8.4 Synthesis Optimization from ASM Charts

Show techniques used to optimize the implementations that we derive from ASM charts.

Since each register-transfer (RT) implementation defines both a control unit and a datapath, we can approach the optimization of these parts separately:

We have already seen how to minimize the **control unit**:

- the size of the control unit can be reduced by Boolean minimization and technology mapping techniques.
- minimizing the number of states by merging equivalent states.
- reduce the size of the next-state and output logic by encoding the states properly.

Minimizing the **datapath**:

Three general techniques based on the three major component types used in the datapath:

- The datapath is defined by variable assignments using **storage components**.
- Variables will be assigned new values through arithmetic, logic, and shift operations that are performed by **functional units**.
- **Buses** are used to transfer data from storage components to functional units and back to storage components.

Thus, we want to minimize the storage components, functional units and buses.

Storage components

Not all variables are alive at the same time, thus it is possible for certain variables to share the same register or the same location in a register file or a memory.

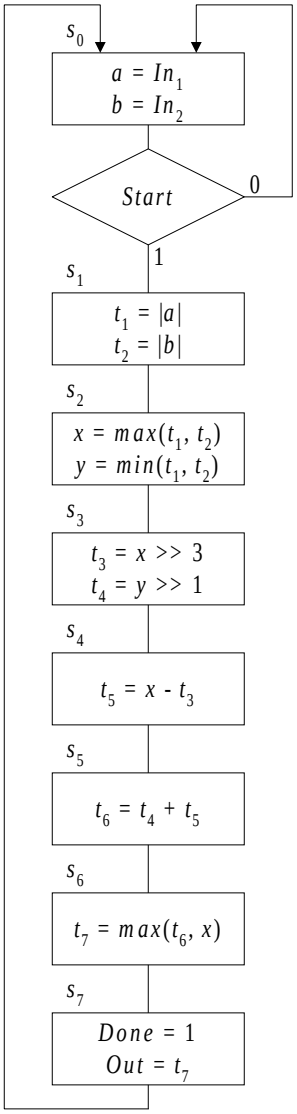
Even if certain variables are alive at the same time, they may not be accessed at the same time, thus they can share the same I/O port.

Functional units

When operations are executed in different states, they could share the same functional unit.

Buses

Again, since different connections will be used in different states, we can group connections into buses, which enables us to reduce the number of wires in the datapath.



Example: Optimize a small custom ASIC to compute the square-root approximation of two signed integers, a and b using the formula

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

where

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

Note:

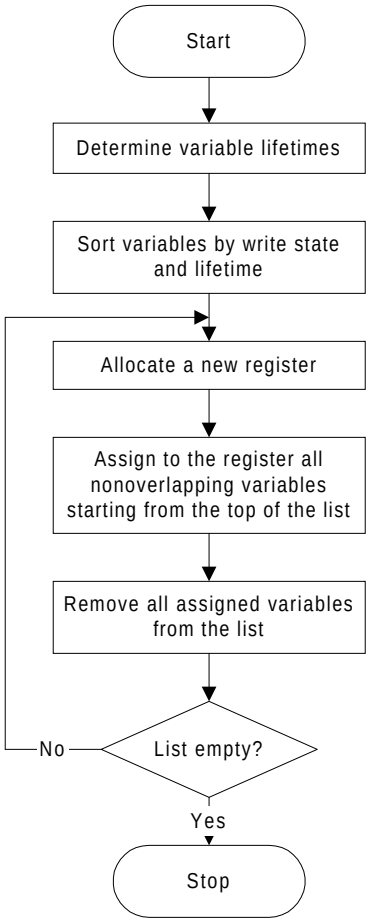
$y >> 1 = 0.5y$
$x >> 2 = 0.25x$
$x >> 3 = 0.125x$
$1x - 0.125x = 0.875x$

(1) **Register / memory sharing**

The **lifetime** of a variable is defined as the set of states in which that variable is **alive**, which includes the state following the state in which it is assigned a new value (write state), every state in which it is used (read state), and all the states on each path between the write state and a read state. e.g. t_4 was assigned a new value in s_3, so it is alive in s_4.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
a	×						
b	×						
t_1		×					
t_2		×					
x			×	×	×	×	
y			×				
$\boldsymbol{t_4}$				×	×		
$\boldsymbol{t_3}$				×			
t_5					×		
t_6						×	
t_7							×
Number of live variables	2	2	2	3	3	2	1

Variable usage
(with sorted list of variables – t_3 and t_4 switched)



The maximum number of variables alive in a single state – three live variables – in s_4 and s_5.

