CSE 140 Lecture 15
System Design II

CK Cheng

CSE Dept.

UC San Diego

Design Process

- Describe system in programs
- Data subsystem
  - List data operations
  - Map operations to functional blocks
  - Add interconnect for data transport
  - Input control signals and output conditions
- Control Subsystem
  - Derive the sequence according to the hardware program
  - Create the sequential machine
  - Input conditions and output control signals

Example: Multiplication

Arithmetic

$Z=X \times Y$

- M=0
- For i=n-1 to 0
  - If $Y_i=1$, $M=M+X* 2^i$
- Z=M

Input X, Y

Output Z

Variable M, i

- M=0
- For i=n-1 to 0
  - If $Y_{n-1}=1$, M=M+X
  - Shift Y left by one bit
  - If i != 0, shift M left by one bit
- Z=M

Implementation: Example

Multiply(X, Y, Z, start, done)

{ Input X[15:0], Y[15:0] type bit-vector, start type boolean;
Local-Object A[15:0], B[15:0] ,M[31:0], i[4:0] type bit-vector;
Output Z[31:0] type bit-vector, done type boolean;

S0: If start' goto S0 || done ←1;

S1: A ← X || B ← Y || i←0 || M←0 || done ←0;

S2: If B_{15} = 0 goto S4 || i←i+1;

S3: M ← M+A;

S4: if i>= 16, goto S6

S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;

S6: Z:←M || done← 1|| goto S0;

}

Step 0: Syntax

S1: A ← X || B ← Y || i←0 || M←0 || done ← 0;

S2: If B_{15} = 0 goto S4 || i←i+1;

S3: M ← M+A;

S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;

S6: Z:←M || done← 1|| goto S0;

Step 0: Syntax

Multiply(X, Y, Z, start, done)

{ Input X[15:0], Y[15:0] type bit-vector, start type boolean;

Local-Object A[15:0], B[15:0] ,M[31:0], i[4:0] type bit-vector;

Output Z[31:0] type bit-vector, done type boolean;

S0: If start’ goto S0 || done ⟵1;

S1: A ⟵ X || B ⟵ Y || i⟵0 || M⟵0 || done ⟵0;

S2: If B_{15} = 0 goto S4 || i⟵i+1;

S3: M ⟵ M+A;

S4: if i>= 16, goto S6

S5: M⟵Shift(M,L,1) || B⟵Shift(B,L,1) || goto S2;

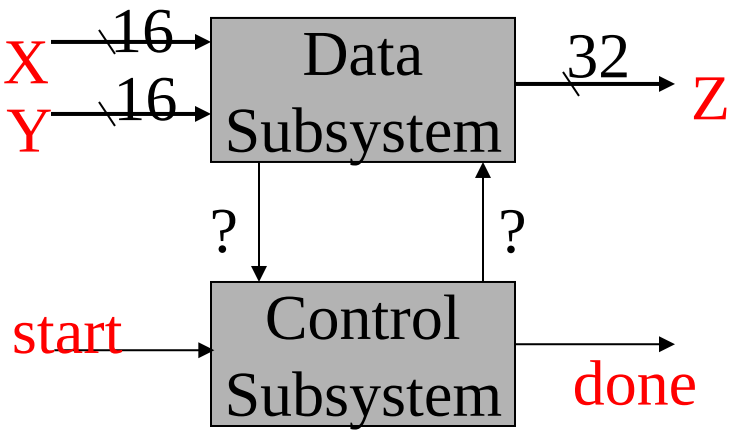
S6: Z:⟵M || done⟵ 1|| goto S0;

}

Step 1: Identify Input and Output of data and control subsystems

Multiply(X, Y, Z, start, done)
{ Input: X[15:0], Y[15:0] type bit-vector,
start type boolean;
Local-Object : A[15:0], B[15:0] ,M[31:0],
i[4:0] type bit-vector;
Output Z[31:0] type bit-vector,
done type boolean;
S0: If start' goto S0 || done ←1;
S1: A ← X || B ← Y || i←0 || M←0 || done ←0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ← M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:←M || done← 1|| goto S0;
}

Z=XY

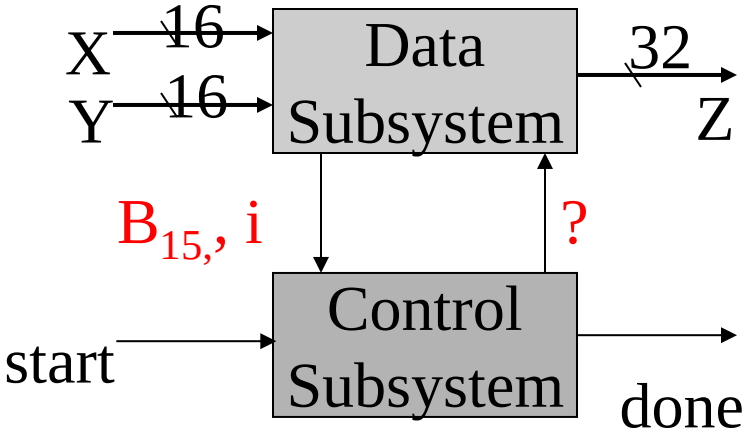


Step 2: Identify Condition Bits to Control Subsystem

Multiply(X, Y, Z, start, done)

{ Input: X[15:0], Y[15:0] type bit-vector,
start type boolean;
Local-Object : A[15:0], B[15:0] ,M[31:0],
i[4:0] type bit-vector;
Output Z[31:0] type bit-vector,
done type boolean;

S0: If start' goto S0 || done ← 1;
S1: A ← X || B← Y || i← 0 || M← 0 || done ← 0;
S2: If B_{15} = 0 goto S4 || i← i+1;
S3: M ← M+A;
S4: if i>= 16, goto S6
S5: M← Shift(M,L,1) || B← Shift(B,L,1) || goto S2;
S6: Z:← M || done← 1|| goto S0;
}



Step 3: Identify Data Subsystem Operations

Multiply(X, Y, Z, start, done)

{ Input: X[15:0], Y[15:0] type bit-vector,
start type boolean;
Local-Object : A[15:0], B[15:0] ,M[31:0],
i[4:0] type bit-vector;
Output Z[31:0] type bit-vector,
done type boolean;

S0: If start' goto S0 || done ←1;

S1: A ← X || B ← Y || i←0 || M←0 || done ← 0;

S2: If B_{15} = 0 goto S4 || i←i+1;

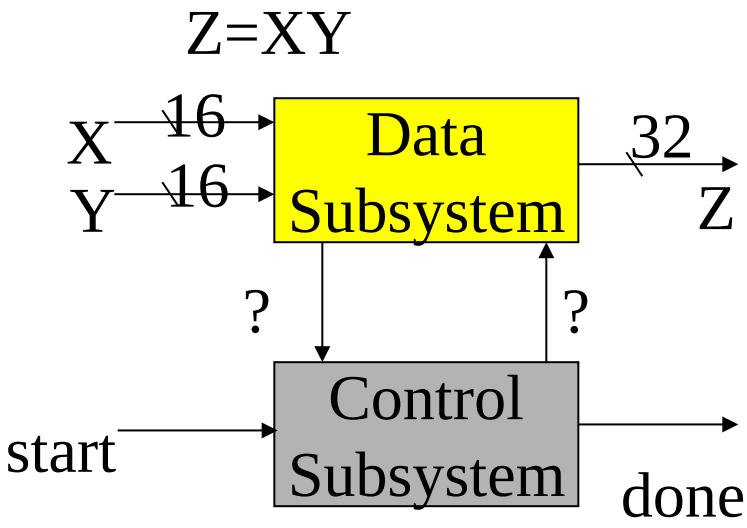
S3: M ← M+A;

S4: if i>= 16, goto S6

S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;

S6: Z:← M || done← 1|| goto S0;

}



Step 4: Map Data Operations to Implementable functions

Multiply(X, Y, Z, start, done)

