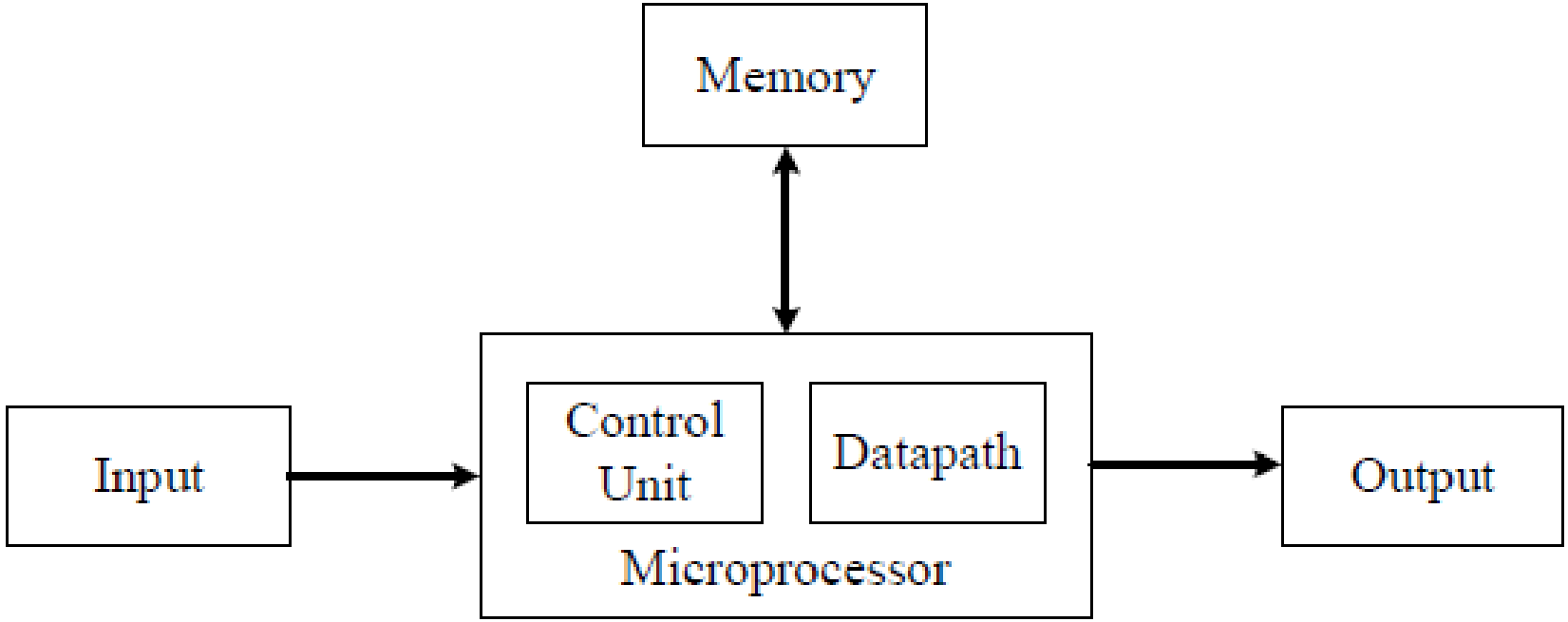
ELECTRICAL
Engineering KMITNB

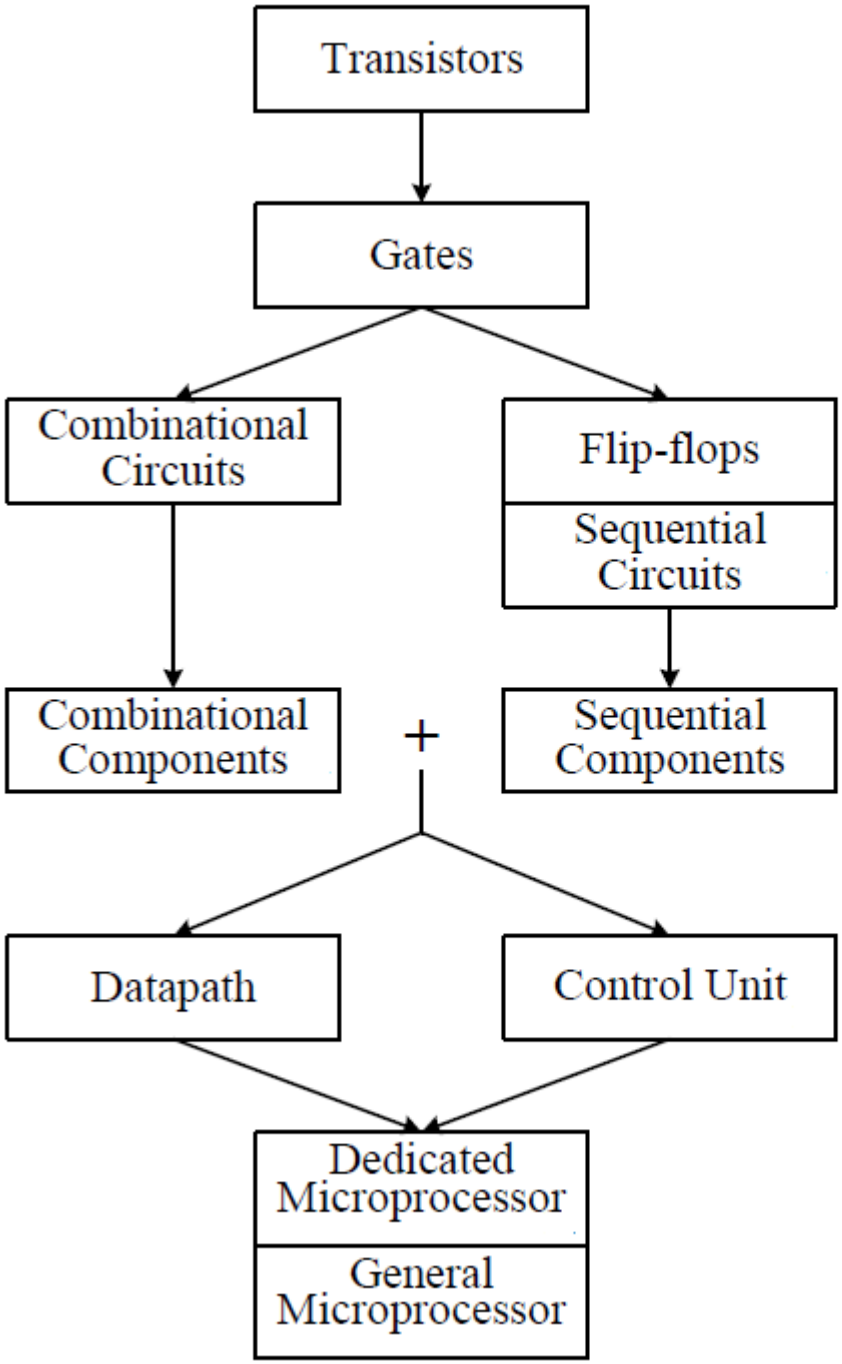
Digital Circuits and Logic Design

Lecture 9: Dedicated Microprocessors

Model of a Computer



Von Neumann model of a computer

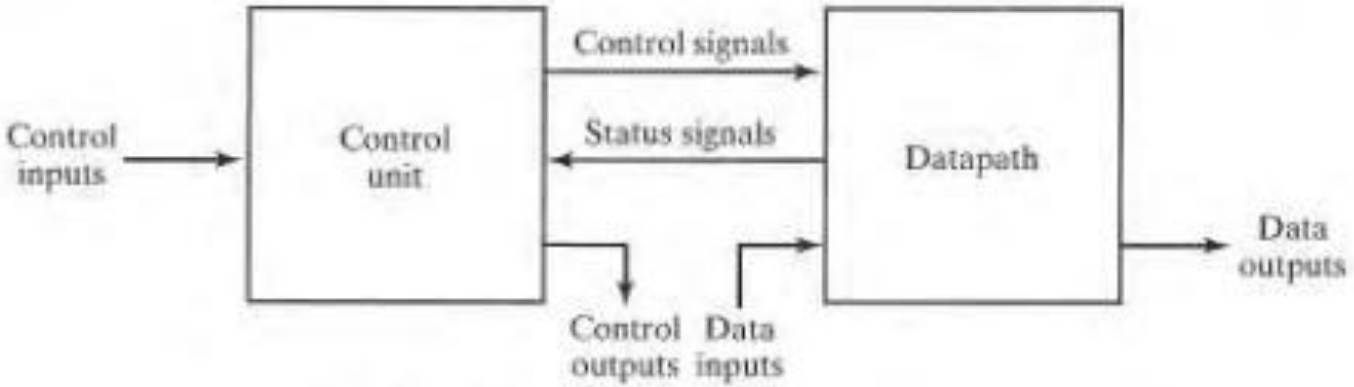


Summary of how the parts of microprocessor fit together

Dedicated Microprocessors

Dedicated microprocessors, also known as Application Specific Integrated Circuits (ASICs), on the other hand, are dedicated to performing only one task. The instructions for performing that one task are, therefore, hardwired into the processor itself, and once manufactured, cannot be modified again.

Datapath and Control Unit

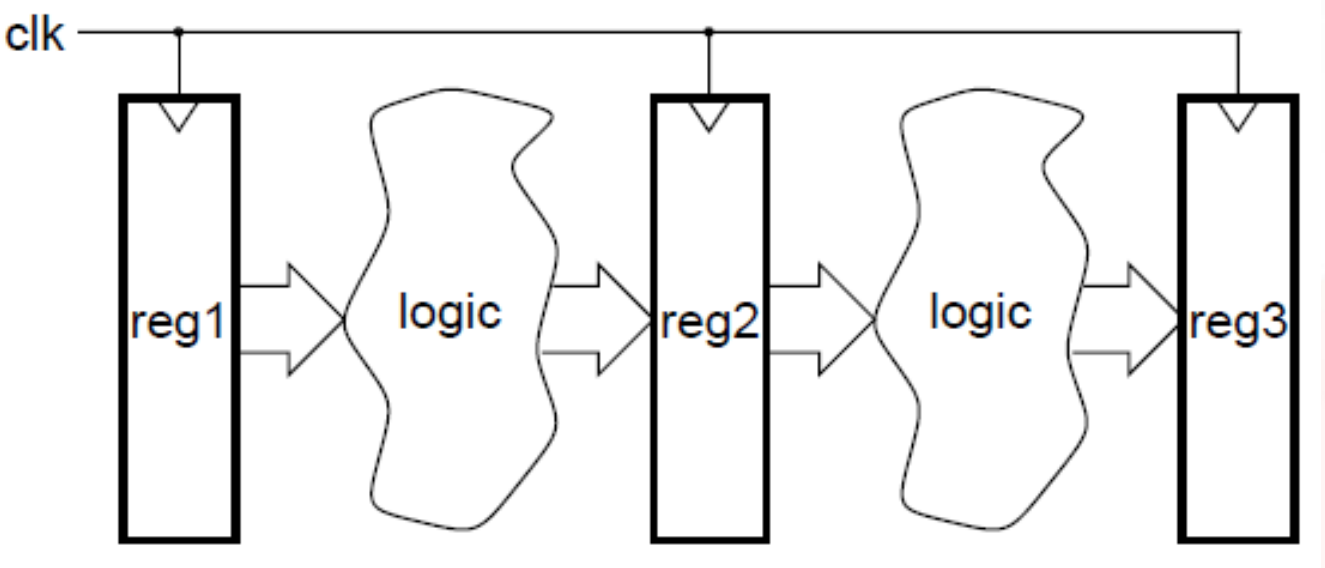


Datapath – Data processing operation
Control Unit –Determines the sequence of datapath operation

Register Transfer Level (RTL)

- *register transfer level (RTL)* is a level of abstraction used in describing the operation of a synchronous digital circuit. In RTL design, a circuit's behavior is defined in terms of the flow of signals (*or transfer of data*) between hardware registers, and the logical operations performed on those signals

The RTL Model



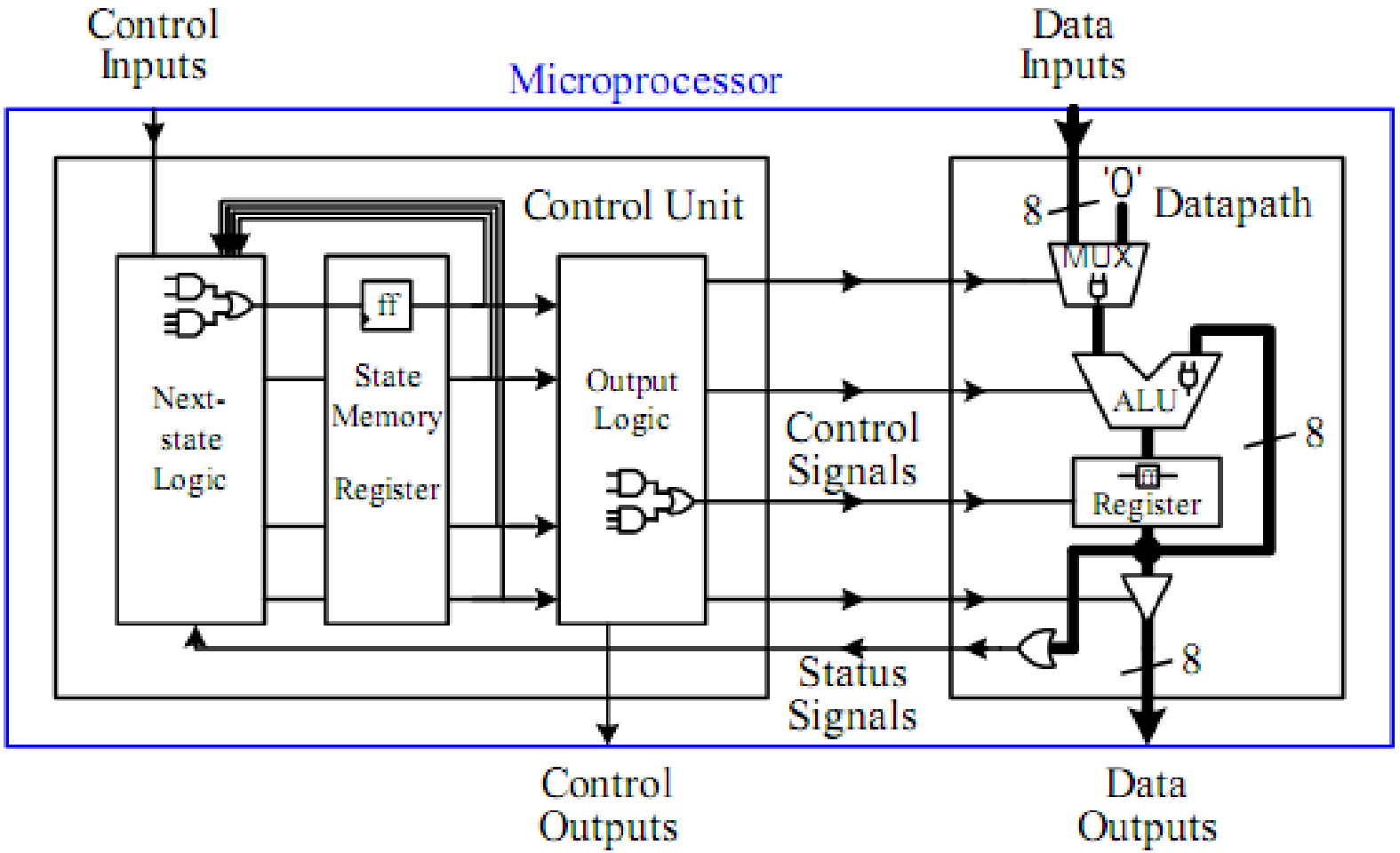
- In the register transfer level model we split the complete system state up into **registers and** consider the flow of information ‘in bulk’ from one register to the next on each clock tick

ELECTRICAL
Engineering KMITNB

RTL and HDL

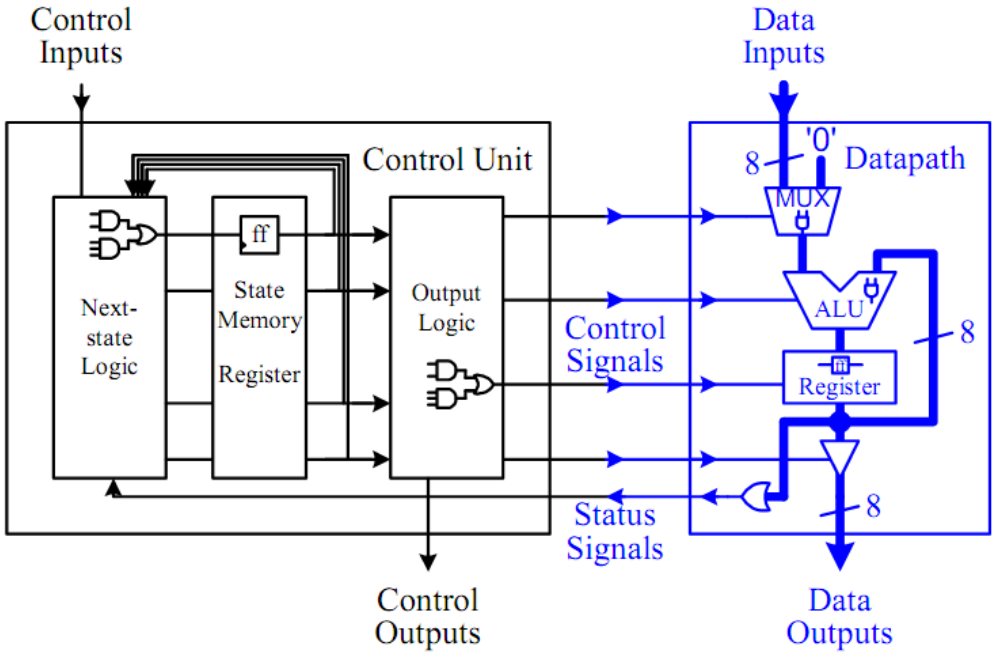
- Register transfer level abstraction is used in hardware description languages (HDLs) like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

Basic Internal Parts of Microprocessor

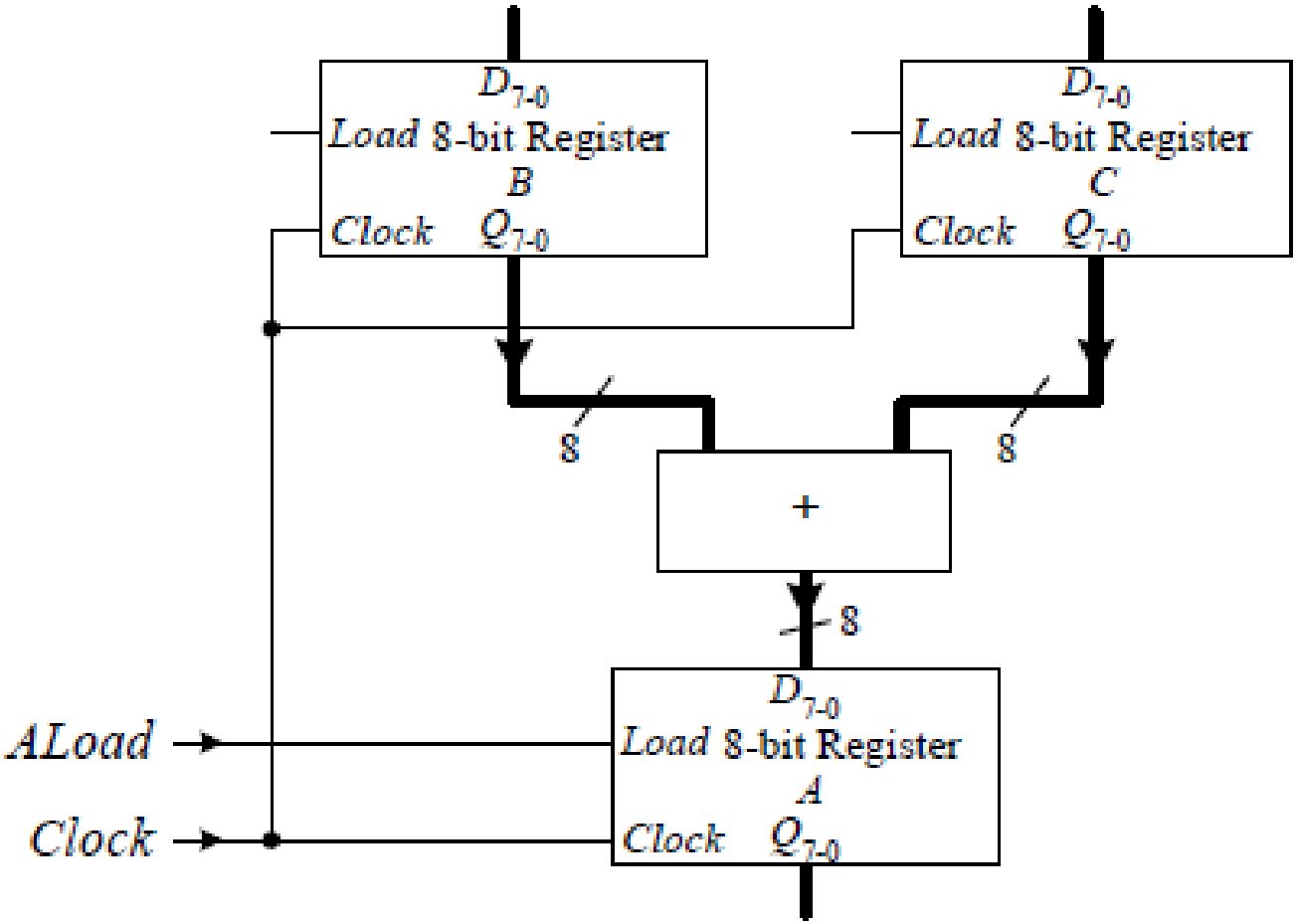


Designing Dedicated Datapaths

- The goal for designing a dedicated datapath is to build a circuit for solving a single specific problem