Thus, we can combine variables into three groups so that each group contains variables that are not alive at the same time.

The **left-edge algorithm**, tries to pack as many variables as possible into each register.

- Sort the variables by their write state.
- If two variables have the same write state, the priority is given to the variable with the longer lifetime.
- If two variables have the same write state and the same lifetime, the priority will be assigned at random.

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

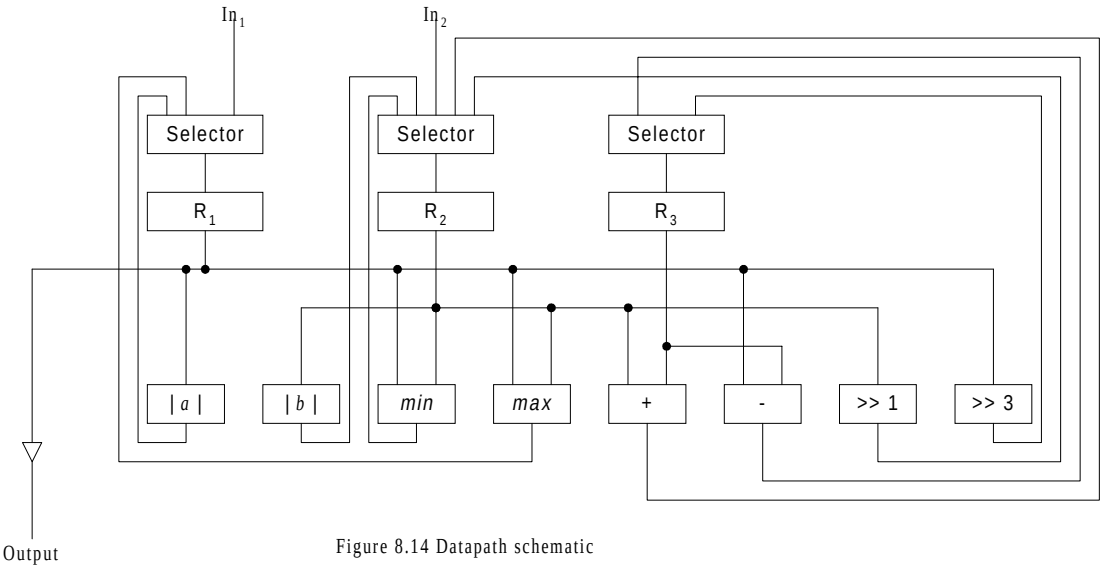


Figure 8.14 Datapath schematic

We cannot use fewer than three registers in the above design. However, there are many possible datapath designs with three registers. We can select one that minimizes the connectivity cost.
The total number of selector inputs is 10 for the datapath shown in Figure 8.14.

(2) **Functional-unit sharing**

Register assignments	Functional-unit sharing
$R_1 = [a, t_1, x, t_7]$	$FU_1 = [\ \|a\|\ ,\ \max\ ]$
$R_2 = [b, t_2, y, t_3, t_5, t_6]$	$FU_2 = [\ \|b\|\ ,\ \min\ ,\ +\ ,\ -\ ]$
$R_3 = [t_4]$	$SH_1 = [\ >>\ 1\ ]$
	$SH_2 = [\ >>\ 3\ ]$

