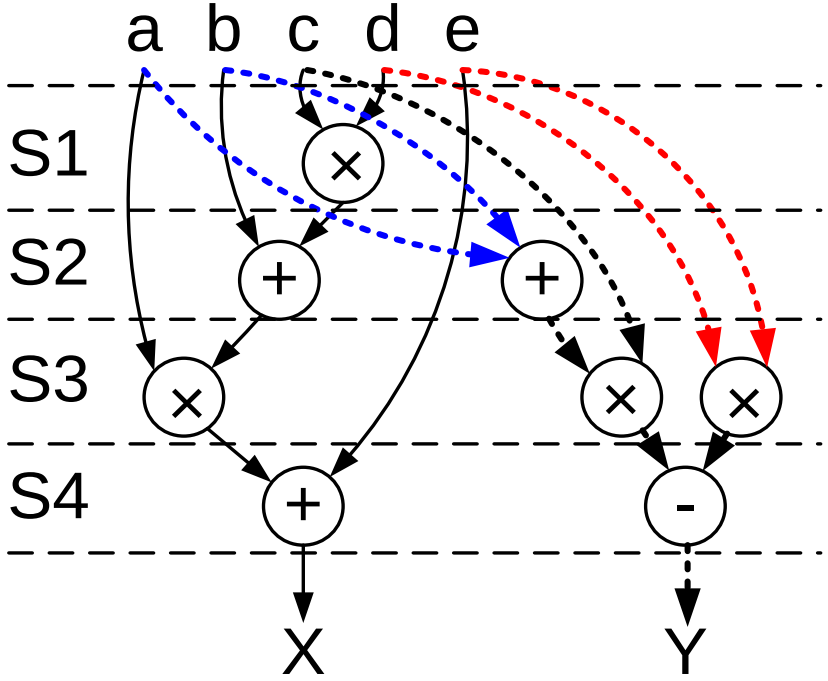
$$X = a \times (b + c \times d) + e$$

$$Y = (a + b) \times c - d \times e$$

ASAP



ALAP（time constraint is 4）

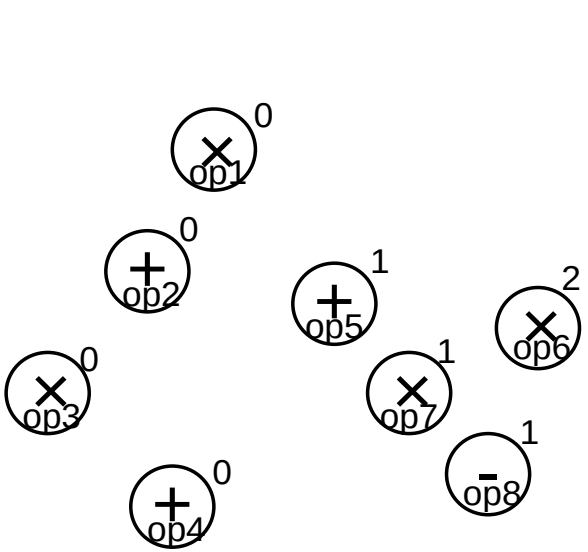


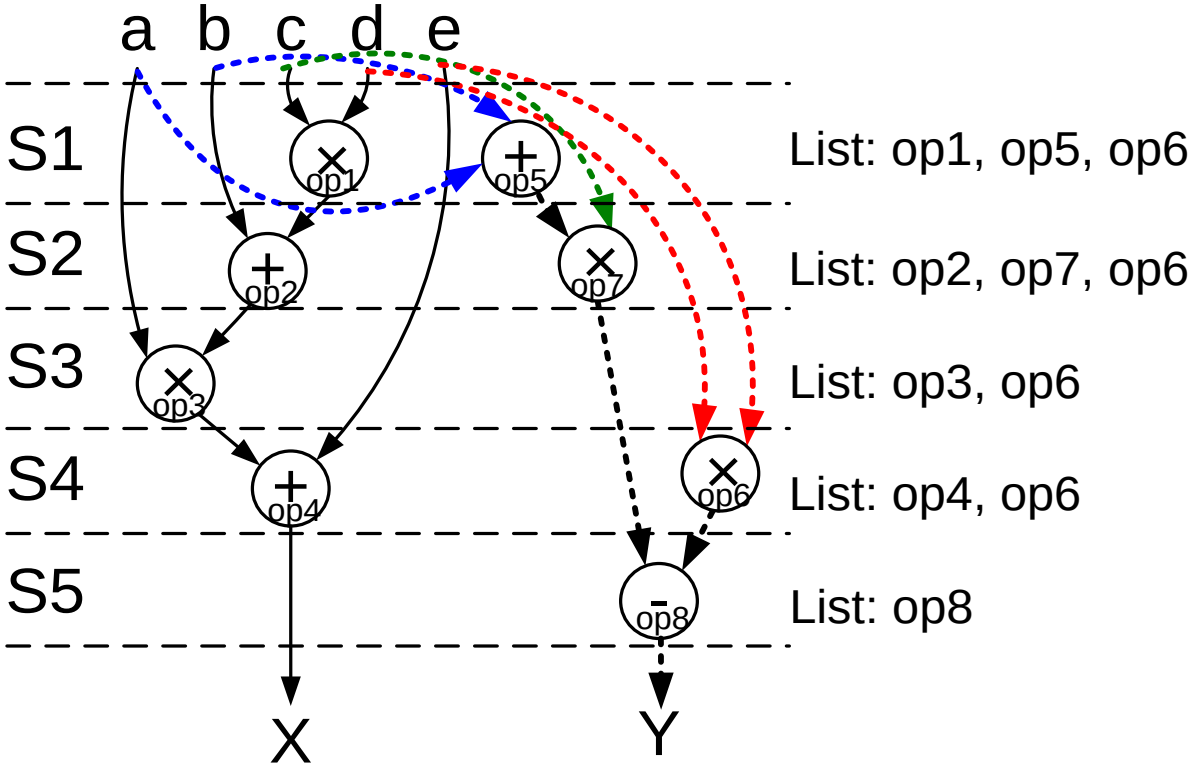
An Example for Synthesis (4/7)

