Computer Architecture

CH3 Computer Arithmetic (II)
Integer Multiplication and Division

Prof. Ren-Shuo Liu
NTHU EE
Fall 2017

Outline

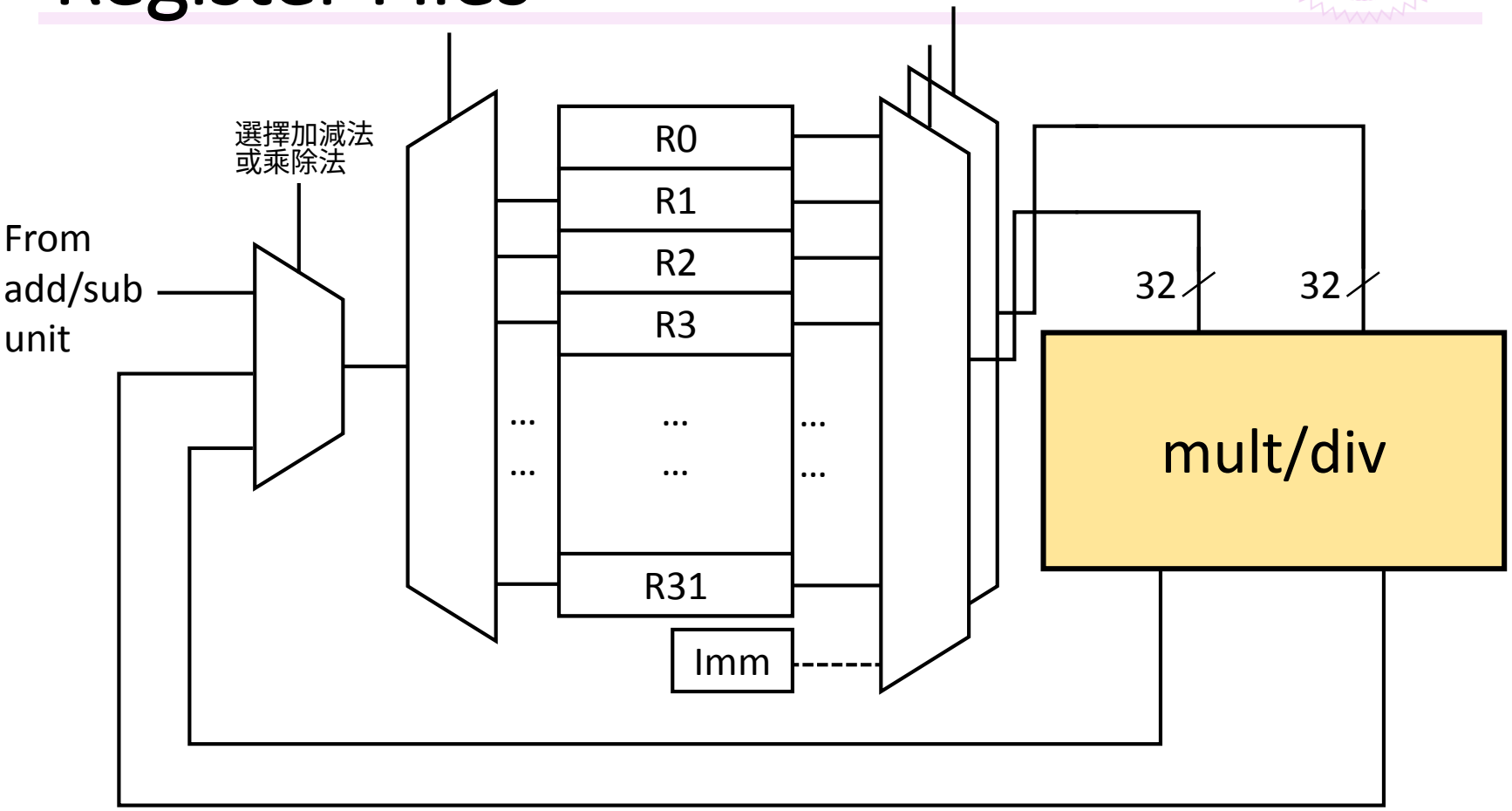
- Multiplication
  - Shift-add multiplier
- Division
  - Shift-subtract divider

RISC-V Instructions

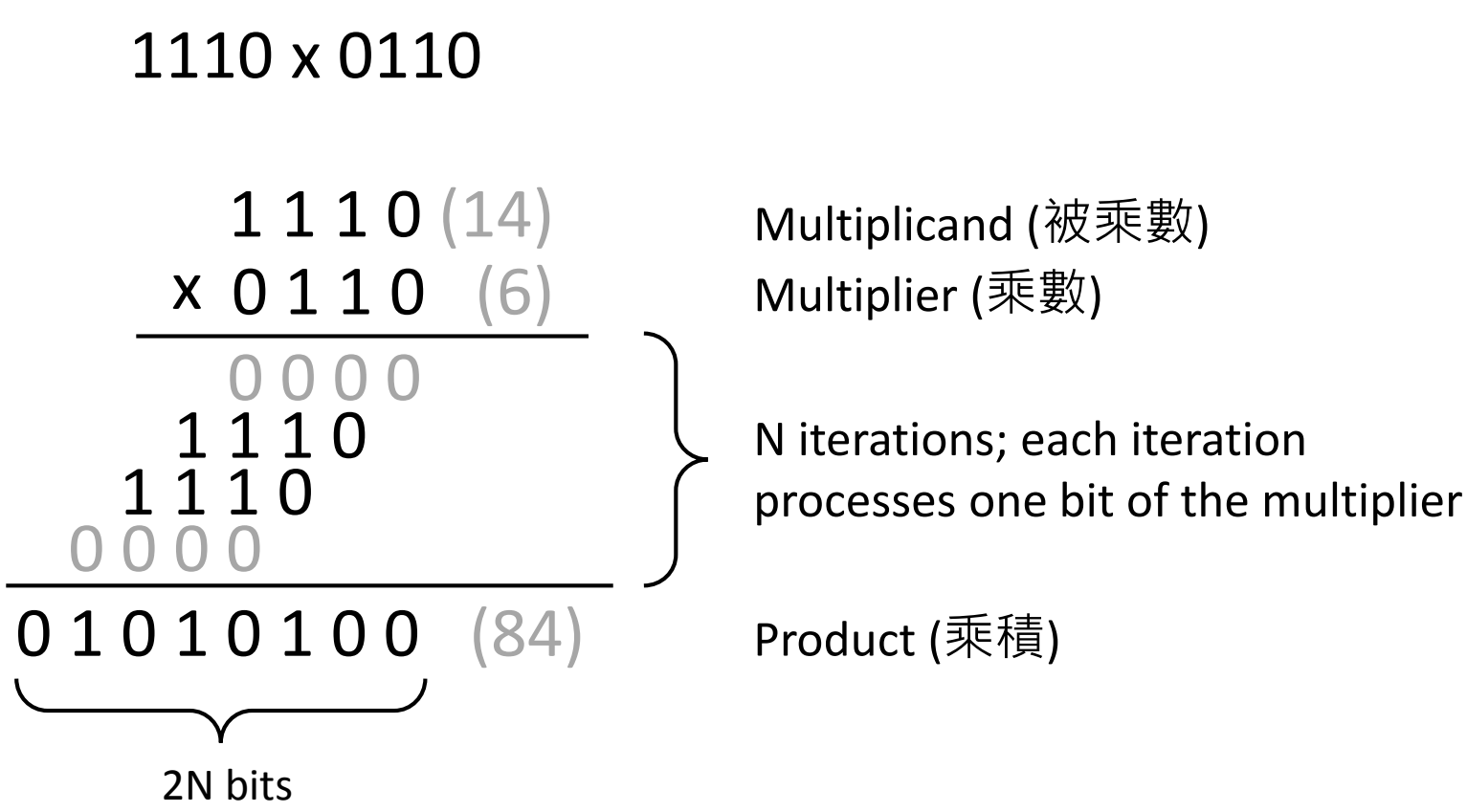
- Multiplication 共64bits，可選擇乘上面32bits或下面32bits
  - `mulh  $d, $s1, $s2 # $d=$s1*$s2 (high 32-bit)`
  - `mul   $d, $s1, $s2 # $d=$s1*$s2 (low 32-bit)`

- Division
  - `div $d, $s1, $s2   # $d = $s1 / $s2`
  - `rem $d, $s1, $s2   # $d = $s1 % $s2`

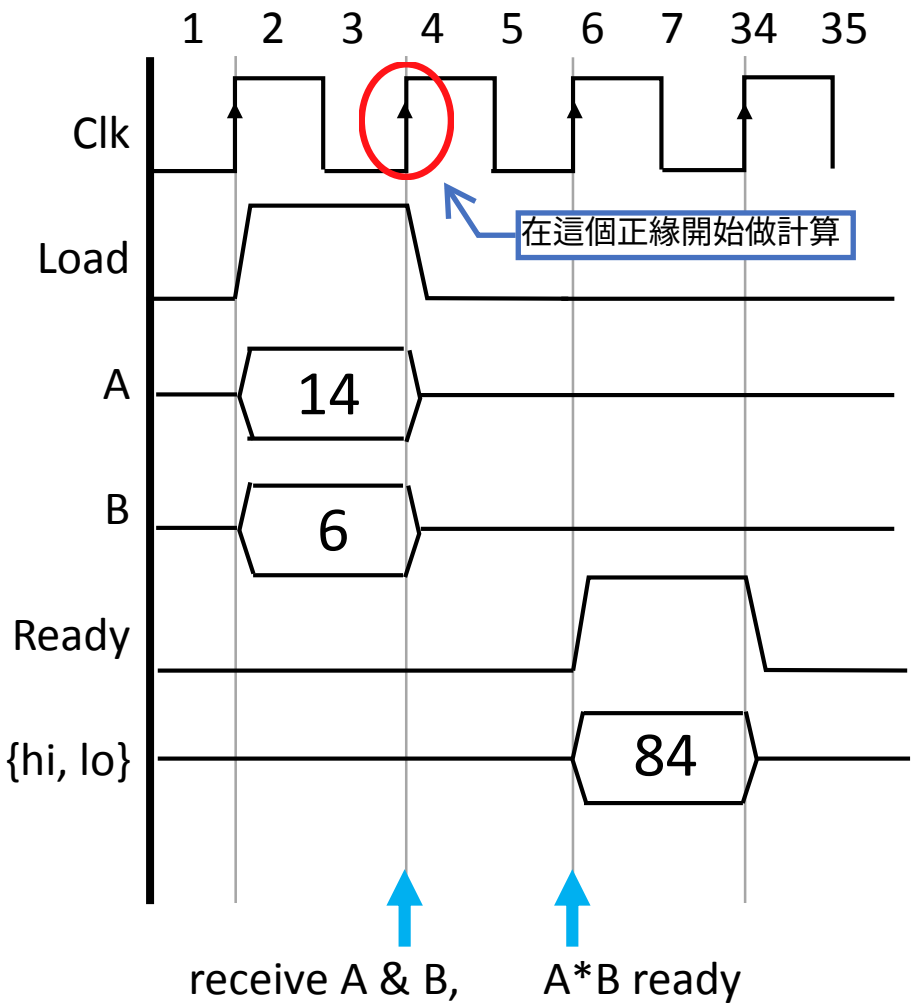
Multiply Unit, Divide Unit, and Register Files

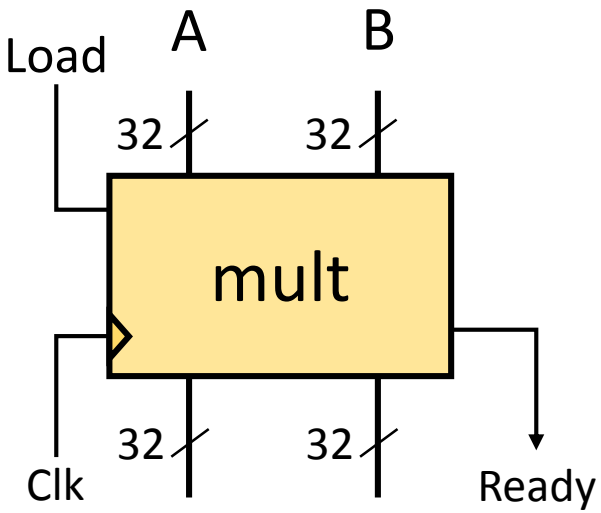


Basic N-Bit Multiplication

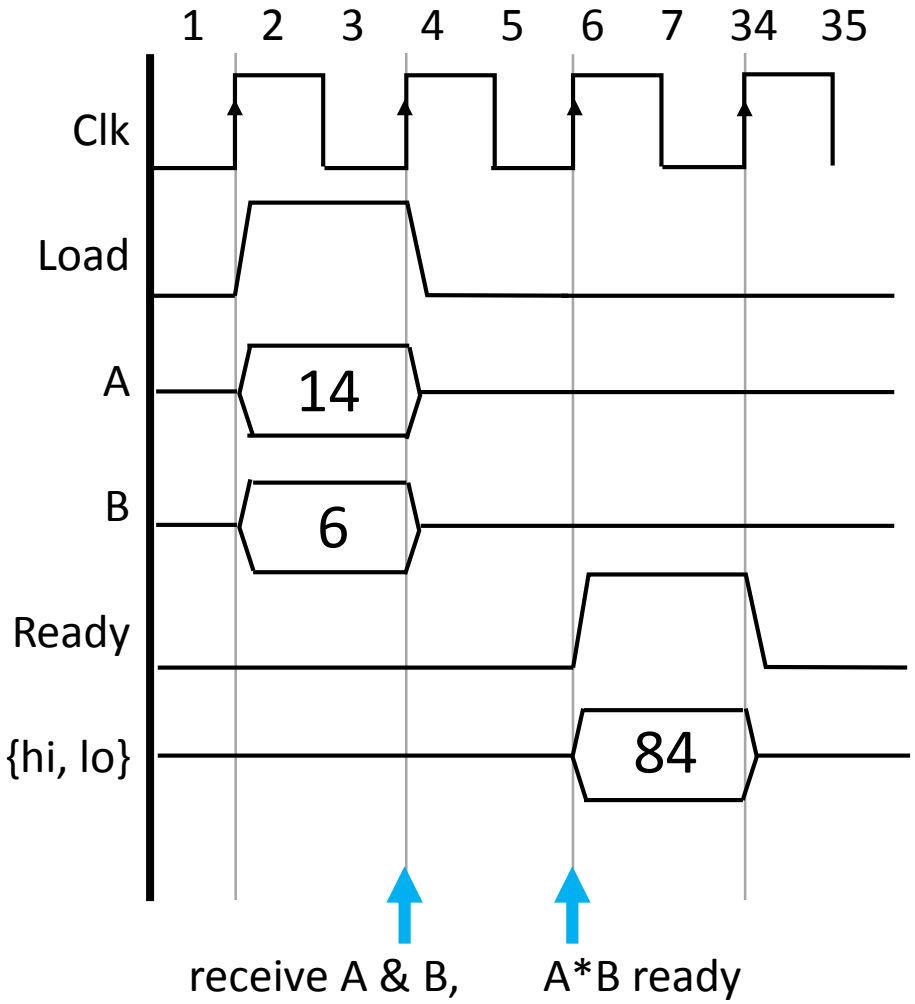


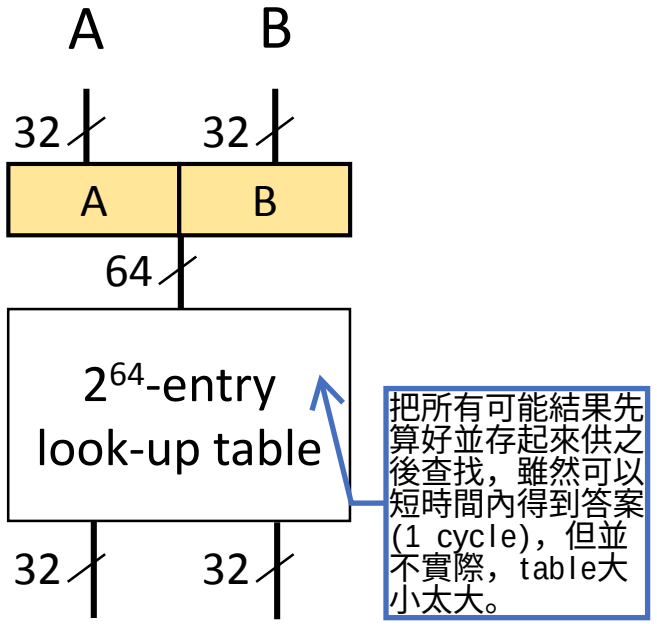
Single-Step Multiply





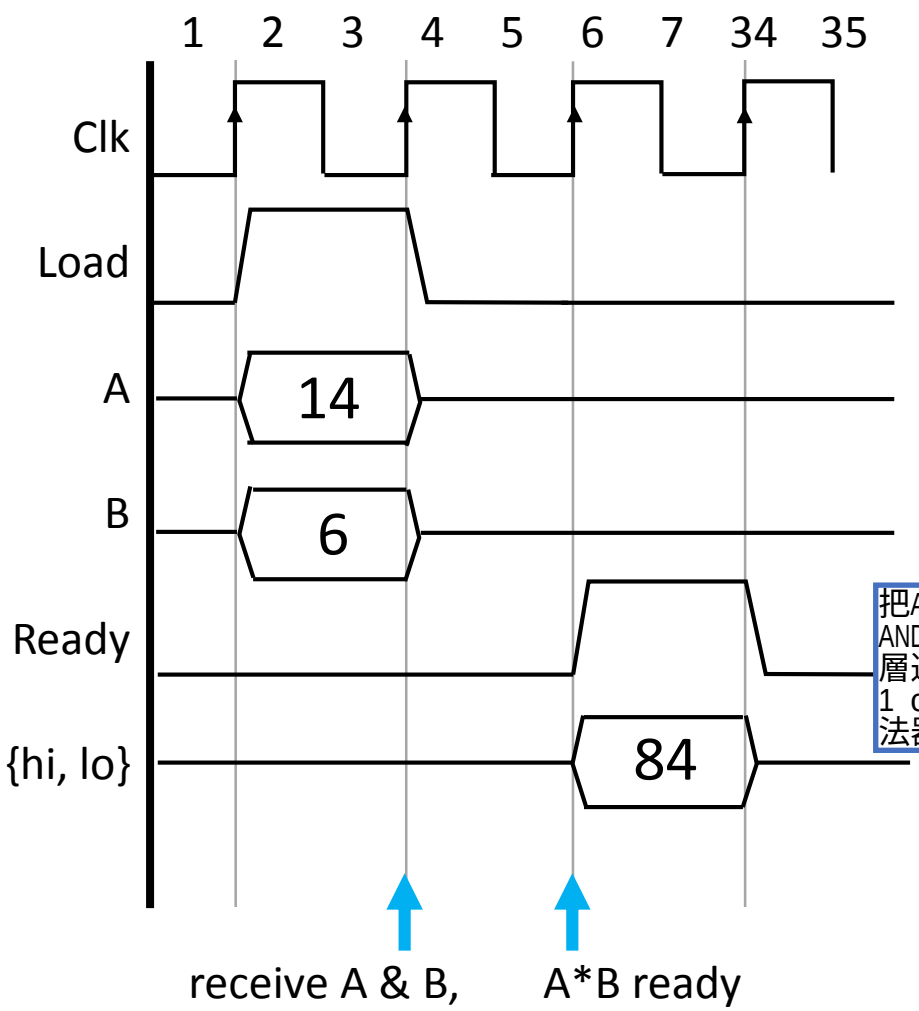
Single-Step Multiply (Table-Based)