{ Input: X[15:0], Y[15:0] type bit-vector,
start type boolean;
Local-Object : A[15:0], B[15:0] ,M[31:0],
i[4:0] type bit-vector;
Output Z[31:0] type bit-vector,
done type boolean;

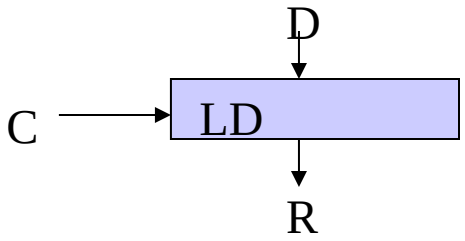
S0: If start' goto S0 || done←1;
S1: A ← X || B ← Y || i←0 || M←0 || done ← 0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ← M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:← M || done← 1|| goto S0;
}

A ← X
B ←Y
M←0
i←0
i←i+ 1
M←M+A
M←Shift(M,L,1)
B←Shift(B,L,1)
Z:←M

operation
A ← Load (X)
B ← Load (Y)
M← Clear(M)
i← Clear(i)
i ← INC(i)
M← Add(M,A)
M ← SHL(M)
B ← SHL(B)
Wires

Step 5: Implement the Data Subsystem from Standard Modules

Registers: If C then R ← D

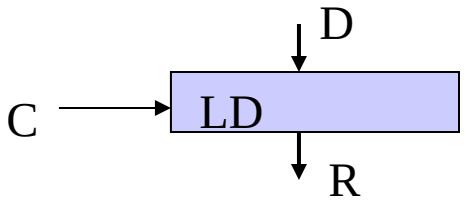


operation

A ← Load (X)
B ← Load (Y)
M← Clear(M)
i← Clear(i)
i ← INC(i)
M← Add(M,A)
M ← SHL(M)
B ← SHL(B)

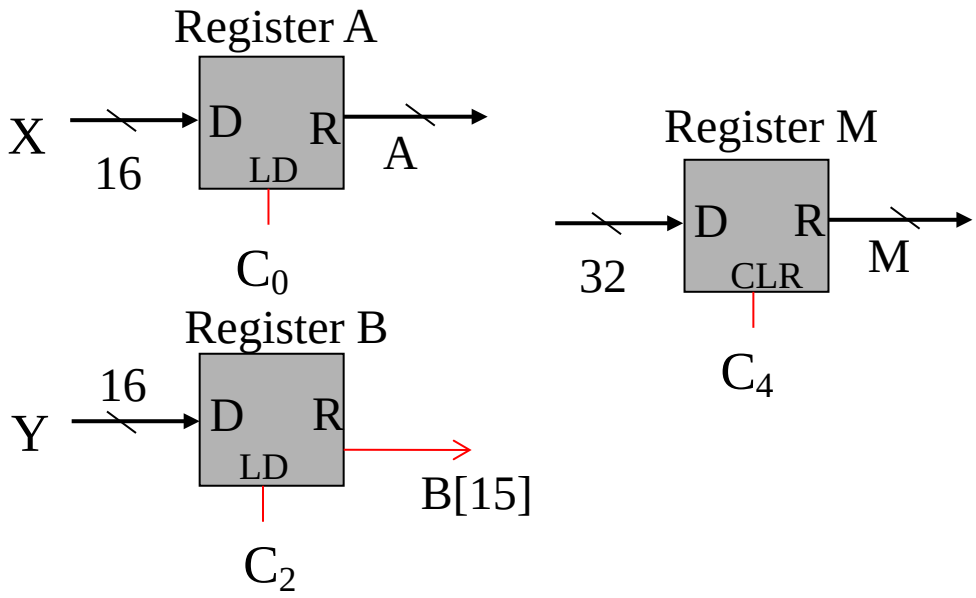
Storage Component: Registers with control signals

Registers: If C then R ← D



operation

A ← Load (X)
B ← Load (Y)
M← Clear(M)
i← Clear(i)
i ← INC(i)
M← Add(M,A)
M ← SHL(M)
B ← SHL(B)



Function Modules: Adder, Shifter

operation

A ← Load (X)

B ← Load (Y)

M← Clear(M)

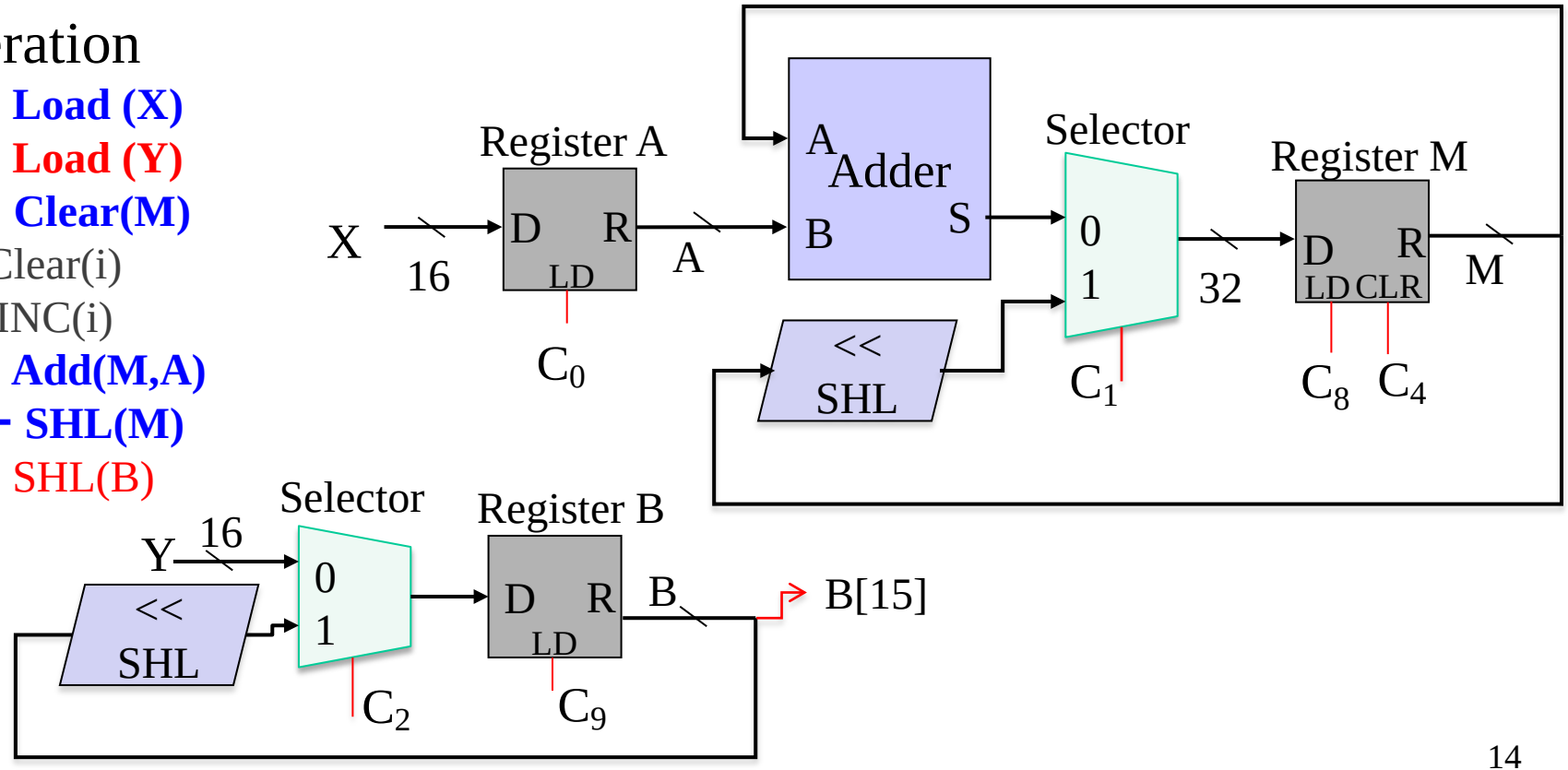
i← Clear(i)

i ← INC(i)

M← Add(M,A)

M ← SHL(M)