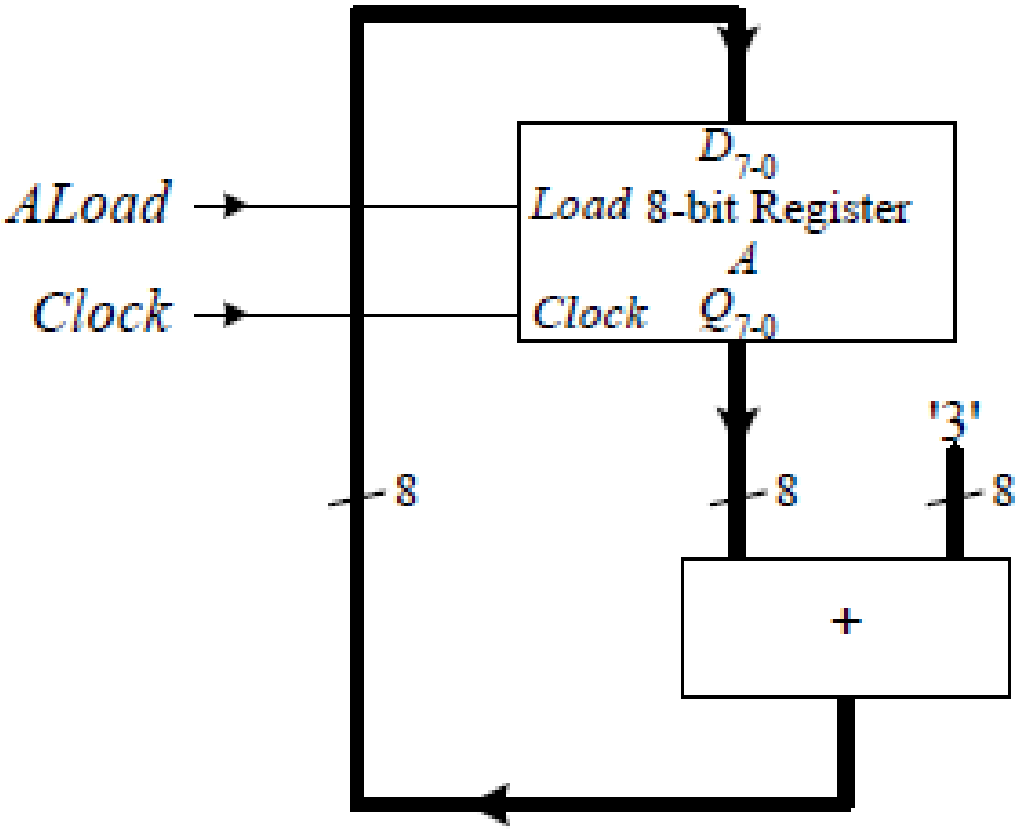


Sample Dedicated Datapath

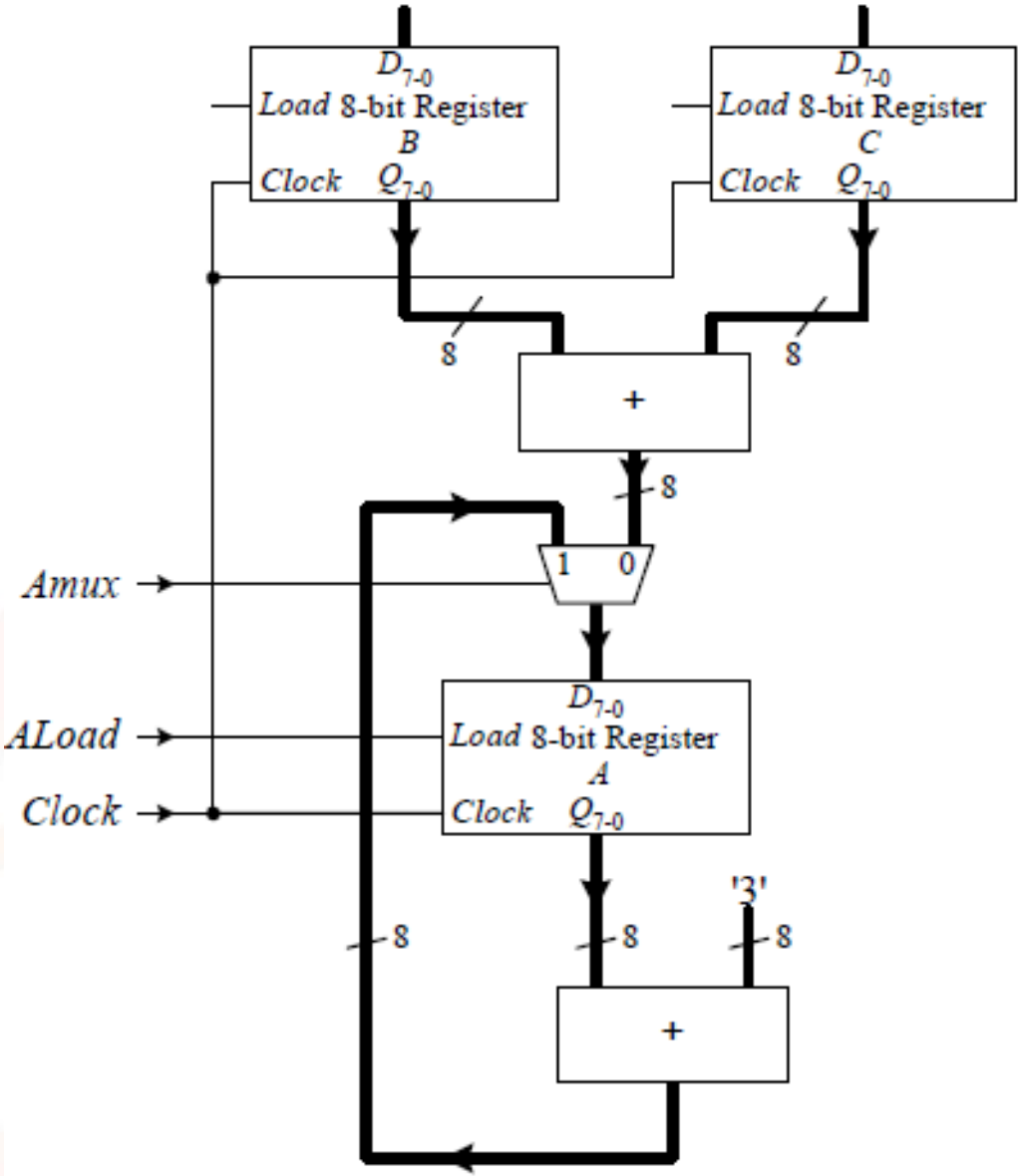


for performing $A = B + C$.

Sample Dedicated Datapath

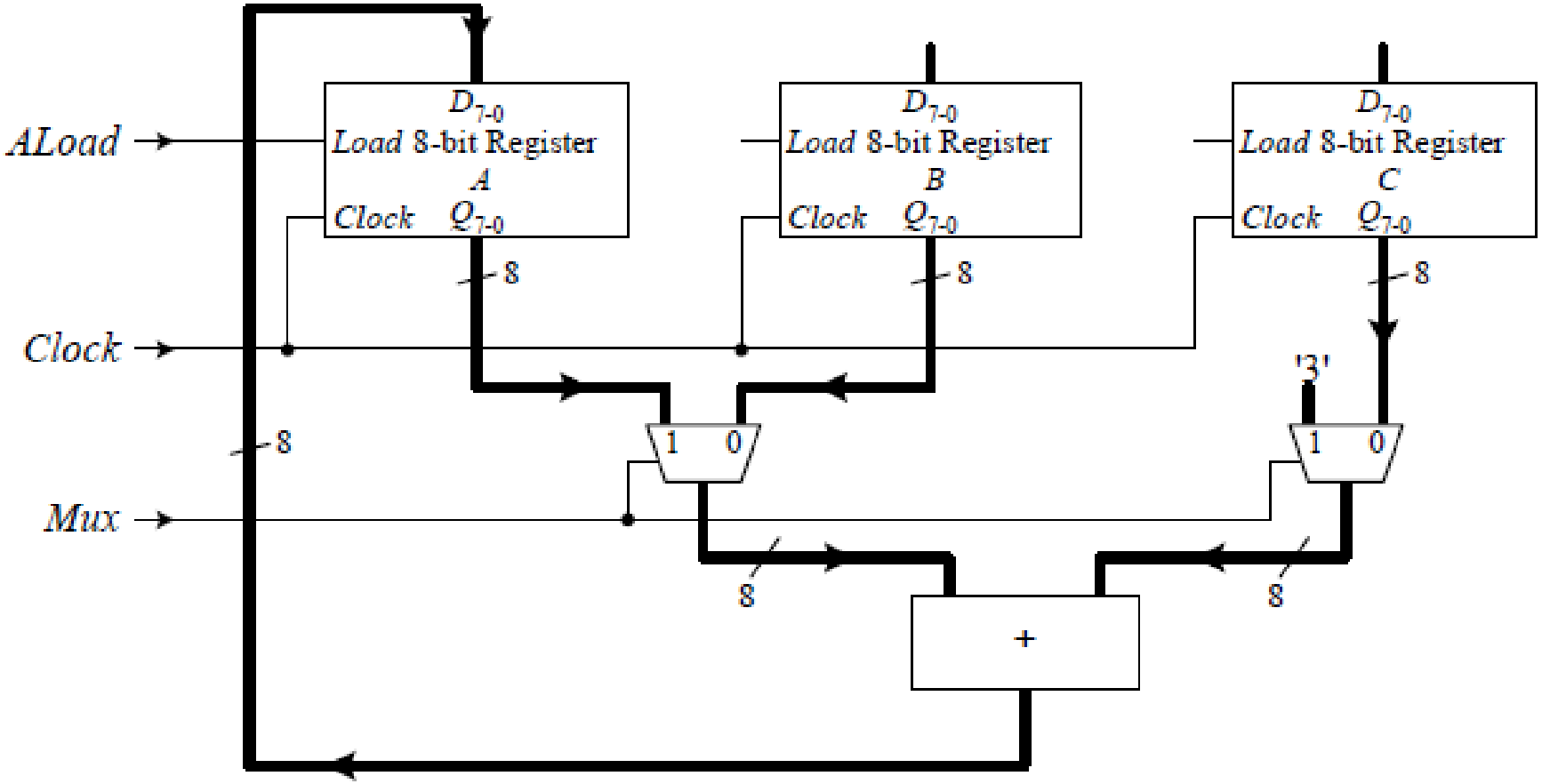


for performing $A = A + 3$



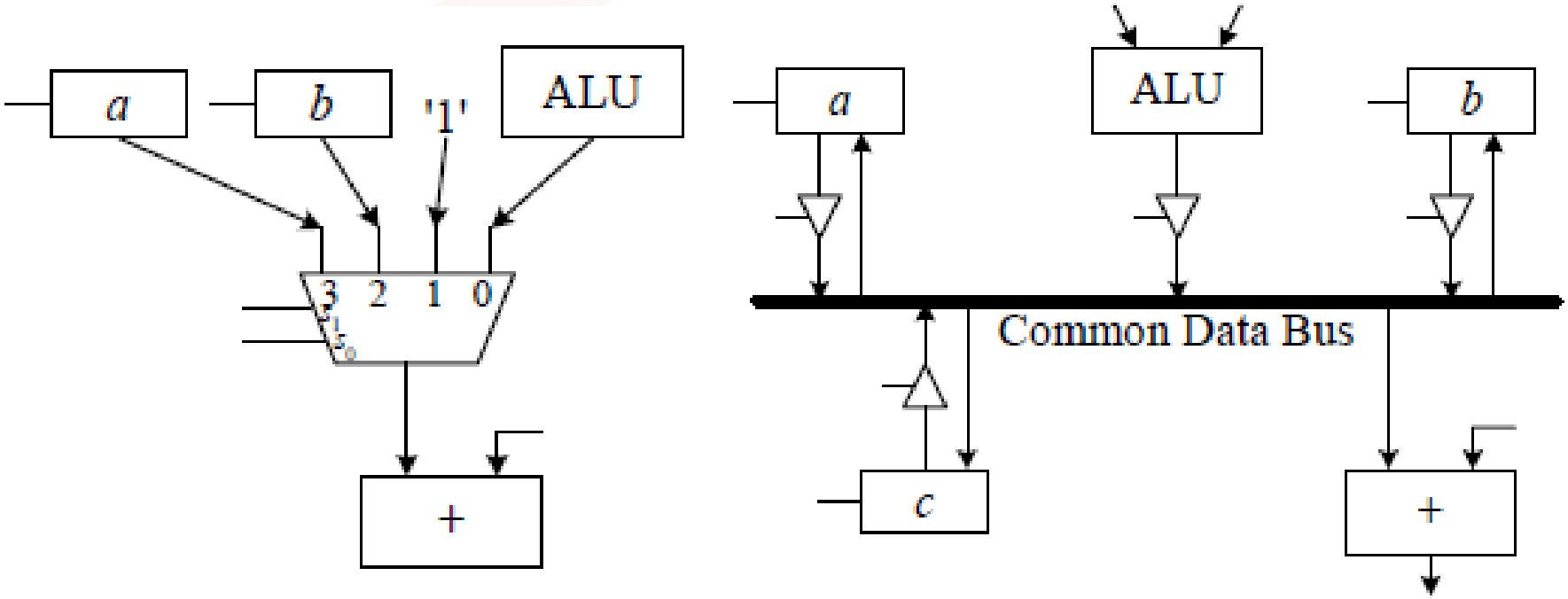
Datapath for performing $A = A + 3$ and $A = B + C$

Datapath



Datapath for performing A=A+3 and A=B+C using only one adder.

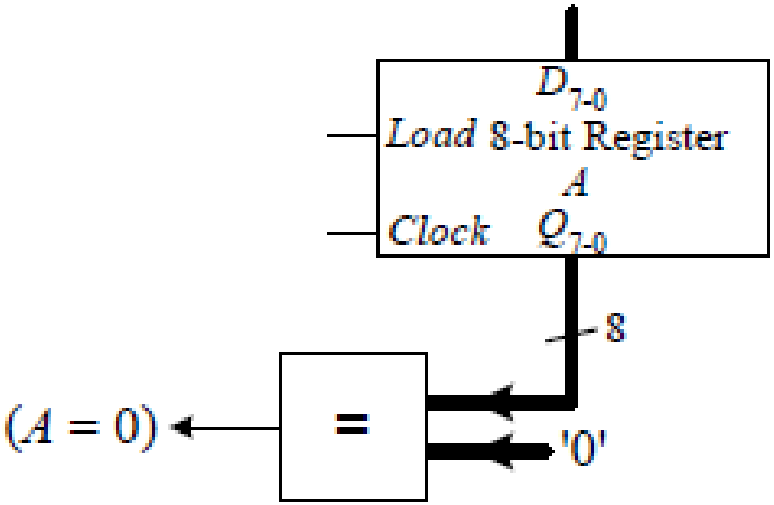
Multiple Sources



Datapath for IF-THEN

$$\text{IF } (A = 0) \text{ THEN } \ldots$$

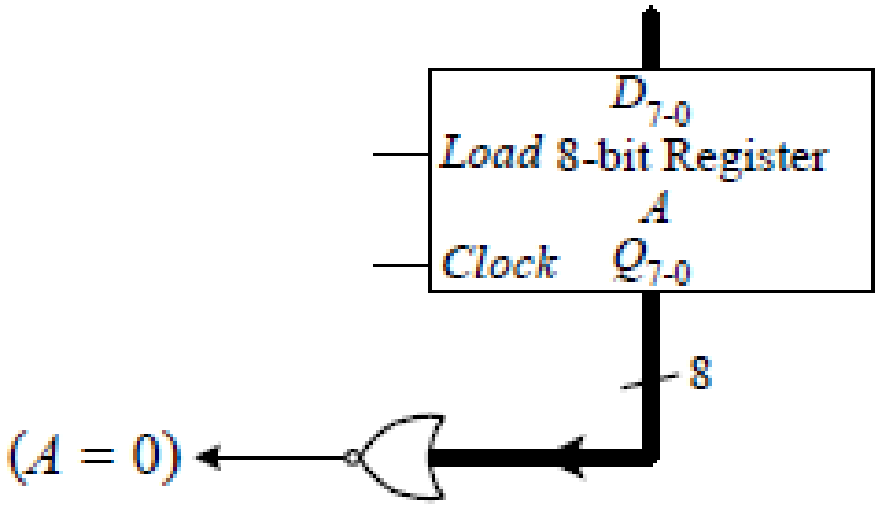
the datapath must, therefore, have an equality comparator that compares the value from the *A register with the* constant 0, as shown in figure below.



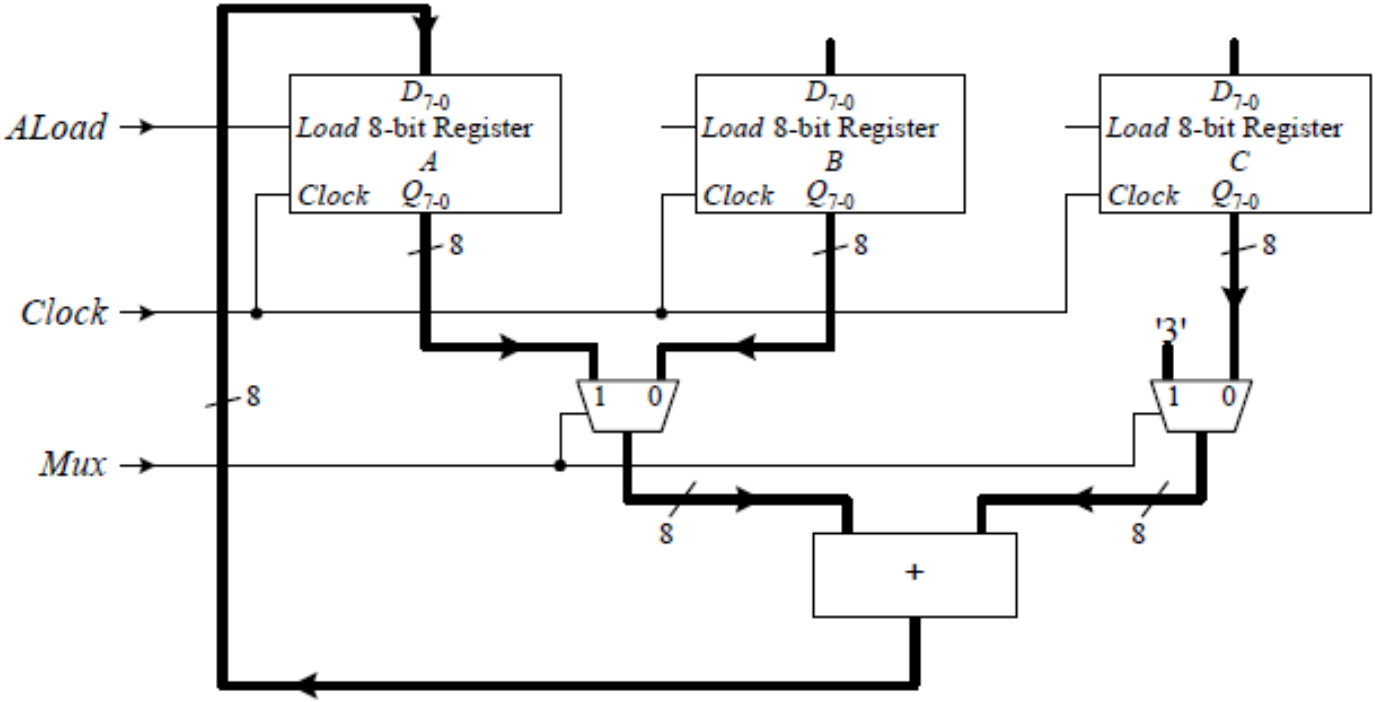
Datapath for IF-THEN

$$\text{IF } (A = 0) \text{ THEN } \ldots$$

The circuit for the equality comparator with the constant 0 is simply a NOR gate



Control Word Table

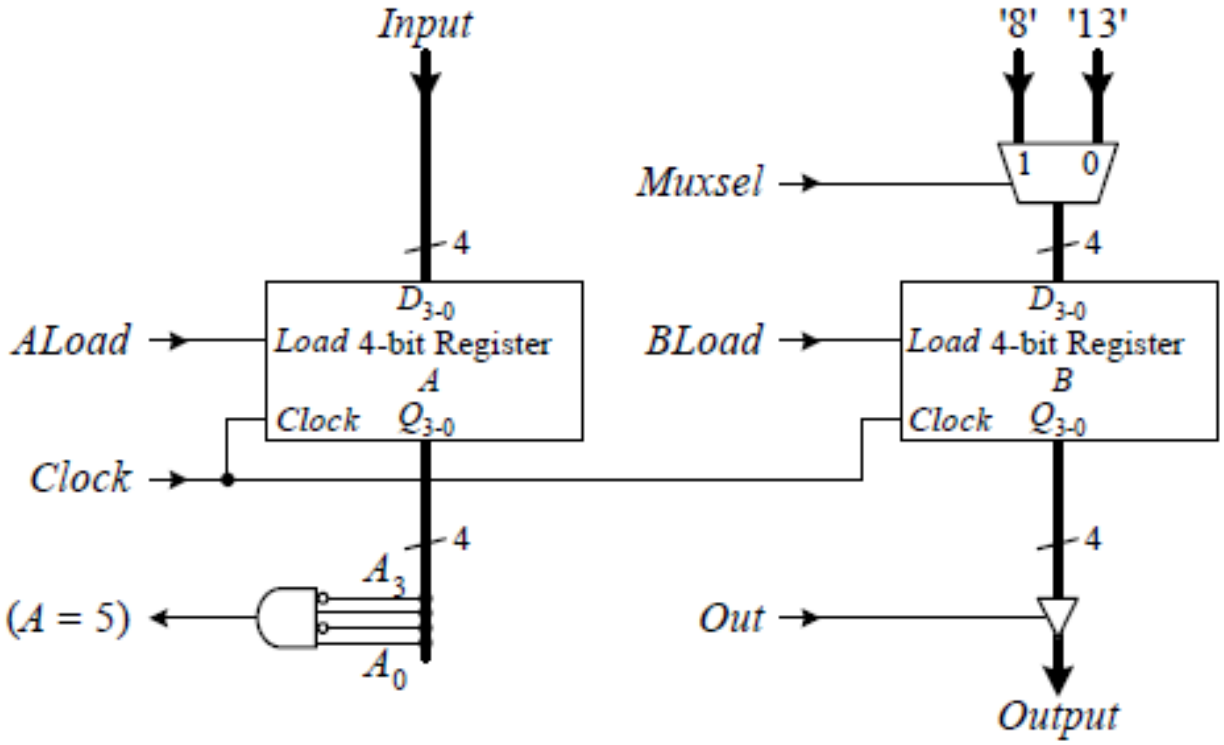


Control Word	Instruction	*ALoad*	*Mux*
1	$A = A + 3$	1	1
2	$A = B + C$	1	0

Simple if-then-else

```
1          INPUT A
2          IF (A = 5) THEN
3             B = 8
4          ELSE
5             B = 13
6          END IF
7          OUTPUT B
```

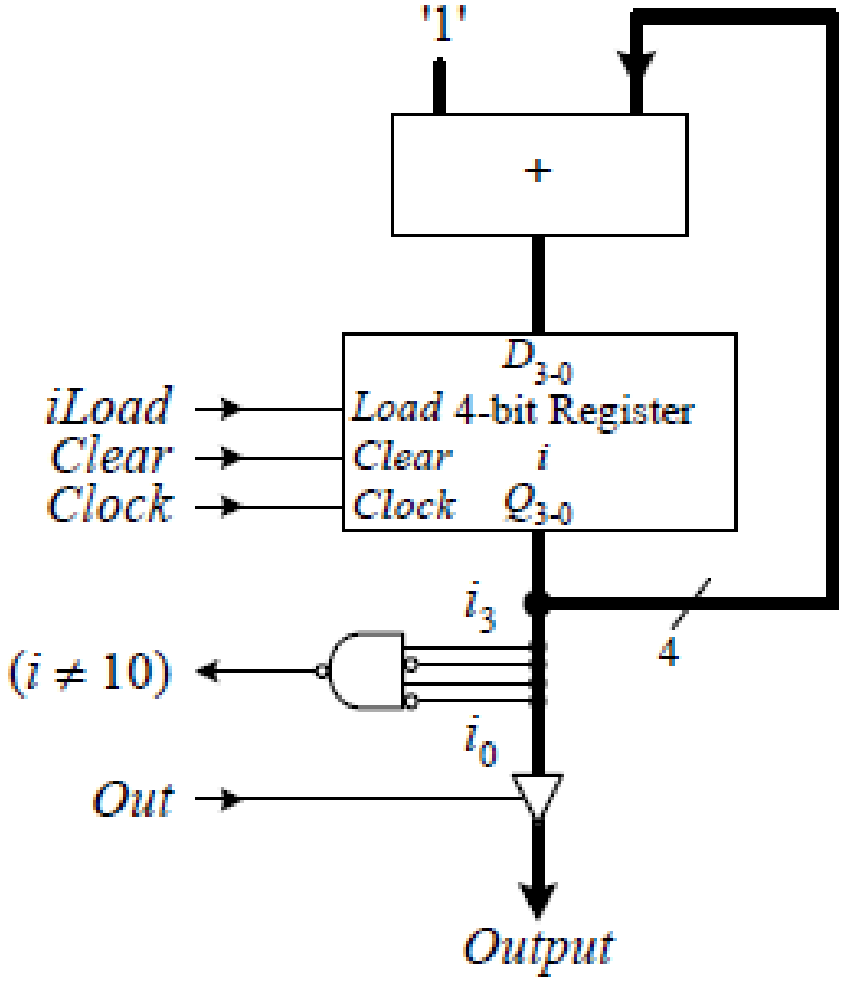
Control Word	Instruction	*ALoad*	*Muxsel*	*BLoad*	*Out*
1	INPUT *A*	1	×	0	0
2	*B* = 8	0	1	1	0
3	*B* = 13	0	0	1	0
4	OUTPUT *B*	0	×	0	1