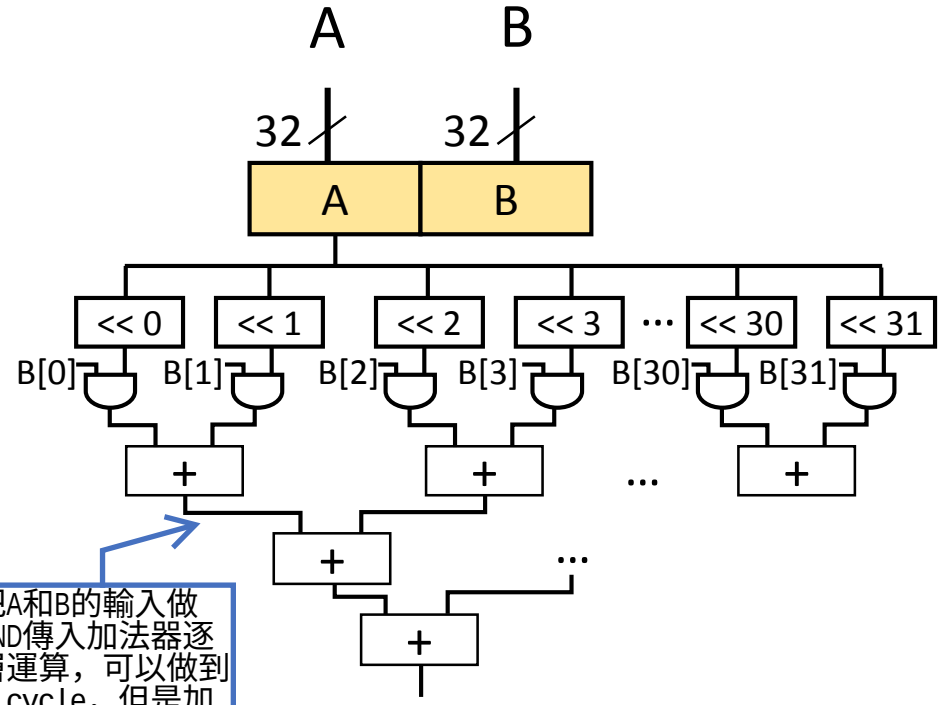




- Theoretically doable
- Rreasonable for small N (e.g., 4)
- **Impractical** for large N (e.g., 32)

Single-Step Multiply (Adder-Based)

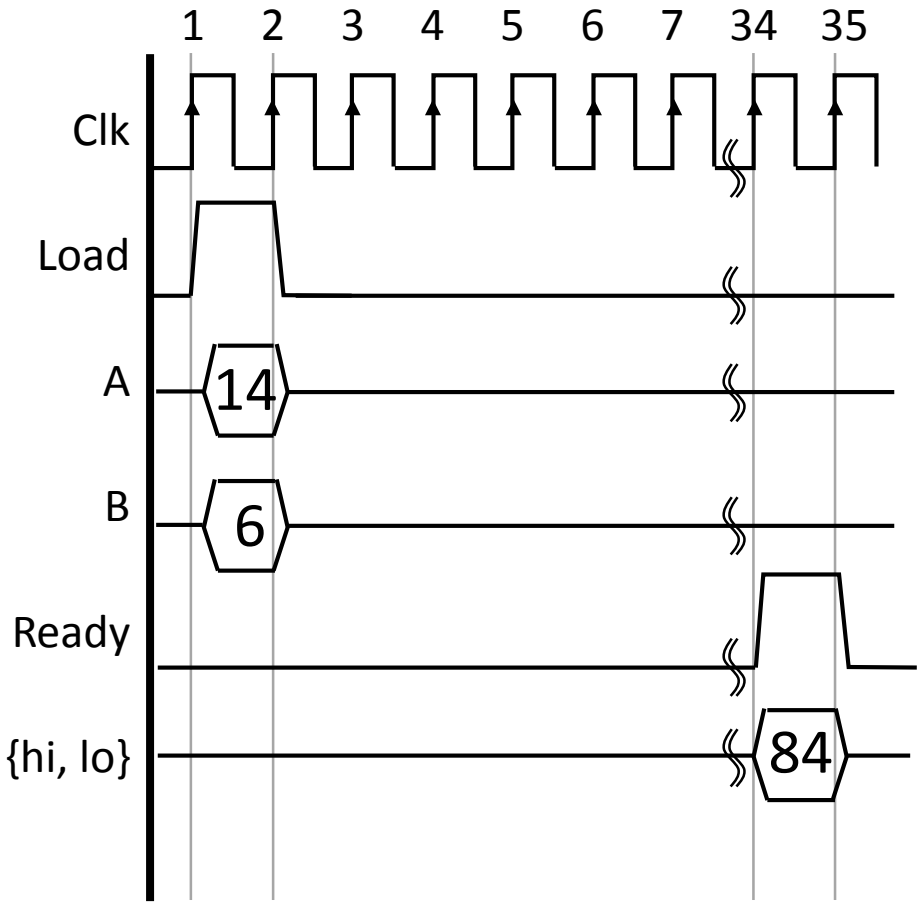


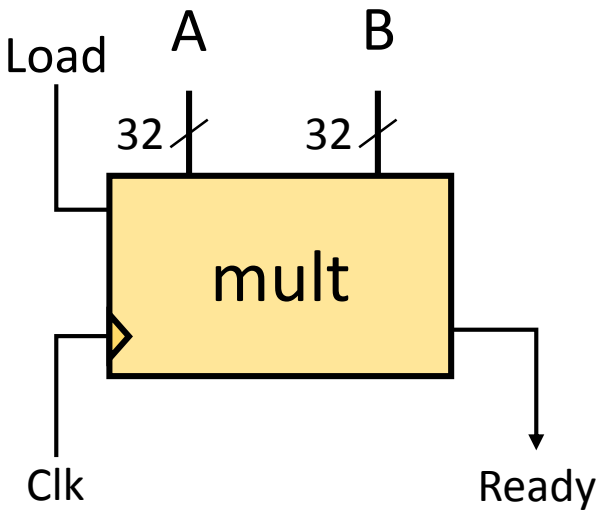


- Total (16+8+4+2+1)=31 adders

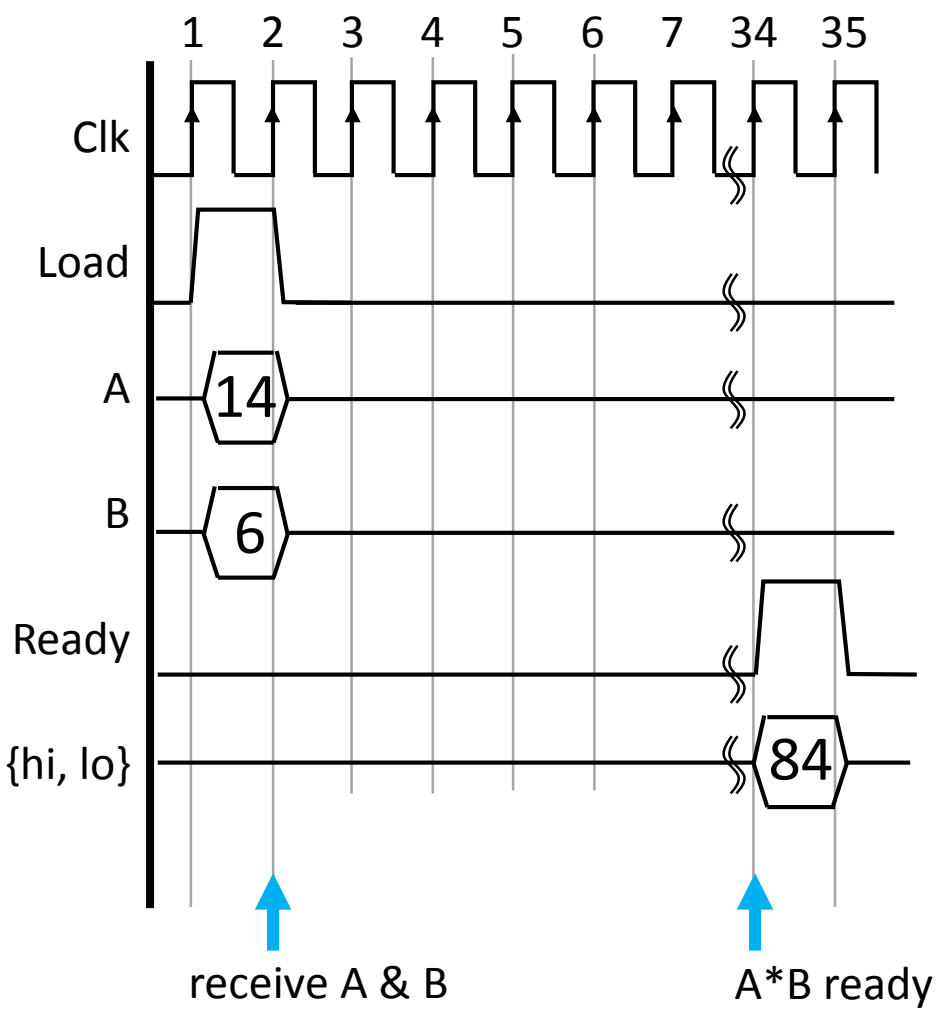
若要求在2cycle內做8bit相乘運算:
則需要4個Adders，拆成一半去做

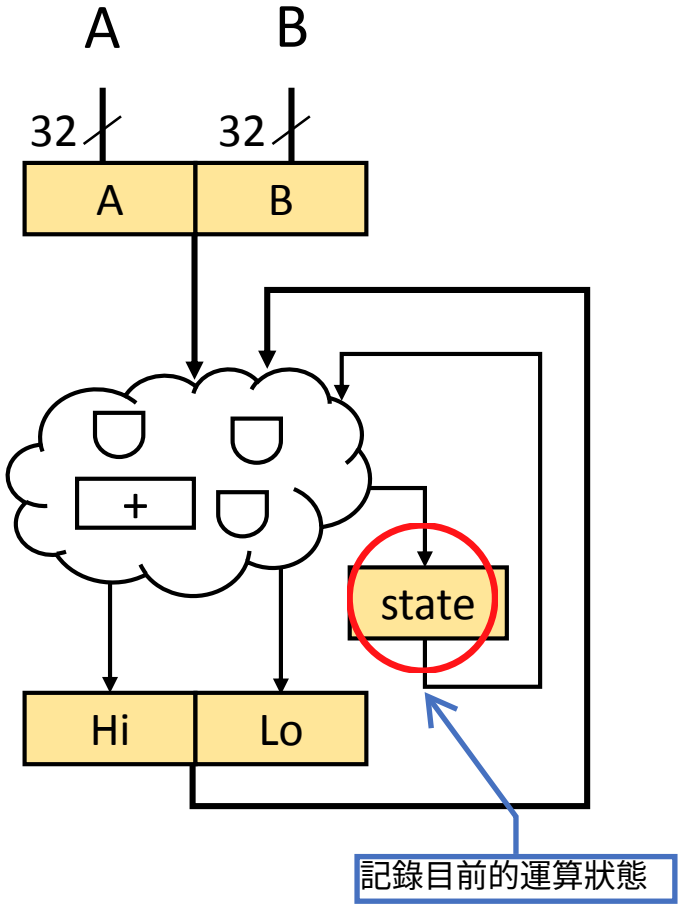
N-Step Multiply



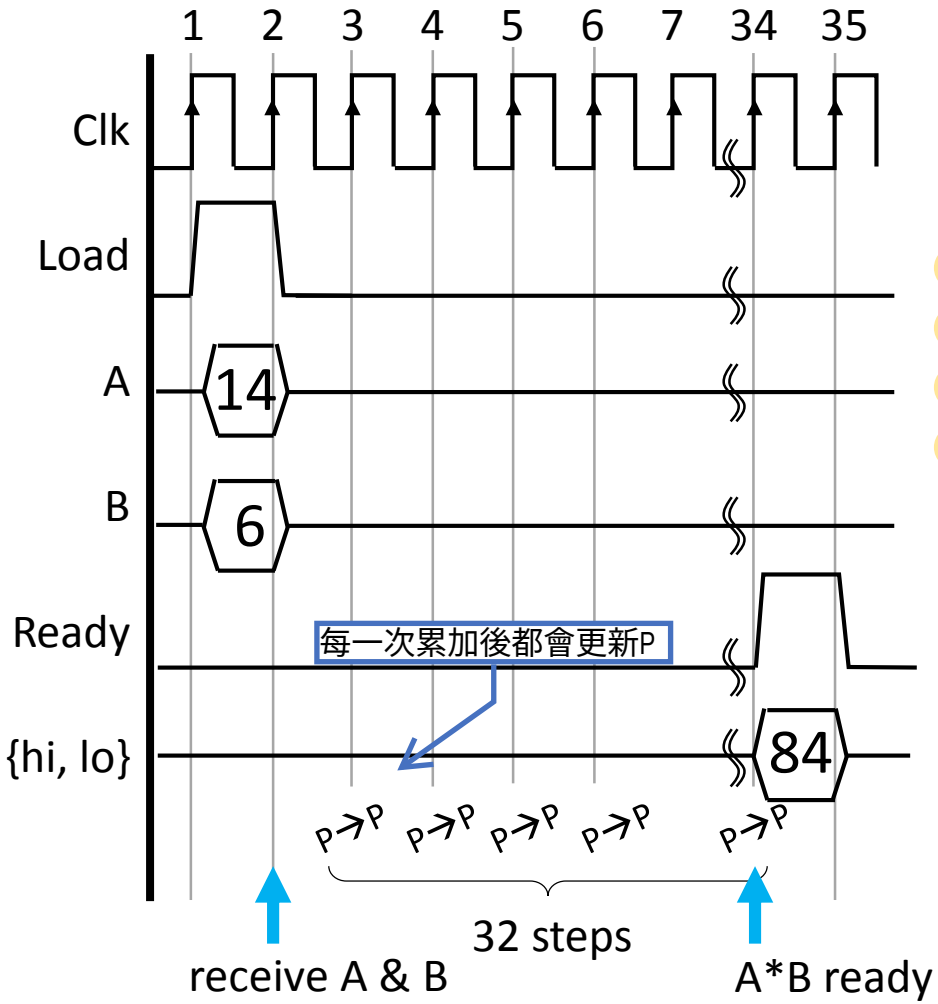


N-Step Multiply





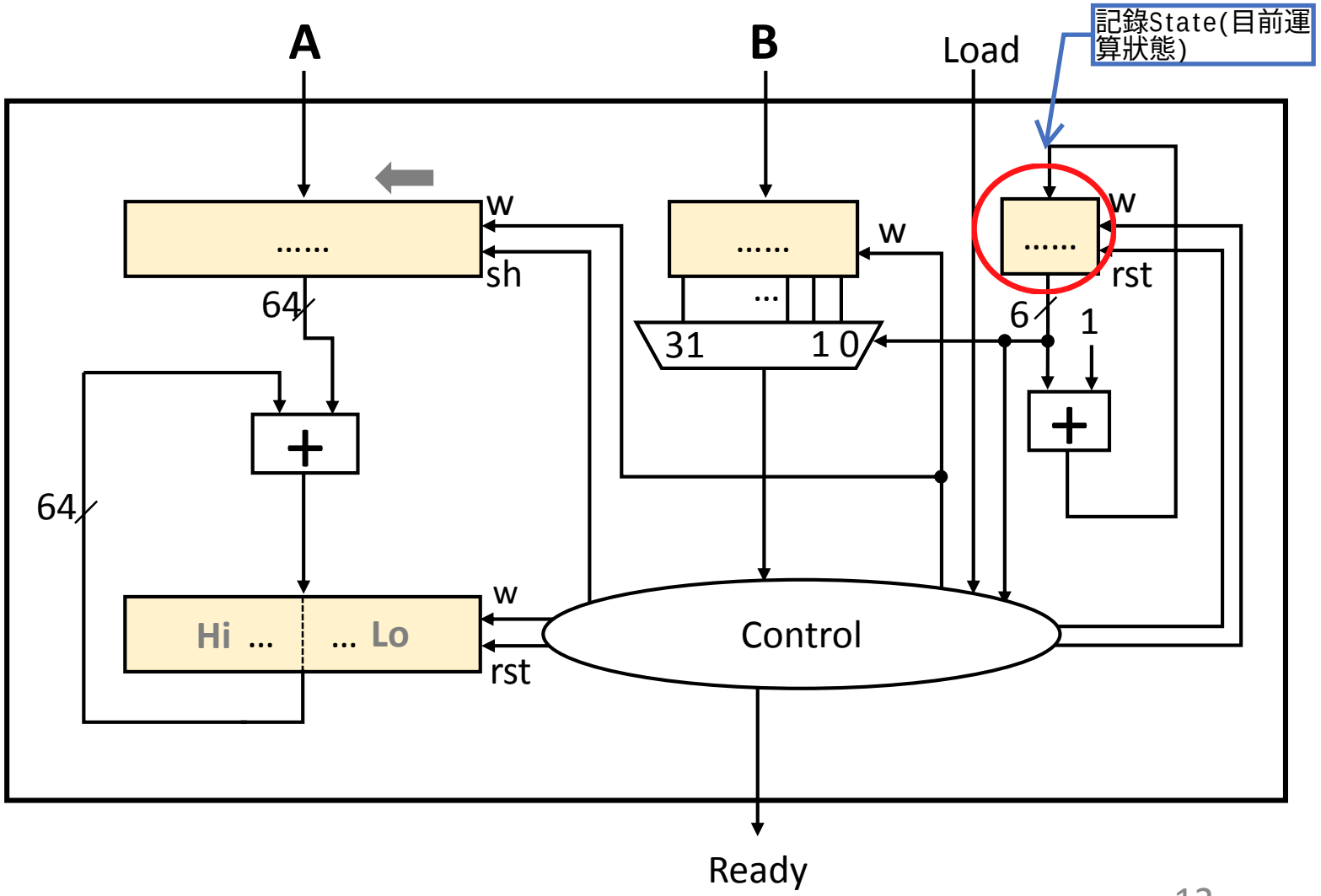
N-Step Multiply



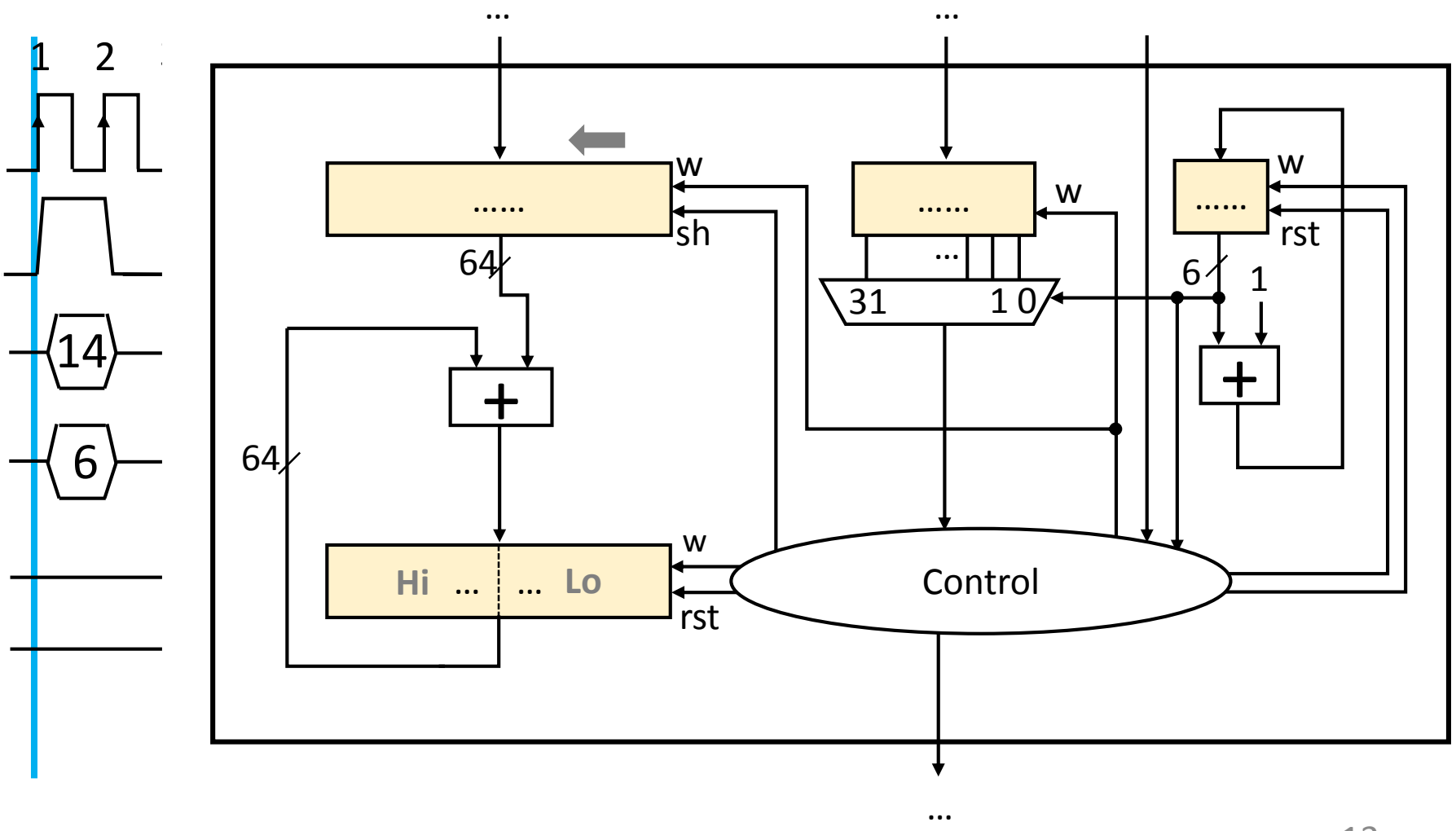
P stands for {Hi, Lo}

```
          1 1 1 0
        x 0 1 1 0
        ---------
P = P     0 0 0 0
P = P + A<<1)   1 1 1 0
P = (P + A<<2)  1 1 1 0
P = P   0 0 0 0
        ---------
  0 1 0 1 0 1 0 0
```