B ← SHL(B)



Function Modules: Adder, Shifter, Counter

operation

A ← Load (X)

B ← Load (Y)

M← Clear(M)

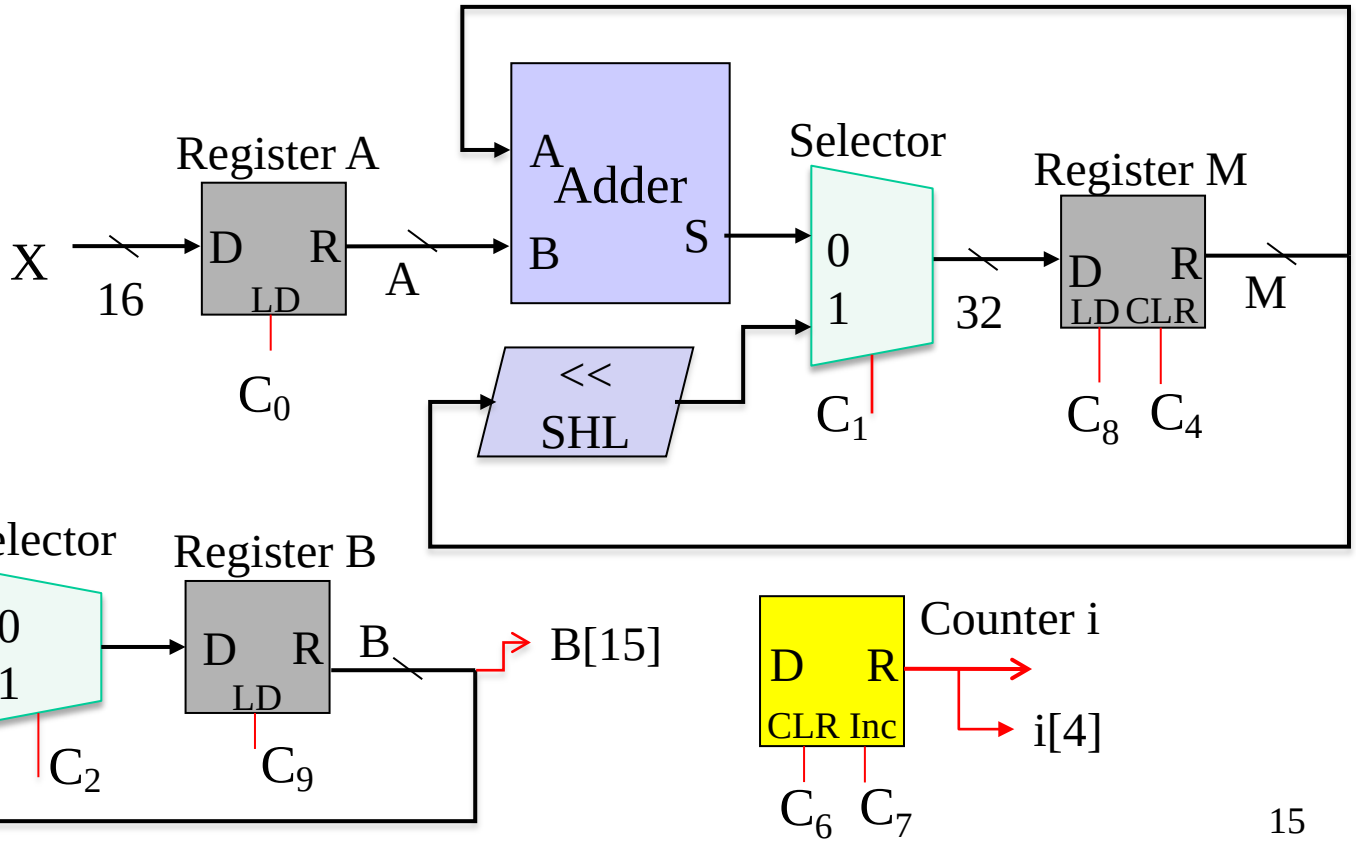
i← Clear(i)

i ← INC(i)

M← Add(M,A)

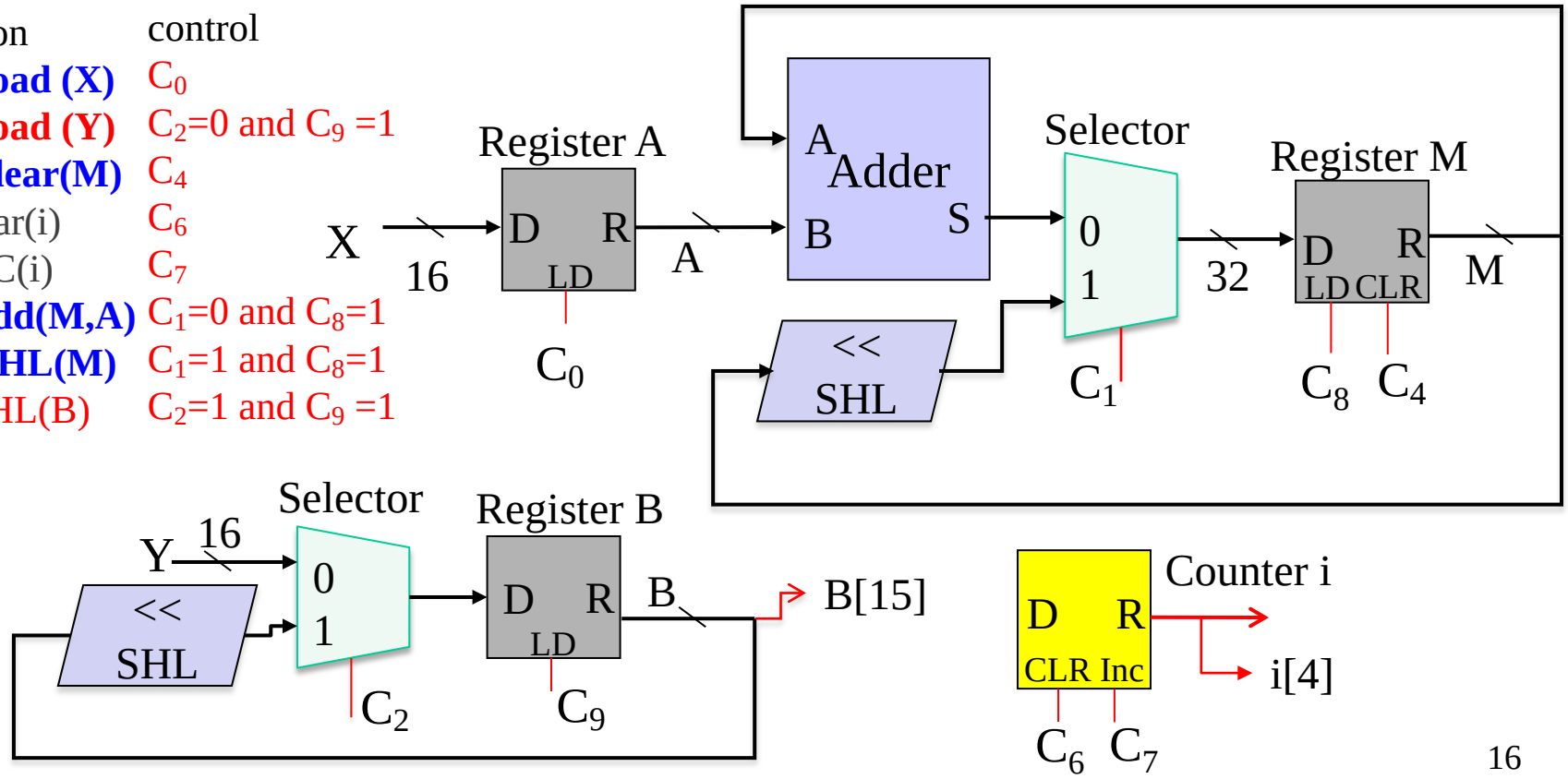
M ← SHL(M)

B ← SHL(B)



