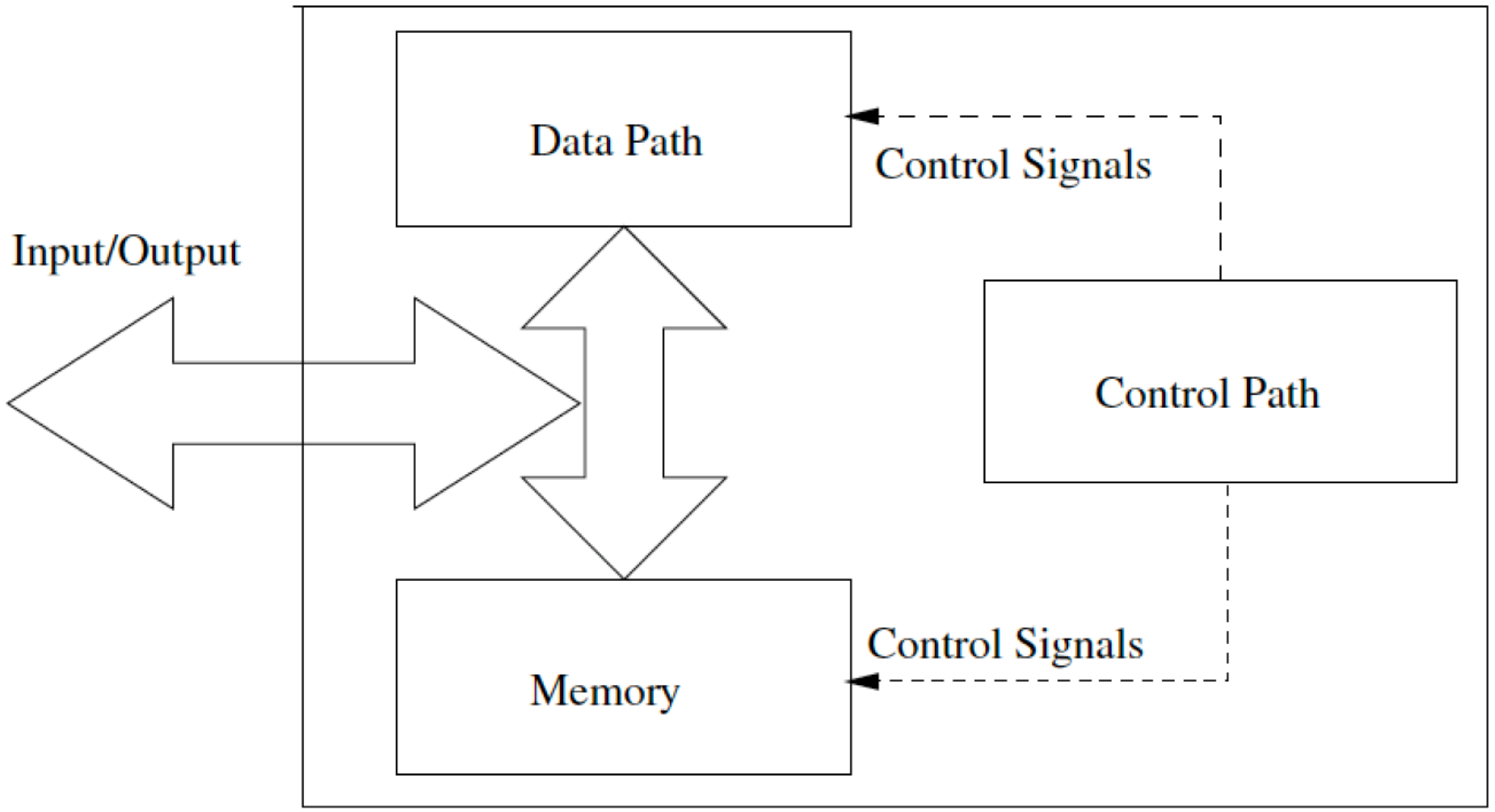
Digital Logic Design + Computer Architecture