Algorithm for Counting 1 to 10

```
1          i = 0
2          WHILE (i ≠ 10){
3             i = i + 1
4             OUTPUT i
5             }
```

Control Word	Instruction	*iLoad*	*Clear*	*Out*
1	$i = 0$	0	1	0
2	$i = i + 1$	1	0	0
3	OUTPUT i	0	0	1

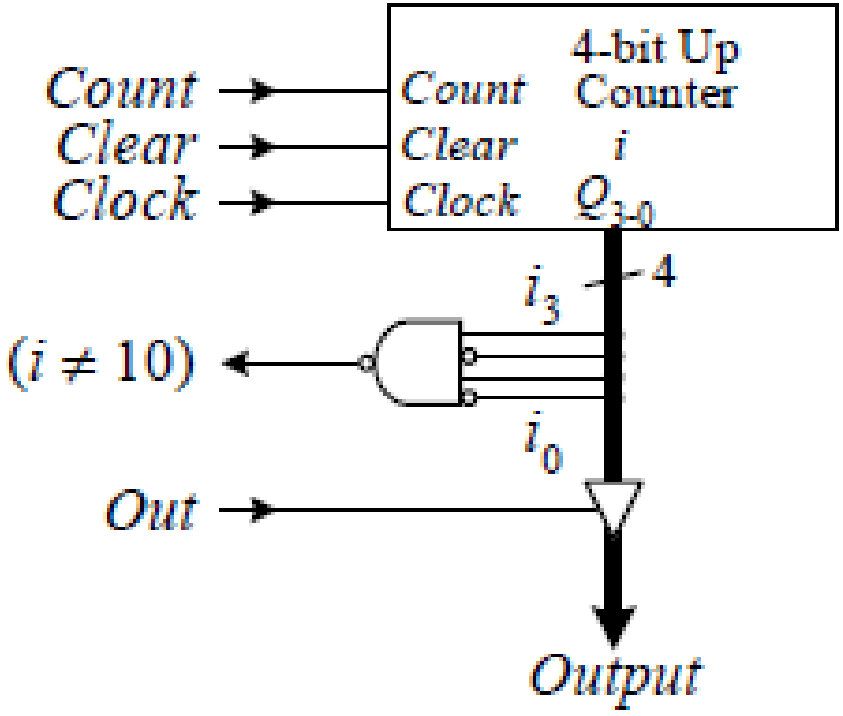


Dedicated datapath using a separate adder and control word

Algorithm for Counting 1 to 10

```
1          i = 0
2          WHILE (i ≠ 10){
3             i = i + 1
4             OUTPUT i
5             }
```

Control Word	Instruction	*Count*	*Clear*	*Out*
1	$i = 0$	0	1	0
2	$i = i + 1$	1	0	0
3	OUTPUT i	0	0	1



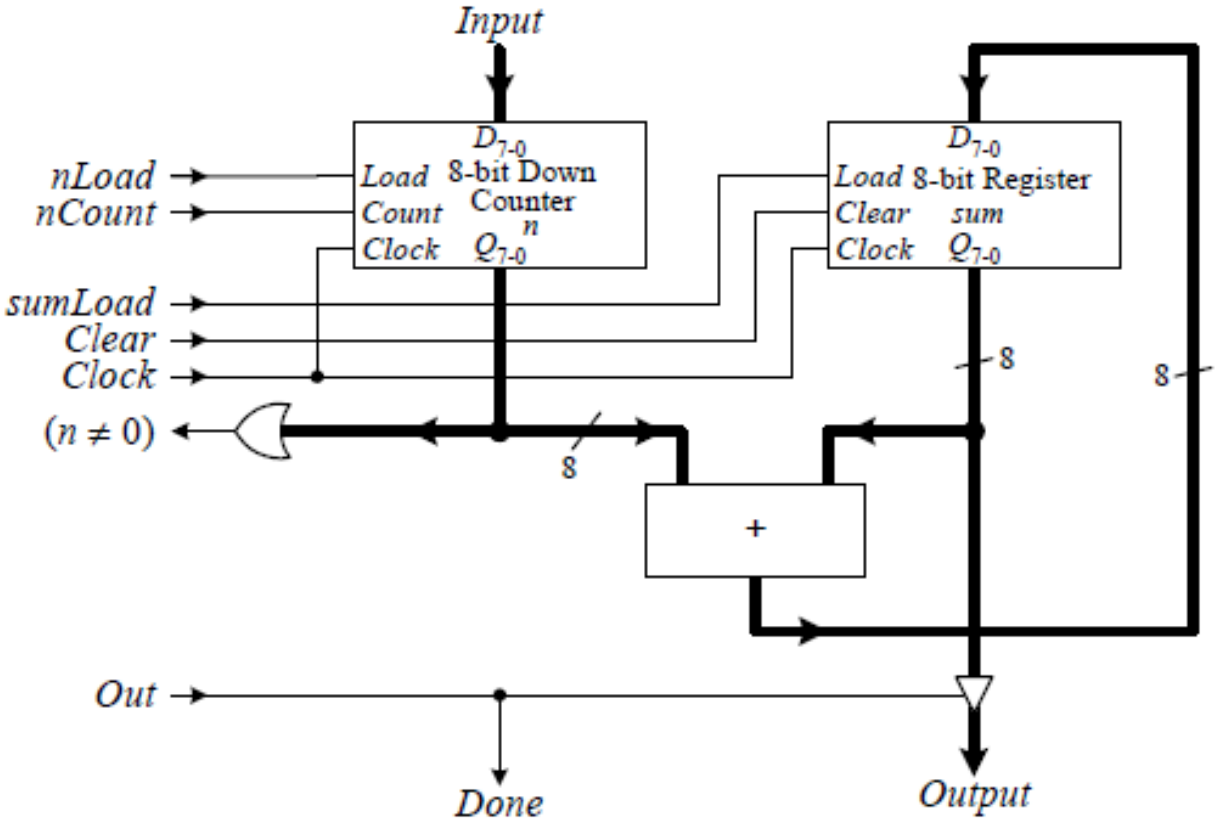
Dedicated datapath using a single up-counter and control word

Ex. Summation of *n Down to 1*

Construct an 8-bit dedicated datapath and control word to generate and add the numbers from *n down to 1, where n is an 8-bit* user input number.

```
1          sum = 0
2          INPUT n
3          WHILE (n ≠ 0){
4              sum = sum + n
5              n = n - 1
6          }
7          OUTPUT sum
```

Sol. Dedicated Datapath



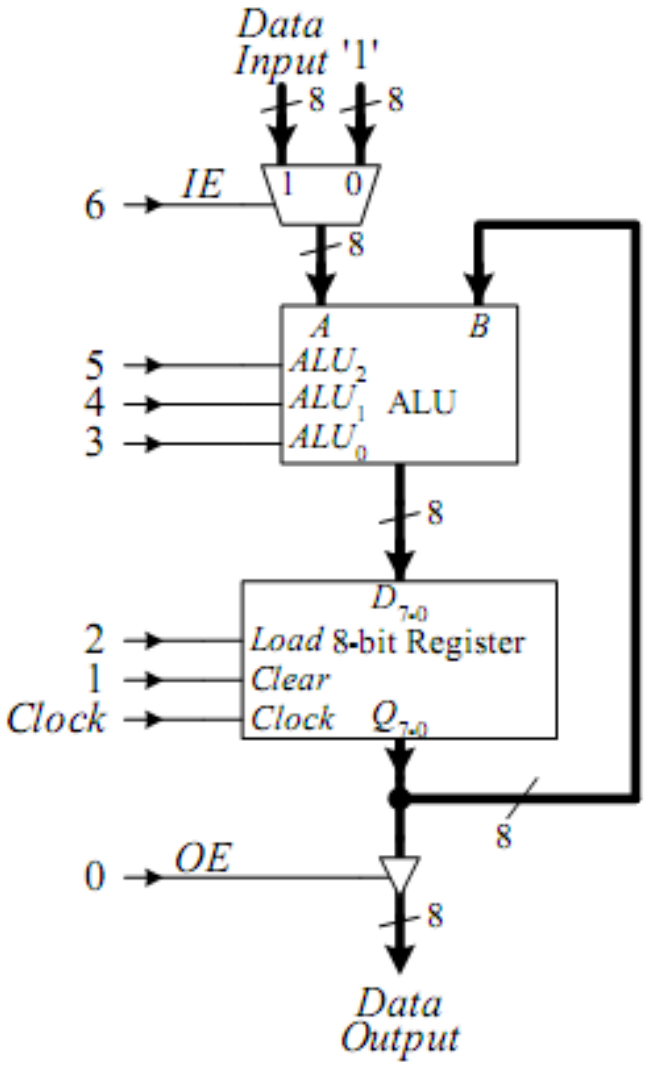
Control Word	Instruction	*nLoad*	*nCount*	*sumLoad*	*Clear*	*Out*
1	$sum = 0$	0	0	0	1	0
2	INPUT n	1	0	0	0	0
3	$sum = sum + n$	0	0	1	0	0
4	$n = n - 1$	0	1	0	0	0
5	OUTPUT *sum*	0	0	0	0	1

ELECTRICAL
Engineering KMITNB

General Datapaths

- a general datapath is more general than dedicated datapath in the sense that it can be used to solve various problems instead of just one specific problem, as long as it has all of the required functional units and enough registers for storing all of the temporary data.

General Datapaths

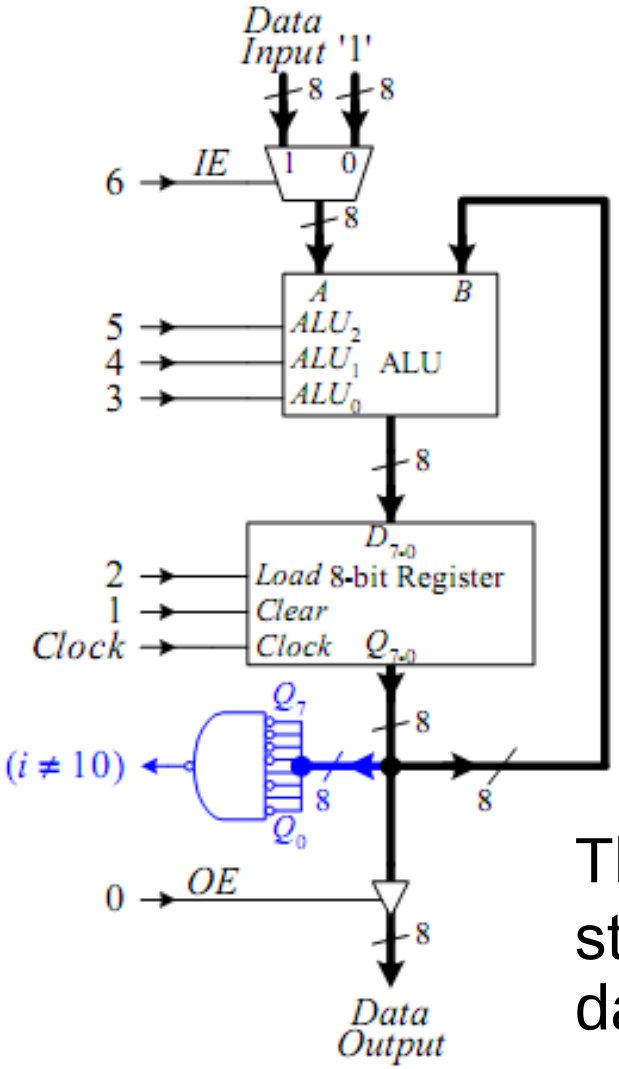


ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$

Control word	*IE* 6	ALU_2 5	ALU_1 4	ALU_0 3	*Load* 2	*Clear* 1	*OE* 0
Value	1	0	0	0	1	0	0

Simple General Datapath

Using General Datapaths




```
1          i = 0
2          WHILE (i ≠ 10){
3             i = i + 1
4             OUTPUT i
5             }
```

Control Word	Instruction	*IE* 6	$ALU_2\ ALU_1\ ALU_0$ 5–3	*Load* 2	*Clear* 1	*OE* 0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	OUTPUT *i*	×	×××	0	0	1

The comparator circuit for generating the status signal (i ≠ 10) added to the general datapath.

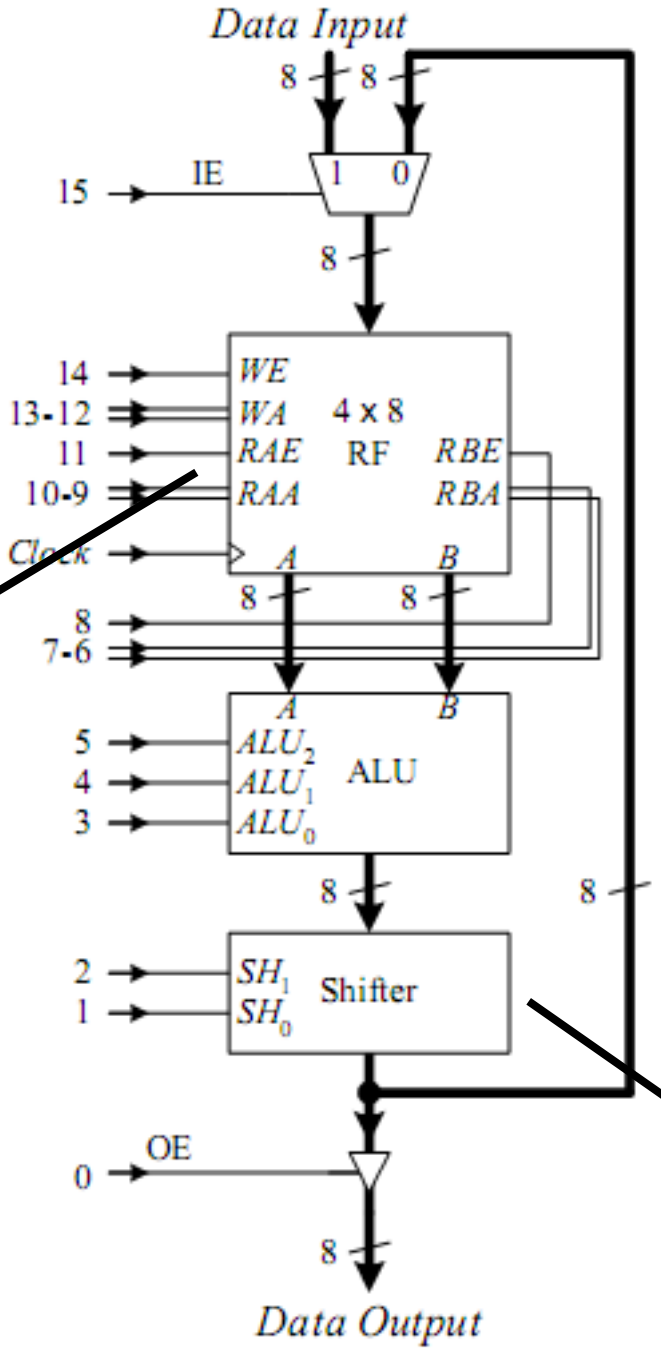
Using General Datapaths

- Just like for the dedicated datapath, the general datapath must also provide the status signal for the condition (i ≠ 10) to the control unit. Using this status signal, the control unit will determine whether to repeat control words 2 and 3 in the loop or to terminate.

Control Word	Instruction	*IE* 6	$ALU_2\ ALU_1\ ALU_0$ 5–3	*Load* 2	*Clear* 1	*OE* 0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	OUTPUT i	×	×××	0	0	1

A More Complex General Datapath

- When a particular general datapath does not contain all of the functional units and/or registers needed to perform all of the required operations specified in the algorithm that we are trying to implement,
- then we need to select a more complex datapath. When working with general datapaths, the goal is to find the simplest and smallest datapath that matches the requirements of the problem as closely as possible.



4x8 Register File

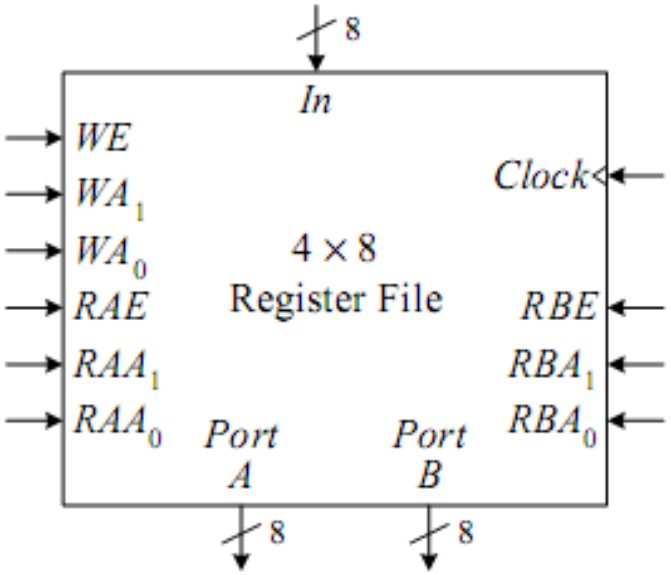
ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$

SH_1	SH_0	Operation
0	0	Pass through
0	1	Shift left and fill with 0
1	0	Shift right and fill with 0
1	1	Rotate right

Shift Register

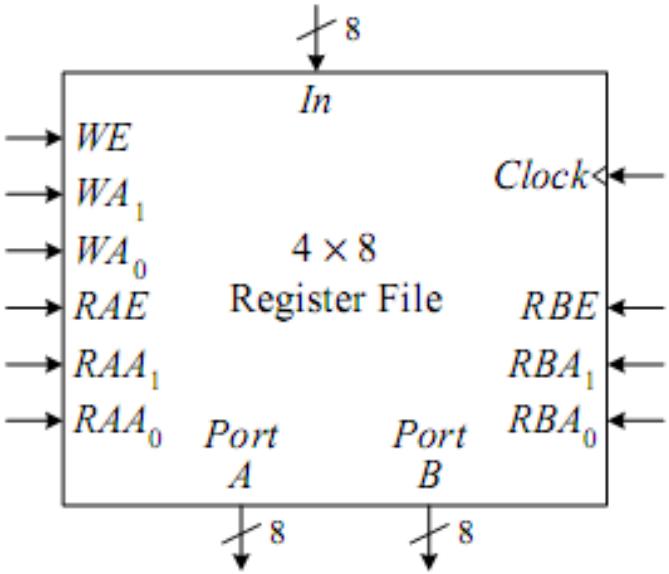
4X8 Register File

The logic symbol for a 4 × 8 register file (four registers, each being 8-bits wide) is shown in Figure below. The 8-bit write port is labeled In, and the two 8-bit read ports are labeled Port A and Port B. WE is the active-high write enable line. To write a value into the register file, this line must be asserted. The WA1 and WA0 are the two address lines for selecting the write location.

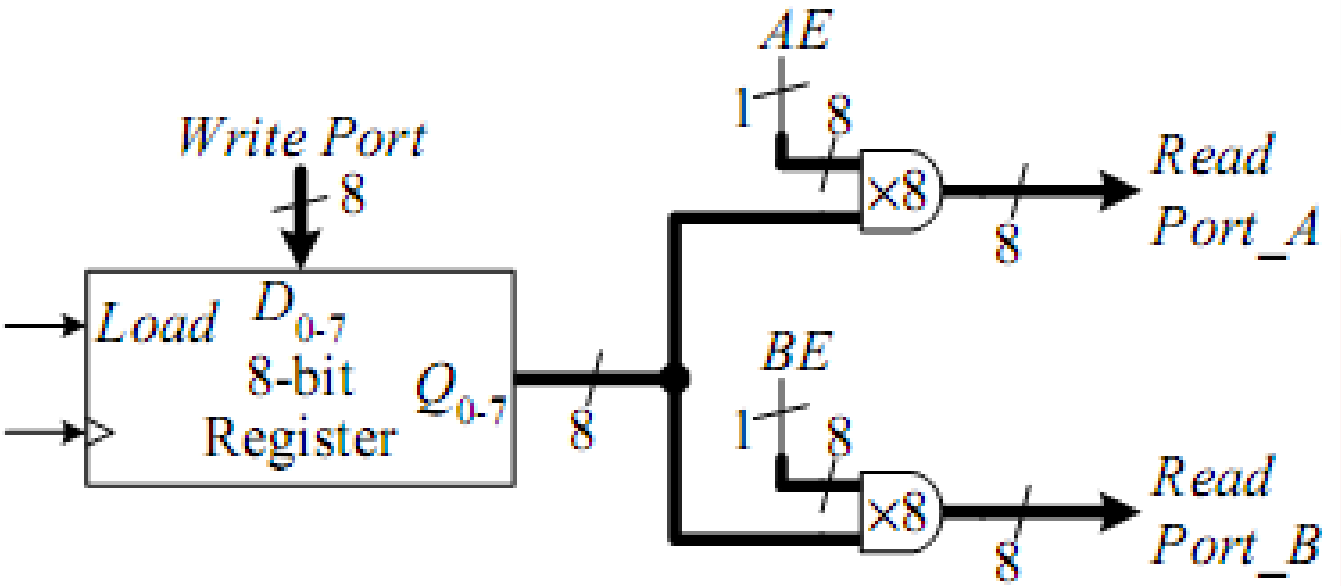


4X8 Register File (cont.)