Scheduling: resource-constrained

One multiplier and two adders

mobility

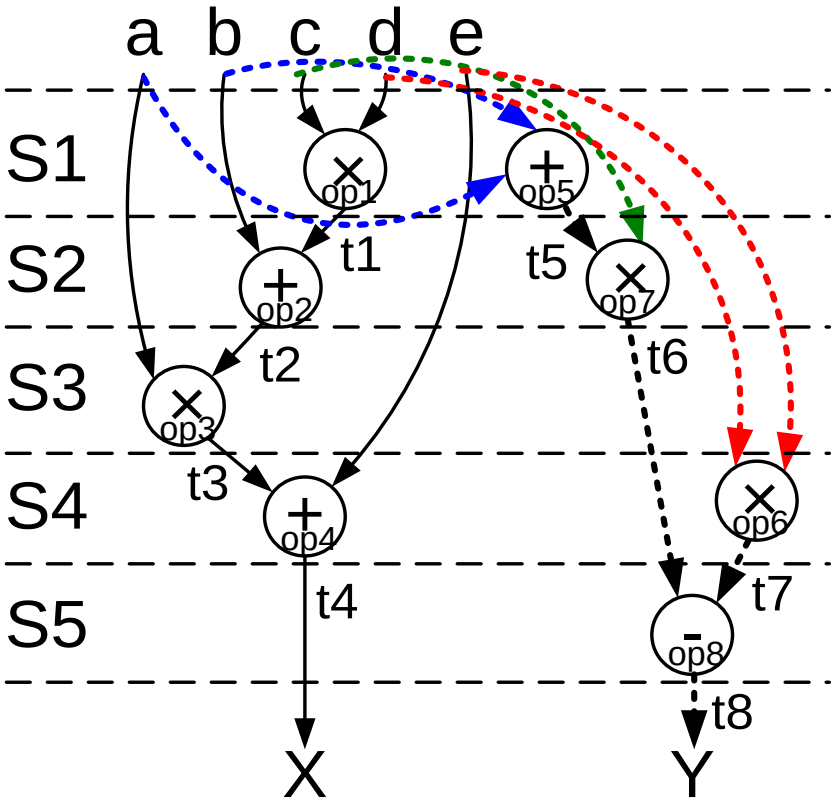




An Example for Synthesis (5/7)

Resources allocation :

Datapath optimizations are considered.



Functional Unit Allocation

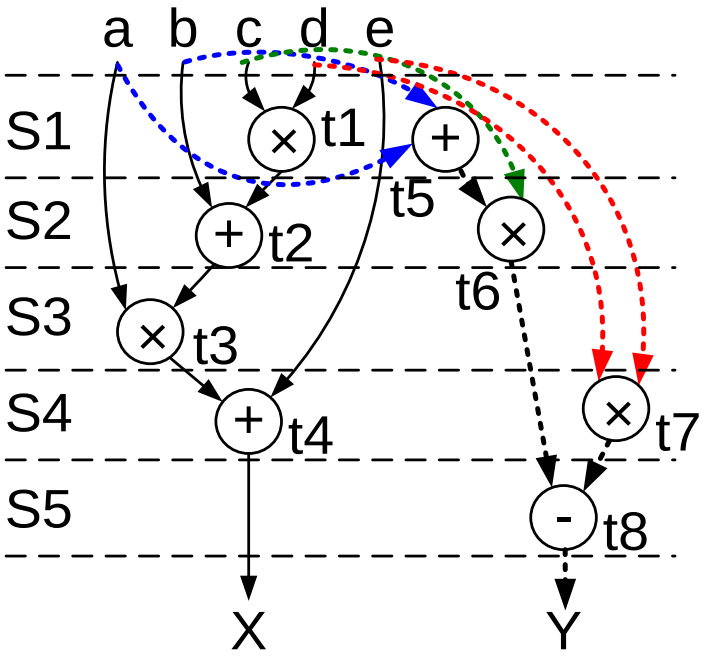
	Multi(*)	adder(+1)	adder(+2)
S_1	op1	op5	
S_2	op7	op2	
S_3	op3		
S_4	op6	op4	
S_5		op8	

An Example for Synthesis (6/7)

Register Allocation (lifetime of each operand)

	a	b	c	d	e	t1	t2	t3	t4	t5	t6	T7
S1	X	X	X	X	X							
S2	X	X	X	X	X	X				X		
S3	X			X	X		X				X	
S4				X	X			X			X	
s5									X		X	X

At least 7 registers are required



An Example for Synthesis (7/7)

	a	b	c	d	e	t1	t2	t3	t4	t5	t6	t7
S1	X	X	X	X	X							
S2	X	X	X	X	X	X				X		
S3	X			X	X		X				X	
S4				X	X			X			X	
s5									X		X	X

Left edge algorithm

R1:a, t3, t4 R2:b, t2, t7

R3:c, t6 R4:d

R5:e R6:t1

R7:t5

. . . .