Step 6: Map Control Signals to Operations

operation	control
A ← Load (X)	C_0
B ← Load (Y)	C_2=0 and C_9 =1
M← Clear(M)	C_4
i← Clear(i)	C_6
i ← INC(i)	C_7
M← Add(M,A)	C_1=0 and C_8=1
M ← SHL(M)	C_1=1 and C_8=1
B ← SHL(B)	C_2=1 and C_9 =1



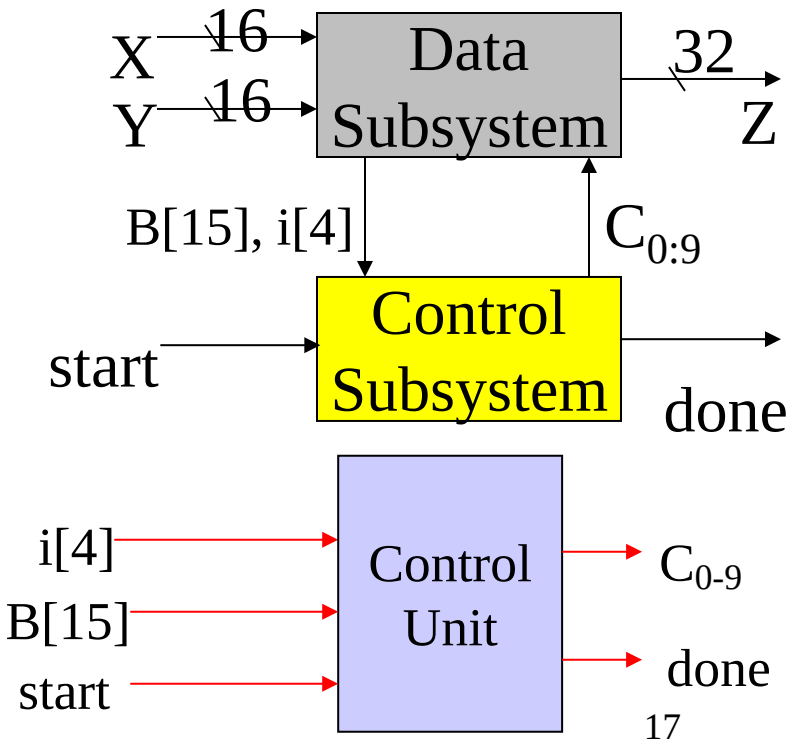
Step 7: Identify Control Path Components

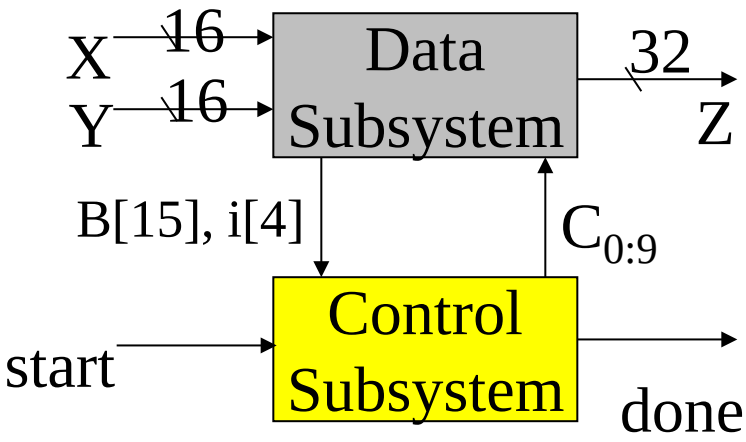
Multiply(X, Y, Z, start, done)

{ Input: X[15:0], Y[15:0] type bit-vector,
start type boolean;
Local-Object : A[15:0], B[15:0] ,M[31:0],
i[4:0] type bit-vector;
Output Z[31:0] type bit-vector,
done type boolean;

S0: If start' goto S0 || done←1;
S1: A← X || B ←Y || i←0 || M←0 || done ←0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ←M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:←M || done←1|| goto S0;
}

Z=XY

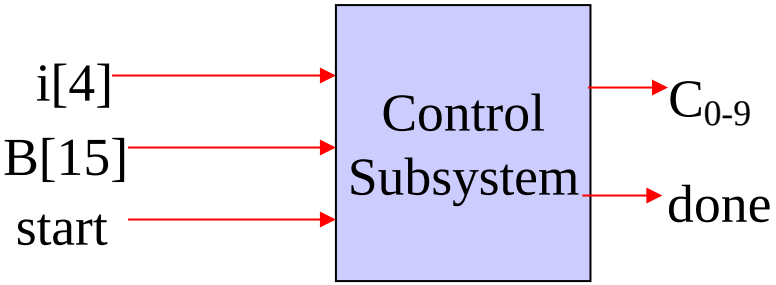



X
16
Y
16
Data
Subsystem
32
Z
B[15], i[4]
C0:9
start
Control
Subsystem
done

PI Q: Which of the following can be used to sequence the order of computation of our algorithm

A. A counter
B. A finite state machine
C. All of the above
D. A combinational circuit

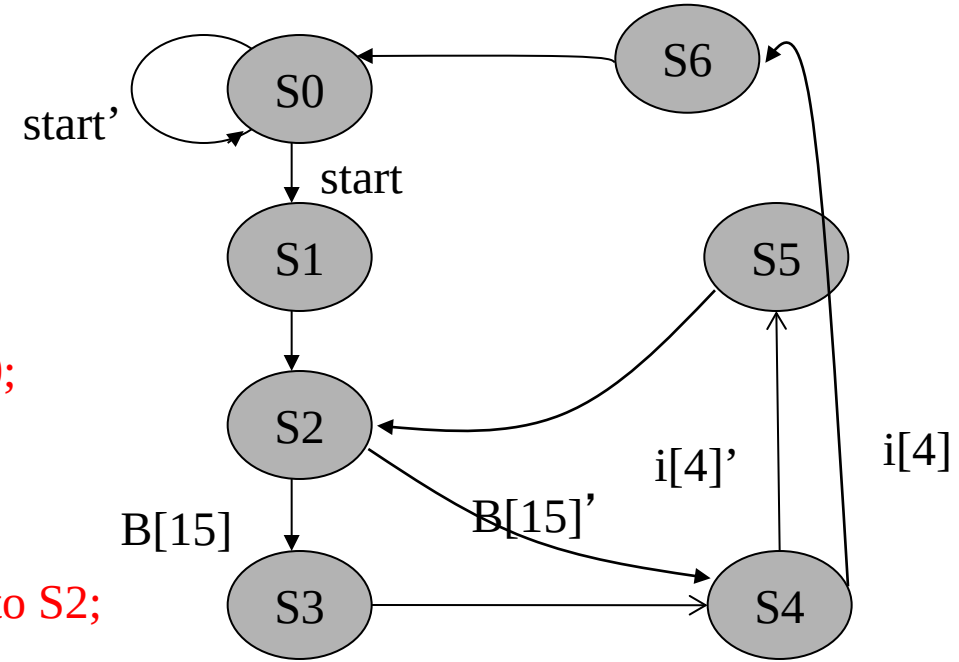
Design of the Control Subsystem



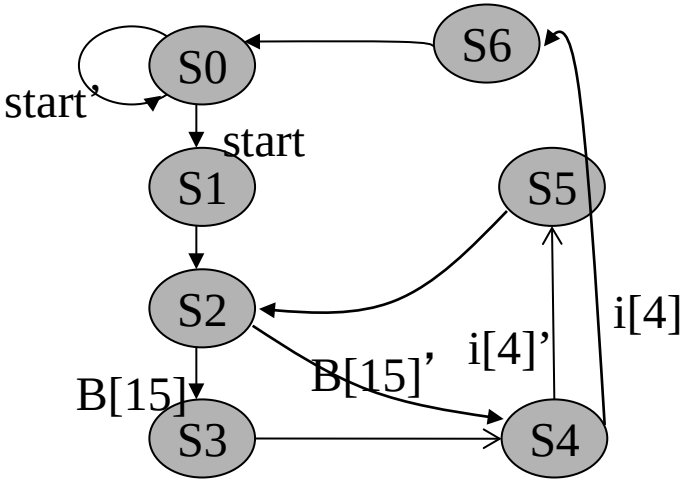
Multiply(X, Y, Z, start, done)
{
S0: If start' goto S0 || done←1;
S1: A← X || B ←Y || i←0 || M←0 || done ←0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ←M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:←M || done←1|| goto S0
}