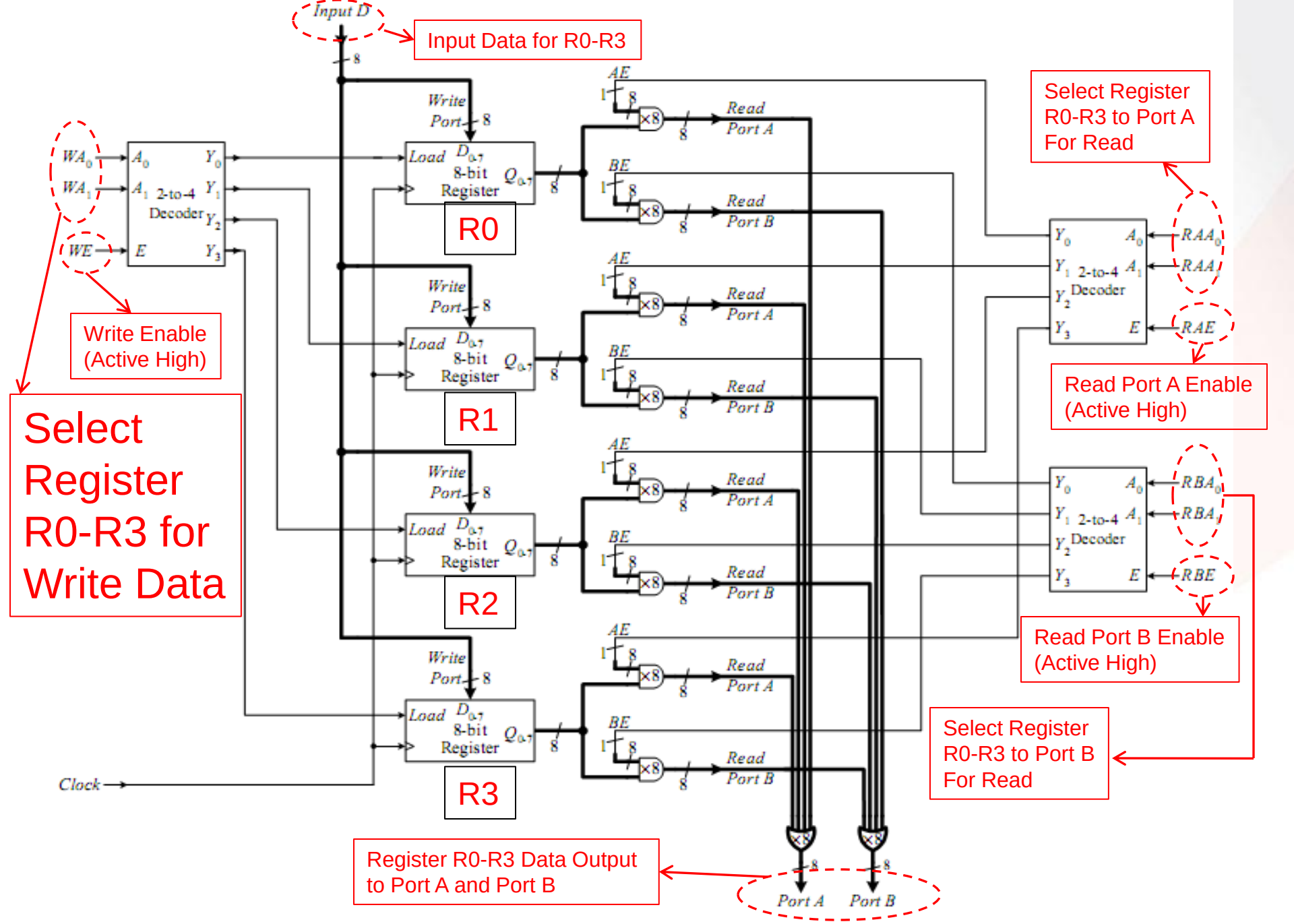
Since there are four locations in this register file, two address lines are needed. The RAE line is the read enable line for Port A. The two read address select lines for Port A are RAA1 and RAA0. For Port B, we have the Port B enable line, RBE, and the two address lines, RBA1 and RBA0.



4X8 Register File (cont.)

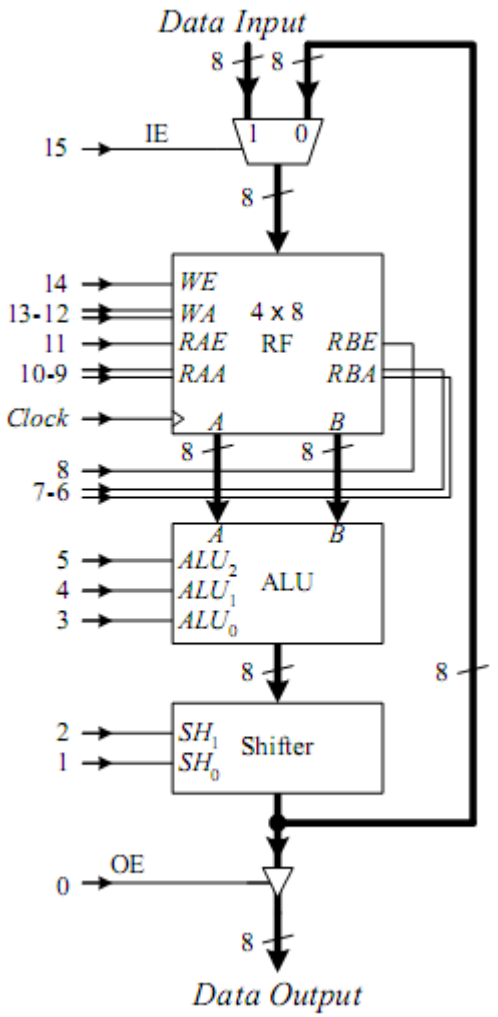


An 8-bit wide register file cell with one write port and two read ports


Input D
Input Data for R0-R3
Write Port
Load
8-bit Register
R0
R1
R2
R3
AE
BE
Read Port A
Read Port B
WA0
WA1
WE
2-to-4 Decoder
Write Enable (Active High)
Select Register R0-R3 for Write Data
Clock
Select Register R0-R3 to Port A For Read
RAA0
RAA1
RAE
Read Port A Enable (Active High)
RBA0
RBA1
RBE
Read Port B Enable (Active High)
Select Register R0-R3 to Port B For Read
Register R0-R3 Data Output to Port A and Port B
Port A
Port B

Sum the numbers from n down to 1.

```
1          sum = 0
2          INPUT n
3          WHILE (n ≠ 0){
4              sum = sum + n
5              n = n - 1
6          }
7          OUTPUT sum
```



Control Word	Instruction	*IE* 15	*WE* 14	$WA_{1,0}$ 13–12	*RAE* 11	$RAA_{1,0}$ 10–9	*RBE* 8	$RBA_{1,0}$ 7–6	$ALU_{2,1,0}$ 5–3	$SH_{1,0}$ 2–1	*OE* 0
1	*sum* = 0	0	1	00	1	00	1	00	101 (subtract)	00	0
2	INPUT *n*	1	1	01	0	××	0	××	×××	××	0
3	*sum* = *sum* + *n*	0	1	00	1	00	1	01	100 (add)	00	0
4	*n* = *n* – 1	0	1	01	1	01	0	××	111 (decrement)	00	0
5	OUTPUT *sum*	×	0	××	1	00	0	××	000 (pass)	00	1

ELECTRICAL
Engineering KMITNB

Ex1.

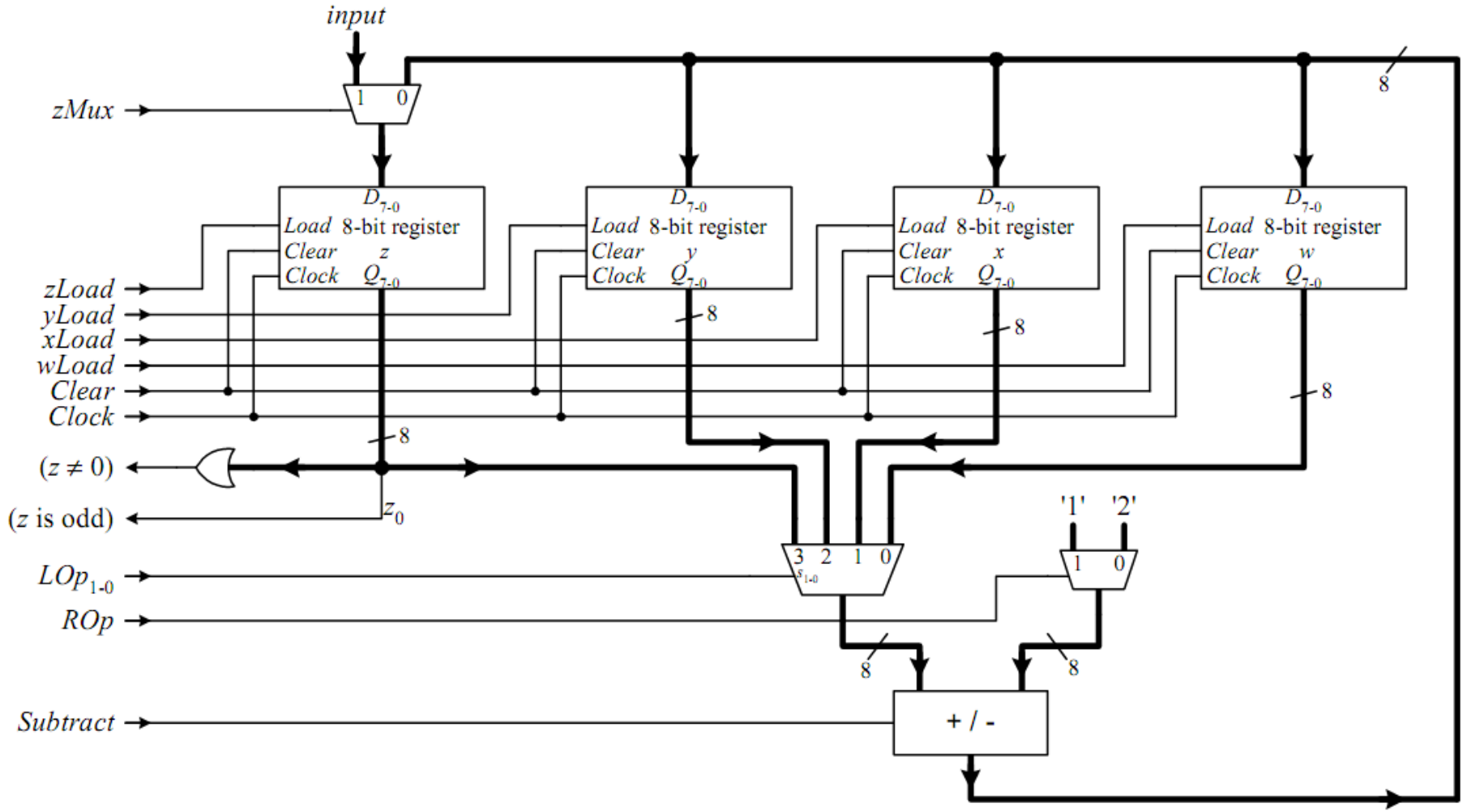
Design an 8-bit dedicated datapath for the algorithm in Figure P9.29, and write the control words for it. Use only one adder-subtractor unit for all the addition and subtraction operations. Label clearly all of the control and status signals.

```
w = 0
x = 0
y = 0
INPUT z
WHILE (z ≠ 0) {
  w = w - 2
  IF (z is an odd number) THEN
     x = x + 2
  ELSE
     y = y + 1
  END IF
  z = z - 1
  }
```

Figure P9.29.

ELECTRICAL
Engineering KMITNB

Sol1.

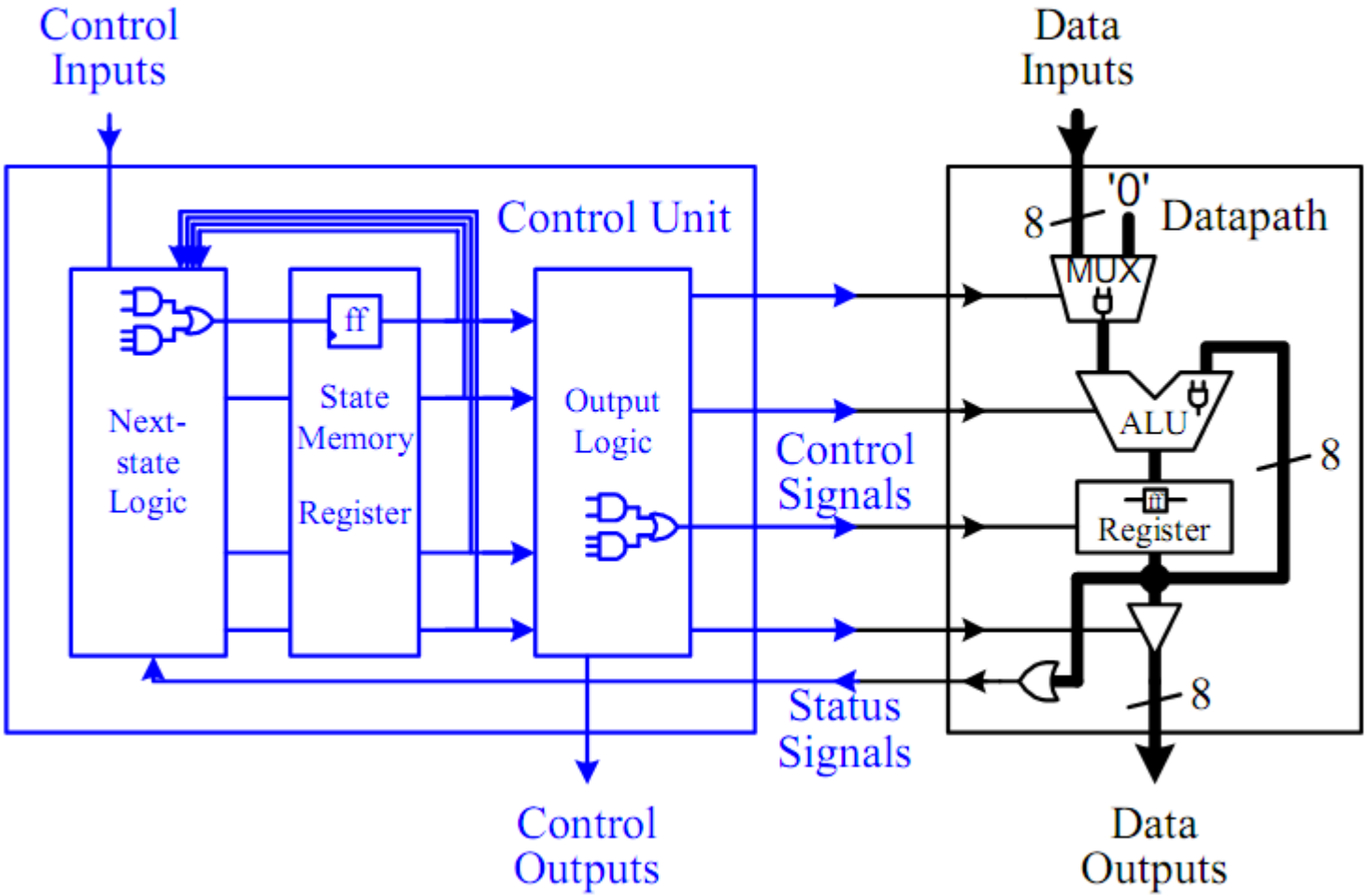


H/W: Count 0's and 1's

- **Construct a 8-bit dedicated datapath and control word for solving the following problem:**

Input an 8-bit number. Output a 1 if the number has the same number of 0's and 1's, otherwise, output a 0. (E.g., the number 10111011 will output a 0; whereas, the number 10100011 will output a 1.)

```
INPUT n
countbit = 0              // for counting the number of zero and one bits
counteight = 0            // for looping eight times
WHILE (counteight ≠ 8) {
   IF (n0 = 1) THEN       // test whether bit 0 of n is a 1
      countbit = countbit + 1
   ELSE
      countbit = countbit - 1
   END IF
   n = n >> 1             // shift n right one bit
   counteight = counteight + 1;
   }
IF (countbit = 0) THEN
   OUTPUT 1
ELSE
   OUTPUT 0
END IF
ASSERT Done
```


Control Inputs
Data Inputs
Control Unit
Next-state Logic
ff
State Memory Register
Output Logic
Control Signals
Status Signals
Control Outputs
'0'
8
Datapath
MUX
ALU
Register
8
8
Data Outputs

ELECTRICAL
Engineering KMITNB

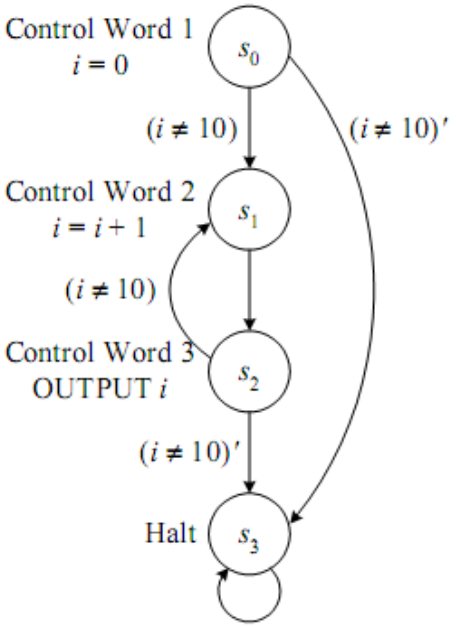
Control Unit

- The control unit inside the microprocessor is a finite state machine. By stepping through a sequence of states,
- the control unit controls the operations of the datapath. For each state that the control unit is in, the output logic that is inside the control unit will generate all of the appropriate control signals for the datapath to perform one data operation.

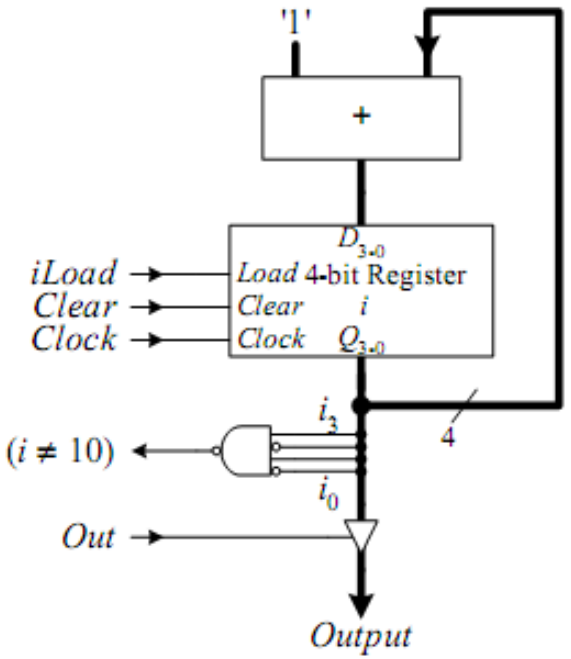
Ex: Counting 1 to 10

Design Control Unit

```
1          i = 0
2          WHILE (i ≠ 10){
3             i = i + 1
4             OUTPUT i
5          }
```



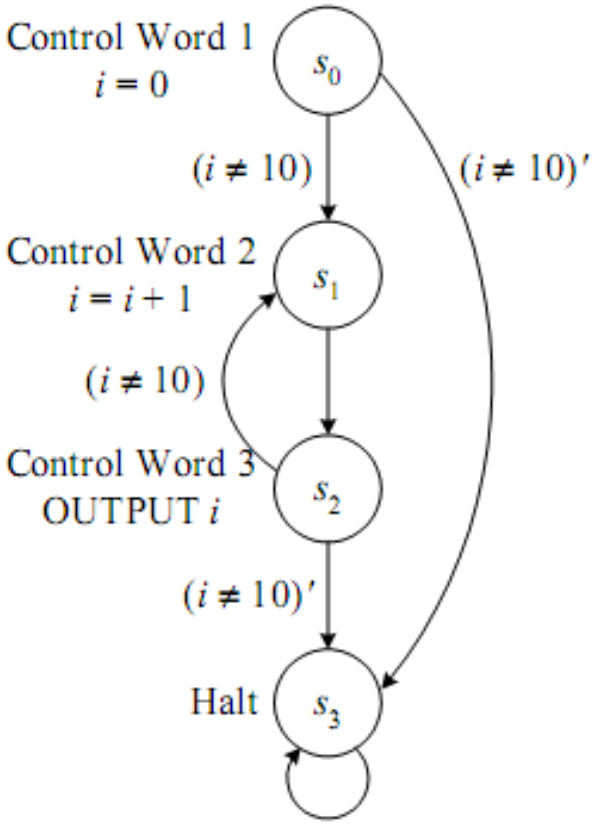
(a)



(b)

Control Word	Instruction	*iLoad*	*Clear*	*Out*
1	$i = 0$	0	1	0
2	$i = i + 1$	1	0	0
3	OUTPUT i	0	0	1

Design Control Unit



(a)

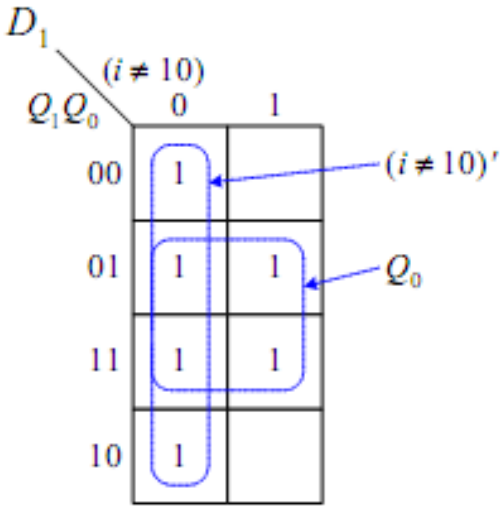
Current State Q_1Q_0	Next State $Q_{1next}\ Q_{0next}$	
	$(i \neq 10)'$	$(i \neq 10)$
s_0 00	s_3 11	s_1 01
s_1 01	s_2 10	s_2 10
s_2 10	s_3 11	s_1 01
s_3 11	s_3 11	s_3 11

(b)

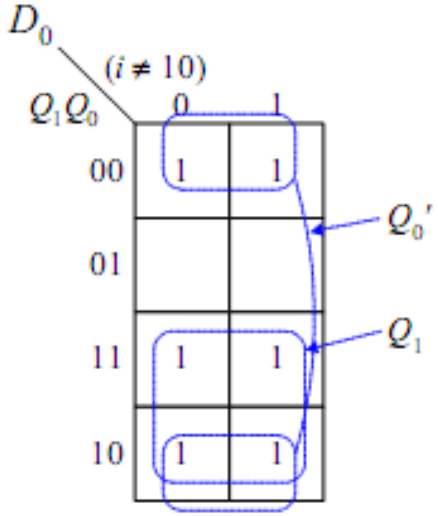
Current State Q_1Q_0	Implementation $D_1\ D_0$	
	$(i \neq 10)'$	$(i \neq 10)$
00	11	01
01	10	10
10	11	01
11	11	11

(c)

Design Control Unit



$D_1 = (i \neq 10)' + Q_0$



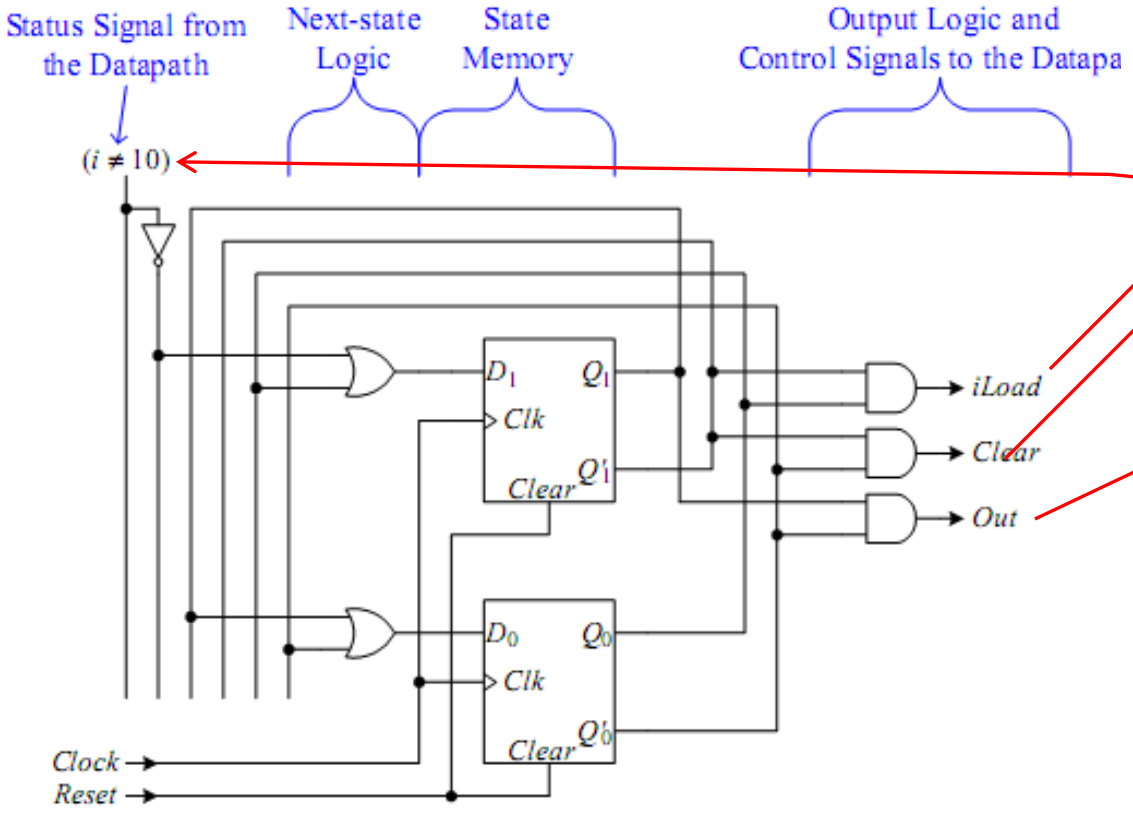
$D_0 = Q_1 + Q_0'$

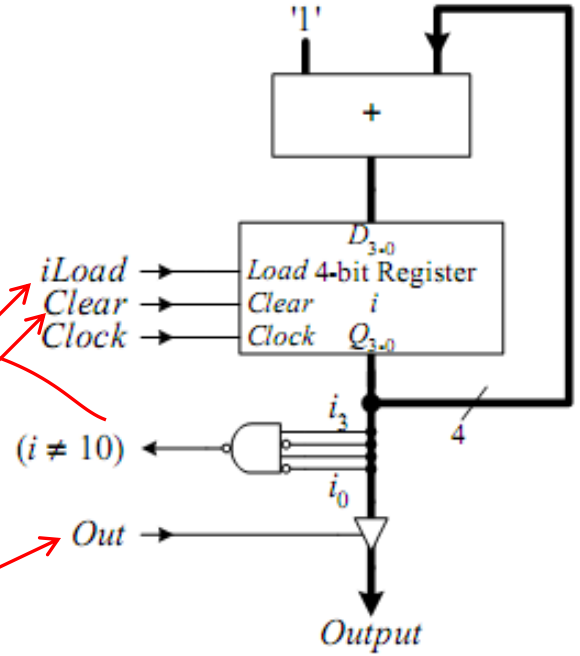
(d)