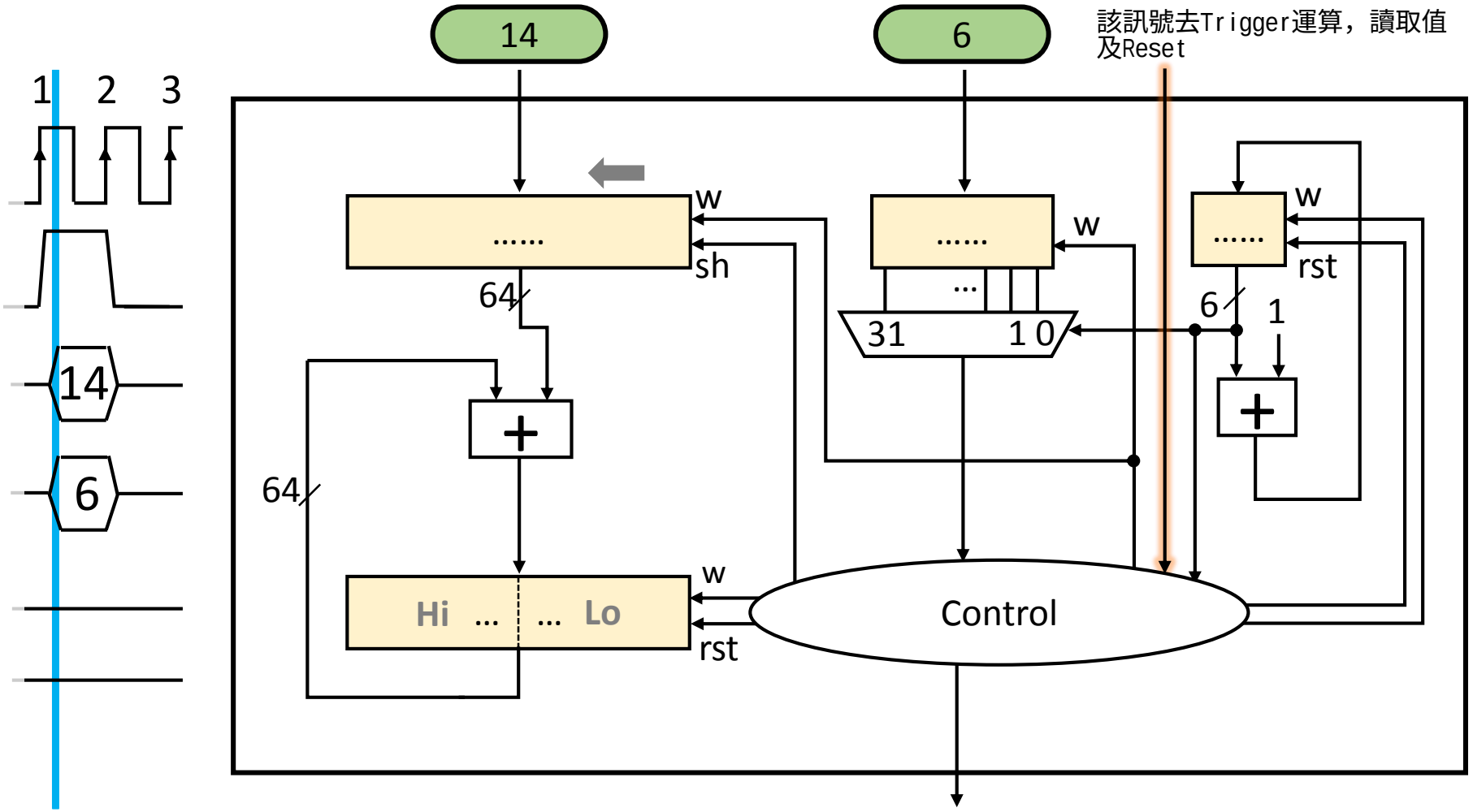
Basic Multiply Unit



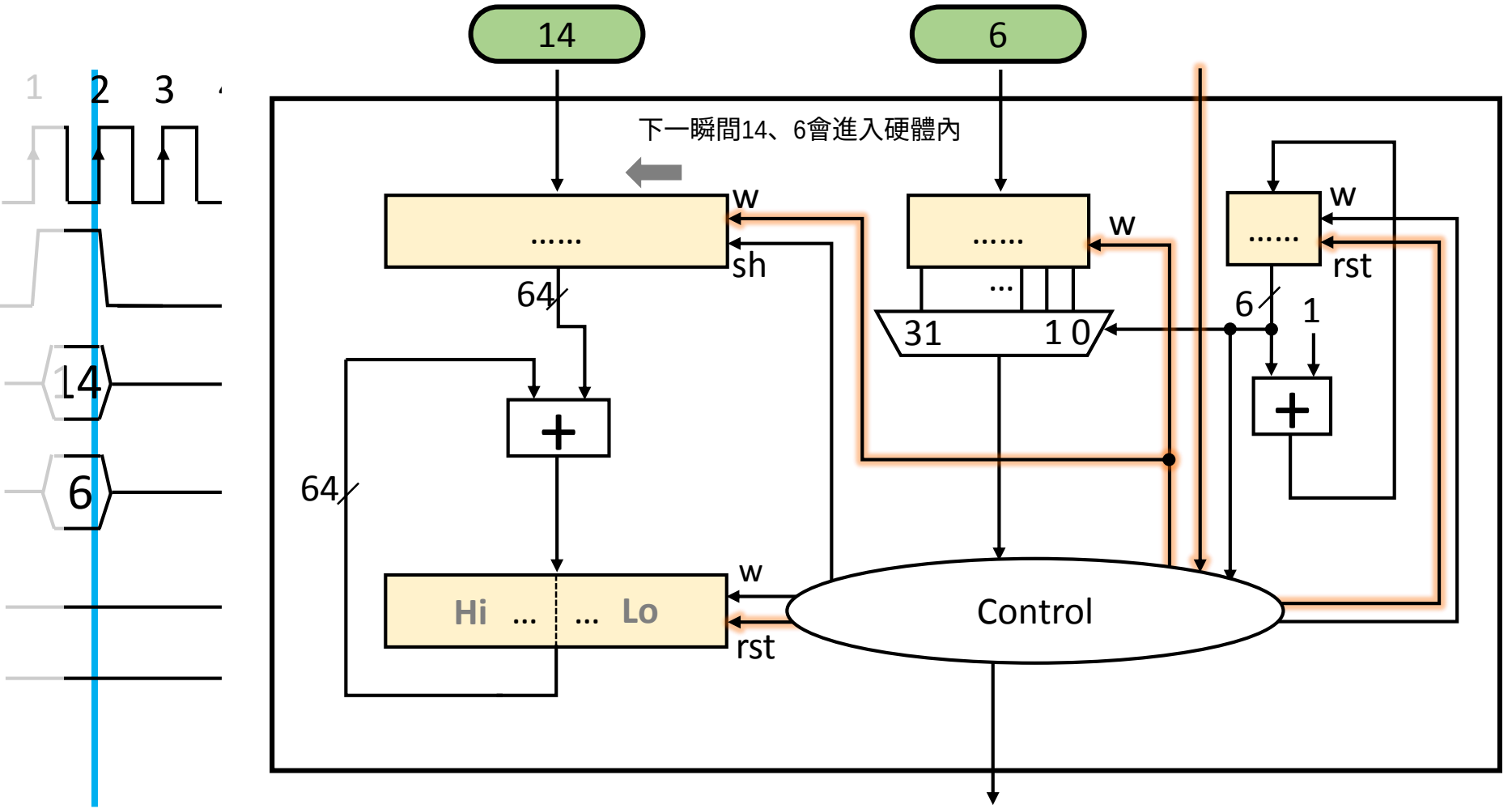
t = 0.99



t = 1.2

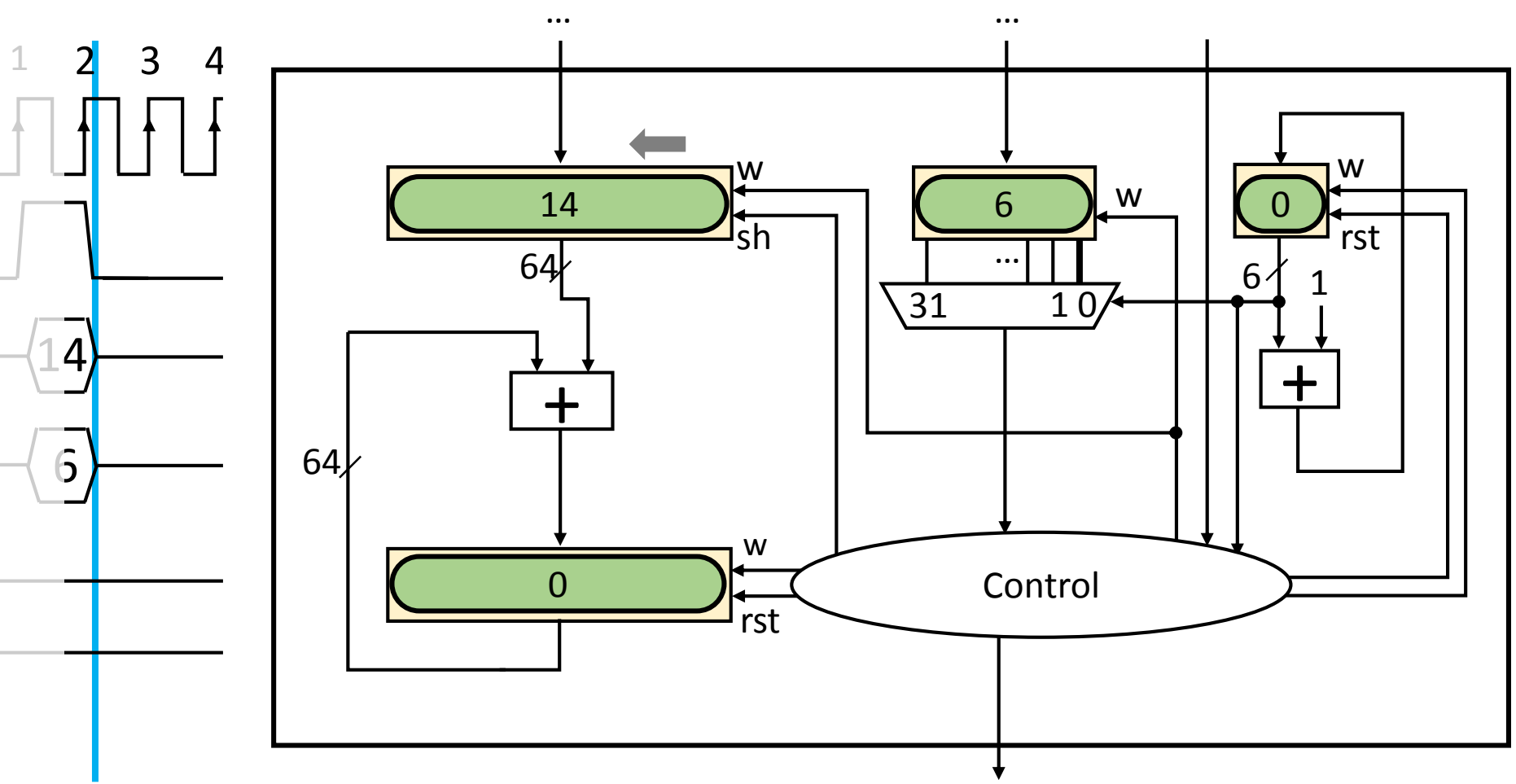

1
2
3
14
6
14
6
該訊號去Trigger運算，讀取值及Reset
.....
w
sh
64
.....
w
...
31
1 0
.....
w
rst
6
1
+
64
+
w
Hi ... ... Lo
rst
Control

t = 1.99


1
2
3
14
6
14
6
下一瞬間14、6會進入硬體內
w
.....
sh
64
64
+
w
Hi ... ... Lo
rst
Control
w
.....
31
1 0
w
.....
rst
6
1
+

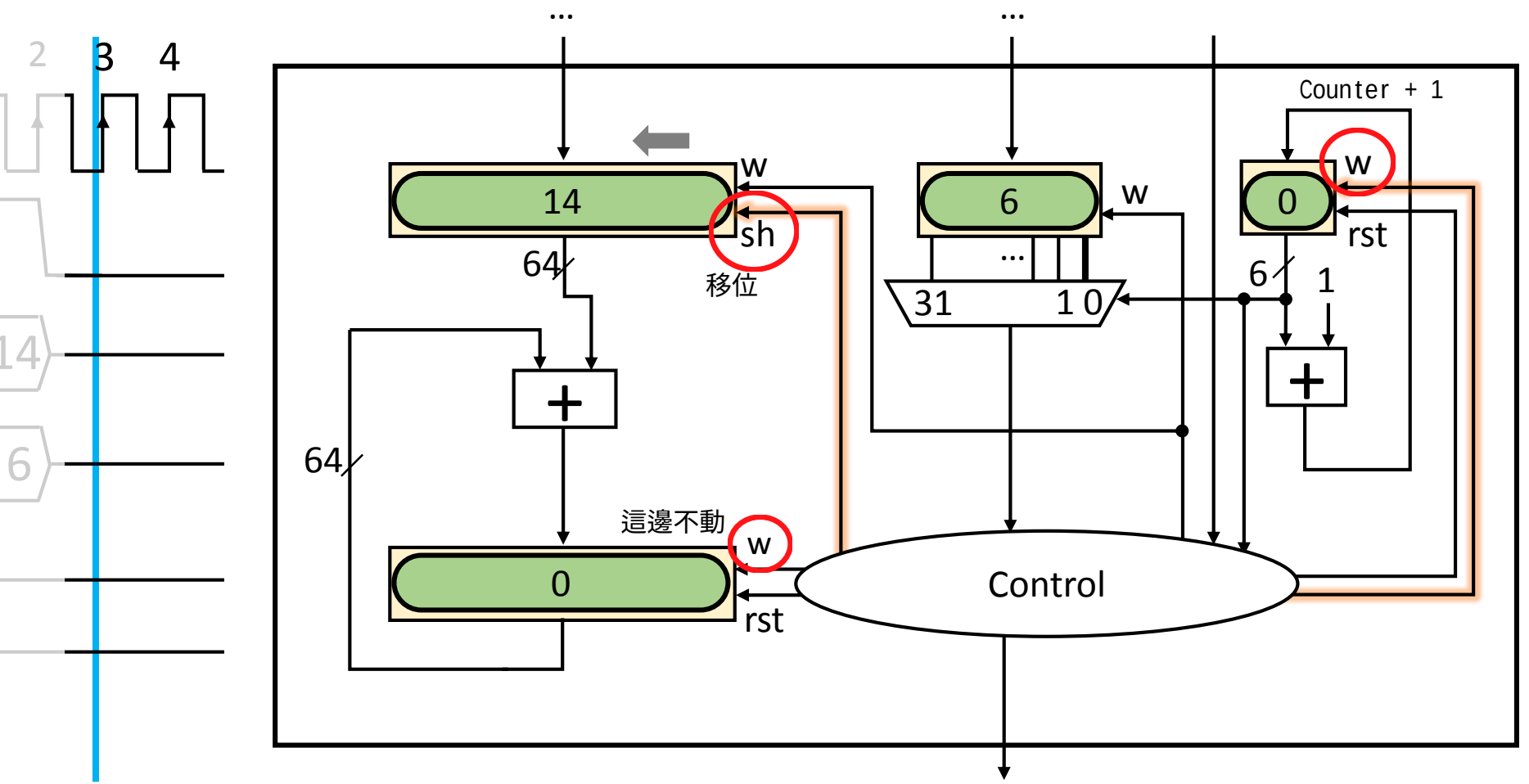
t = 2.2

- Received A & B. Reset Hi & Lo.
- Start preparing next {Hi, Lo} based on B[0]=0.



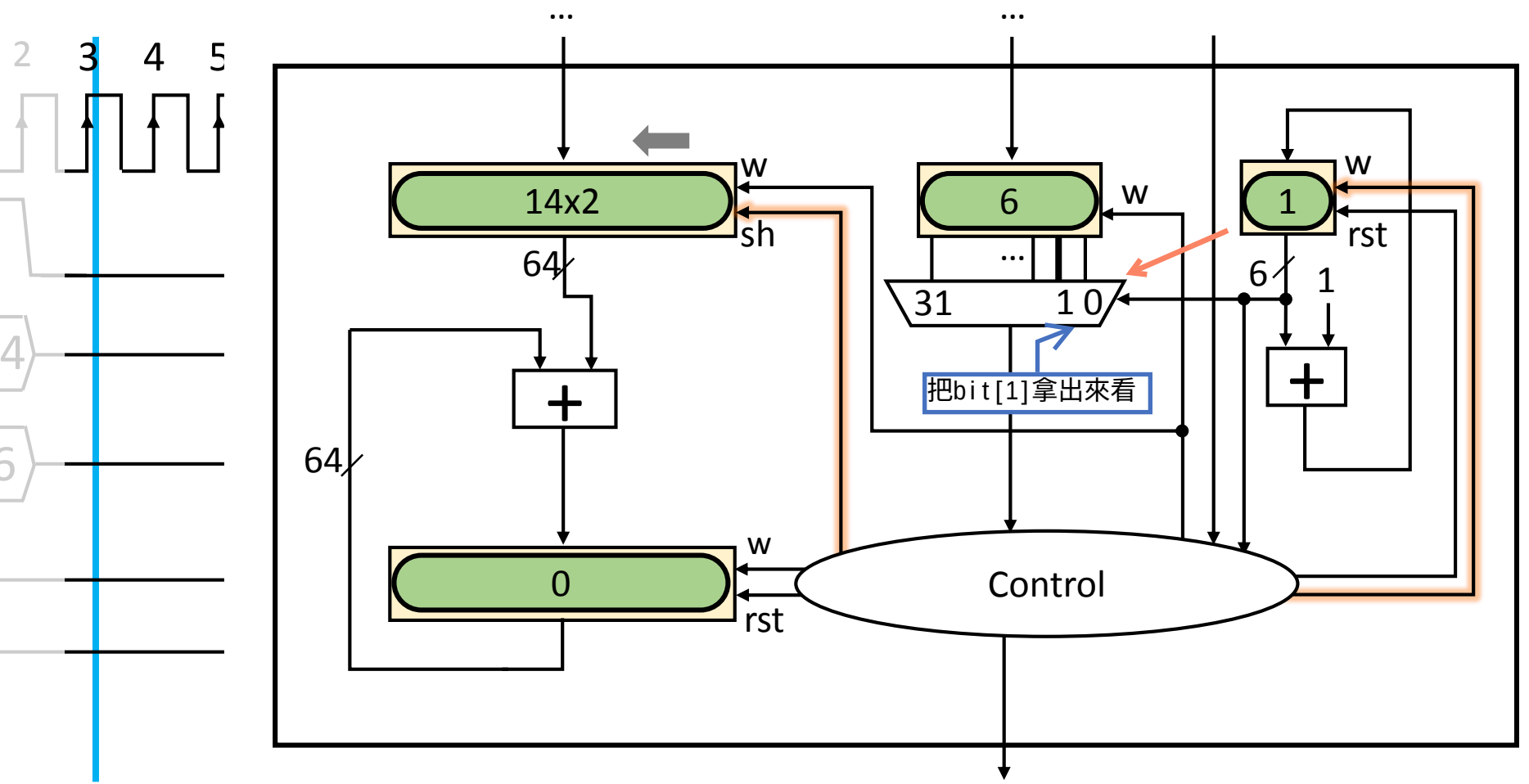
t = 2.99

- Prepared next {Hi, Lo} based on B[0]=0.
- Start transitioning to the next {Hi, Lo}