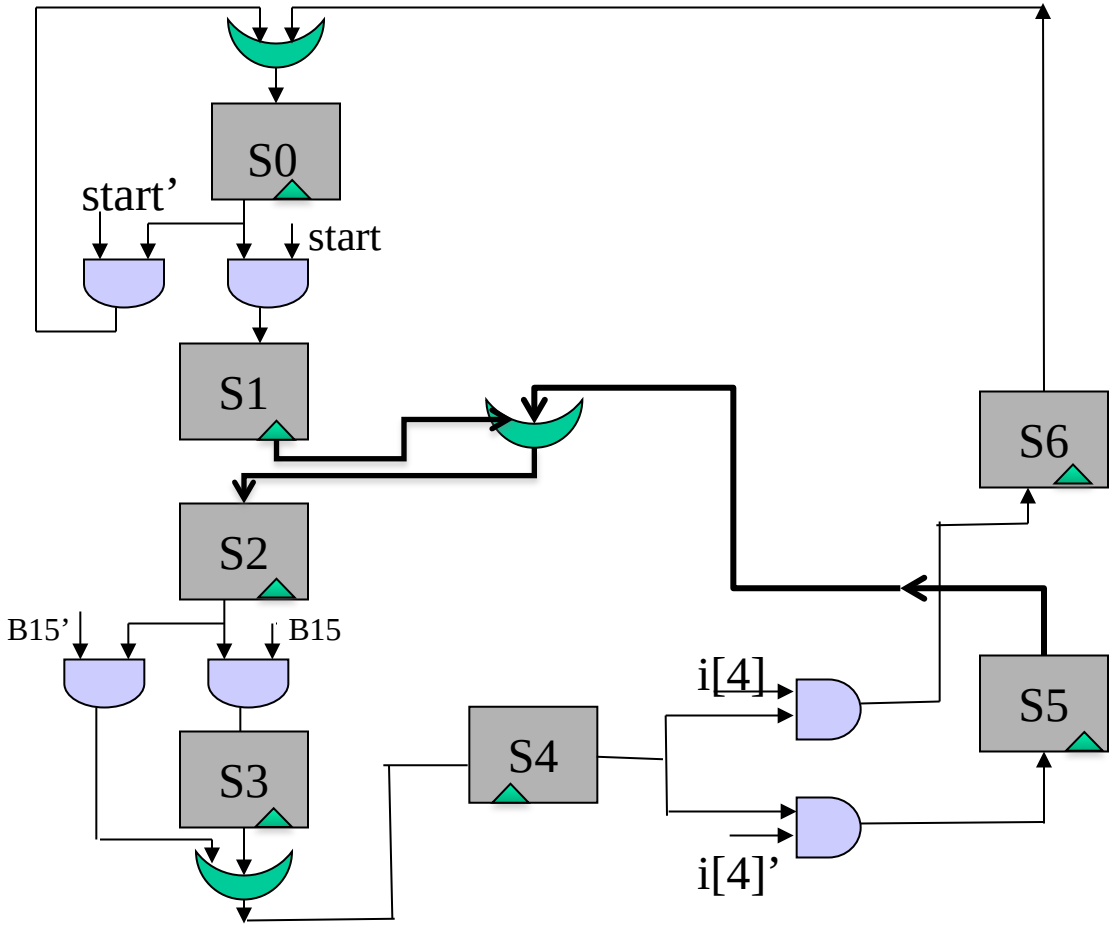
Control Subsystem

Multiply(X, Y, Z, start, done)
{
S0: If start’ goto S0 || done←1;
S1: A← X || B ←Y || i←0 || M←0 || done ←0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ←M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:←M || done←1|| goto S0
}



One-Hot State Machine





Control Subsystem: One-Hot State Machine Design

Input: State Diagram

1. Use a flip flop to replace each state.
2. Set the flip flop which corresponds to the initial state and reset the rest flip flops.
3. Use an OR gate to collect all inward edges.
4. Use a Demux to distribute the outward edges.

operation	control
A ← Load (X)	C_0
B ← Load (Y)	C_2=0 and C_9 =1
M← Clear(M)	C_4
i← Clear(i)	C_6
i ← INC(i)	C_7
M← Add(M,A)	C_1=0 and C_8=1
M ← SHL(M)	C_1=1 and C_8=1
B ← SHL(B)	C_2=1 and C_9 =1

Multiply(X, Y, Z, start, done) {
S0: If start' goto S0 || done←1;
S1: A← X || B ←Y || i←0 || M←0 || done ←0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ←M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:←M || done←1|| goto S0;}

	C0 Load A	C1 Feed M	C2 Feed B	C4	C6	C7	C8 Load M	C9 Load B	done
S0	0			0	0	0	0	0	1
S1	1			1	1	0	0	1	0
S2	0			0	0	1	0	0	0
S3	0			0	0	0	1	0	0
S4	0			0	0	0	0	0	0
S5	0			0	0	0	1	1	0
S6	0			0	0	0	0	0	1

operation	control
A ← Load (X)	C_0
B ← Load (Y)	C_2=0 and C_9 =1
M← Clear(M)	C_4
i← Clear(i)	C_6
i ← INC(i)	C_7
M← Add(M,A)	C_1=0 and C_8=1
M ← SHL(M)	C_1=1 and C_8=1
B ← SHL(B)	C_2=1 and C_9 =1

Multiply(X, Y, Z, start, done) {
S0: If start' goto S0 || done←1;
S1: A← X || B ←Y || i←0 || M←0 || done ←0;
S2: If B_{15} = 0 goto S4 || i←i+1;
S3: M ←M+A;
S4: if i>= 16, goto S6
S5: M←Shift(M,L,1) || B←Shift(B,L,1) || goto S2;
S6: Z:←M || done←1|| goto S0;}

	C0 Load A	C1 Feed M	C2 Feed B	C4	C6	C7	C8 Load M	C9 Load B	done
S0	0	X	X	0	0	0	0	0	1
S1	1	X	0	1	1	0	0	1	0
S2	0	X	X	0	0	1	0	0	0
S3	0	0	X	0	0	0	1	0	0
S4	0	X	X	0	0	0	0	0	0
S5	0	1	1	0	0	0	1	1	0
S6	0	X	X	0	0	0	0	0	1

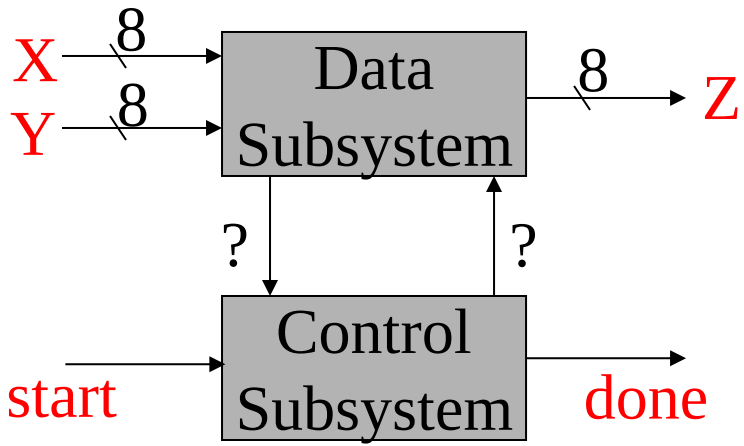
Implementation: Example

Given a hardware program, implement data path and control subsystems

```
{ Input X[7:0], Y[7:0] type bit-vector, start type boolean;
  Local-Object A[7:0], B[7:0] type bit-vector;
  Output Z[7:0] type bit-vector, done type boolean;
 Wait: If start' goto Wait || done←1;
   S1: A ←X || B ←Y|| done ←0;
   S2: If B >= 0 goto S4;
   S3: B ←-B;
   S4: If A >= B goto S6;
   S5: A ←A + 1 || B ←B-1 || goto S4;
   S6: Z ←4 * A || done ←1 || goto Wait;
}
```

