Introduction to Computer Organization and Architecture

Topic 4. Computer arithmetic

Han, Nguyen Dinh (han.nguyendinh@hust.edu.vn)

Faculty of Mathematics and Informatics
Hanoi University of Science and Technology

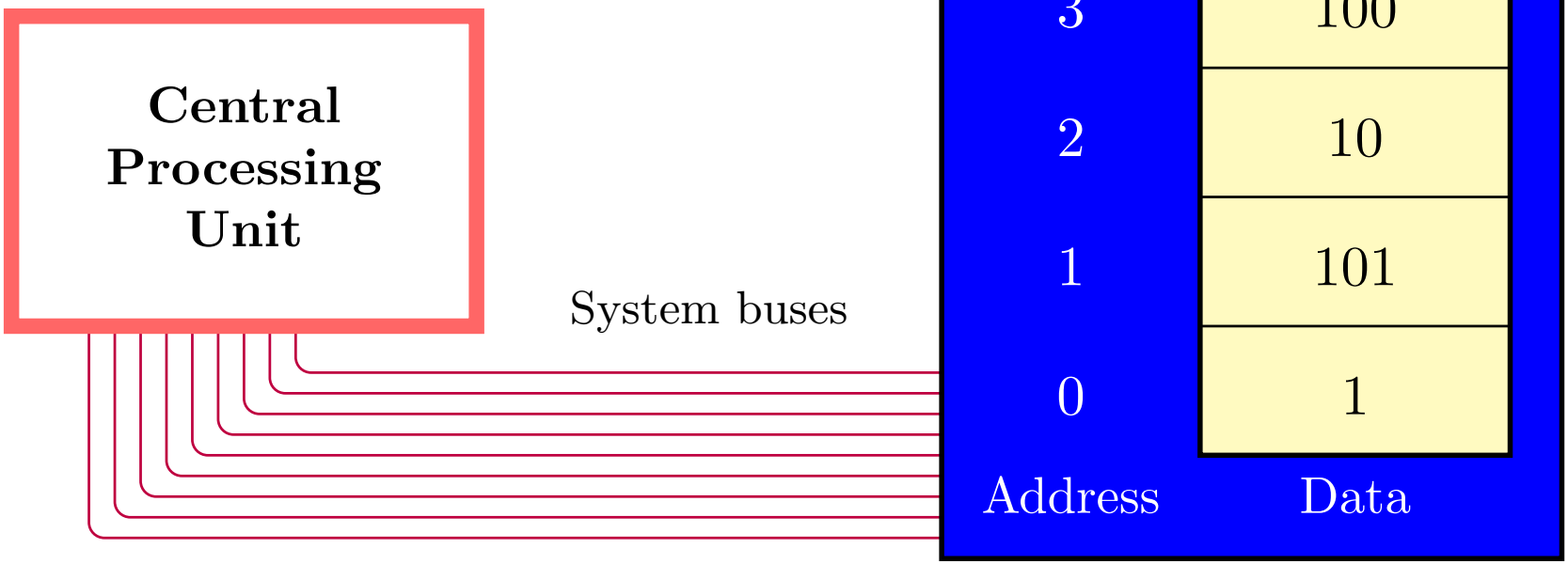
Outline

1

Number representation

Representing instructions in the computer

Instructions are represented as numbers. Then, programs can be stored in memory to be read or written just like numbers



Number representation

Numbers can be represented in any base; humans prefer base 10, and base 2 is best for computers. Computer words are composed of bits; thus words can be represented as binary numbers. In any number base, the value of ith digit d is

$$d \times \text{Base}^i$$

where i starts at 0 and increases from right to left.
For example, 1011_{two} represents

$$\begin{aligned}
& (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)_{ten} \\
= \; & (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)_{ten} \\
= \; & 8 + 0 + 2 + 1_{ten} \\
= \; & 11_{ten}
\end{aligned}$$

Number representation

The MIPS word is 32 bits long, so we can represent 2^{32} patterns. The phrase *least significant bit* is used to refer to the rightmost bit (i.e. bit 0) and *most significant bit* to the leftmost bit (i.e. bit 31). It is natural to let the combinations represent the numbers from 0 to $2^{32} - 1$:

$$
\begin{aligned}
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} &= 0_{ten} \\
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} &= 1_{ten} \\
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} &= 2_{ten} \\
\ldots & \ldots \\
1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} &= 4,294,967,293_{ten} \\
1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} &= 4,294,967,294_{ten} \\
1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} &= 4,294,967,295_{ten}
\end{aligned}
$$