Sayandeep Saha

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Sequential Circuits

General View of a Hardware: Once Again



Binary GCD Algorithm

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
while (u!=v) do
    if u and v are even then
        z = 2gcd(u/2, v/2)
    end
    else if (u is odd and v is even) then
        z = gcd(u, v/2)
    end
    else if (u is even and v is odd) then
        z = gcd(u/2, v)
    end
    else
        if (u ≥ v) then
            z = gcd((u − v)/2, v)
        end
        else
            z = gcd(u, (v − u)/2)
        end
    end
end
```

Binary GCD Algorithm: Closer to Hardware

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
while (u != v) do
    if u and v are even then
        z = 2gcd(u/2, v/2)
    end
    else if (u is odd and v is even) then
        z = gcd(u, v/2)
    end
    else if (u is even and v is odd) then
        z = gcd(u/2, v)
    end
    else
        if (u ≥ v) then
            z = gcd((u − v)/2, v)
        end
        else
            z = gcd(u, (v − u)/2)
        end
    end
end
```

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
register X_R, Y_R;
X_R = u; Y_R = v; count = 0;
while (X_R != Y_R) do
    if (X_R[0] = 0 and Y_R[0] = 0) then
        X_R = right shift(X_R)
        Y_R = right shift(Y_R)
        count = count + 1
    end
    else if (X_R[0] = 1 and Y_R[0] = 1) then
        Y_R = right shift(Y_R)
    end
    else if (X_R[0] = 0 and Y_R[0] = 1) then
        X_R = right shift(X_R)
    end
    else
        if (X_R ≥ Y_R) then
            X_R = right shift(X_R − Y_R)
        end
        else
            Y_R = right shift(Y_R − X_R)
        end
    end
end
while (count > 0) do
    X_R = left shift(X_R)
    count = count − 1
end
```

Binary GCD Algorithm: Closer to Hardware

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

while $(u!=v)$ **do**
  if u *and* v *are even* **then**
    $z = 2gcd(u/2, v/2)$
  end
  else if $(u$ *is odd and* v *is even*$)$ **then**
    $z = gcd(u, v/2)$
  end
  else if $(u$ *is even and* v *is odd*$)$ **then**
    $z = gcd(u/2, v)$
  end
  else
    if $(u \geq v)$ **then**
      $z = gcd((u-v)/2, v)$
    end
    else
      $z = gcd(u, (v-u)/2)$
    end
  end
end

How to interpret as hardware?

Keep the count of the 2's getting multiplied

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

register X_R, Y_R;
$X_R = u$; $Y_R = v$; $count = 0$;
while $(X_R! = Y_R)$ **do**
  if $(X_R[0] = 0$ *and* $Y_R[0] = 0)$ **then**
    $X_R = \text{right shift}(X_R)$
    $Y_R = \text{right shift}(Y_R)$
    $count = count + 1$
  end
  else if $(X_R[0] = 1$ *and* $Y_R[0] = 1)$ **then**
    $Y_R = \text{right shift}(Y_R)$
  end
  else if $(X_R[0] = 0$ *and* $Y_R[0] = 1)$ **then**
    $X_R = \text{right shift}(X_R)$
  end
  else
    if $(X_R \geq Y_R)$ **then**
      $X_R = \text{right shift}(X_R - Y_R)$
    end
    else
      $Y_R = \text{right shift}(Y_R - X_R)$
    end
end
while $(count > 0)$ **do**
  $X_R = \text{left shift}(X_R)$
  $count = count - 1$
end

Constructing the Datapath

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
register X_R, Y_R;
X_R = u; Y_R = v; count = 0;
while (X_R != Y_R) do
    if (X_R[0] = 0 and Y_R[0] = 0) then
        X_R = right shift(X_R)
        Y_R = right shift(Y_R)
        count = count + 1
    end
    else if (X_R[0] = 1 and Y_R[0] = 1) then
        Y_R = right shift(Y_R)
    end
    else if (X_R[0] = 0 and Y_R[0] = 1) then
        X_R = right shift(X_R)
    end
    else
        if (X_R ≥ Y_R) then
            X_R = right shift(X_R − Y_R)
        end
        else
            Y_R = right shift(Y_R − X_R)
        end
end
while (count > 0) do
    X_R = left shift(X_R)
    count = count − 1
end
```