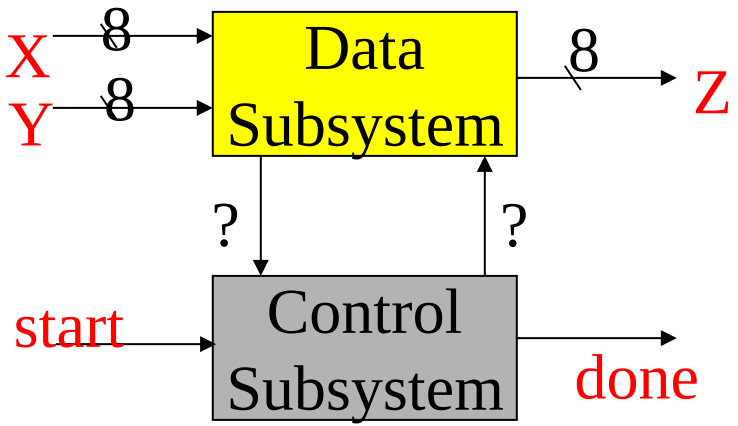
Step 1: Identify Input and Output of data and control subsystems

```
Some_function
{ Input X[7:0], Y[7:0] type bit-vector,
          start type boolean;
  Local-Object A[7:0], B[7:0] type bit-vector;
  Output Z[7:0] type bit-vector,
          done type boolean;
Wait: If start' goto Wait || done←1;
   S1: A ←X || B ←Y|| done ←0;
   S2: If B >= 0 goto S4;
   S3: B ←-B;
   S4: If A >= B goto S6;
   S5: A ←A + 1 || B ←B-1 || goto S4;
   S6: Z ←4 * A || done ←1 || goto Wait;
}
```



Step 2: Identify Data Subsystem Operations

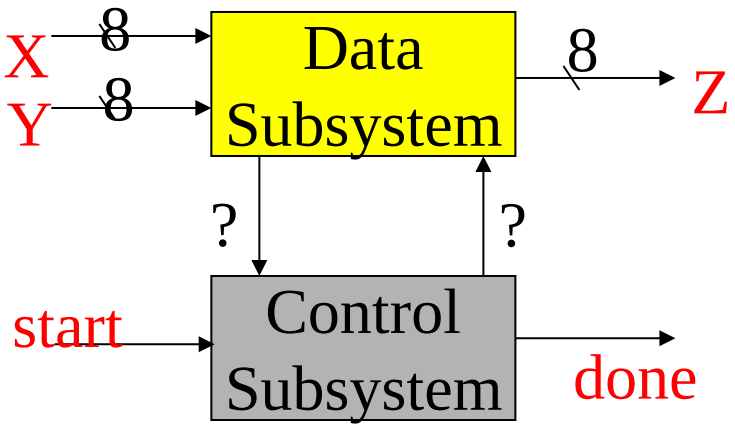
Some_function
{ Input X[7:0], Y[7:0] type bit-vector,
start type boolean;
Local-Object A[7:0], B[7:0] type bit-vector;
Output Z[7:0] type bit-vector,
done type boolean;
Wait: If start' goto Wait || done←1;
S1: A ←X || B ←Y|| done ←0;
S2: If B >= 0 goto S4;
S3: B ←-B;
S4: If A >= B goto S6;
S5: A ←A + 1 || B ←B-1 || goto S4;
S6: Z ←4 * A || done ←1 || goto Wait;
}



$$Z = \begin{cases} 4\,\text{Ceiling}[\,(X + |Y|\,)/\,2] & \text{if } X < |Y| \\ 4X & \text{otherwise} \end{cases}$$

Step 2: Identify Data Subsystem Operations

Some_function
{ Input X[7:0], Y[7:0] type bit-vector,
start type boolean;
Local-Object A[7:0], B[7:0] type bit-vector;
Output Z[7:0] type bit-vector,
done type boolean;
Wait: If start' goto Wait || done←1;
S1: A ←X || B ← Y|| done <= 0;
S2: If B >= 0 goto S4;
S3: B ← -B;
S4: If A >= B goto S6;
S5: A ← A + 1 || B← B-1 || goto S4;
S6: Z ← 4 * A || done ← 1 || goto Wait;
}



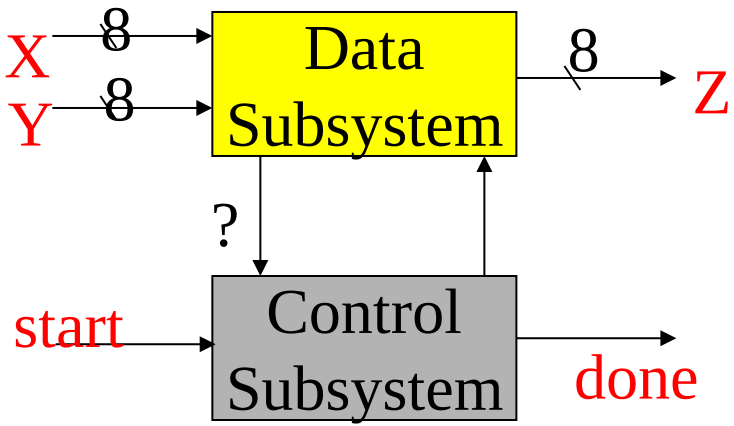
Step 2: Map Data Operations to Implementable functions

```
{Input X[7:0], Y[7:0] type bit-vector,
        start type boolean;
  Local-Object A[7:0], B[7:0] type bit-vector;
  Output Z[7:0] type bit-vector,
        done type boolean;
Wait: If start' goto Wait || done←1;
   S1: A ←X || B ←Y|| done <= 0;
   S2: If B >= 0 goto S4;
   S3: B ←-B;
   S4: If A >= B goto S6;
   S5: A ←A + 1 || B ← B-1 || goto S4;
   S6: Z ← 4 * A || done ← 1 || goto Wait;
}
```

	operation
A ← X	A ← Load (X)
B ← Y	B ← Load (Y)
B ← -B	B ← CS (B)
A >= B	Comp (A, B)
A ← A + 1	A ← INC (A)
B ← B – 1	B ← DEC (B)
Z ← 4A	Z ← SHL(A)

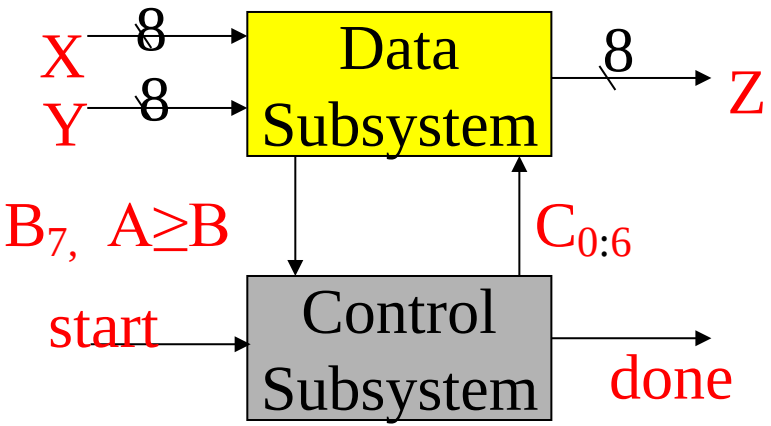
Step 3: Tag each Data Operations with a Control Signal

	operation
A ← X	A ← Load (X)
B ← Y	B ← Load (Y)
B ← -B	B ← CS (B)
A >= B	Comp (A, B)
A ← A + 1	A ← INC (A)
B ← B – 1	B ← DEC (B)
Z ← 4A	Z ← SHL(A)