Q_1Q_0	*iLoad*	*Clear*	*Out*
00	0	1	0
01	1	0	0
10	0	0	1
11	0	0	0

$iLoad = Q_1'Q_0$
$Clear = Q_1'Q_0'$
$Out = Q_1Q_0'$

Dedicated Microprocessor for Counting 1 to 10





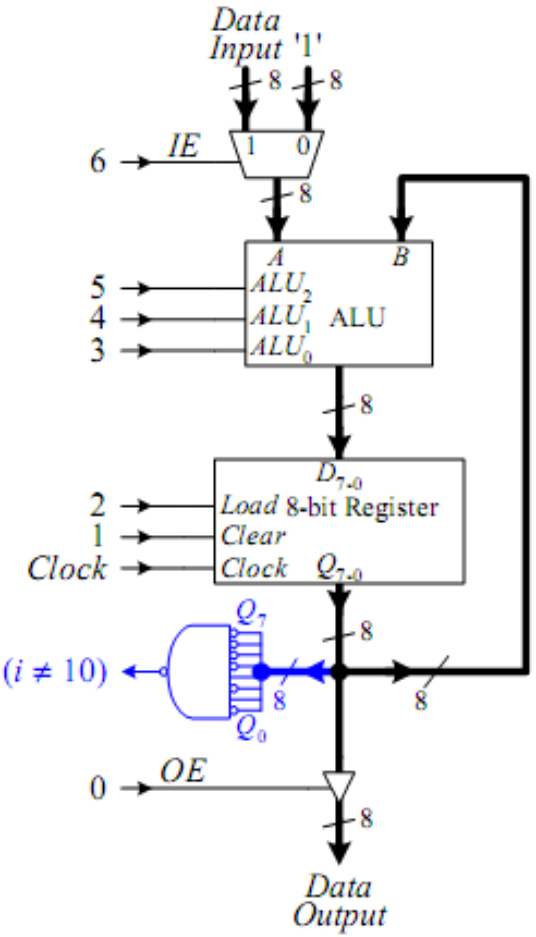
(b)

Control Word	Instruction	*iLoad*	*Clear*	*Out*
1	$i = 0$	0	1	0
2	$i = i + 1$	1	0	0
3	OUTPUT i	0	0	1

Control Unit

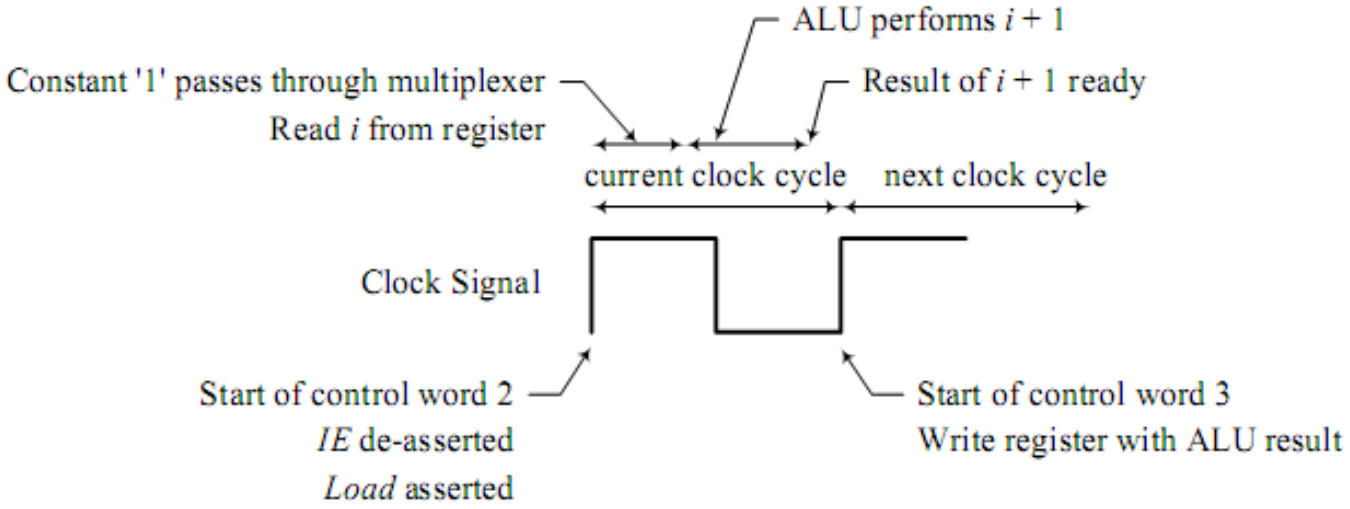
Datapath

Timing Issues

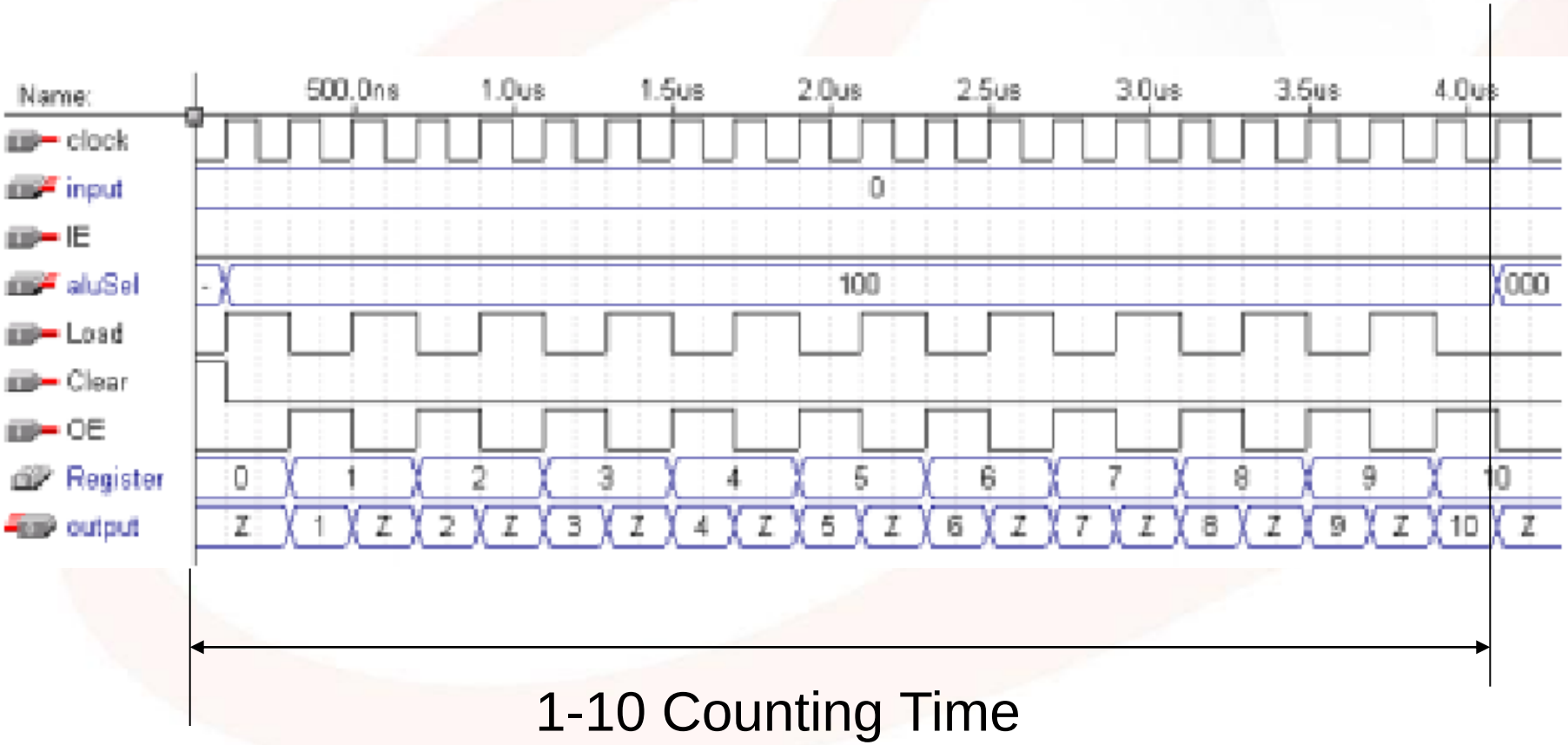



```
1          i = 0
2          WHILE (i ≠ 10){
3             i = i + 1
4             OUTPUT i
5             }
```

Control Word	Instruction	*IE* 6	$ALU_2\,ALU_1\,ALU_0$ 5–3	*Load* 2	*Clear* 1	*OE* 0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	OUTPUT *i*	×	×××	0	0	1



Timing Issues

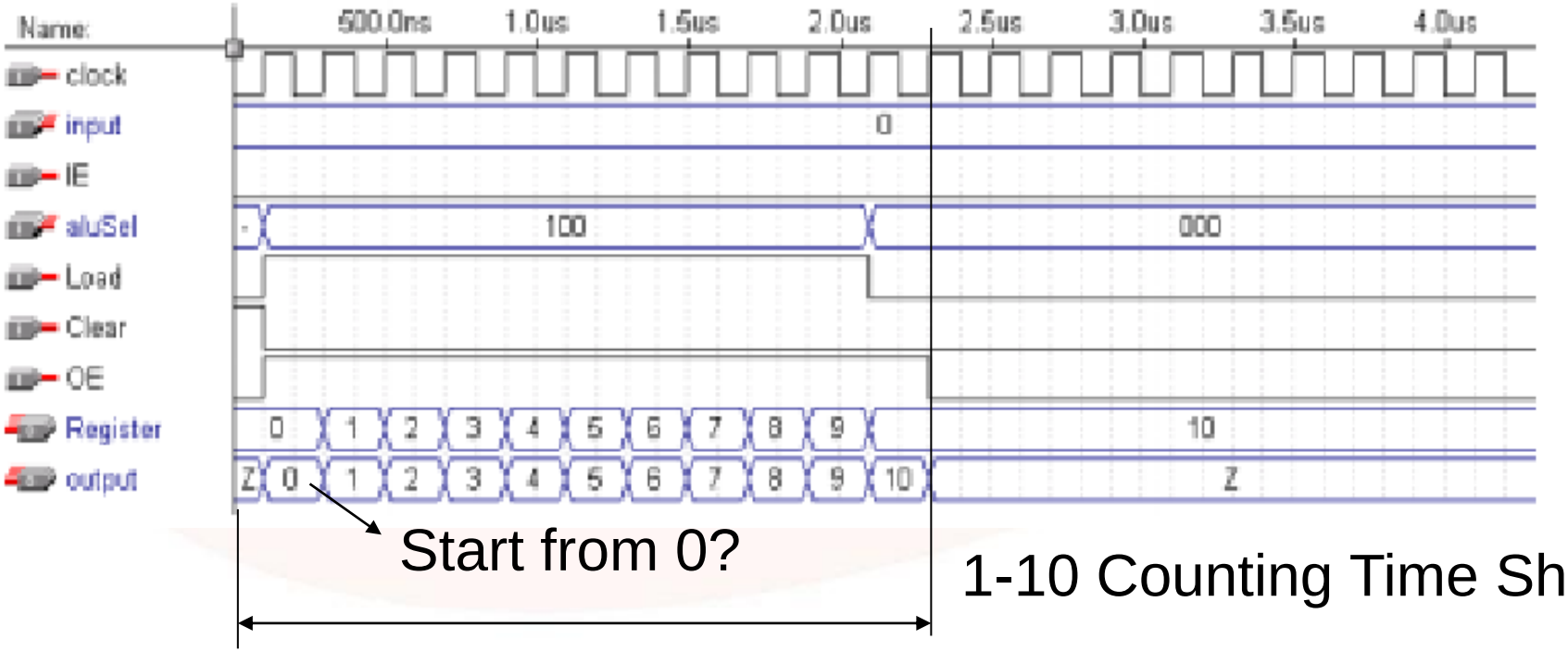


1-10 Counting Time

Timing Issues

Change Control Word →

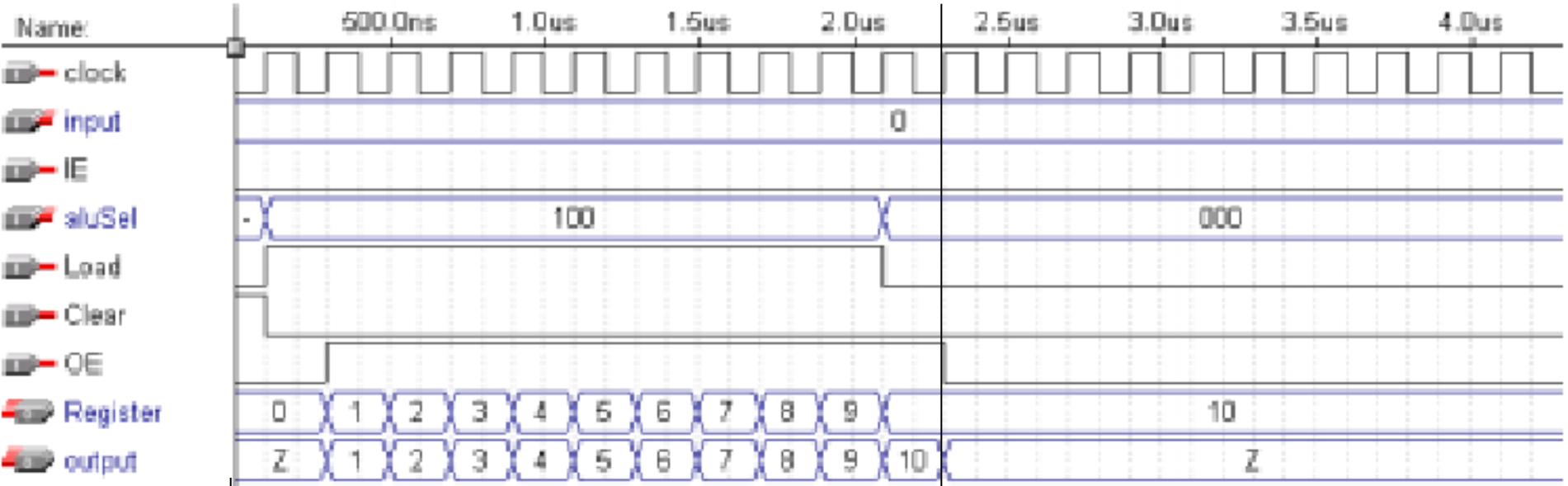
Control Word	Instruction	*IE* 6	$ALU_2\ ALU_1\ ALU_0$ 5–3	*Load* 2	*Clear* 1	*OE* 0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$ & OUTPUT i	0	100 (add)	1	0	1



Timing Issues

Optimize Control Word →

Control Word	Instruction	*IE* 6	$ALU_2\ ALU_1\ ALU_0$ 5–3	*Load* 2	*Clear* 1	*OE* 0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	$i = i + 1$ & OUTPUT i	0	100 (add)	1	0	1



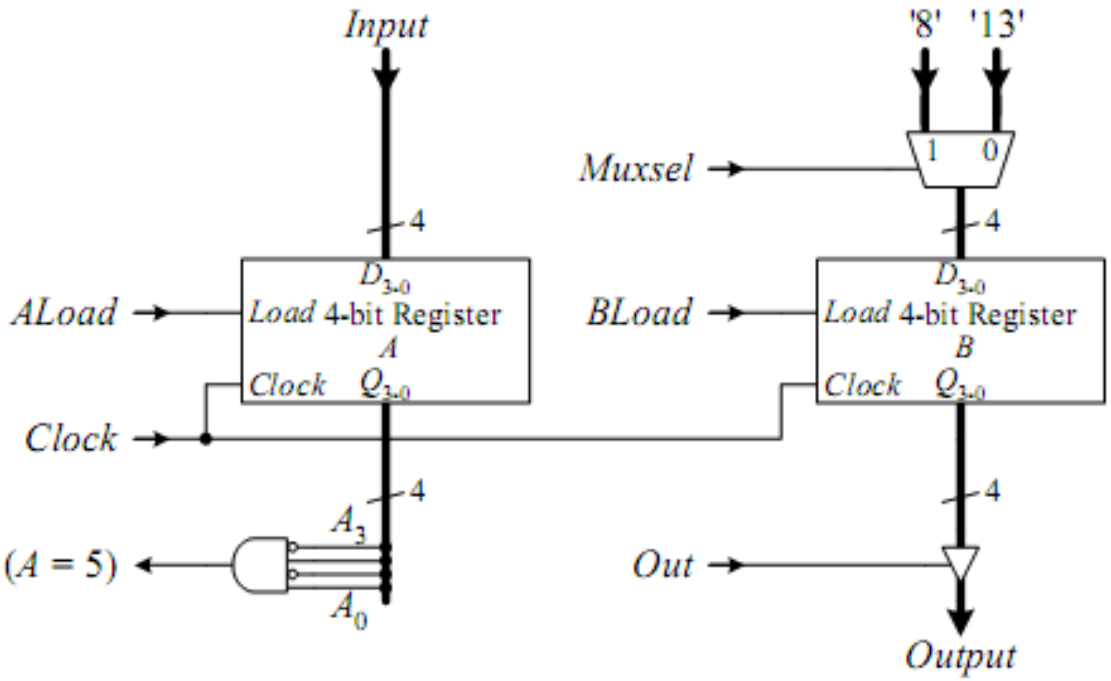
1-10 Counting Time

Start from 1

Design Control Unit -> if-then-else

```
1          INPUT A
2          IF (A = 5) THEN
3             B = 8
4          ELSE
5             B = 13
6          END IF
7          OUTPUT B
```

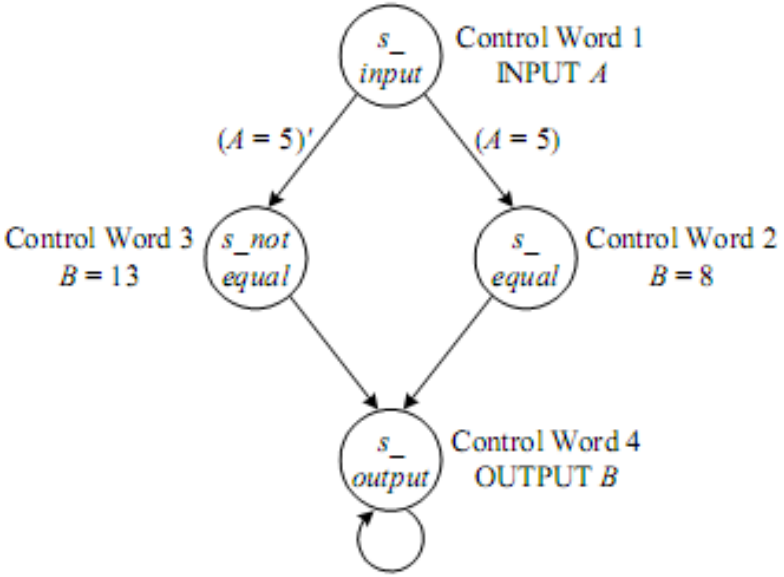
Control Word	Instruction	*ALoad*	*Muxsel*	*BLoad*	*Out*
1	INPUT *A*	1	×	0	0
2	*B* = 8	0	1	1	0
3	*B* = 13	0	0	1	0
4	OUTPUT *B*	0	×	0	1