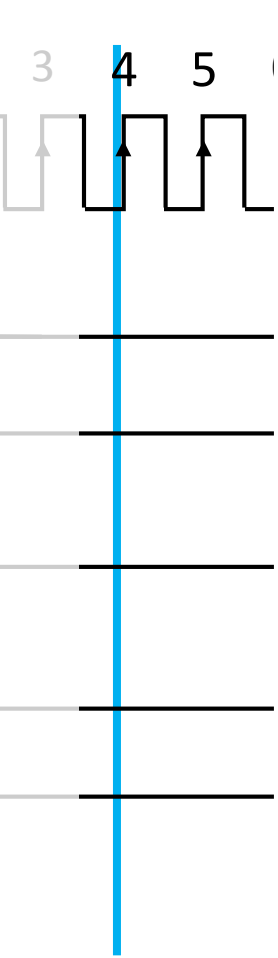
t = 3.2

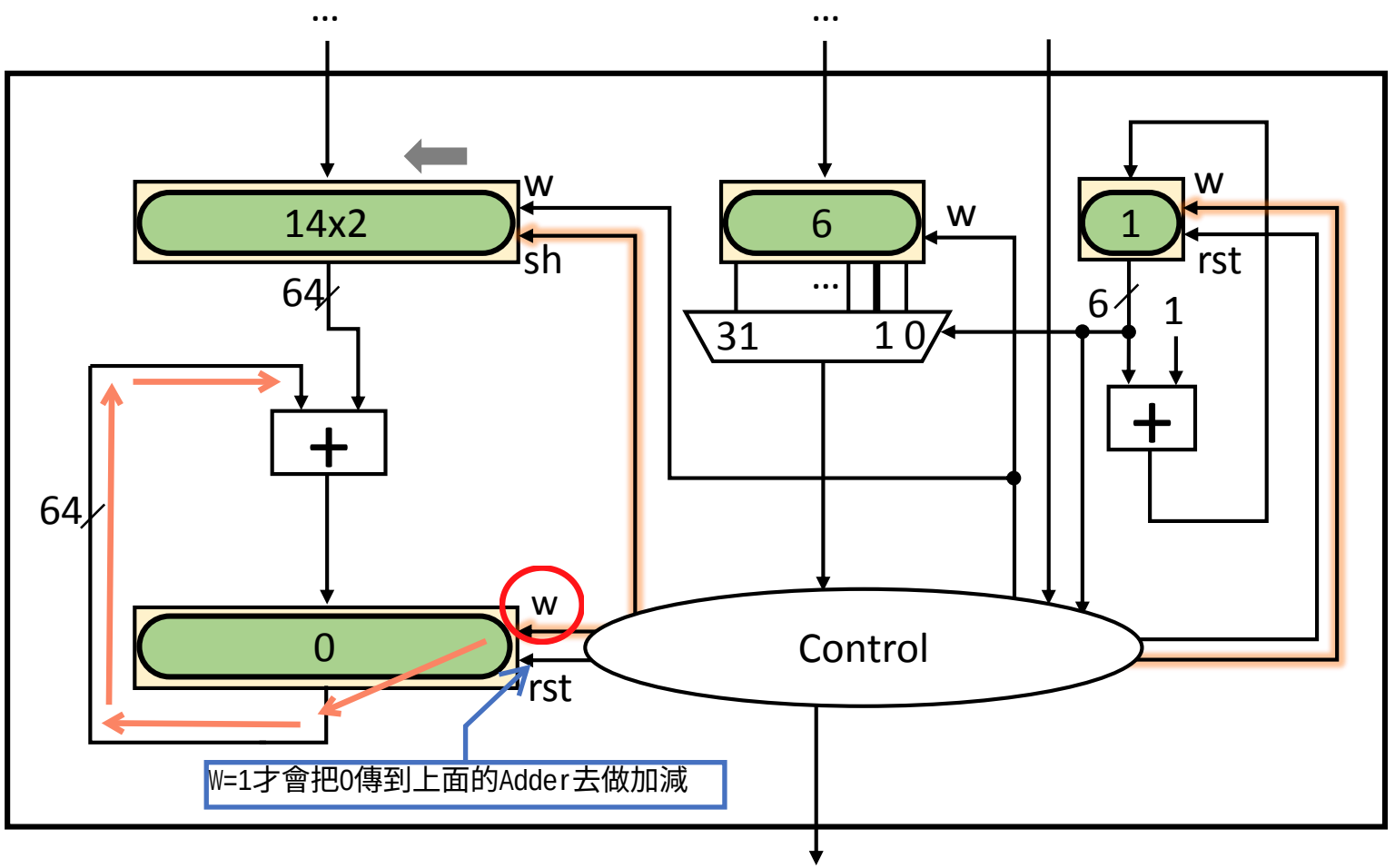
- Hi & Lo changed.
- Start preparing next {Hi, Lo} based on B[1]=1.



t = 3.99

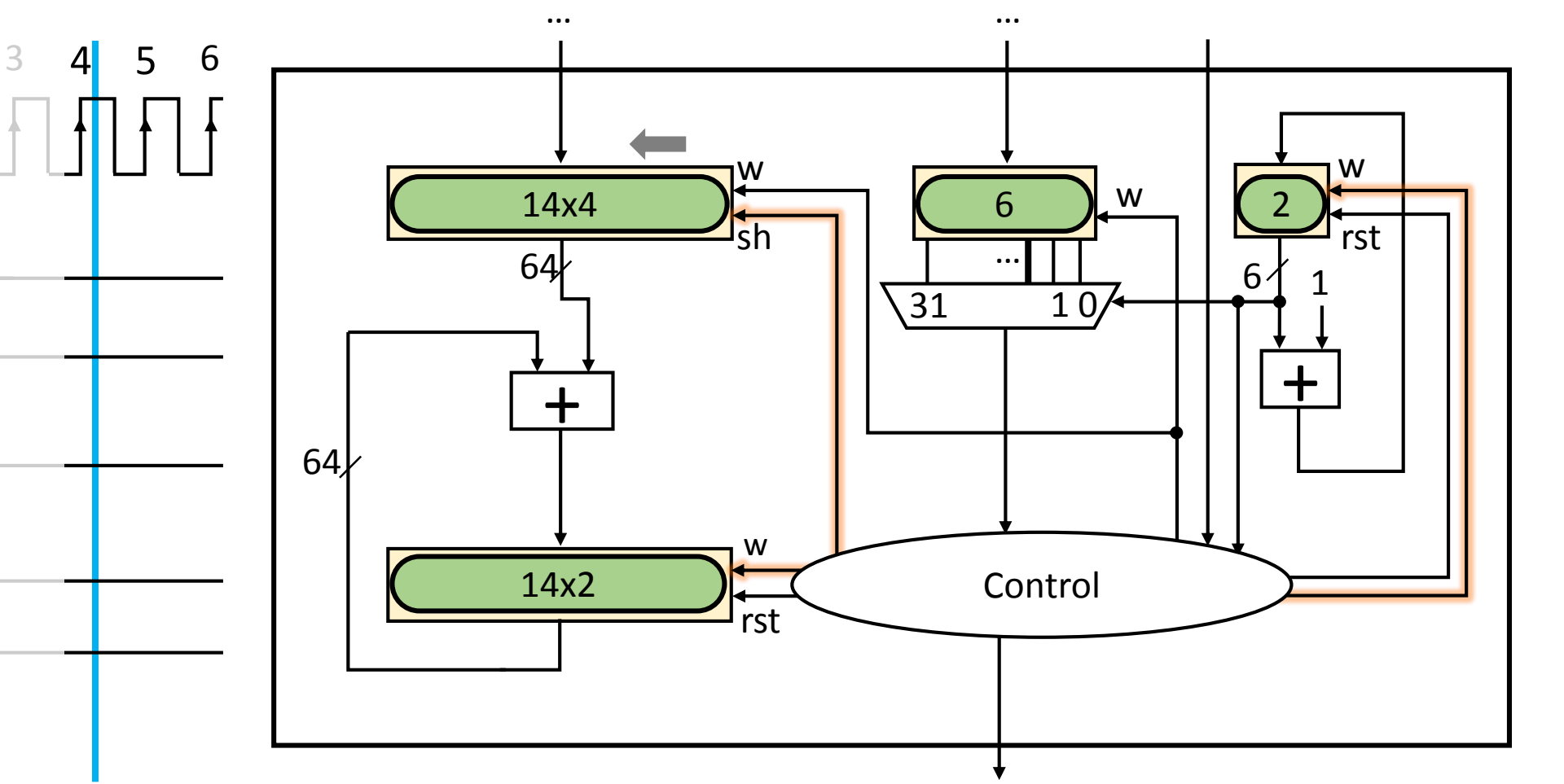
- Prepared next {Hi, Lo} based on B[1]=1.
- Start transitioning to the next {Hi, Lo}





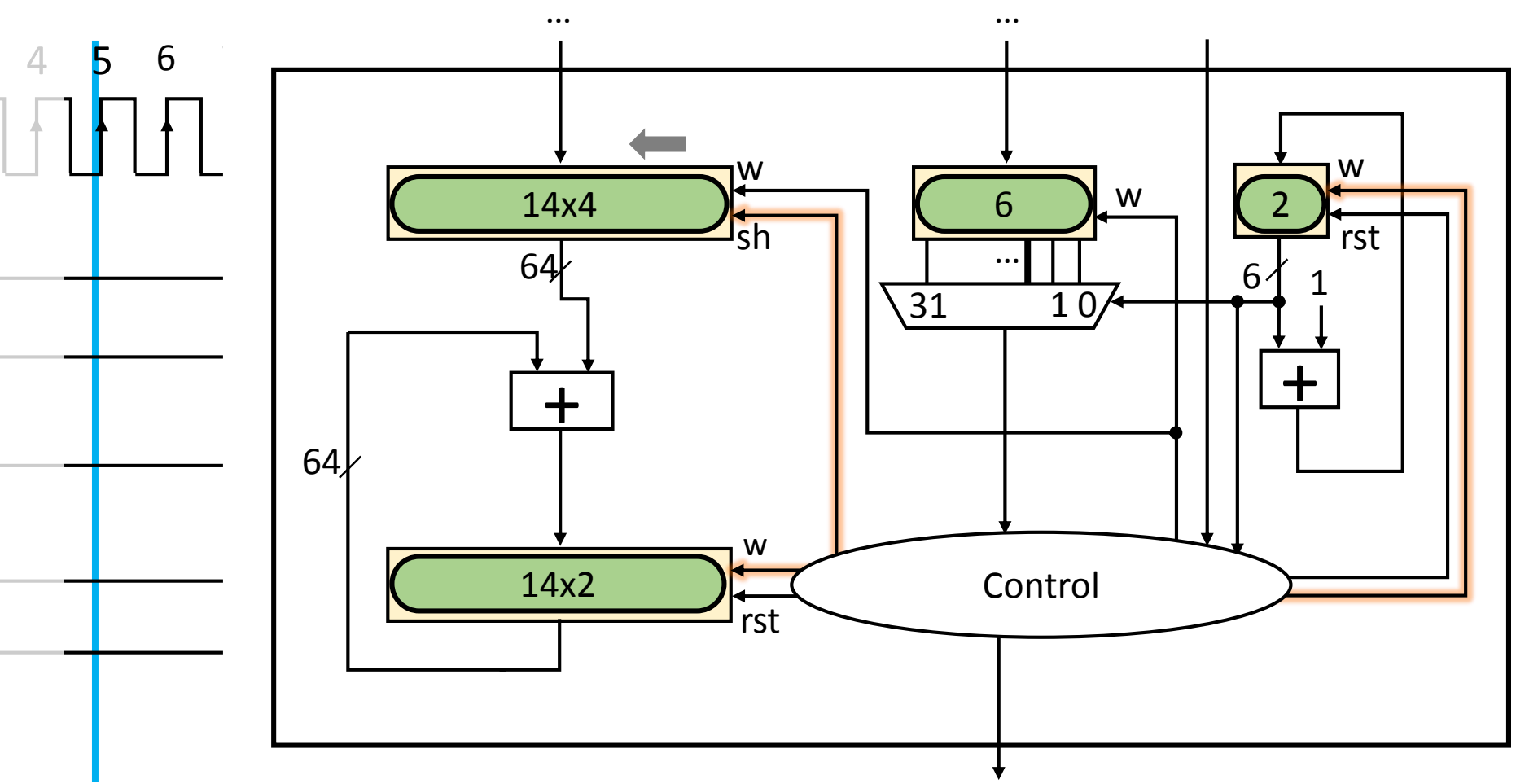
t = 4.2

- Hi & Lo changed.
- Start preparing next {Hi, Lo} based on B[2]=1.



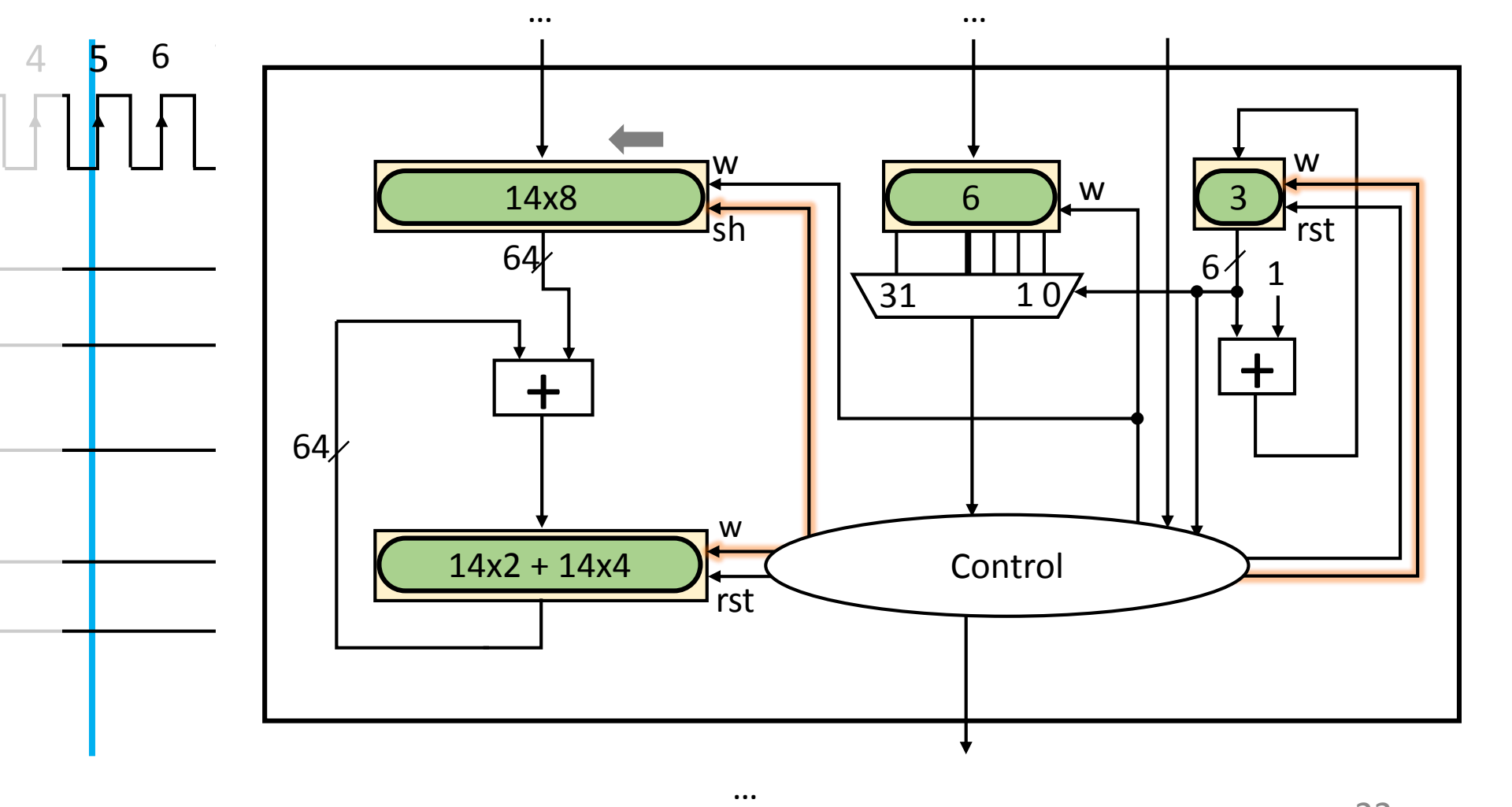
t = 4.99

- Prepared next {Hi, Lo} based on B[2]=1.
- Start transitioning to the next {Hi, Lo}



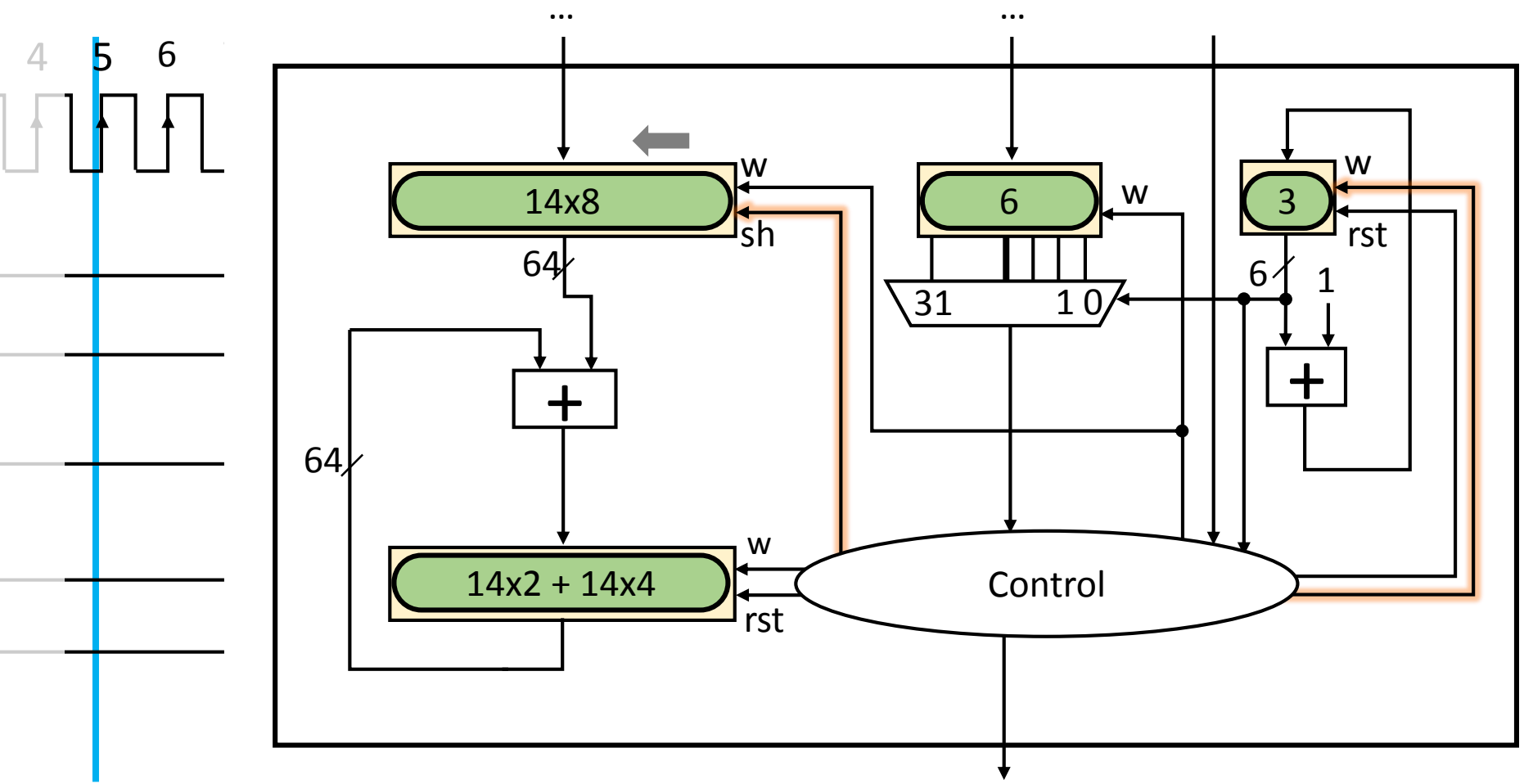
t = 5.2

- Hi & Lo changed.
- Start preparing next {Hi, Lo} based on B[3]=0.