Number representation

Computer programs calculate both positive and negative numbers, so we need a representation that distinguishes the positive from the negative. The solution that made the hardware simple is: leading 0s mean positive, and leading 1s mean negative. This convention for representing signed binary numbers is called *two's complement* representation:

$$
\begin{aligned}
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} &= 0_{ten} \\
\ldots & \ldots \\
0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} &= 2,147,483,647_{ten} \\
1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} &= -2,147,483,648_{ten} \\
\ldots & \ldots \\
1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} &= -1_{ten}
\end{aligned}
$$

Number representation

Two's complement representation has the advantage that all negative numbers have a 1 in the most significant bit. This particular bit is often called the *sign bit*. In order to negative a two's complement binary number, first we need to invert the number (i.e. every 0 to 1 and every 1 to 0), then add one to the result. For example, we have

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

Negating this number by inverting the bits and adding one,

$$\begin{array}{rrcr} & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} & = & -3_{ten} \\ + & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} & = & 1_{ten} \\ \hline = & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} & = & -2_{ten} \end{array}$$

2

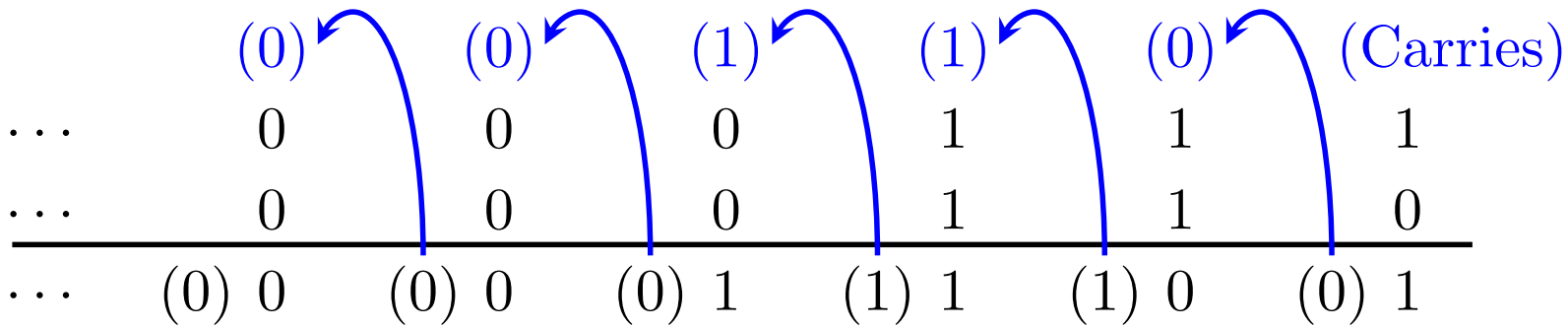
Addition and subtraction

Addition and subtraction

Addition is just what you would expect in computers. Let's try adding 6_{ten} to 7_{ten} as follows

$$\begin{array}{lll} & \texttt{0000 0000 0000 0000 0000 0000 0000 0111}_{two} = & 7_{ten} \\ + & \texttt{0000 0000 0000 0000 0000 0000 0000 0110}_{two} = & 6_{ten} \\ \hline = & \texttt{0000 0000 0000 0000 0000 0000 0000 1011}_{two} = & 13_{ten} \end{array}$$

Digits are added bit by bit from right to left, with carries passed to the next digit to the left, just as you would do by hand



Addition and subtraction

Subtraction uses addition: the appropriate operand is simply negated before being added. Now let's try subtracting 6_{ten} from 7_{ten}. Actually, this can be done directly

$$\begin{array}{cll} & \texttt{0000 0000 0000 0000 0000 0000 0000 0111}_{two} = & 7_{ten} \\ - & \texttt{0000 0000 0000 0000 0000 0000 0000 0110}_{two} = & 6_{ten} \\ \hline = & \texttt{0000 0000 0000 0000 0000 0000 0000 0001}_{two} = & 1_{ten} \end{array}$$

or via addition using the two's complement representation of the appropriate operand (i.e. -6_{ten}) as follows

$$\begin{array}{cll} & \texttt{0000 0000 0000 0000 0000 0000 0000 0111}_{two} = & 7_{ten} \\ + & \texttt{1111 1111 1111 1111 1111 1111 1111 1010}_{two} = & -6_{ten} \\ \hline = & \texttt{0000 0000 0000 0000 0000 0000 0000 0001}_{two} = & 1_{ten} \end{array}$$

3

Constructing an arithmetic logic unit

Let's run

pieces of Verilog codes

given in Section B.4, Appendix B of the textbook!