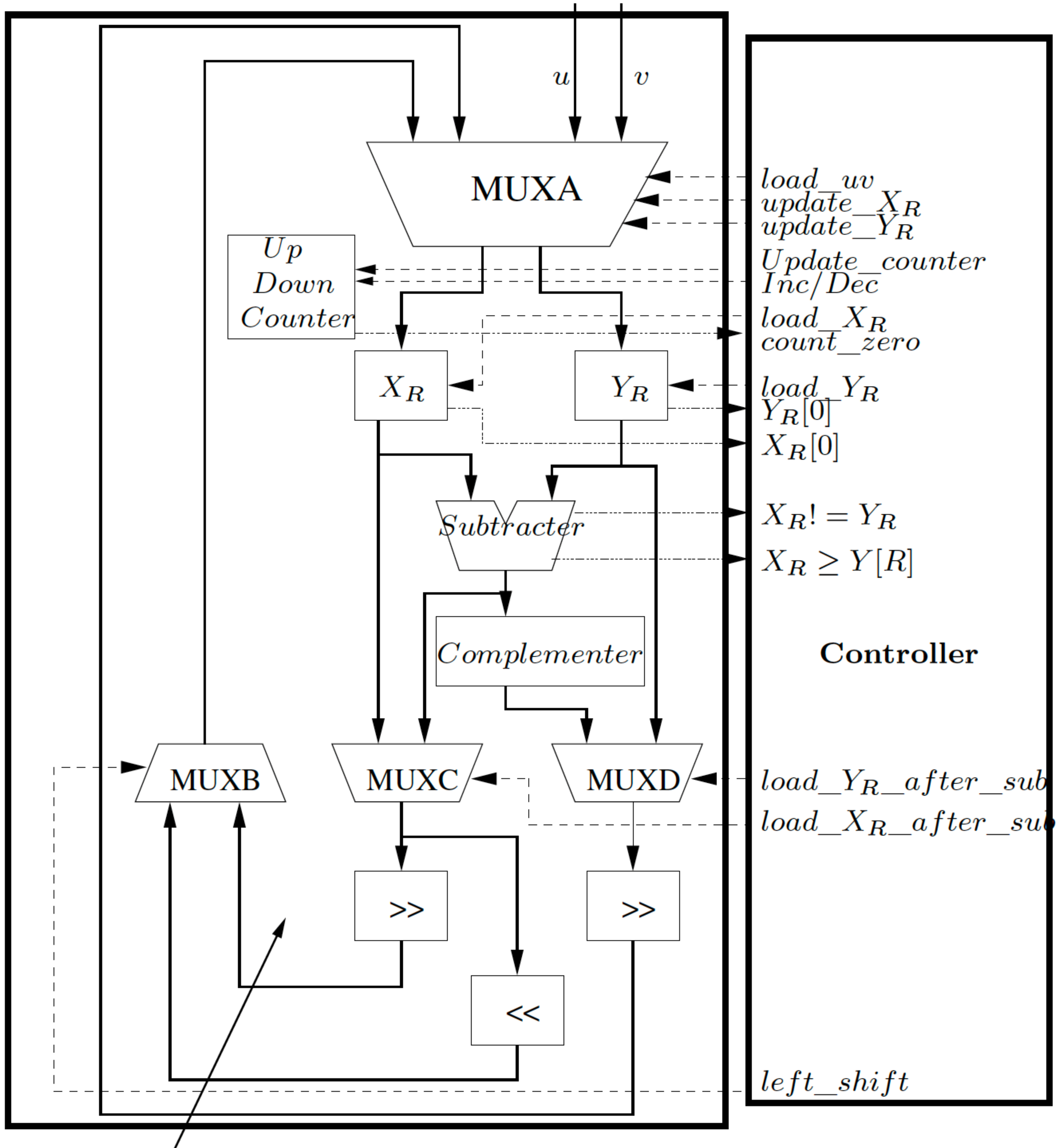
- **Required hardware components**
  - Two Registers for holding u and v (sequential)
- **Datapath elements**
  - Subtractor
  - Complementer
  - Right shifter
  - Left shifter
  - Counter (sequential)
  - Multiplexors — to route the control signals