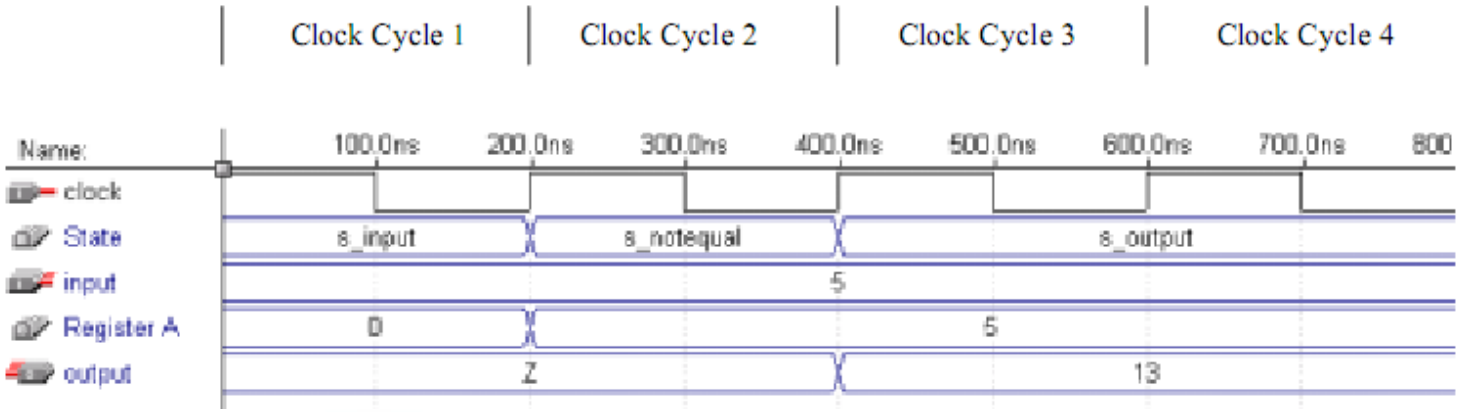
Datapath

Design Control Unit -> if-then-else

Incorrect State Diagram

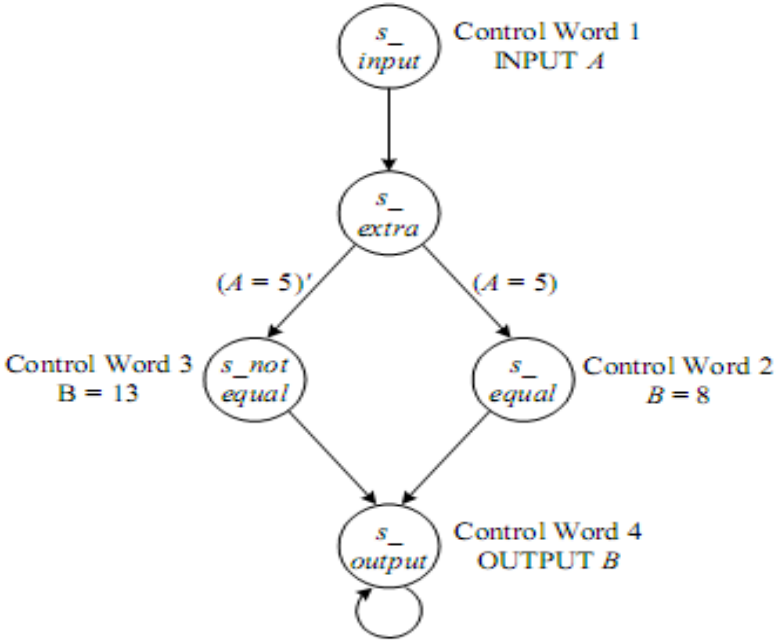


Incorrect Timing Diagram

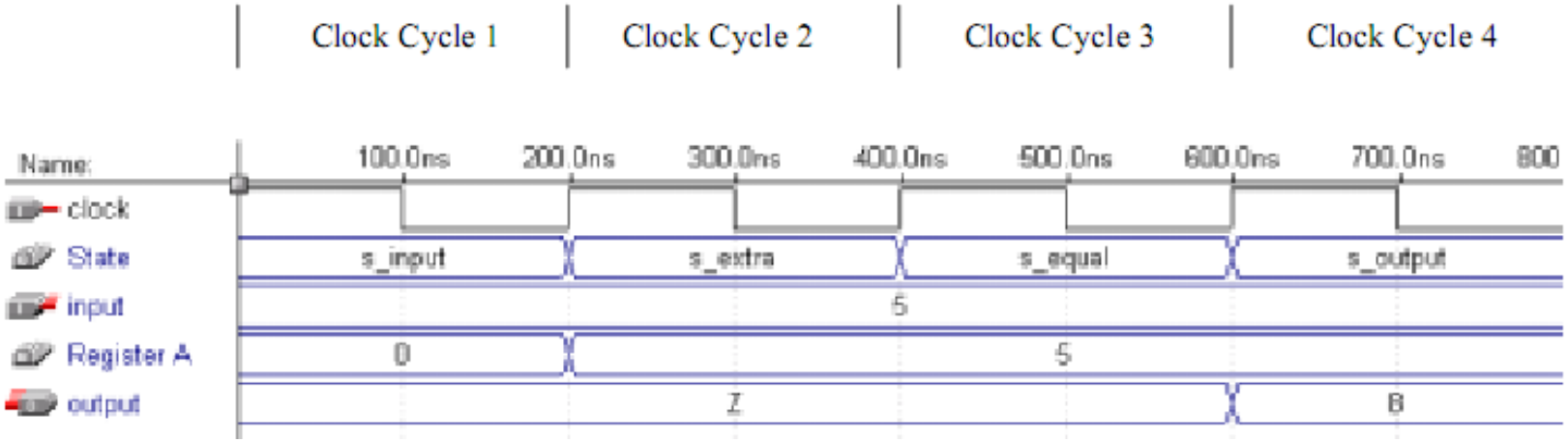


Design Control Unit -> if-then-else

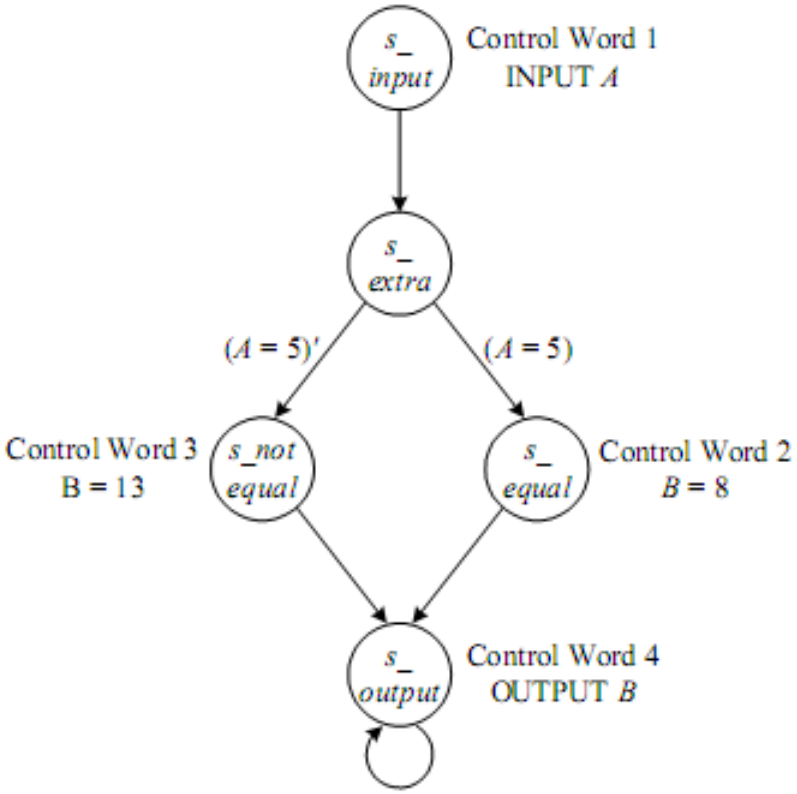
Correct State Diagram



Correct Timing Diagram



Design Control Unit -> if-then-else



Current State $Q_2Q_1Q_0$	Next State $Q_{2next}\,Q_{1next}\,Q_{0next}$	
	$(A = 5)'$	$(A = 5)$
s_input 000	*s_extra* 001	*s_extra* 001
s_extra 001	*s_notequal* 010	*s_equal* 011
s_notequal 010	*s_output* 100	*s_output* 100
s_equal 011	*s_output* 100	*s_output* 100
s_output 100	*s_output* 100	*s_output* 100
Unused 101	000	000
Unused 110	000	000
Unused 111	000	000

Design Control Unit -> if-then-else

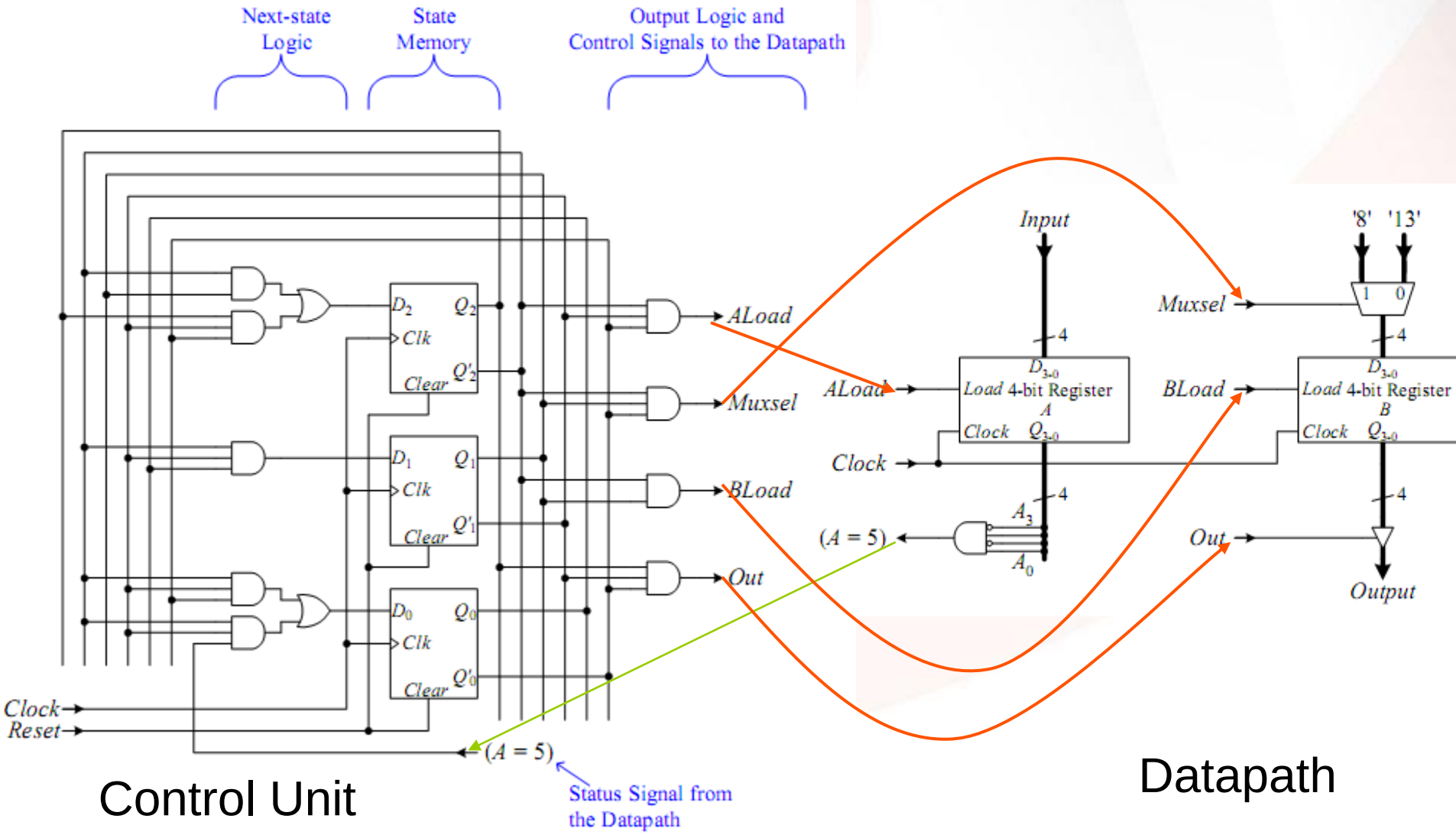
Current State $Q_2Q_1Q_0$	Next State $D_2D_1D_0$	
	$(A = 5)'$	$(A = 5)$
000	001	001
001	010	011
010	100	100
011	100	100
100	100	100
101	000	000
110	000	000
111	000	000

$$D_2 = Q_2'Q_1 + Q_2Q_1'Q_0'$$
$$D_1 = Q_2'Q_1'Q_0$$
$$D_0 = Q_2'Q_1'Q_0' + Q_2'Q_1'(A = 5)$$

$Q_2Q_1Q_0$	Instruction	*ALoad*	*Muxsel*	*BLoad*	*Out*
000	INPUT *A*	1	×	0	0
001	No operation	0	×	0	0
010	$B = 8$	0	1	1	0
011	$B = 13$	0	0	1	0
100	OUTPUT *B*	0	×	0	1
101	No operation	0	×	0	0
110	No operation	0	×	0	0
111	No operation	0	×	0	0

$$ALoad = Q_2'Q_1'Q_0$$
$$Muxsel = Q_2'Q_1Q_0'$$
$$BLoad = Q_2'Q_1$$
$$Out = Q_2Q_1'Q_0'$$

Design Control Unit -> if-then-else



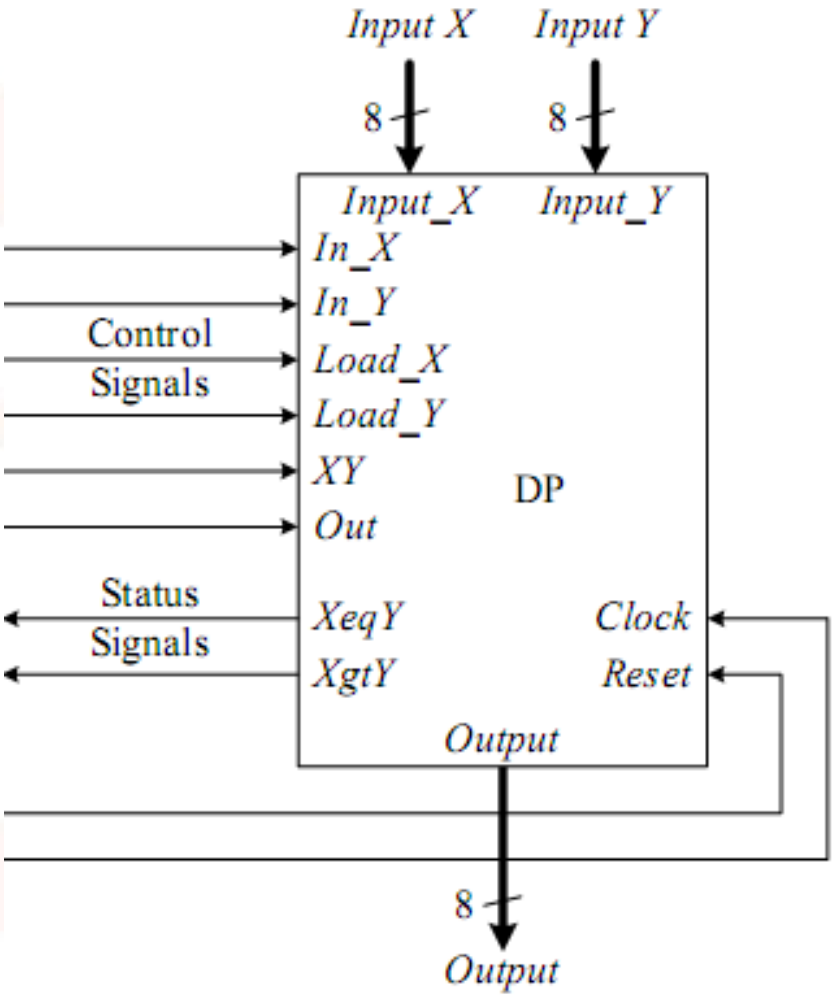
Greatest Common Divisor (GCD)

In this example, we will manually design the complete dedicated microprocessor for evaluating the greatest common divisor (GCD) of two 8-bit unsigned numbers X and Y. For example, GCD(3,5) = 1,GCD(10,4) = 2, and GCD(12,60) = 12.

```
1     input X
2     input Y
3     while (X ≠ Y){
4        if (X > Y) then
5           X = X - Y
6        else
7           Y = Y - X
8        end if
9        }
10    output X
```

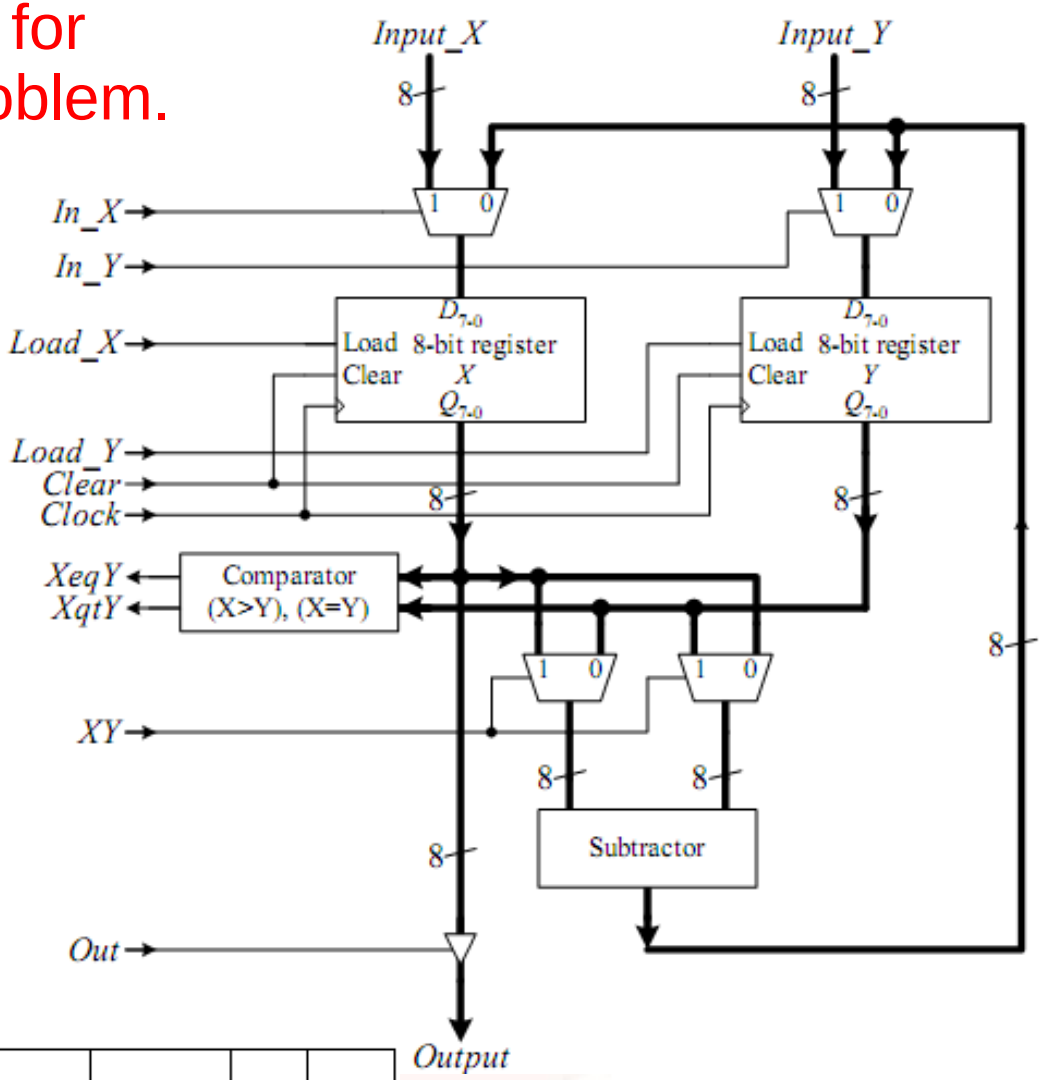
Algorithm for solving the GCD problem.

GCD Datapath



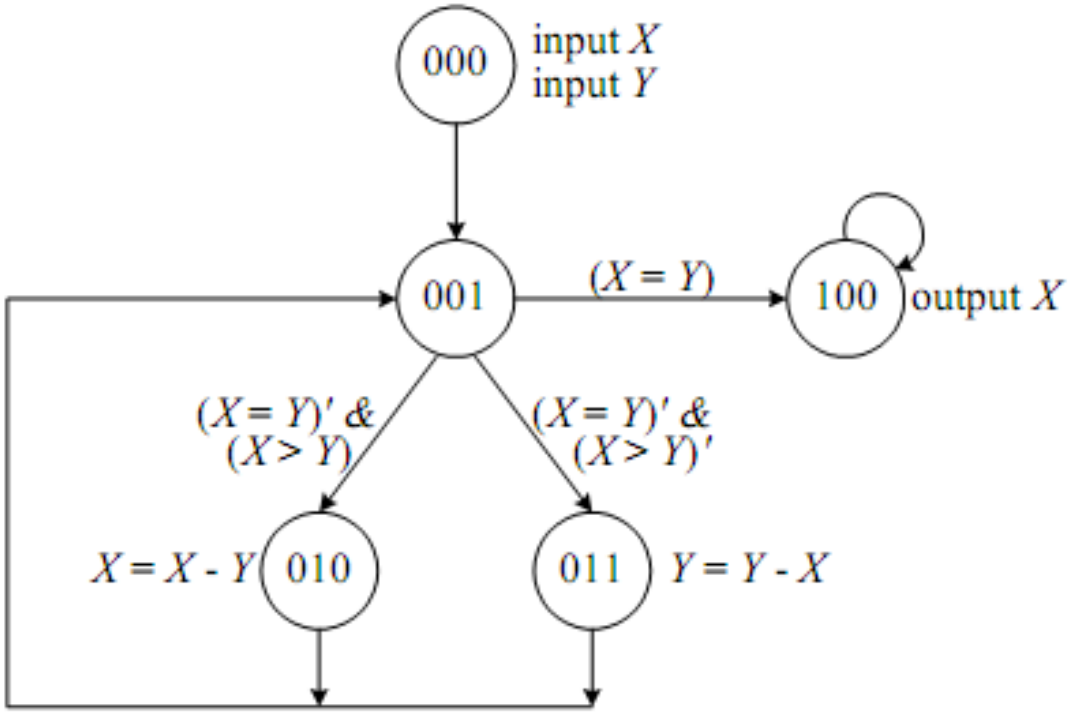
Dedicated datapath for solving the GCD problem.

```
1     input X
2     input Y
3     while (X ≠ Y) {
4        if (X > Y) then
5           X = X - Y
6        else
7           Y = Y - X
8        end if
9        }
10    output X
```



Control Word	State $Q_2\ Q_1\ Q_0$	Instruction	*In_X*	*In_Y*	*Load_X*	*Load_Y*	*XY*	*Out*
0	000	Input *X*, Input *Y*	1	1	1	1	×	0
1	001	none	×	×	0	0	×	0
2	010	$X = X - Y$	0	×	1	0	1	0
3	011	$Y = Y - X$	×	0	0	1	0	0
4	100	Output *X*	×	×	0	0	×	1

```
1       input X
2       input Y
3       while (X ≠ Y) {
4          if (X > Y) then
5             X = X - Y
6          else
7             Y = Y - X
8          end if
9          }
10      output X
```

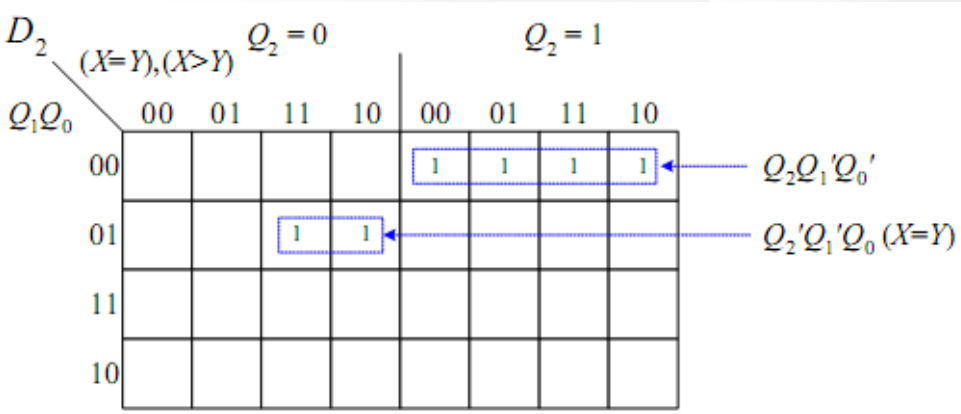


(a)

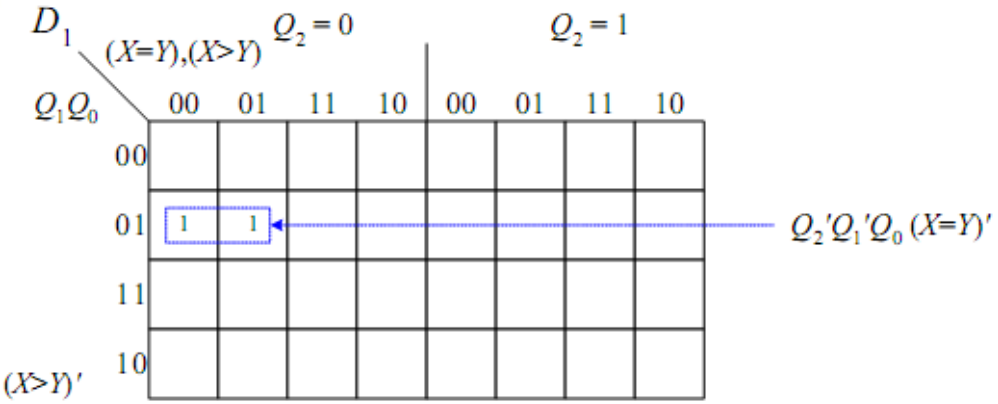
Current State $Q_2Q_1Q_0$	Next State $Q_{2next}Q_{1next}Q_{0next}$ (X=Y), (X>Y)			
	00	01	10	11
000	001	001	001	001
001	011	010	100	100
010	001	001	001	001
011	001	001	001	001
100	100	100	100	100