4

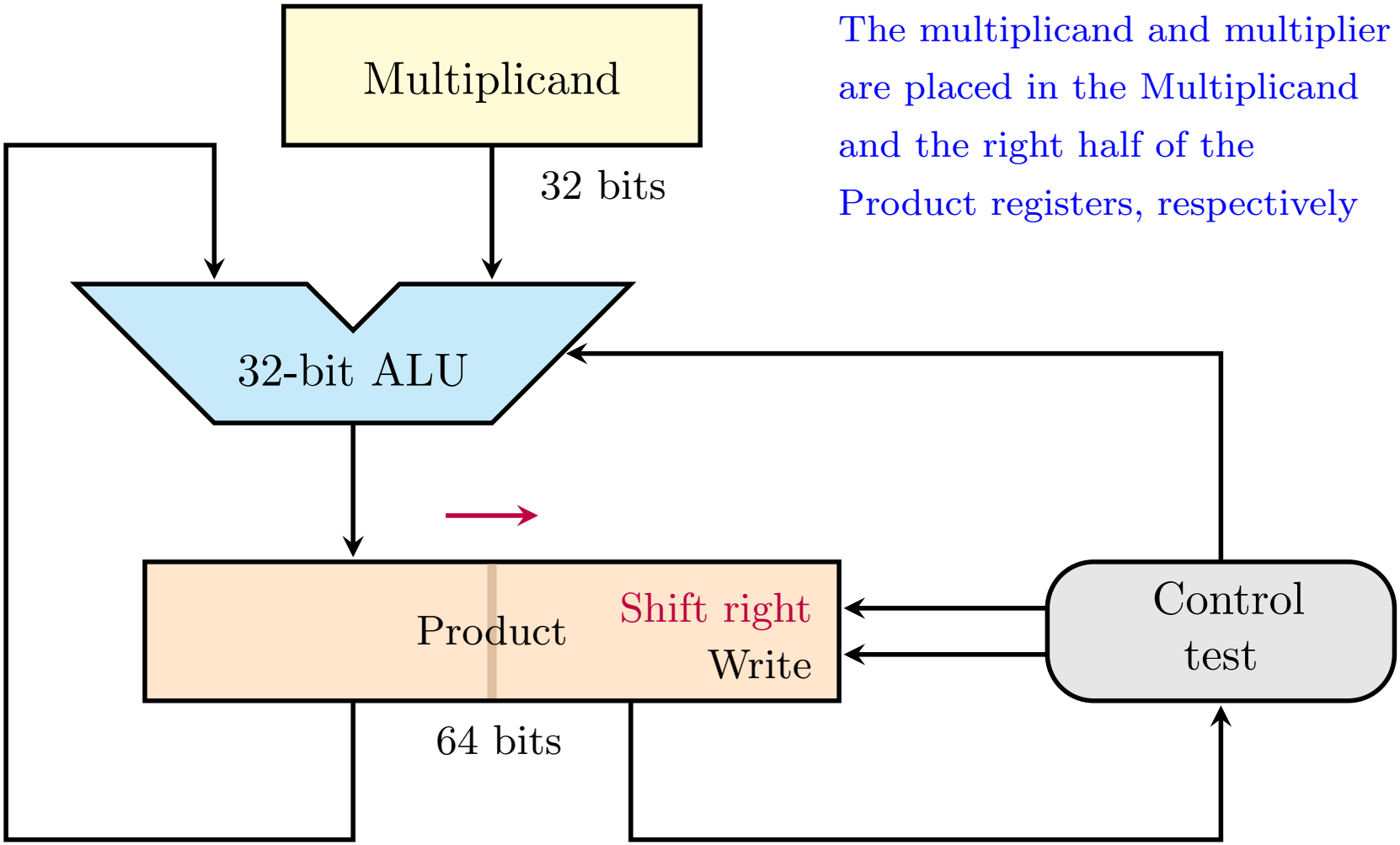
Multiplication

Multiplication

We recall the multiplication algorithm learned in grammar school. Suppose that we multiply two unsigned decimal numbers, and we restrict the decimal digits to 0 and 1. Then, the multiplying 1000_{ten} (multiplicand) by 1001_{ten} (multiplier), that yields the product 1001000_{ten}, is as follows