Step 4: Identify Condition Bits to Control Subsystem

{Input X[7:0], Y[7:0] type bit-vector,
start type boolean;
Local-Object A[7:0], B[7:0] type bit-vector;
Output Z[7:0] type bit-vector,
done type boolean;
Wait: If start' goto Wait || done←1;
S1: A ← X || B ← Y|| done ← 0;
S2: If B ≥ 0 goto S4;
S3: B ← -B;
S4: If A ≥ B goto S6;
S5: A ← A + 1 || B ← B-1 || goto S4;
S6: Z ← 4 * A || done← 1 || goto Wait;
}



Step 5: Implement the Data Subsystem from Standard Modules

operation

A ← Load (X)

B ← Load (Y)

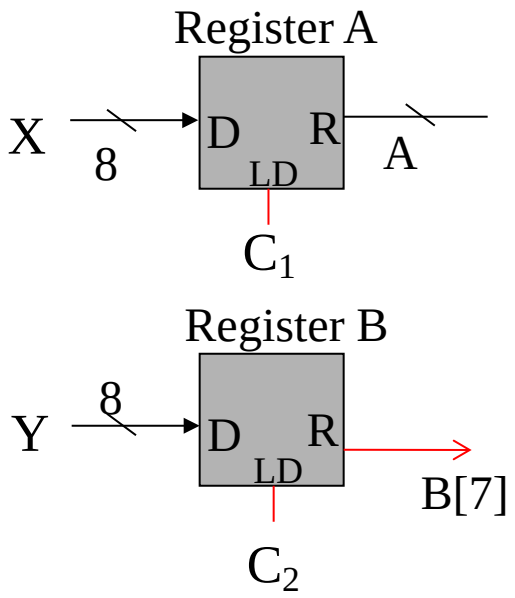
B ← CS (B)

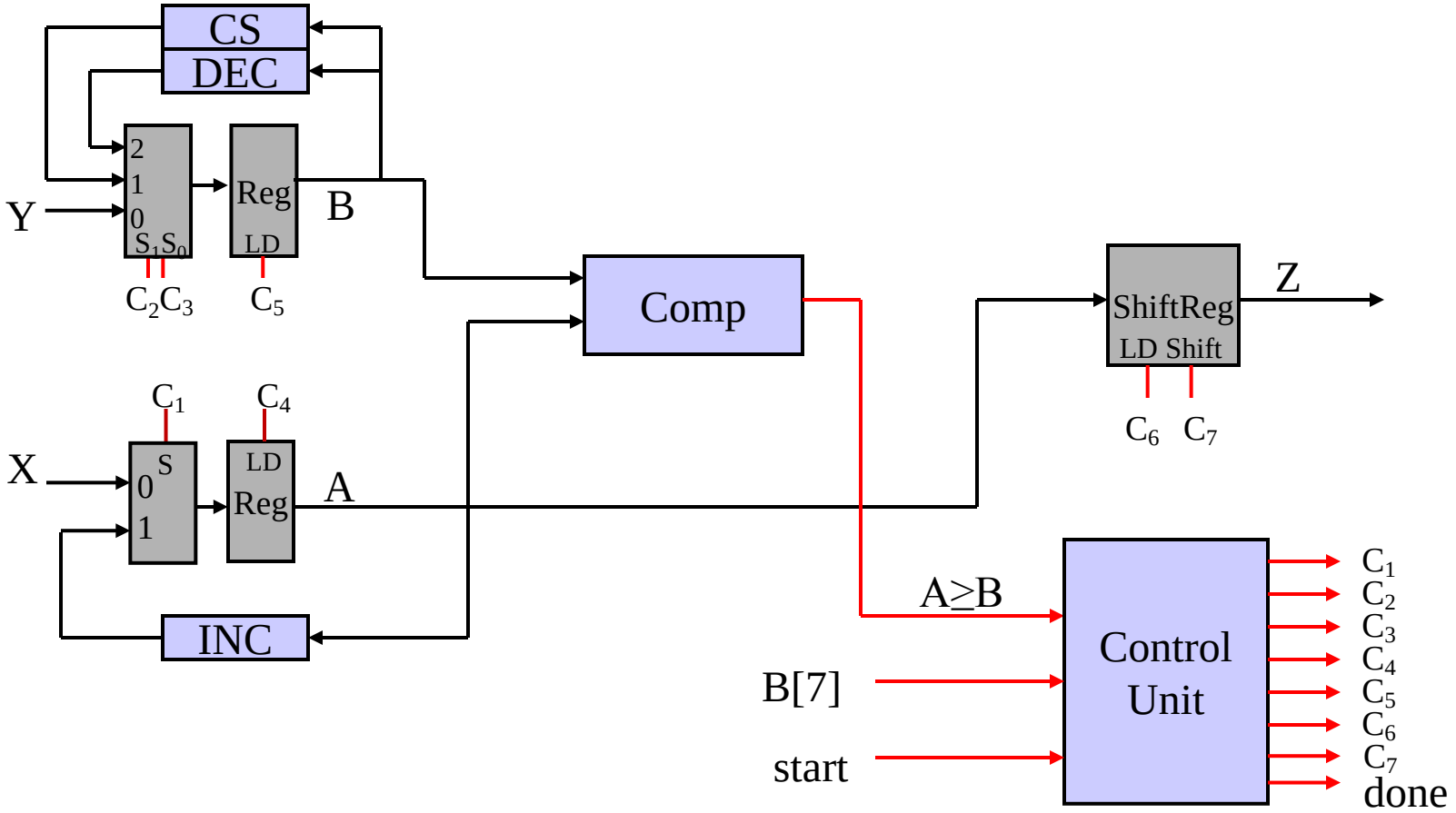
Comp (A, B)

A ← INC (A)

B ← DEC (B)

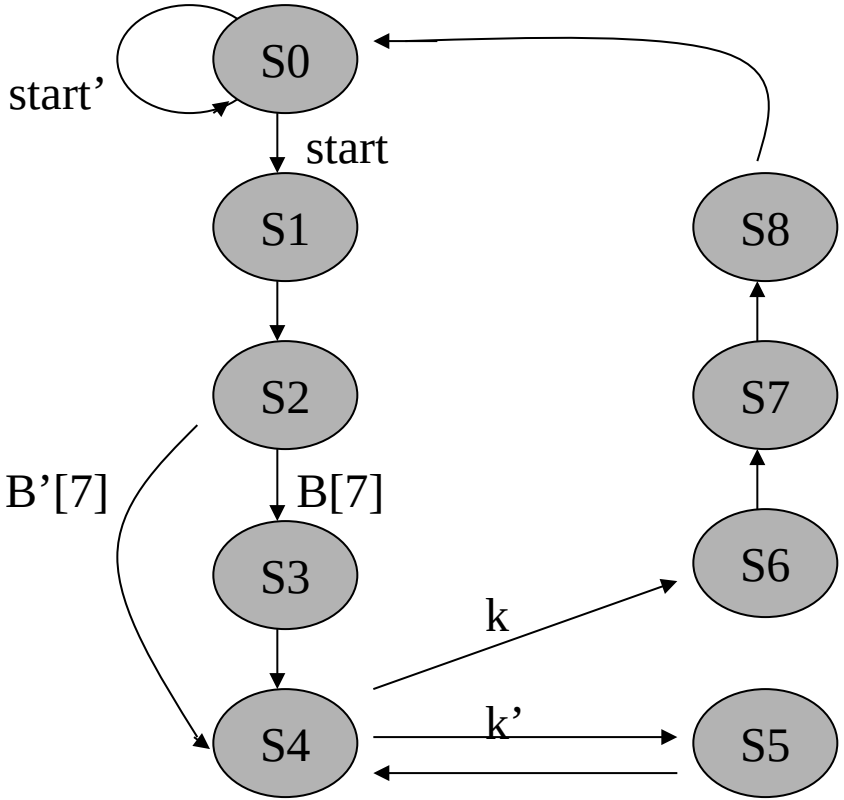
Z ← SHL(A)