ELECTRICAL
Engineering KMITNB

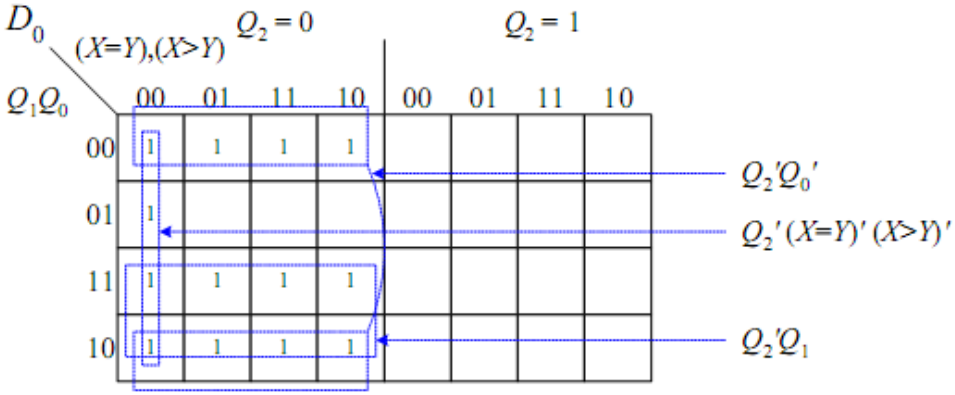
Current State $Q_2Q_1Q_0$	Next State $Q_{2next}Q_{1next}Q_{0next}$ $(X=Y), (X>Y)$			
	00	01	10	11
000	001	001	001	001
001	011	010	100	100
010	001	001	001	001
011	001	001	001	001
100	100	100	100	100



$$D_2 = Q_2Q_1'Q_0' + Q_2'Q_1'Q_0\ (X=Y)$$



$$D_1 = Q_2'Q_1'Q_0\ (X=Y)'$$



$$D_0 = Q_2'Q_0' + Q_2'(X=Y)'(X>Y)' + Q_2'Q_1$$

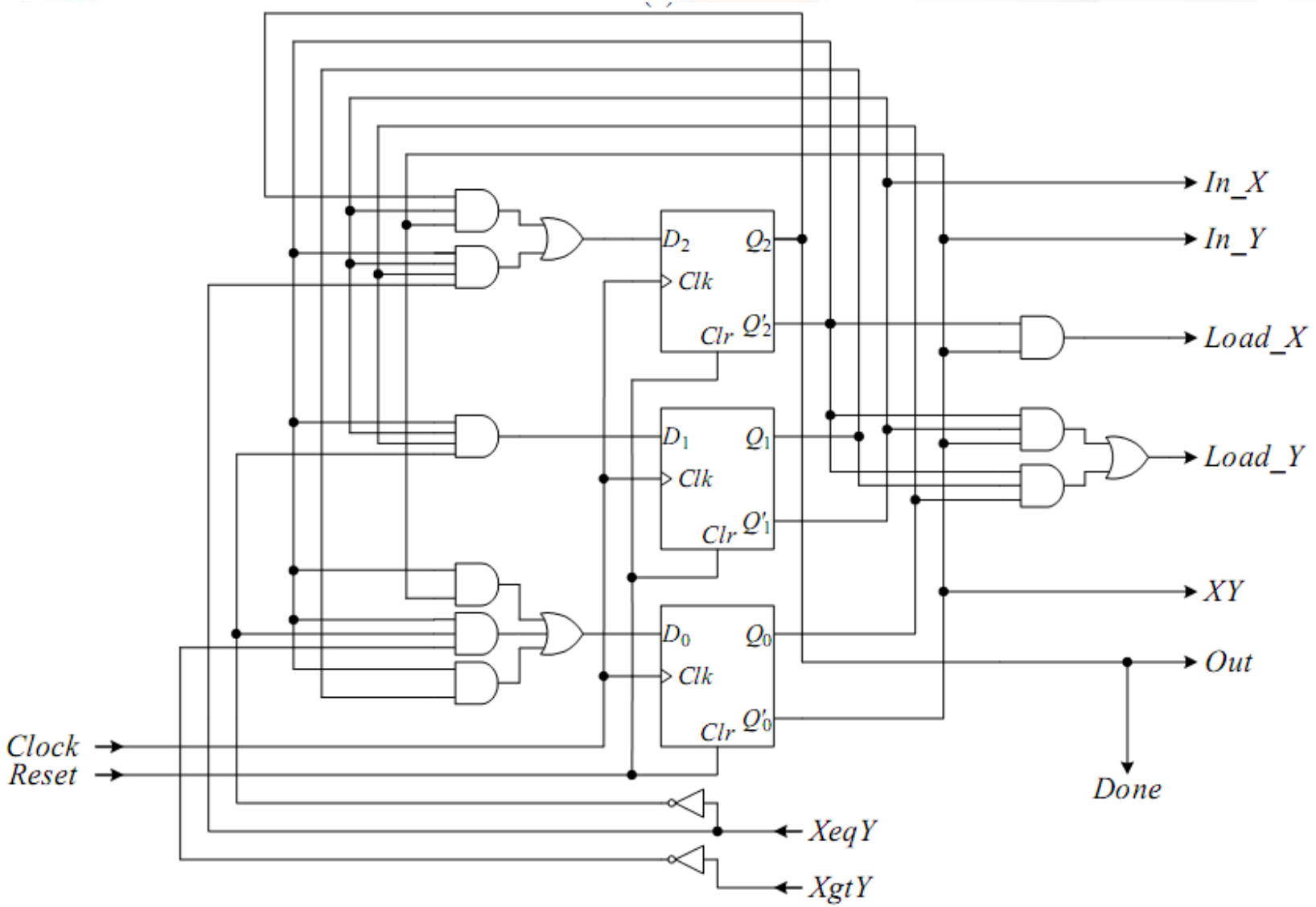
Output Logic stage

Control Word	State $Q_2\,Q_1\,Q_0$	Instruction	*In_X*	*In_Y*	*Load_X*	*Load_Y*	*XY*	*Out*
0	000	Input *X*, Input *Y*	1	1	1	1	×	0
1	001	none	×	×	0	0	×	0
2	010	$X = X - Y$	0	×	1	0	1	0
3	011	$Y = Y - X$	×	0	0	1	0	0
4	100	Output *X*	×	×	0	0	×	1

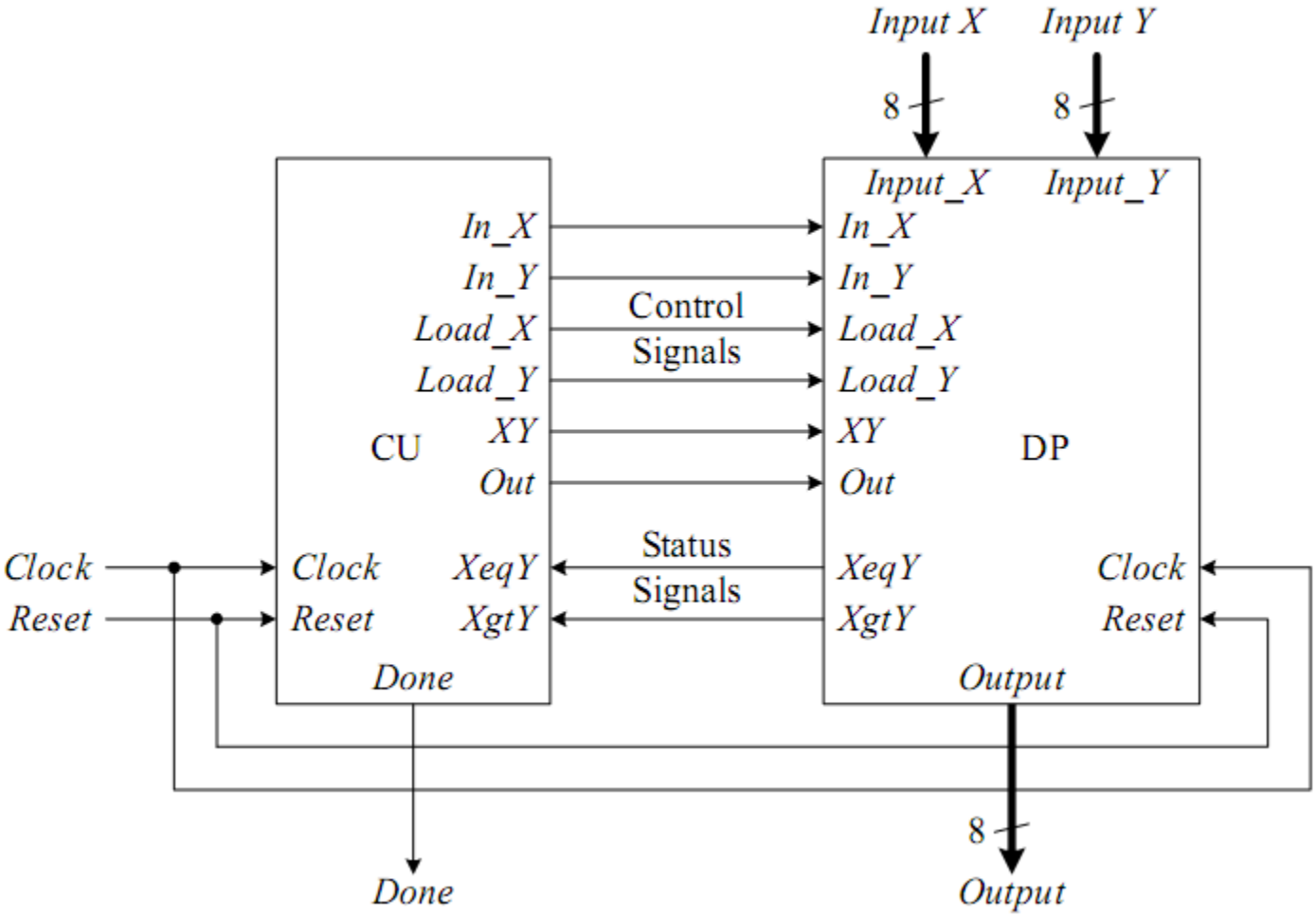
$$In\_X = Q_1'$$
$$In\_Y = Q_0'$$
$$XY = Q_0'$$

$$Load\_X = Q_2'Q_0'$$
$$Load\_Y = Q_2'Q_1'Q_0' + Q_2'Q_1Q_0$$
$$Out = Q_2$$

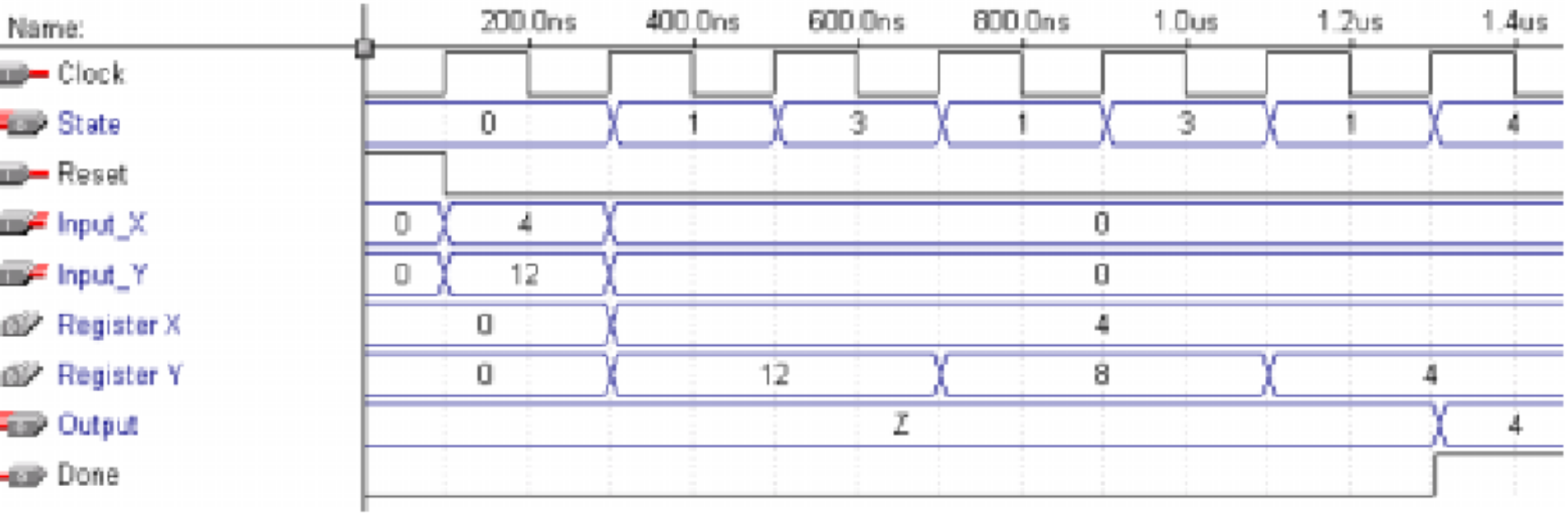
GCD FSM


In_X
In_Y
Load_X
Load_Y
XY
Out
Done
Clock
Reset
XeqY
XgtY
D2
Q2
Clk
Clr
Q'2
D1
Q1
Clk
Clr
Q'1
D0
Q0
Clk
Clr
Q'0

Dedicated Microprocessor for solve GCD



Sample Simulation GCD

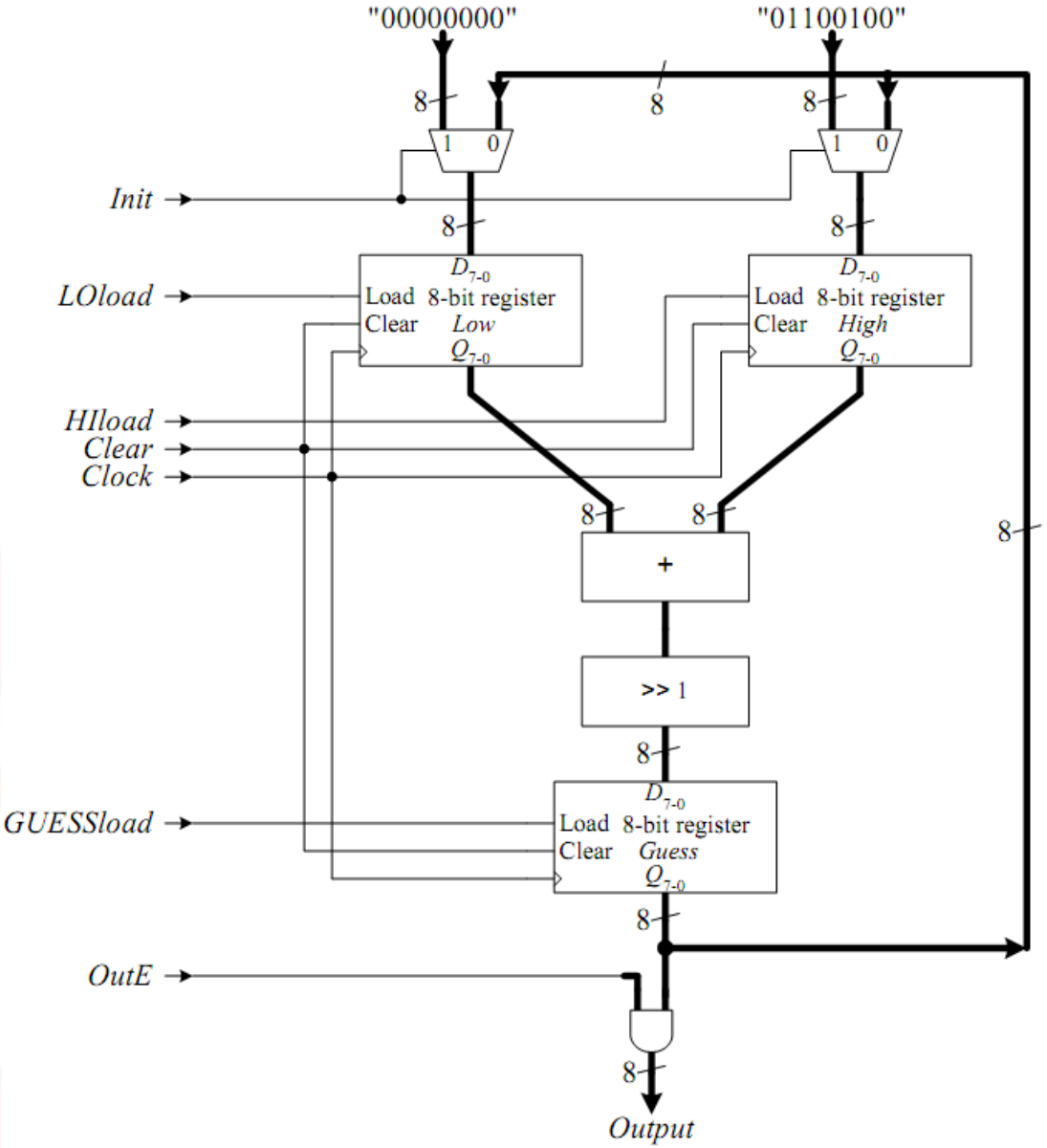


The GCD of numbers 4 and 12 is 4.

Ex1: Hi-Low guessing game

```
1       Low = 0                          // initialize Low
2       High = 100                       // initialize High
3       repeat {
4          Guess = (Low + High) / 2  // calculate guess using binary search
5          output Guess
6          if (lo_button = '1' and hi_button = '0')    // low button pressed
7             Low = Guess
8          else if (lo_button = '0' and hi_button = '1') // hi button pressed
9             High = Guess
10         end if
11      } until (lo_button = '1' and hi_button = '1')  // repeat until both
                                                       // buttons are pressed
12      while (lo_button = '0' and hi_button = '0')
           // blink correct guess
13         output Guess
14         turn off display
15      end while
```

Algorithm for Hi-Low guessing game



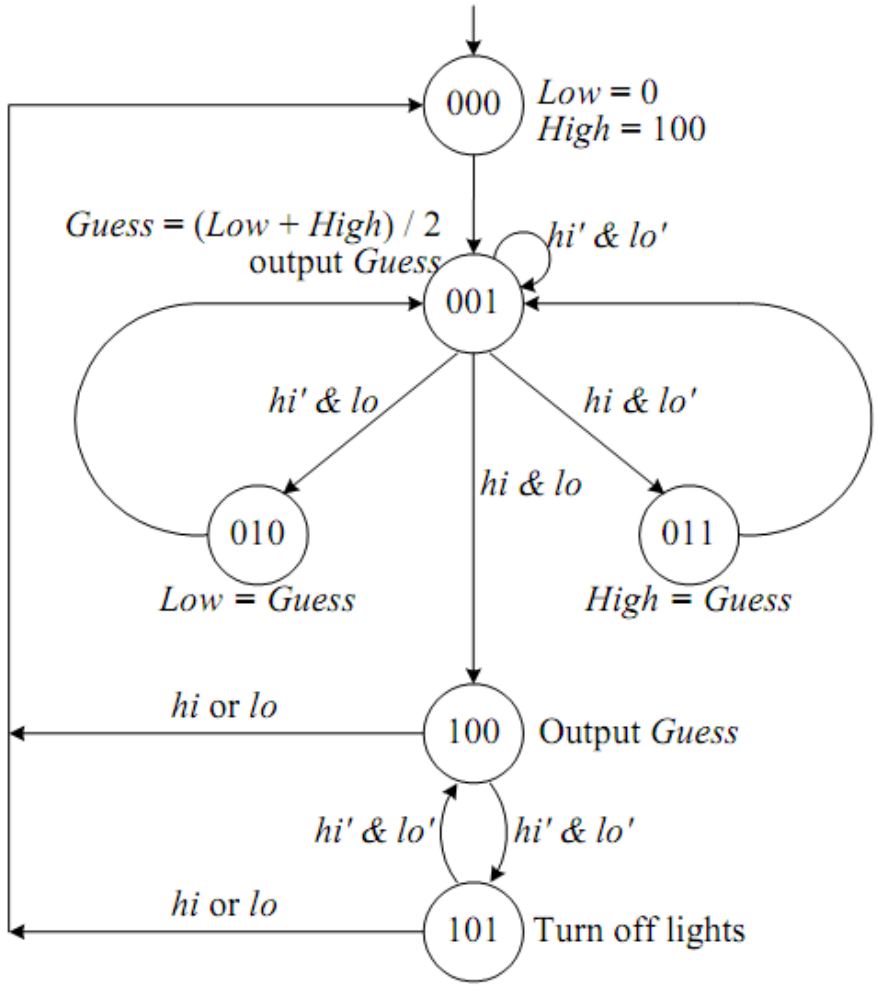
Dedicated datapath for the high-low guessing game.