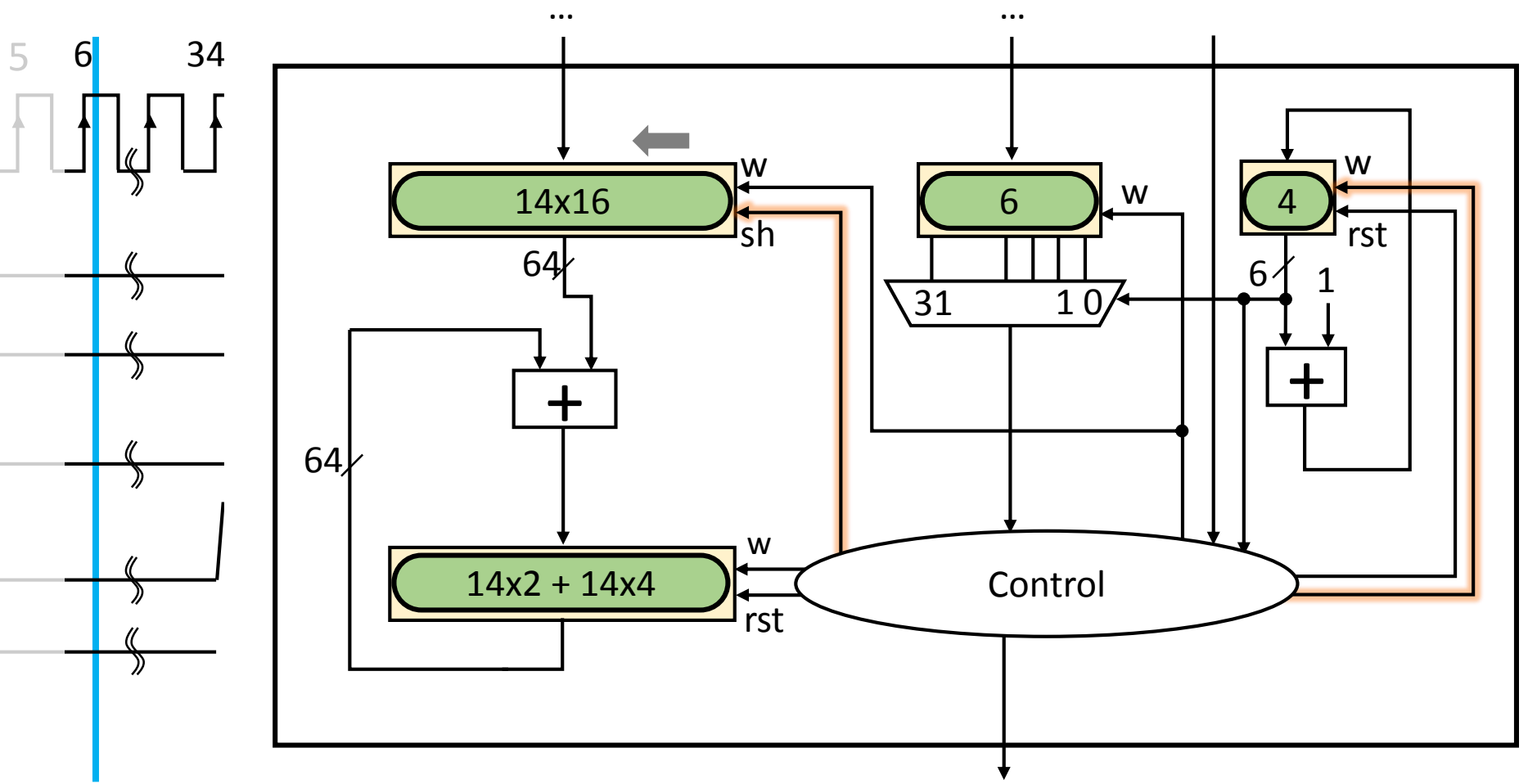
t = 5.99

- Prepared next {Hi, Lo} based on B[3]=0.
- Start transitioning to the next {Hi, Lo}



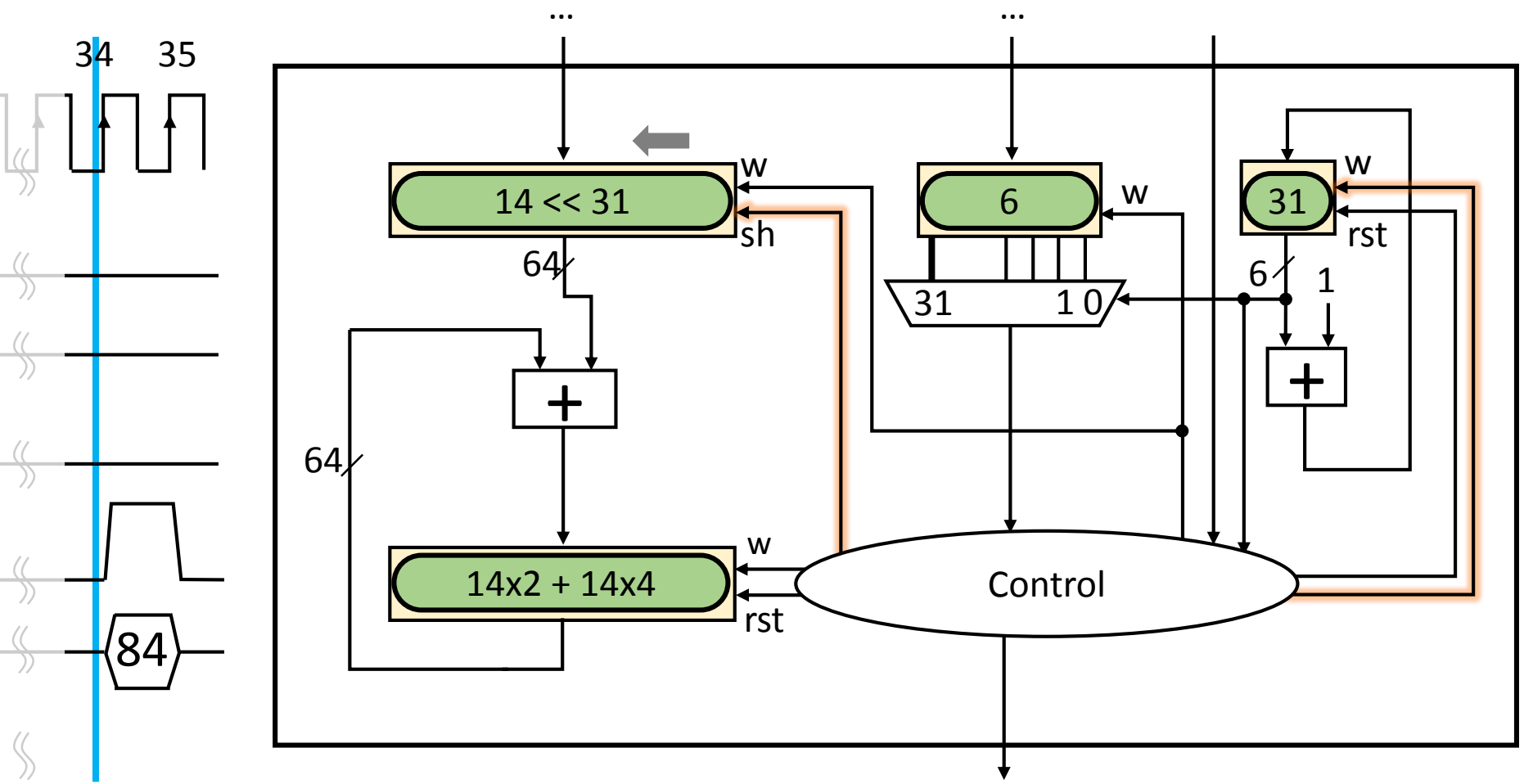
t = 6.2

- Hi & Lo changed.
- Start preparing next {Hi, Lo} based on B[4]=0.



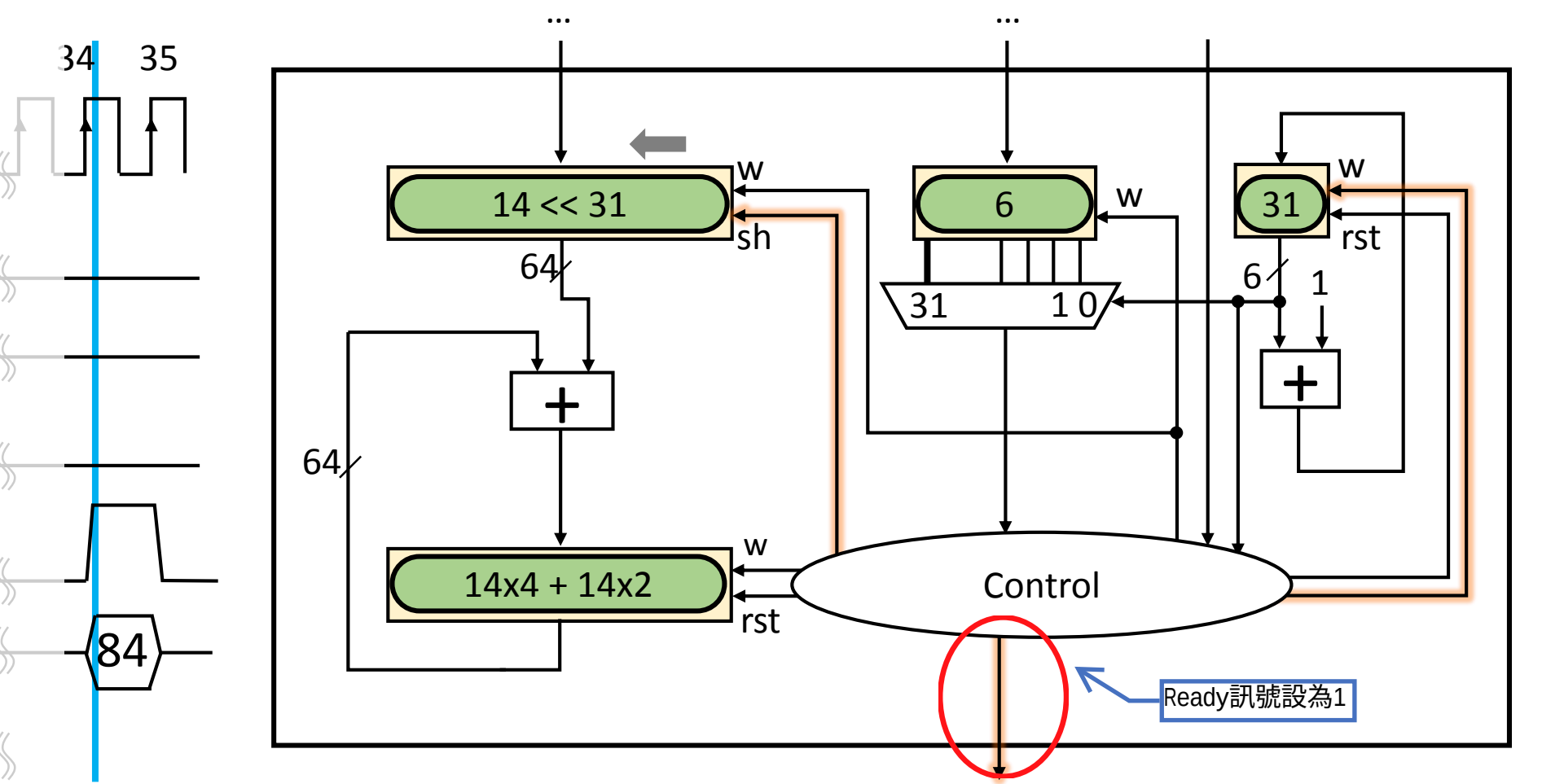
t = 33.99

- Prepared next {Hi, Lo} based on B[31]=0.
- Start transitioning to the next {Hi, Lo}

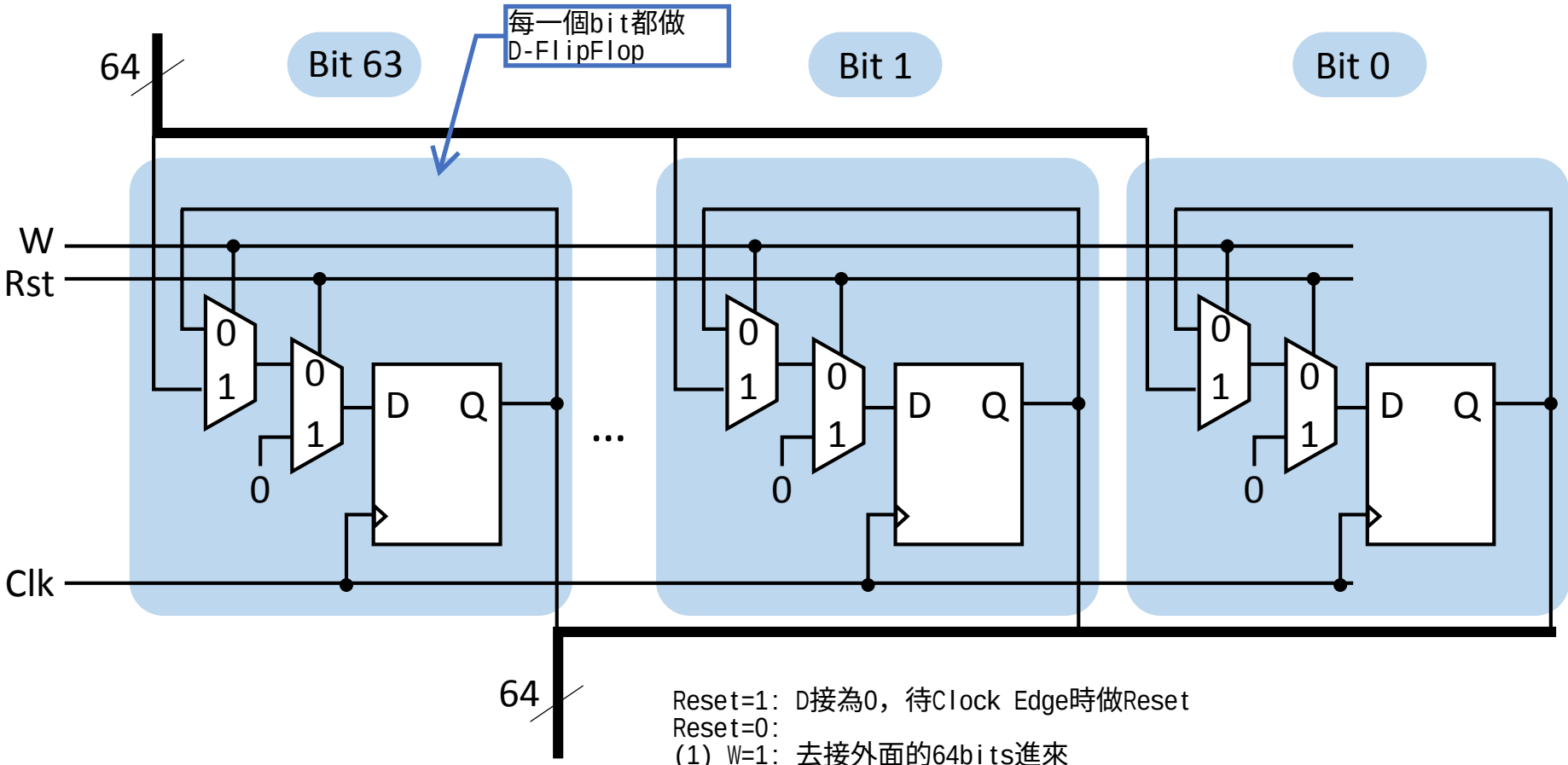


t = 34.2

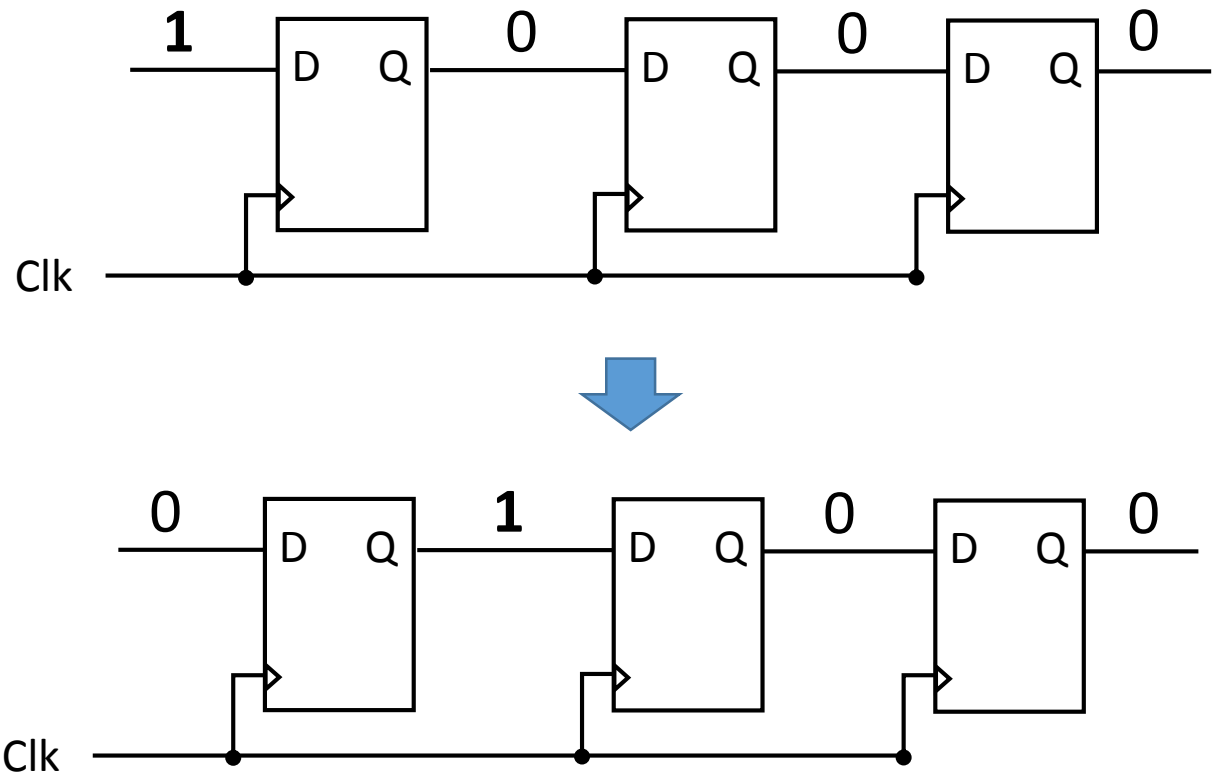
- Hi & Lo are done.
- Raise the Ready signal.