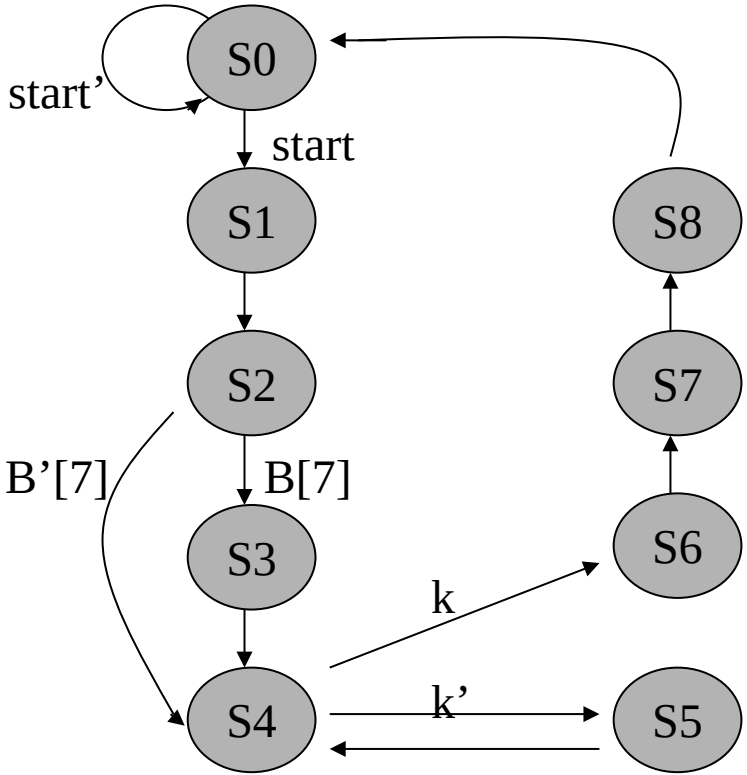

CS
DEC
2
1
0
S1S0
Y
Reg
LD
B
C2C3
C5
Comp
ShiftReg
LD Shift
Z
C6
C7
C1
C4
X
S
0
1
LD
Reg
A
INC
A≥B
B[7]
start
Control
Unit
C1
C2
C3
C4
C5
C6
C7
done

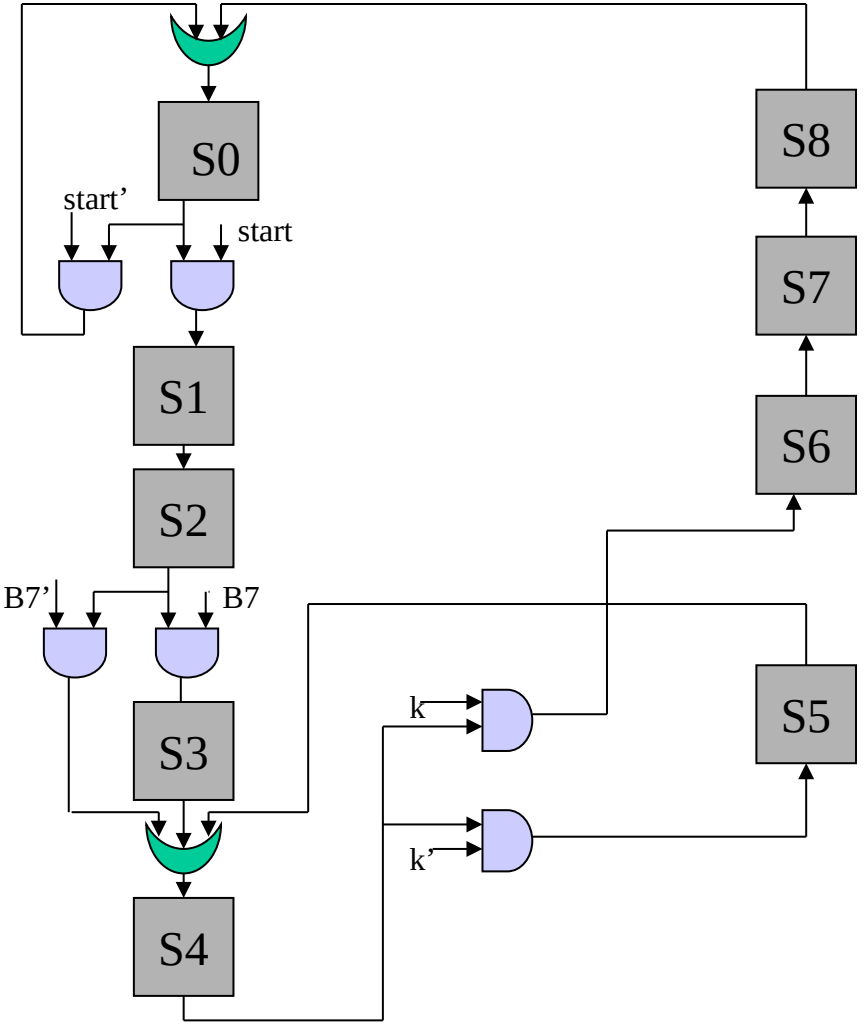
Step 6: Map Control Signals to Operations
Step 7: Identify Control Path Components

S0: If start', goto S0, else goto S1 || done←1
S1: A ← X || B ← Y || done ← 0 || goto S2
S2: If B'<7> goto S4, else goto S3
S3: B ← CS (B) || goto S4
S4: If k goto S6, else goto S5
S5: A ← INC (A) || B ← DEC (B) || goto S4
S6: Z ← A || goto S7
S7: Z ← SHL (z) || goto S8
S8: Z ← SHL (z) || done ← 1 ||goto S0

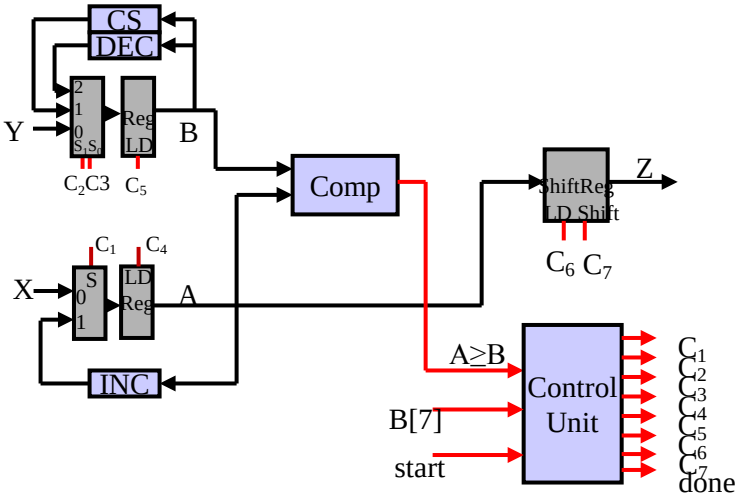


One-Hot State Machine





S0: If start', goto S0, else goto S1 || done<=1
S1: A ← X || B ← Y || done ← 0 || goto S2
S2: If B'<7> goto S4, else goto S3
S3: B ← CS (B) ||goto S4
S4: If k goto S6, else goto S5
S5: A ← INC (A) || B ← DEC (B) || goto S4
S6: Z ← A || goto S7
S7: Z ← SHL (z) || goto S8
S8: Z ← SHL (z) || done<=1 || goto S0



	C1	C2	C3	C4 A	C5 B	C6 Z	C7	done
S0				0	0	0	0	1
S1				1	1	0	0	0
S2				0	0	0	0	0
S3				0	1	0	0	0
S4				0	0	0	0	0
S5				1	1	0	0	0
S6				0	0	1	0	0
S7				0	0	0	1	0
S8				0	0	0	1	1

Summary

- Hardware Allocation
  - Balance between cost and performance
- Resource Sharing and Binding
  - Map operations to hardware
- Interconnect Synthesis
  - Convey signal transports
- Operation Scheduling
  - Sequence the process

Remarks:

1.Implement the control subsystem with one-hot state machine design.

2.Try to reduce the latency of the whole system.