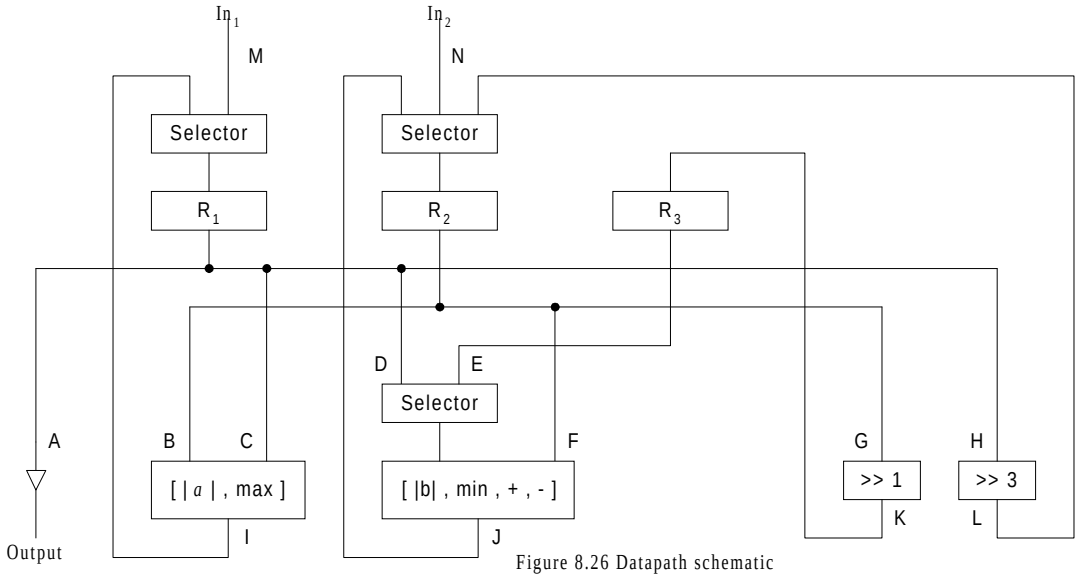


Figure 8.26 Datapath schematic

(3) **Bus sharing**
Assume that we use one register per variable and single-operation functional units.

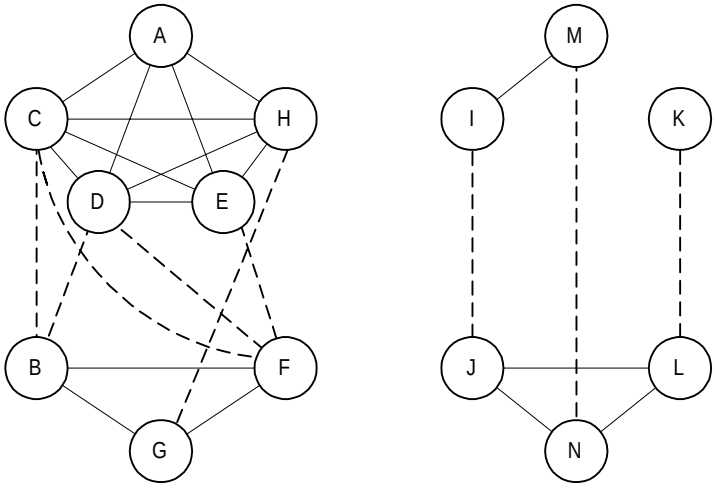
	a	b	t_1	t_2	x	y	t_3	t_4	t_5	t_6	t_7
abs1	I		O								
abs2		I		O							
min			I	I	O						
max			I	I	I	O				I	O
>>3					I		O				
>>1						I		O			
–					I		I		O		
+								I	I	O	

Of these 23 connections, very few are needed in any one state.
The maximum number of connections is used in state s_2 when we need four input connections and two output connections.
Group connections and assigning one bus to each group so as to minimize the connection cost.
The connection cost includes the cost of bust drivers, which are required for every connection of a unit to a bus, and the cost of input selectors, which are required whenever two or more buses are connected to the same input of a storage or functional unit.

Step 1:
Create a **connection use table**.
An × is used to designate the state in which each connection is to be used.

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	Operation
A								×	Out = t_7
B			×				×		max(t_1,?) ; max(t_6, ?)
C		×	×				×		|a| ; max(?, t_2) ; max(?, x)
D			×		×				min(t_1, ?) ; x – ?
E						×			t_4 + ?
F		×	×		×	×			|b| ; min(?, t_2) ; ? – t_3 ; ? + t_5
G				×					>> 1
H				×					>> 3
I		×	×				×		t_1 = |a| ; x = max( ) ; t_7 = max( )
J		×	×		×	×			t_2 = |b| ; y = min( ) ; t_5 = – ; t_6 = +
K				×					t_4 = >> 1
L				×					t_3 = >> 3
M	×								a = In_1
N	×								b = In_2

Step 2:
Construct a **compatibility graph**.
Two nodes are **incompatible** and thus are connected by an incompatibility edge whenever their corresponding connections do not originate from the same source but are to be used at the same time.
e.g. B is incompatible with C because they are used at the same time (s_2) but do not originate from the same source.
Two nodes are connected by **priority edge** whenever their corresponding connections have a common source or a common destination.
e.g. C and D have a common source.
e.g. D and E have a common destination.



Step 3:
Use a graph-partitioning algorithm to group connections in a way that will maximize the number of priority edges included in all group – i.e. trying to cut all the incompatibility edges while cutting as few priority edges as possible.