Ex1: Hi-Low guessing game

Control Word	State $Q_2\ Q_1\ Q_0$	Instruction	*Init*	*HIload*	*LOload*	*GUESSload*	*OutE*
0	000	Initialize	1	1	1	0	1
1	001	Guess	0	0	0	1	1
2	010	Low = Guess	0	0	1	0	1
3	011	High = Guess	0	1	0	0	1
4	100	Correct	0	0	0	0	1
5	101	Flash output	0	0	0	0	0

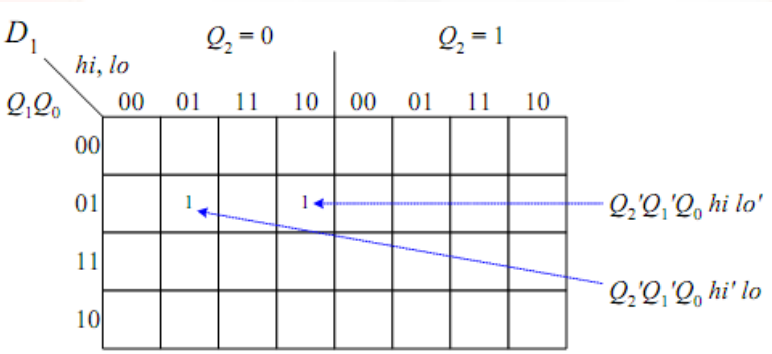
Control word of the Hi-Low guessing game

Sol.: Design Control Unit Hi-Low guessing game

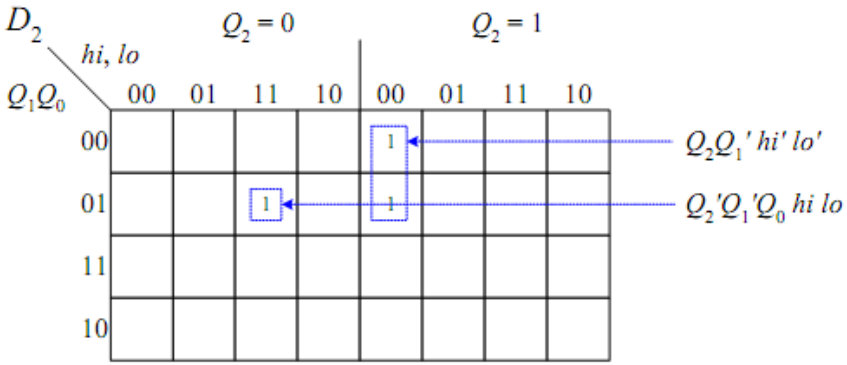


Sol.: Design Control Unit
Hi-Low guessing game

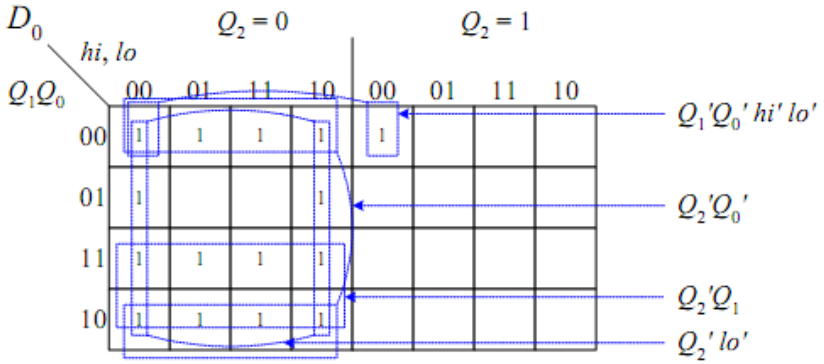
Current State $Q_2Q_1Q_0$	Next State $Q_{2next}\,Q_{1next}\,Q_{0next}$ (Implementation $D_2\,D_1\,D_0$) hi, lo			
	00	01	10	11
000	001	001	001	001
001	001	010	011	100
010	001	001	001	001
011	001	001	001	001
100	101	000	000	000
101	100	000	000	000
110 (not used)	000	000	000	000
111 (not used)	000	000	000	000



$$D_1 = Q_2'Q_1'Q_0\,hi\,lo' + Q_2'Q_1'Q_0\,hi'\,lo$$



$$D_2 = Q_2Q_1'\,hi'\,lo' + Q_2'Q_1'Q_0\,hi\,lo$$



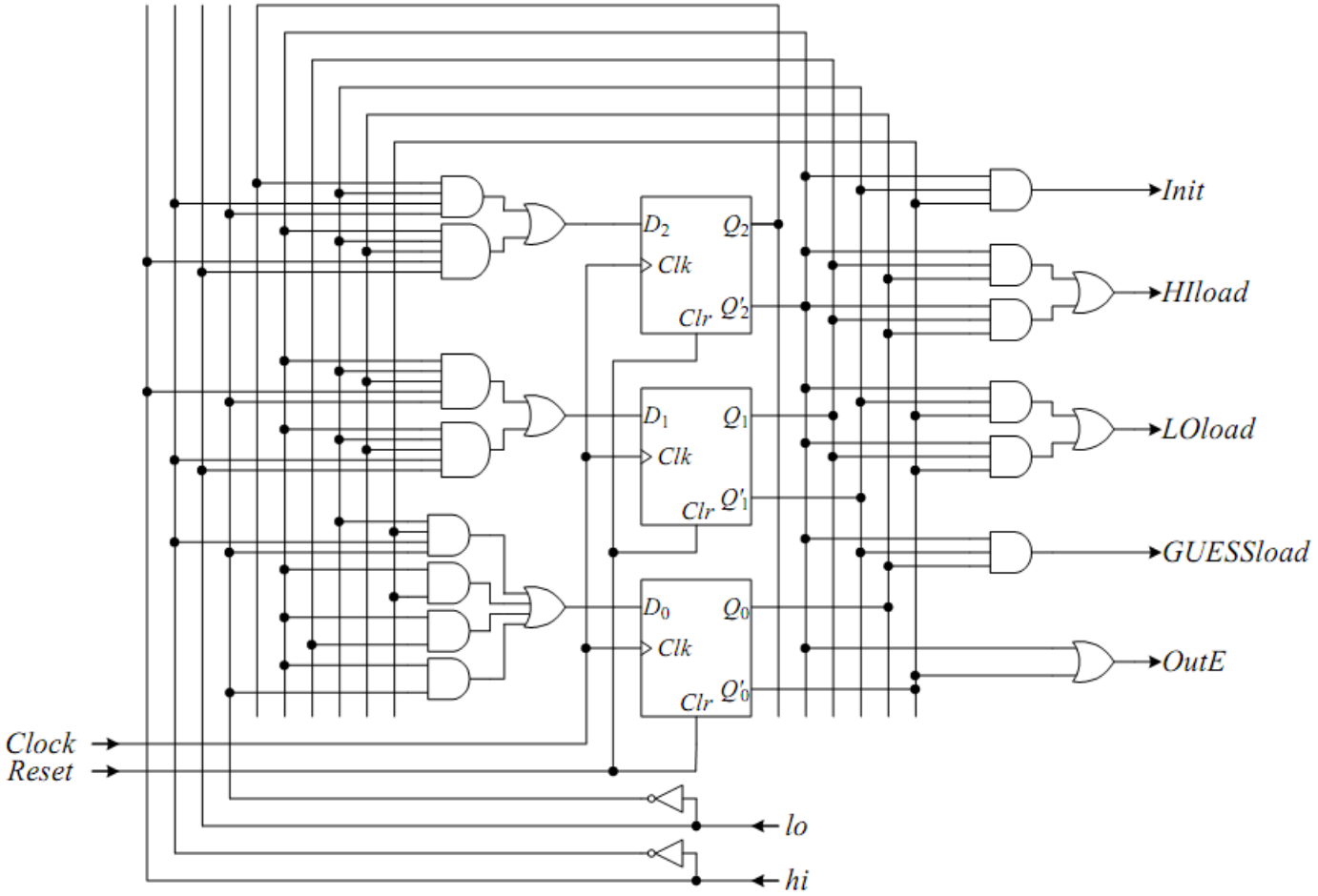
$$D_0 = Q_1'Q_0'\,hi'\,lo' + Q_2'Q_0' + Q_2'Q_1 + Q_2'\,lo'$$

Sol.: Design Control Unit Hi-Low guessing game

Control Word	State $Q_2\ Q_1\ Q_0$	Instruction	*Init*	*HIload*	*LOload*	*GUESSload*	*OutE*
0	000	Initialize	1	1	1	0	1
1	001	Guess	0	0	0	1	1
2	010	Low = Guess	0	0	1	0	1
3	011	High = Guess	0	1	0	0	1
4	100	Correct	0	0	0	0	1
5	101	Flash output	0	0	0	0	0

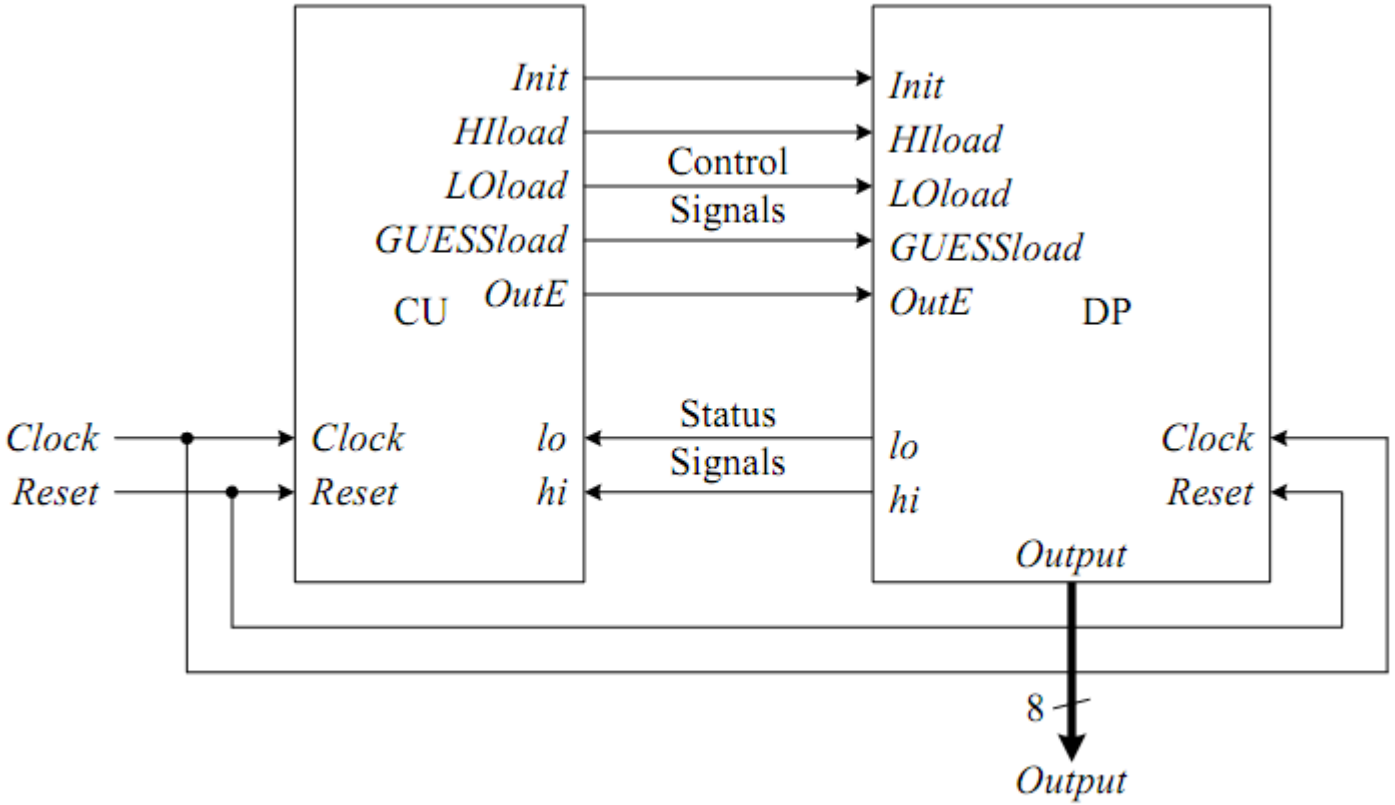
$$Init = Q_2'Q_1'Q_0'$$
$$HIload = Q_2'Q_1'Q_0' + Q_2'Q_1Q_0$$
$$LOload = Q_2'Q_1'Q_0' + Q_2'Q_1Q_0'$$
$$GUESSload = Q_2'Q_1'Q_0$$
$$OutE = Q_2' + Q_0'$$

Sol.: Design Control Unit Hi-Low guessing game

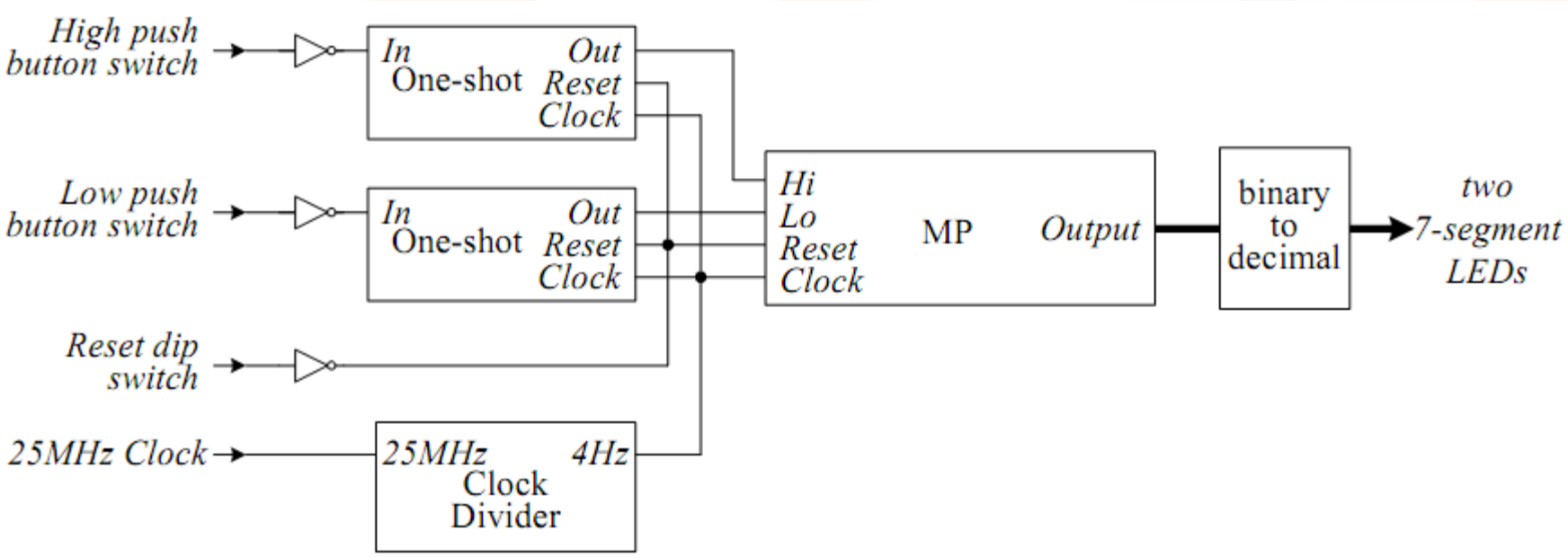


Control Unit

Dedicated microprocessor for the Hi-Low guessing game



Example for implement the Hi-Low guessing game

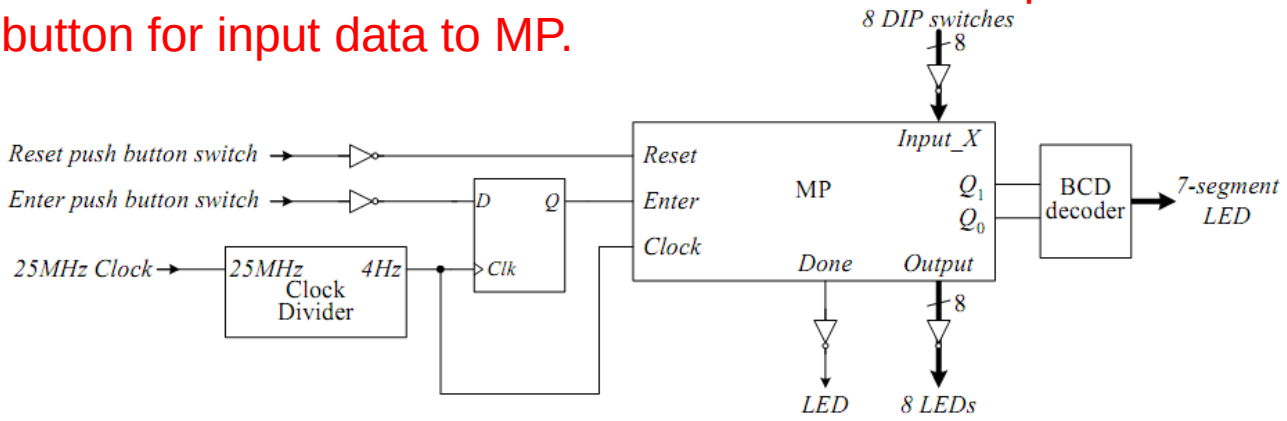


H/W: Finding Largest Number

Design dedicated microprocessor for the algorithm finding largest number.

```
1     Largest = 0              // for storing the current largest number
2     input X                  // enter first number
3     while (X ≠ 0){
4        if (X > Largest) then     // if new number greater?
5           Largest = X            // yes, remember new largest number
6        end if
7        output Largest
8        input X                   // get next number
9     }
```

When set DIP switches for a number finish then press enter button for input data to MP.



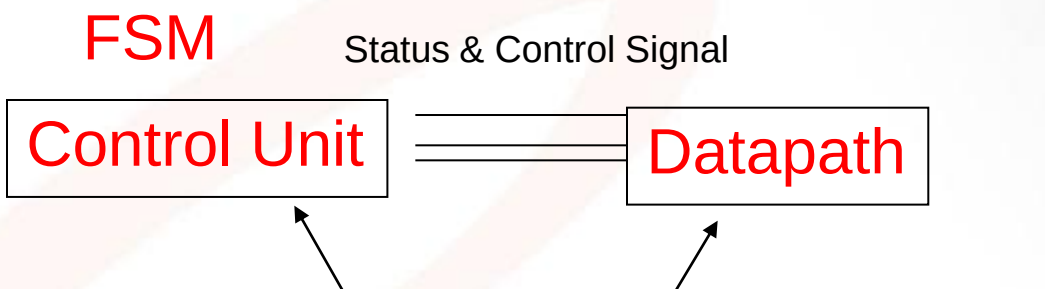
FSM+D

Constructing a microprocessor manually this way uses the FSM+D (FSM plus datapath) model. In this model, both the FSM and the datapath circuits are manually constructed as separate units. The FSM and the datapath are connected together in an enclosing unit using the control and status signals.

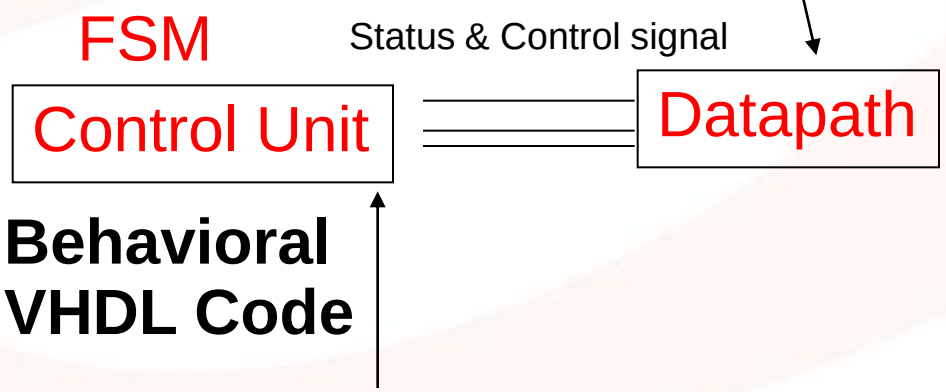
The advantage of using this FSM+D model is that you have full control as to how the datapath is built.

FSM+D

1.

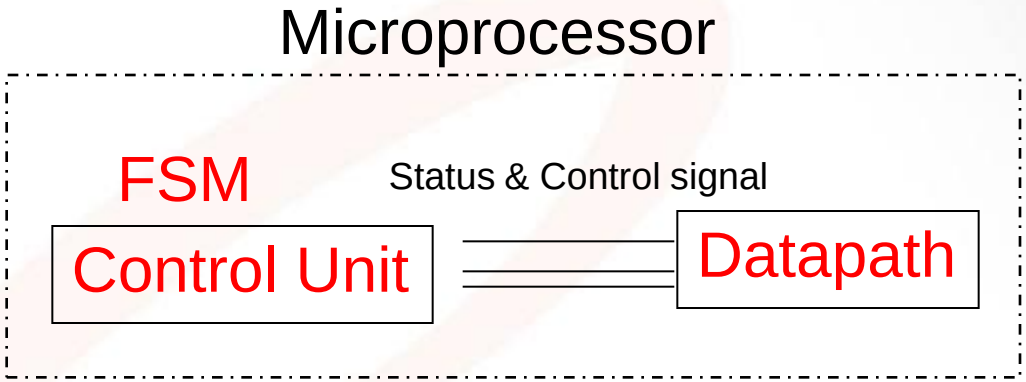


2.



The automatically synthesized FSM and the manually constructed datapath are then connected together like before in an enclosing unit using the control and status signals.

FSMD (FSM with Datapath)



Hardware Description Language

Using this model, you would design the FSM using behavioral VHDL code just like in the FSM+D model. However, instead of constructing the datapath manually as a separate unit, all the datapath operations are embedded within the FSM entity using the built-in VHDL operators.

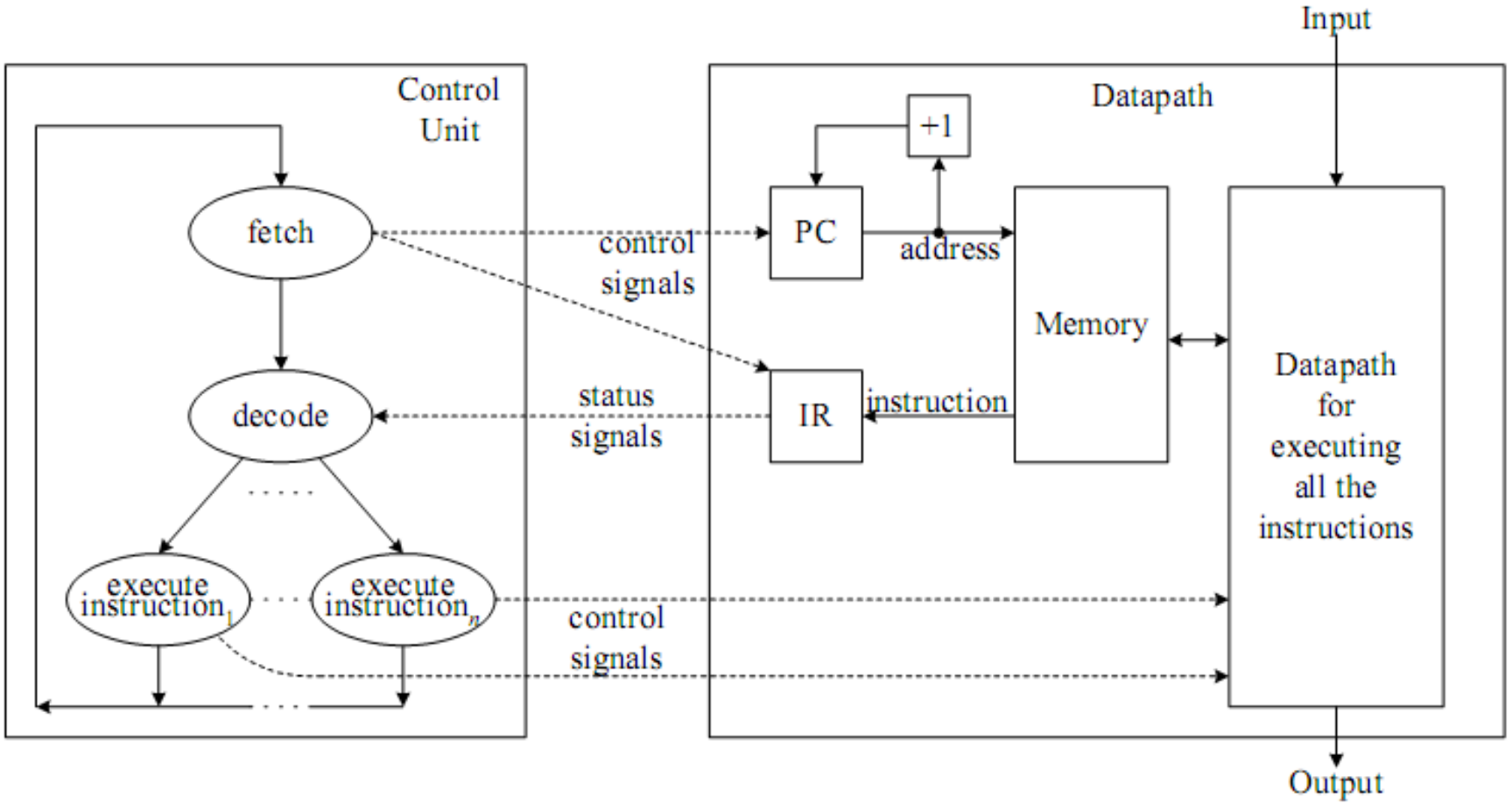
During the synthesis process, the synthesizer will automatically generate a separate FSM and datapath units, but these two units will be automatically connected together as one microprocessor

General-purpose Microprocessor

General-purpose microprocessors are capable of performing a variety of computations. In order to achieve this goal, each computation is not hardwired into the processor, but rather is represented by a sequence of instructions in the form of a program that is stored in the memory, and executed by the microprocessor.

The program in the memory can be easily changed so that another computation can be performed.

General-purpose Microprocessor



Overview of a general-purpose microprocessor.

General-purpose Microprocessor

In designing a CPU, we must first define its instruction set, and how the instructions are encoded and executed.

Once we have decided on the instruction set, we can proceed to designing a datapath that can execute all the instructions in the instruction set.

General-purpose Microprocessor

The control unit for a general-purpose microprocessor basically cycles through three main steps, usually referred to as the instruction cycle:

1) fetches an instruction;
2) decodes the instruction; and
3) executes the instruction.

ELECTRICAL
Engineering KMITNB

The End

Hope to see you again next semester !!!
in
Embedded System…
we will fun……