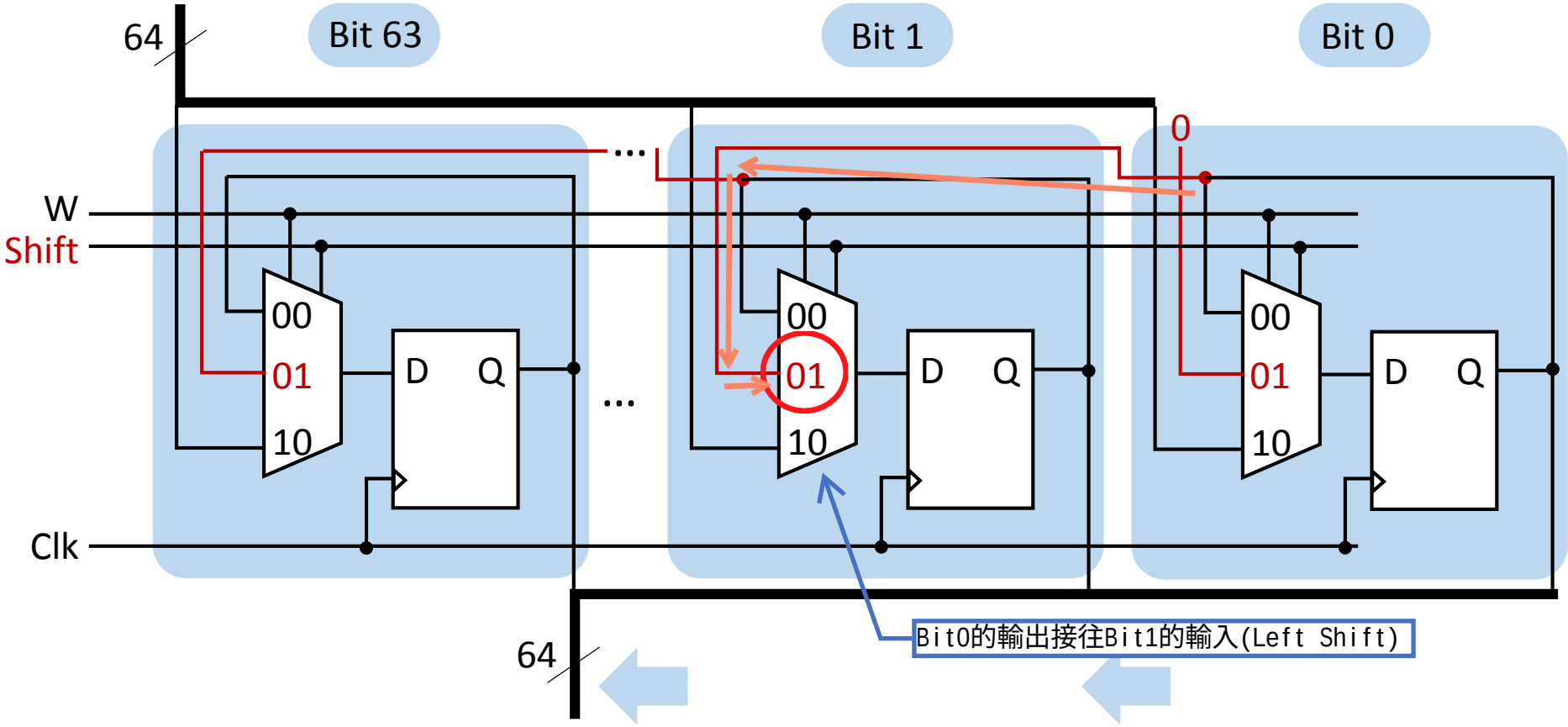
Registers with Reset and Write



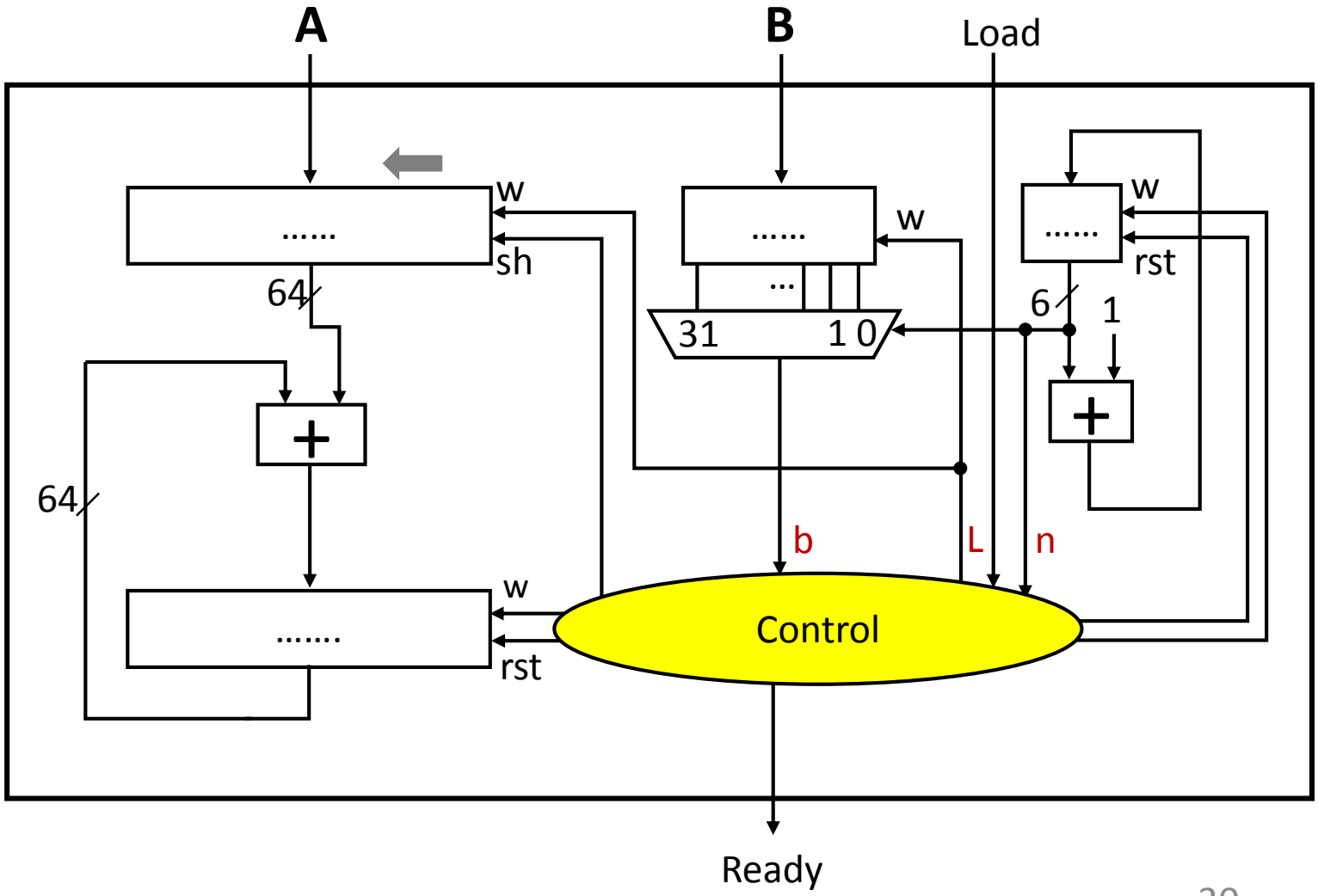
Shift Registers



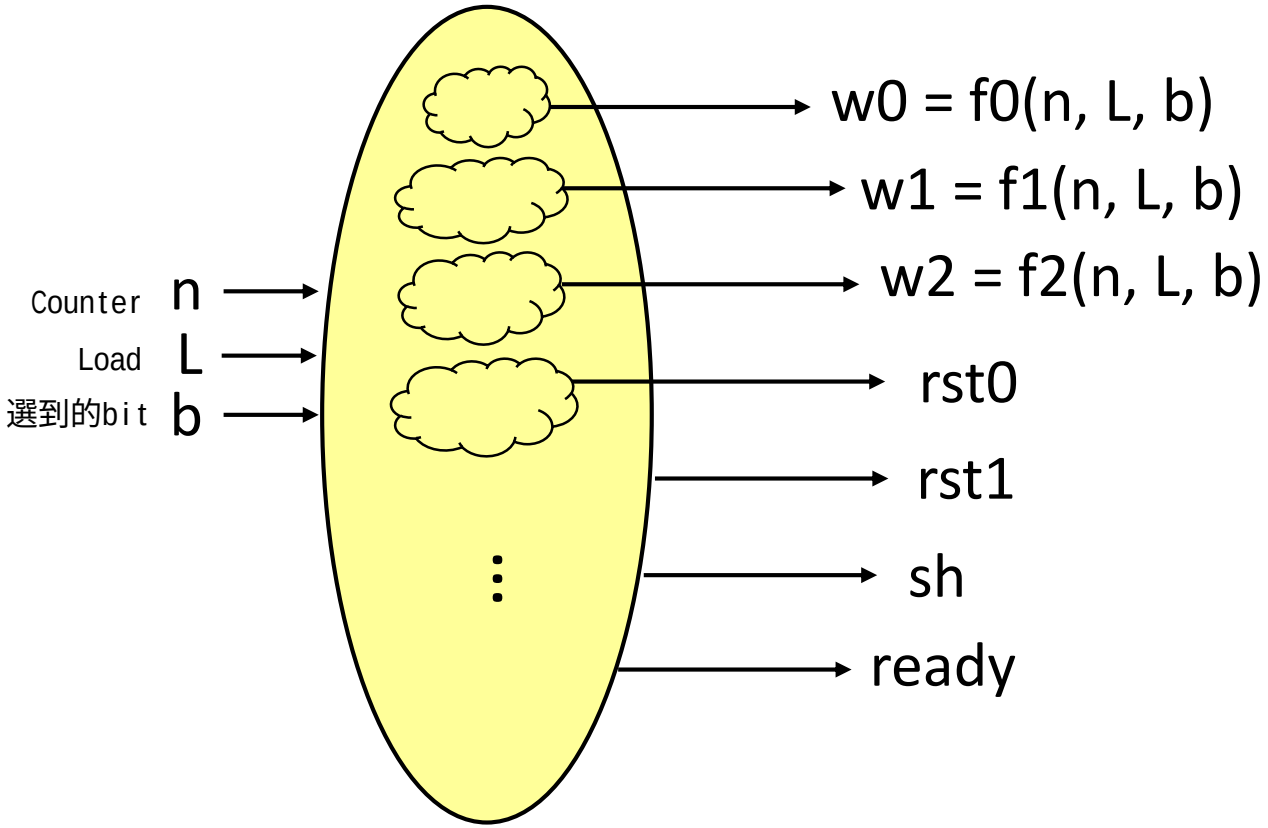
Registers with Write and Left Shift



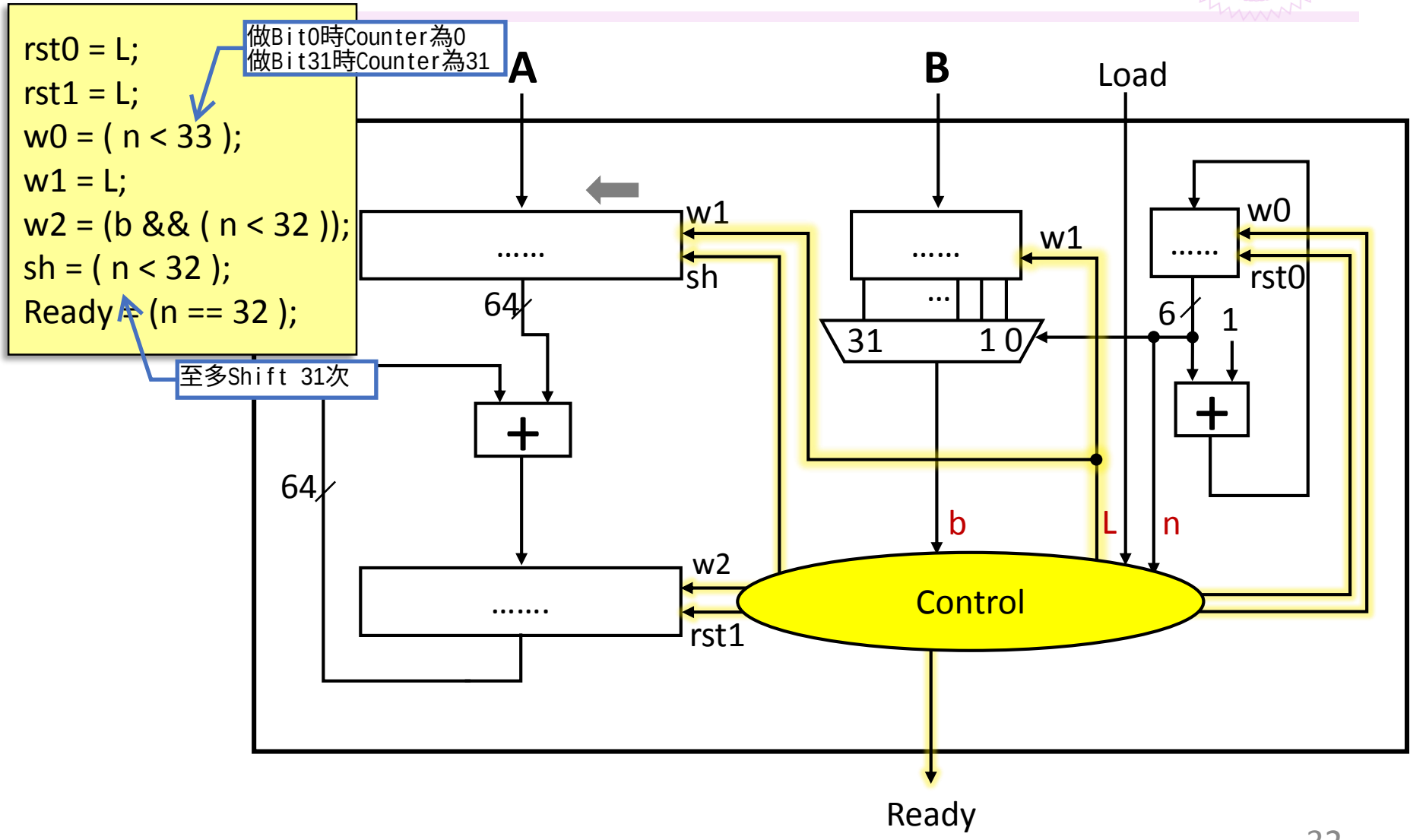
The Control Unit



The Control Unit



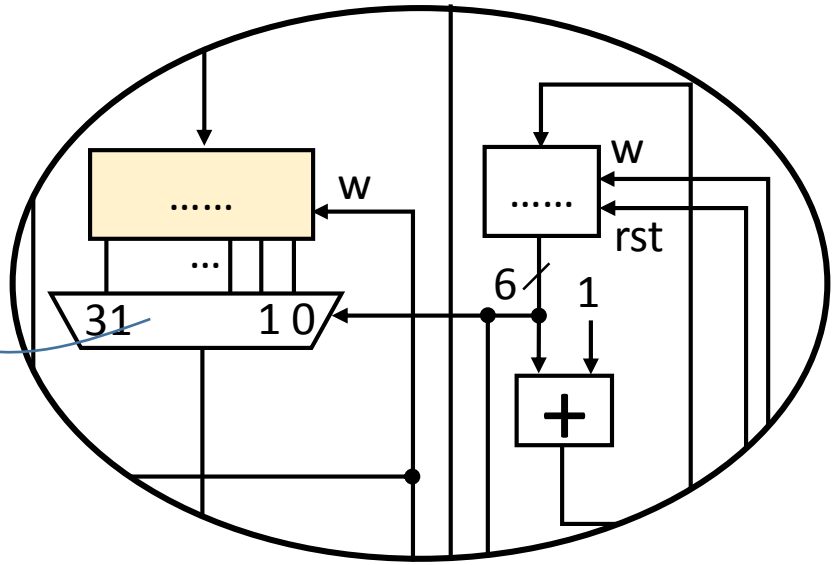
The Control Unit



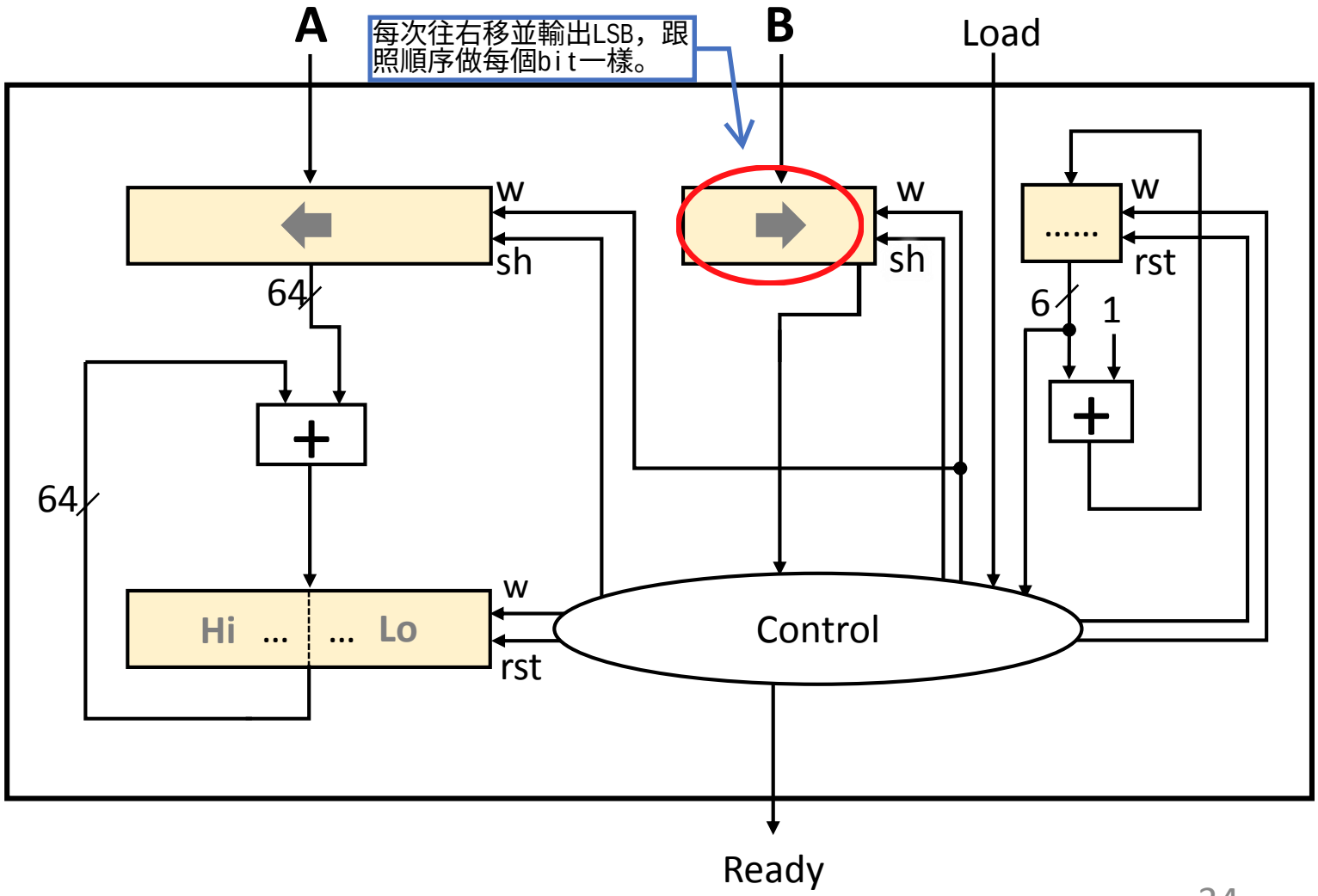
Improved Multiply Unit (V1)

- We can use shift registers here to replace the 32-to-1 multiplexer

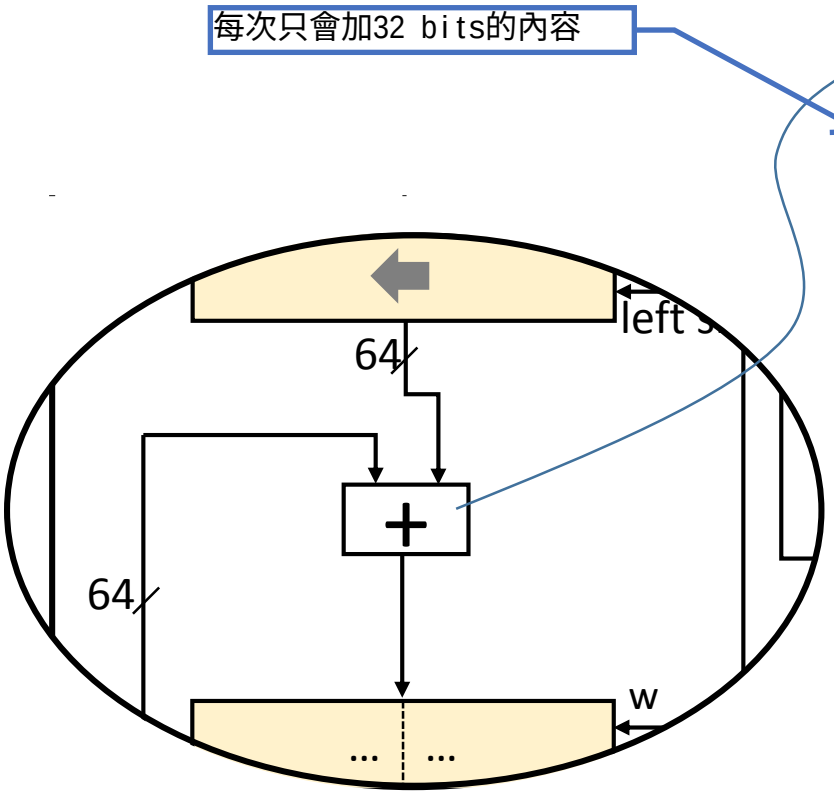
Multiplexer讓我們可以挑選任一bit，而乘法不需要這種功能，只要能照順序Shift就好。