Constructing the Controller

```
register X_R, Y_R;
X_R = u; Y_R = v; count = 0;                          /* State 0 */
while (X_R != Y_R) do
    if (X_R[0] = 0 and Y_R[0] = 0) then               /* State 1 */
        X_R = right shift(X_R)
        Y_R = right shift(Y_R)
        count = count + 1
    end
    else if (X_R[0] = 1 and Y_R[0] = 1) then          /* State 2 */
        Y_R = right shift(Y_R)
    end
    else if (X_R[0] = 0 and Y_R[0] = 1) then          /* State 3 */
        X_R = right shift(X_R)
    end
    else                                              /* State 4 */
        if (X_R ≥ Y_R) then
            X_R = right shift(X_R − Y_R)
        end
        else
            Y_R = right shift(Y_R − X_R)
        end
end
while (count > 0) do                                  /* State 5 */
    X_R = left shift(X_R)
    count = count − 1
end
```

- **Observation**
  - Whenever there is an if-else block, we allocate a state
  - Why? Because they dictate the control flow
  - The while loop is handled by the counter
    - Give the counter value to the controller and it tells you when to stop
  - **Note**: there might be cases when you have no if-else statements, but you have to sequentialize the computation
    - Remember the 4a+b example.
    - Whenever you need to sequentialise, allocate states.

Constructing the Hardware Diagram



Input: Integers u and v

Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

register X_R, Y_R;

$X_R = u$; $Y_R = v$; $count = 0$;

while $(X_R != Y_R)$ **do**

  if $(X_R[0] = 0$ and $Y_R[0] = 0)$ **then**

    $X_R =$ right shift(X_R)

    $Y_R =$ right shift(Y_R)

    $count = count + 1$

  end

  else if $(X_R[0] = 1$ and $Y_R[0] = 1)$ **then**

    $Y_R =$ right shift(Y_R)

  end

  else if $(X_R[0] = 0$ and $Y_R[0] = 1)$ **then**

    $X_R =$ right shift(X_R)

  end

  else

    if $(X_R \geq Y_R)$ **then**

      $X_R =$ right shift$(X_R - Y_R)$

    end

    else

      $Y_R =$ right shift$(Y_R - X_R)$

    end

end

while $(count > 0)$ **do**

  $X_R =$ left shift(X_R)

  $count = count - 1$

end

Constructing the State Transition Table

Present State	Next State					Output Signals										
	0____	100_	110_	101_	111_	*load uv*	*update* X_R	*update* Y_R	*load* X_R	*load* Y_R	*load_*X_R *after_sub*	*load_*Y_R *after_sub*	*Update counter*	*Inc /Dec*	*left shift*	*count zero*
S_0	S_5	S_1	S_2	S_3	S_4	1	0	0	1	1	0	0	0	_	_	_
S_1	S_5	S_1	S_2	S_3	S_4	0	1	1	1	1	0	0	1	1	_	_
S_2	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	0	0	_	_	_
S_3	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	0	0	0	_	_	_
S_4 $(X_R \geq Y_R)$	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	1	0	0	_	_	_
S_4 $(X_R < Y_R)$	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	1	0	_	_	_
S_5	S_5	S_5	S_5	S_5	S_5	0	0	0	0	0	0	0	1	0	1	0

- **Controller receives 4 input signals**
  - XR != YR
  - XR[0]
  - YR[0]
  - XR >= YR

Now go and code it down