Improved Multiply Unit (V1)

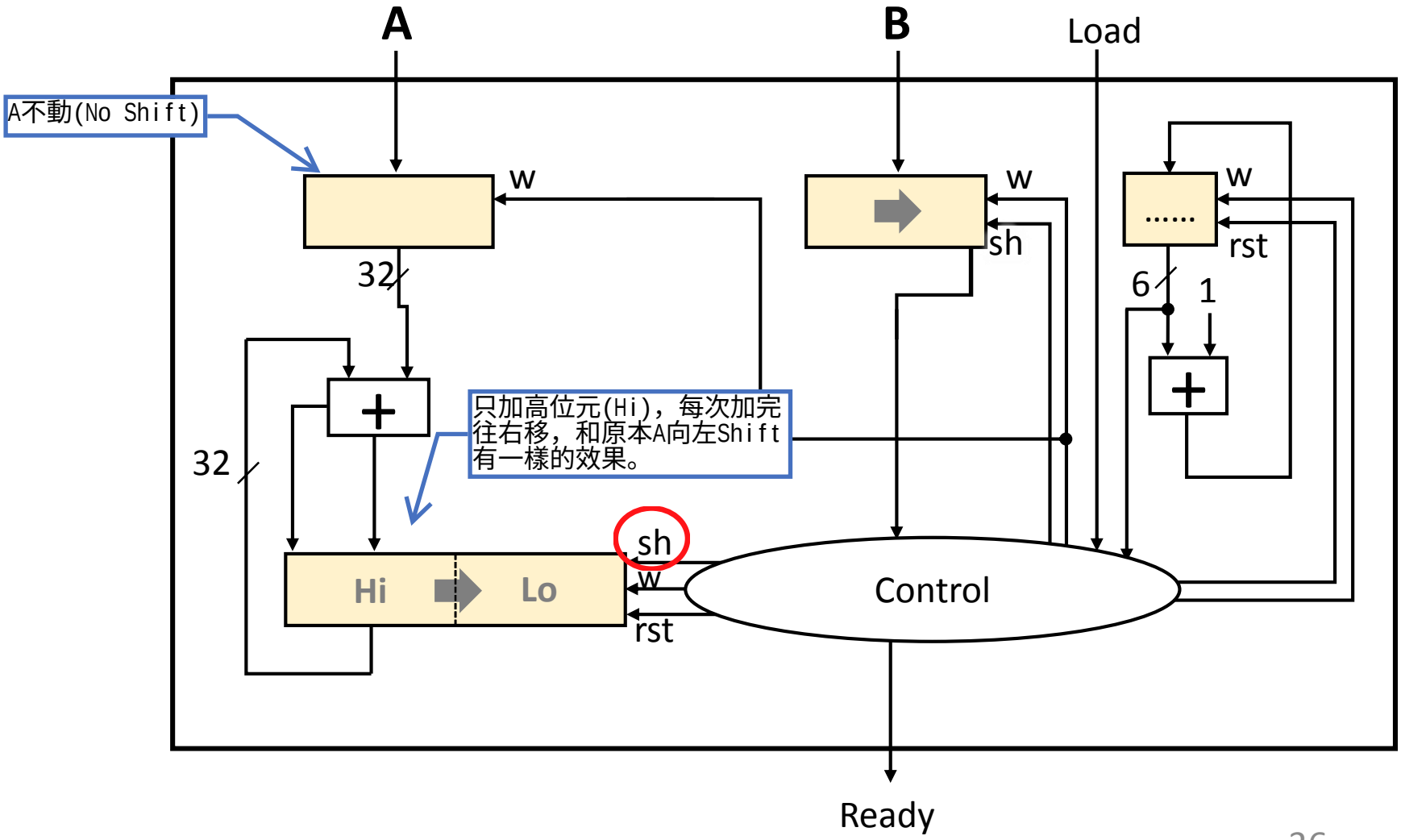


Improved Multiply Unit (V2)



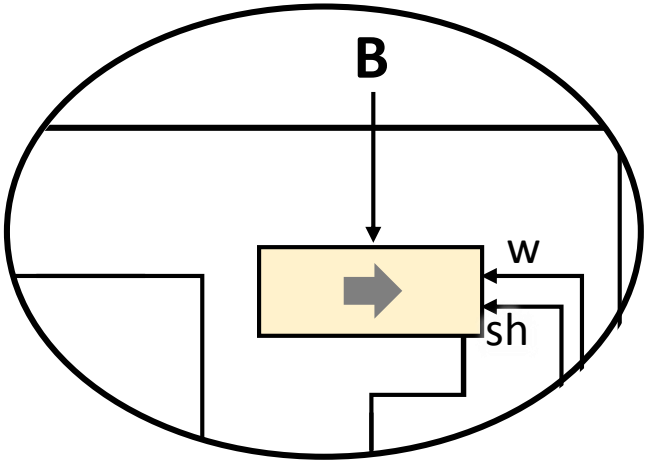
- 64-bit adder is wasteful
  - Each iteration only touches the 32-bit Hi register
- Solution
  - Right shifting {Hi, Lo} instead of left shifting the multiplicand

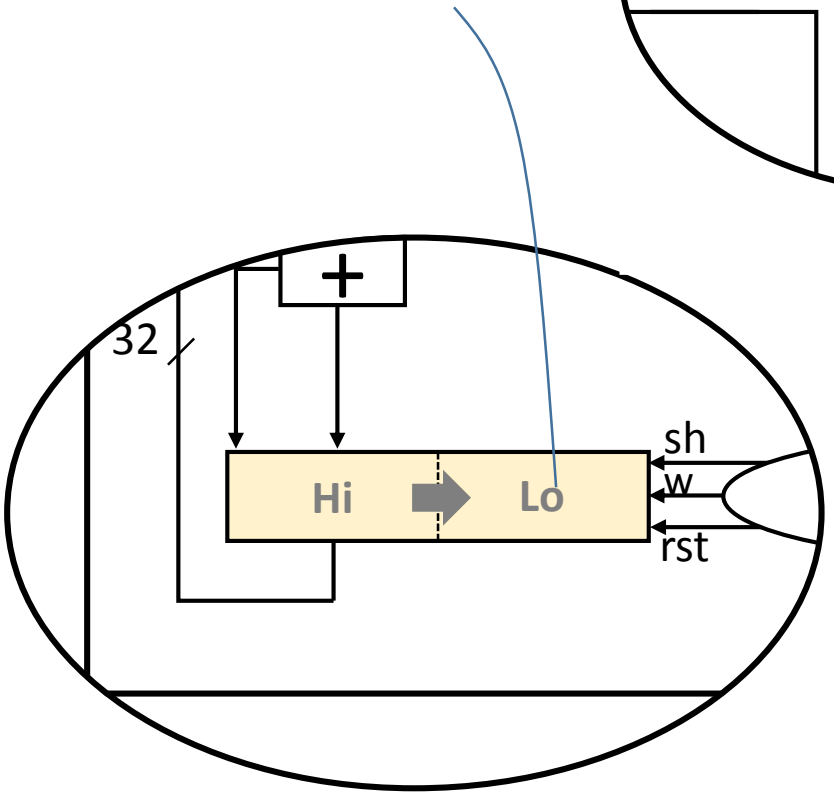
Improved Multiply Unit (V2)



Improved Multiply Unit (V3)

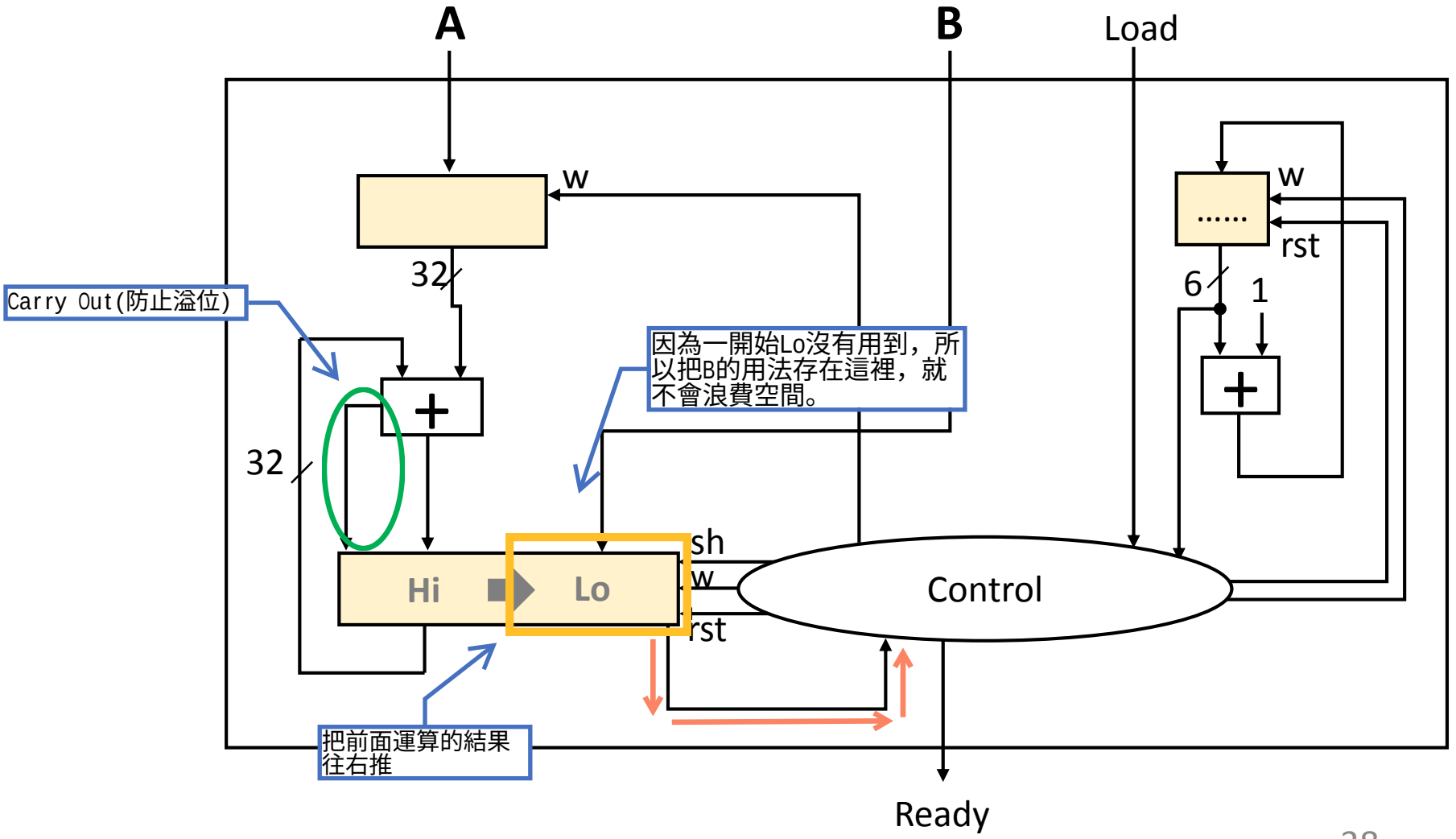
- Lo is not in use in the beginning





→ sharing registers between multiplicand and Lo

Improved Multiply Unit (V3)



Improved Multiply Unit (V3)

- A = 1110 = 14_{ten}, B = 0110 = 6_{ten}

決定下一個State的狀態為何(交錯)

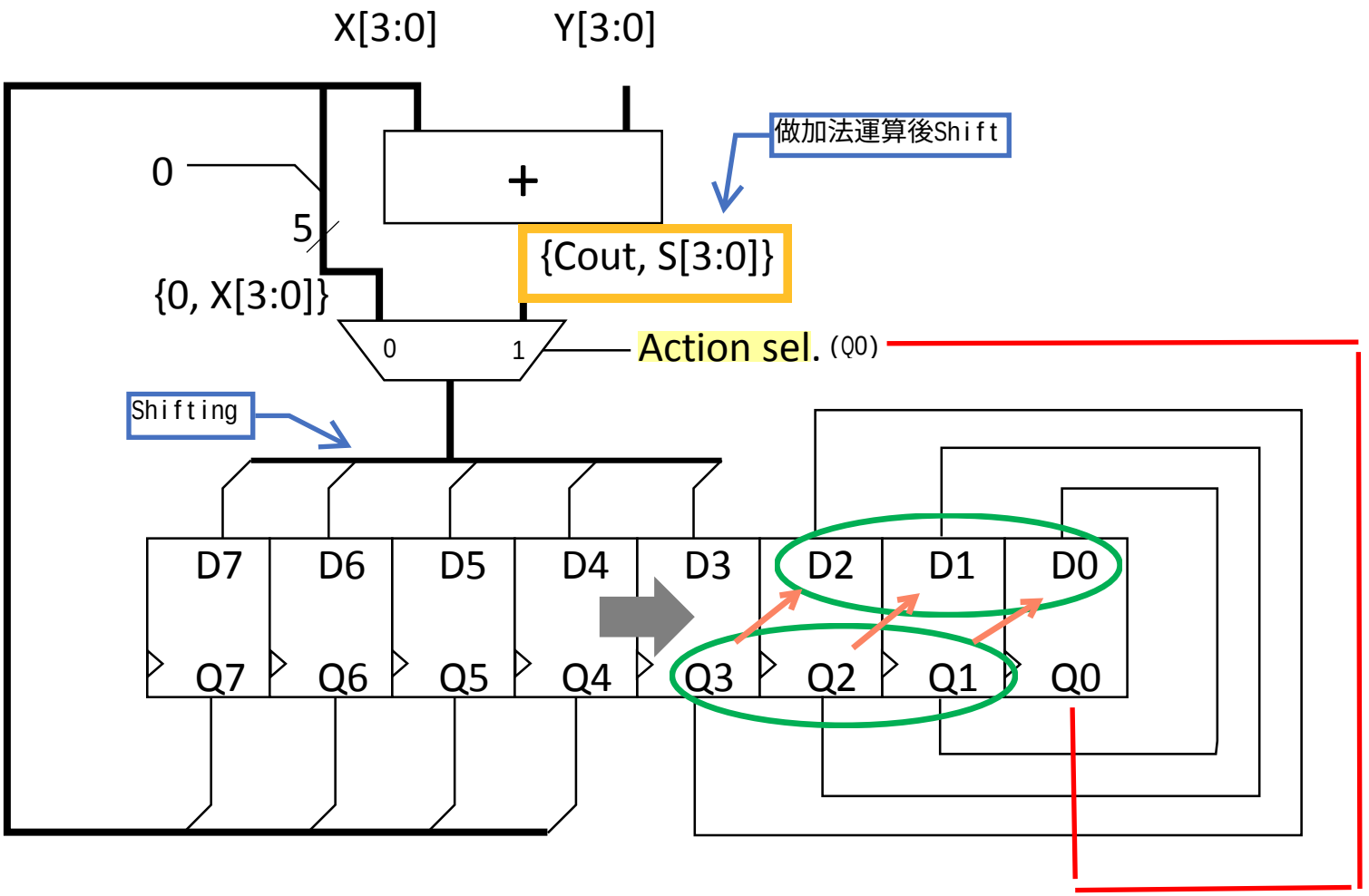
看似兩步，其實在電路設計上是一步做完

	Hi	Lo (B)	Lo[0]	Next Action Decided by Control
0	0000	0110	0	Right shift {Hi, Lo}
1	0000	0011	1	$Hi_D = (Hi_Q + A)$; Right shift {Hi, Lo}
2	0111	0001	1	$Hi_D = (Hi_Q + A)$; Right shift {Hi, Lo}
3	1010	1000	0	Right shift {Hi, Lo}
4	0101	0100		

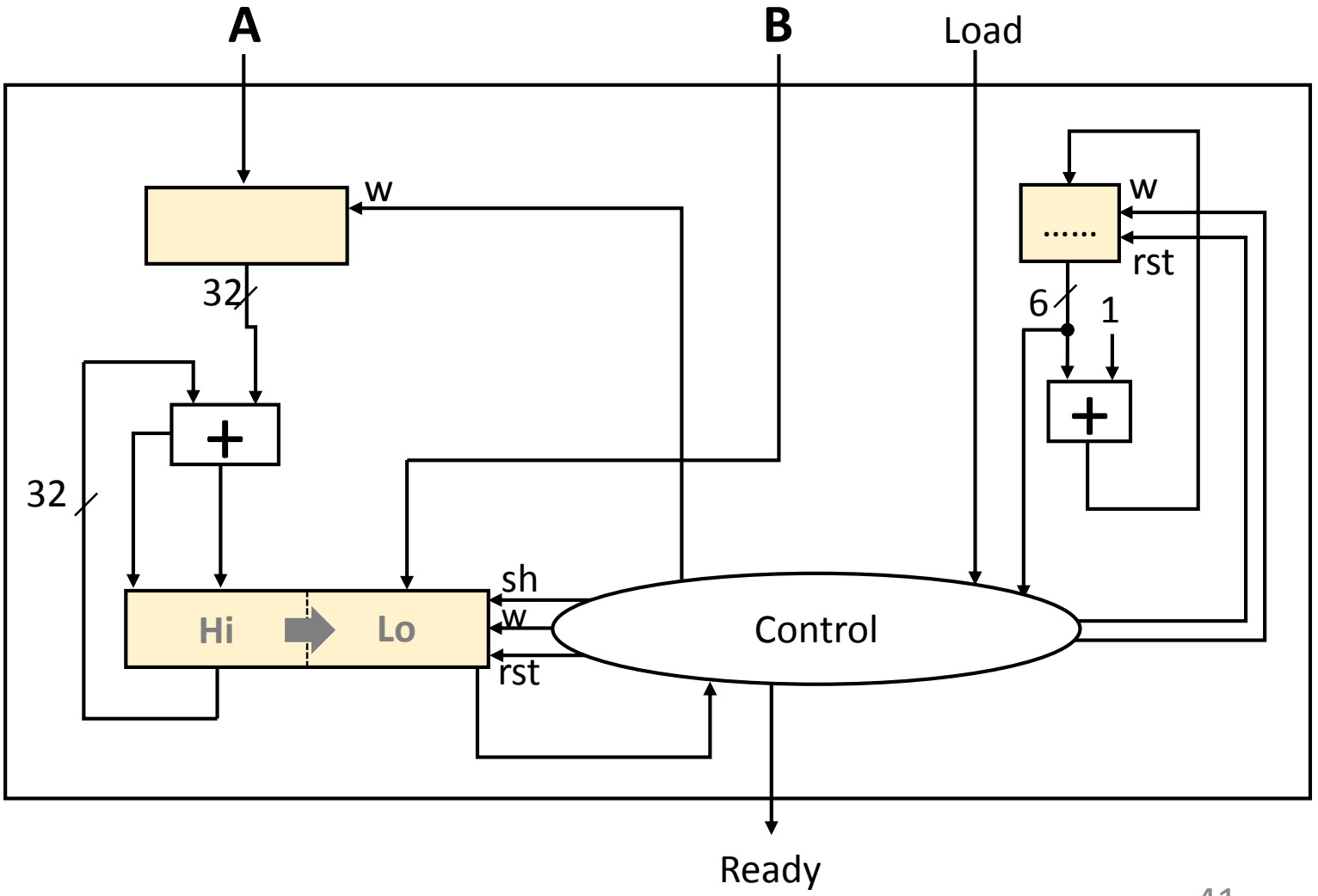
為0時不加直接Shift

{Hi, Lo} = 0101_0100 = 84_{ten}

Simultaneous Add and Right Shift



Improved Multiply Unit (V3)



Signed Multiplication

1110 x 0110 = - ((-1110) x 0110) = - (0010 x 0110)

轉成2's complement做運算

```
    0 0 1 0   (2)
x   0 1 1 0   (6)
-------------
    0 0 0 0
  0 0 1 0
  0 0 1 0
0 0 0 0
-------------
0 0 0 1 1 0 0   (12)
```

Signed multiplication can be done by using the unsigned procedure

Negate

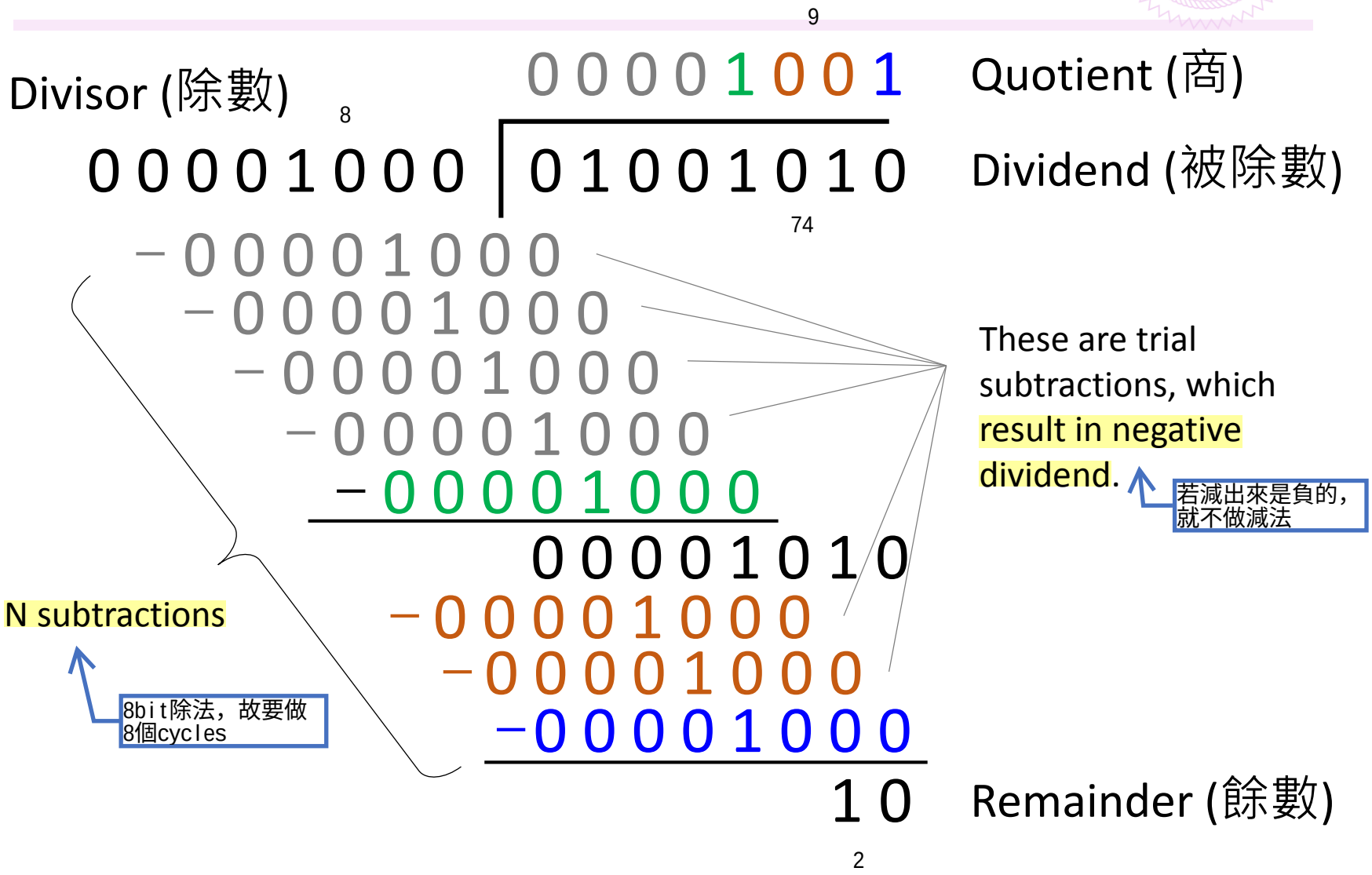
先用絕對值(正數)運算完後再冠負號

1 1 1 0 1 0 0 (-12)

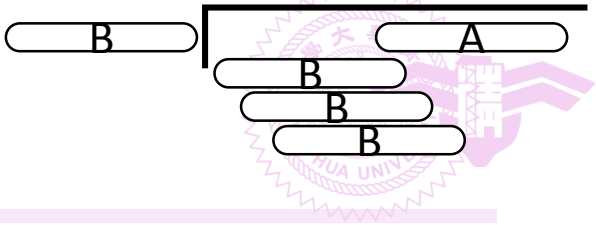
MIPS Division Instructions

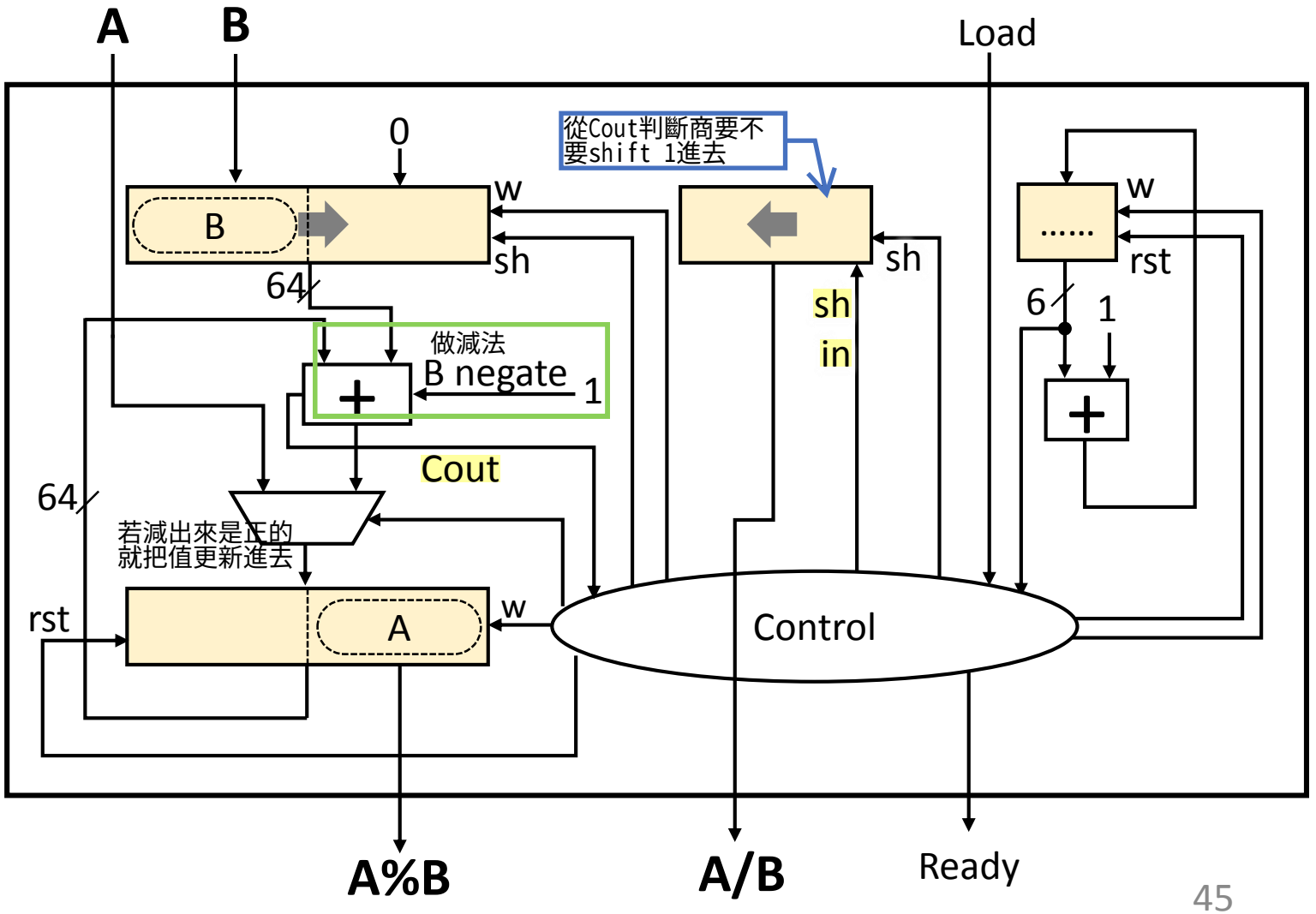
- `DIV $d, $s, $t`
- `REM $d, $s, $t`

Division

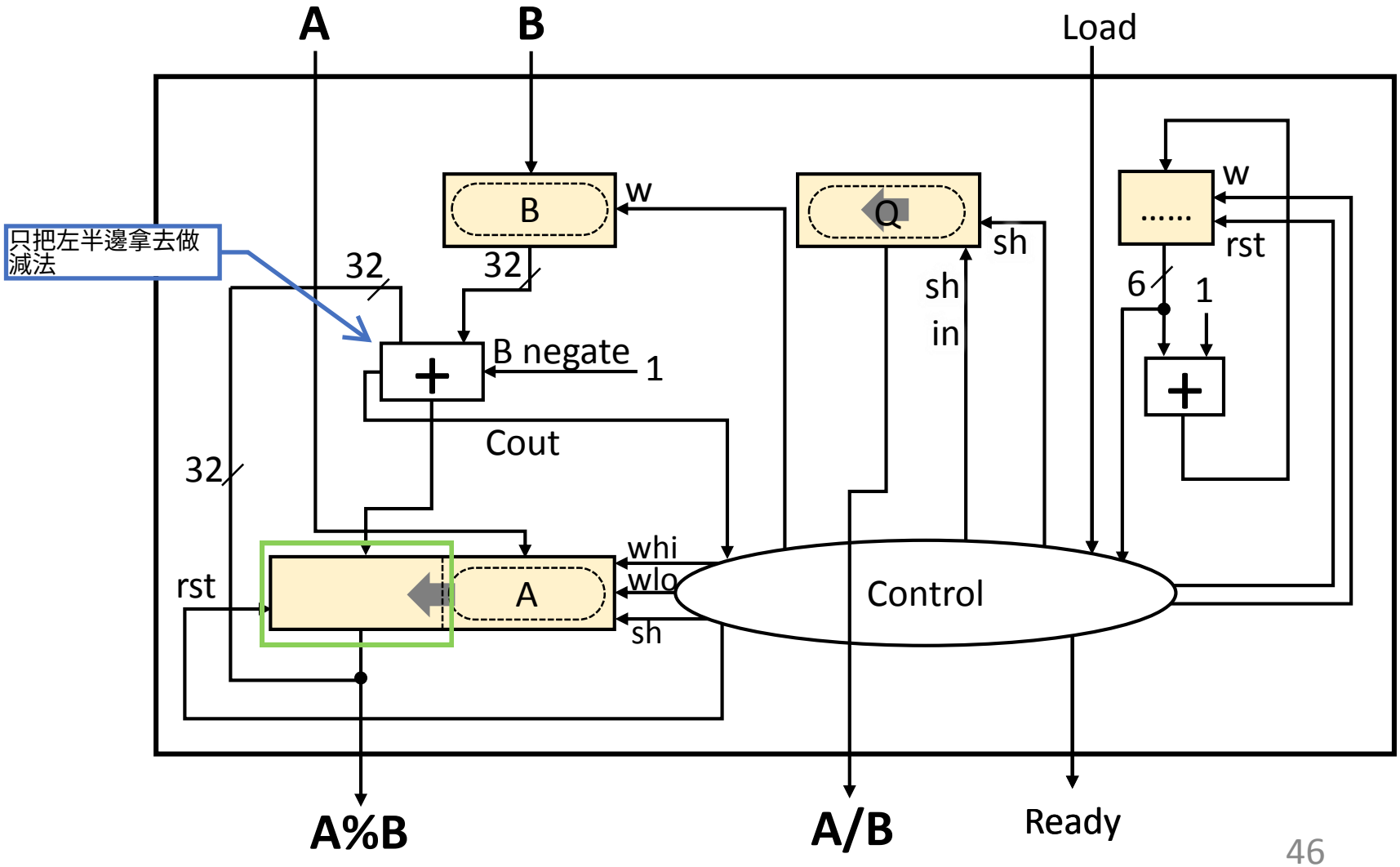

9
0 0 0 0 1 0 0 1
Quotient (商)
Divisor (除數)
8
0 0 0 0 1 0 0 0
0 1 0 0 1 0 1 0
Dividend (被除數)
74
− 0 0 0 0 1 0 0 0
− 0 0 0 0 1 0 0 0
− 0 0 0 0 1 0 0 0
− 0 0 0 0 1 0 0 0
− 0 0 0 0 1 0 0 0
0 0 0 0 1 0 1 0
− 0 0 0 0 1 0 0 0
− 0 0 0 0 1 0 0 0
− 0 0 0 0 1 0 0 0
1 0
Remainder (餘數)
2
These are trial subtractions, which result in negative dividend.
若減出來是負的，就不做減法
N subtractions
8bit除法，故要做8個cycles

Basic Division Unit

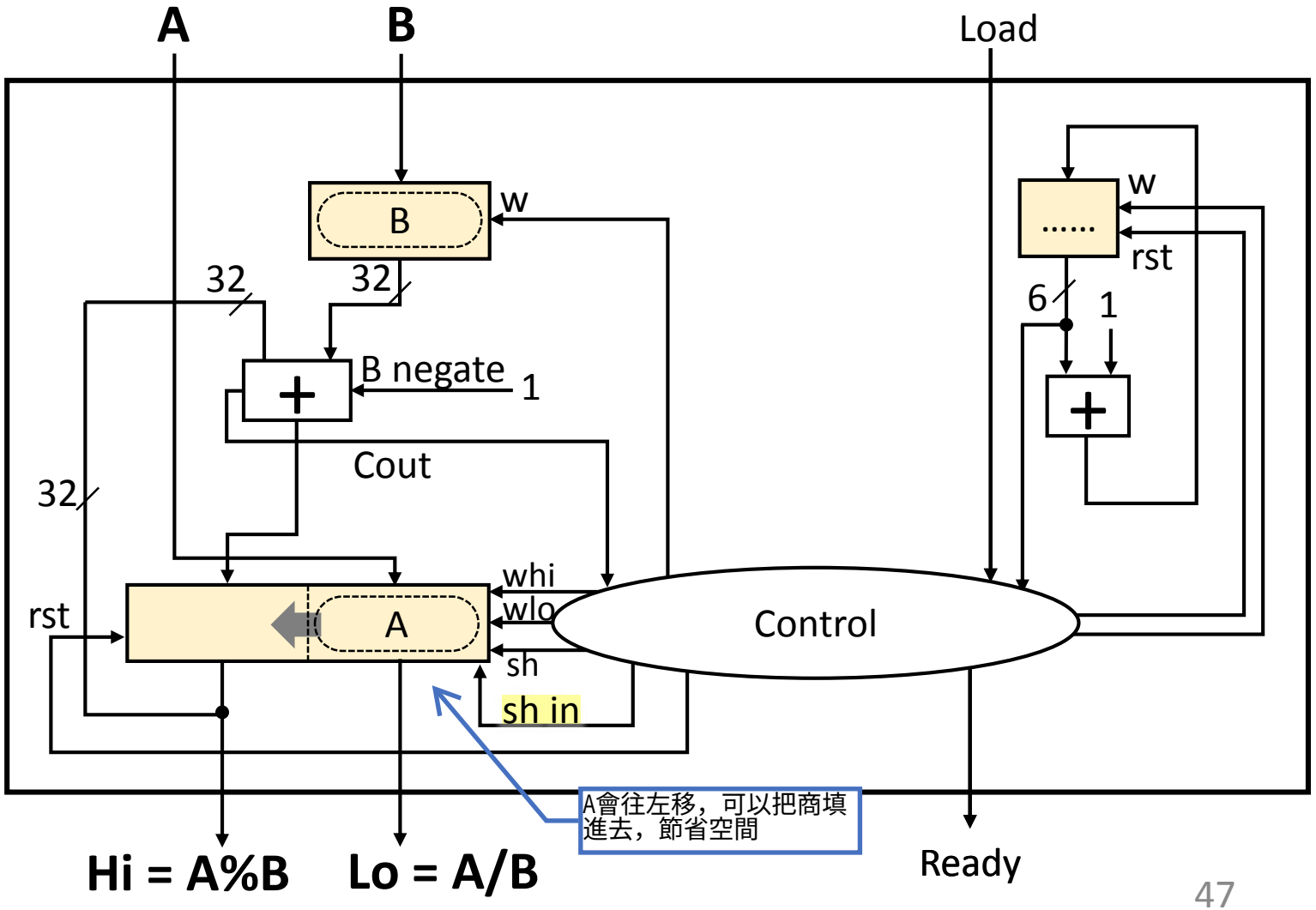




Improved Division Unit (V1)



Improved Division Unit (V2)



Improved Division Unit (V2)

- A = 01001010 = 74_{ten}, B = 00001000 = 8_{ten}

	Hi (R)	Lo (A, Q)	Hi < B?	Todo
0	0 0 0 0 0 0 0 0	0 1 0 0 1 0 1 0		
1	0 0 0 0 0 0 0 0	1 0 0 1 0 1 0 0	1	Shift 0 into {Hi, Lo}
2	0 0 0 0 0 0 0 1	0 0 1 0 1 0 0 0	1	Shift 0 into {Hi, Lo}
3	0 0 0 0 0 0 1 0	0 1 0 1 0 0 0 0	1	Shift 0 into {Hi, Lo}
4	0 0 0 0 0 1 0 0	1 0 1 0 0 0 0 0	1	Shift 0 into {Hi, Lo}
5	0 0 0 0 1 0 0 1	0 1 0 0 0 0 0 0	1	Shift 0 into {Hi, Lo}
6	0 0 0 0 0 1 0	1 0 0 0 0 0 0 1	0	Hi = (Hi-B); Shift 1 into {Hi, Lo}
7	0 0 0 0 0 1 0 1	0 0 0 0 0 0 1 0	1	Shift 0 into {Hi, Lo}
8	0 0 0 0 1 0 1 0	0 0 0 0 0 1 0 0	1	Shift 0 into {Hi, Lo}
9	0 0 0 0 0 0 1 0	0 0 0 0 1 0 0 1	0	Hi = (Hi-B); Shift 1 into {Lo}

Remainder (2) Quotient (9)

最後只改Lo的部分

Signed Division

- Rule
  - The remainder (餘數) should have the same signs as that of dividend (被除數)
  - A ÷ B
    - |Q| = |A| / |B|; sign(Q) = sign(A) XOR sign(B)
    - |R| = |A| % |B|; sign(R) = sign(A)
- Examples
  - 7 ÷ 2 = 3 … 1
  - (-7) ÷ 2 = (-3) … (-1)
  - 7 ÷ (-2) = (-3) … 1
  - (-7) ÷ (-2) = 3 … (-1)

Summary

- Bit-wise logical
  - ✔ and, andi
  - ✔ or, ori
  - ✔ xor, xori
  - ✔ nor
- Arithmetic
  - ✔ add, addu, addi, addiu
  - ✔ sub, subu
  - ✔ mulh, mul
  - ✔ div, rem
- Comparisons
  - ✔ slt, slti, sltu, sltiu

Note

- My slides present slightly different from the textbook
  - Textbook plots the flow charts of performing multiplication and division
  - My slides emphasize logic circuits
  - In the textbook, the dividend is subtracted by the divisor before the subtraction result is checked
  - In my slides, the subtraction result is checked before the result is written into the dividend register
- Please also read the textbook, too

Outline

- Overview
- Integer operations
  - Bit-wise logical operations
  - Additions, subtractions
  - Comparisons
  - Multiplications
  - Divisions
- Floating point operations
  - Additions, subtractions
  - Multiplications