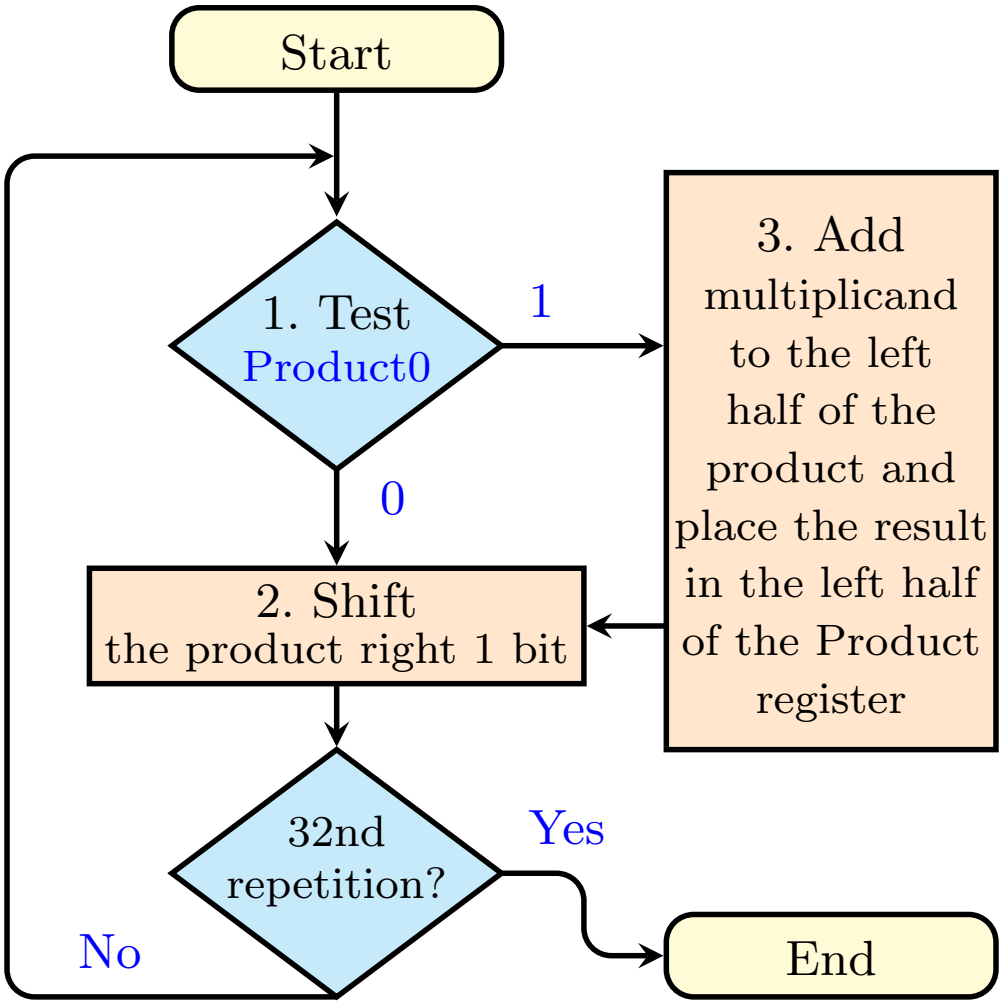
`Multiplicand`	1000_{ten}
`Multiplier`	1001_{ten}
	1000
	0000
	0000
	1000
`Product`	1001000_{ten}

The multiplication hardware



The multiplicand and multiplier are placed in the Multiplicand and the right half of the Product registers, respectively

The multiplication algorithm



Demonstration ($0110 = 0010 \times 0011$)	
Iter.	Product
0	0000 001 1
1	0010 0011 0001 000 1
2	0011 0001 0001 100 0
3	0001 1000 0000 110 0
4	0000 1100 0000 0110

Booth's algorithm

We can use Booth's algorithm to multiply signed numbers. Instead of looking at 1 bit of the multiplier, Booth's algorithm examines 2 bits consisting of the current bit and the bit to the right (i.e. the current bit in the previous step). Depending on this pair of bits, the algorithm works as follows

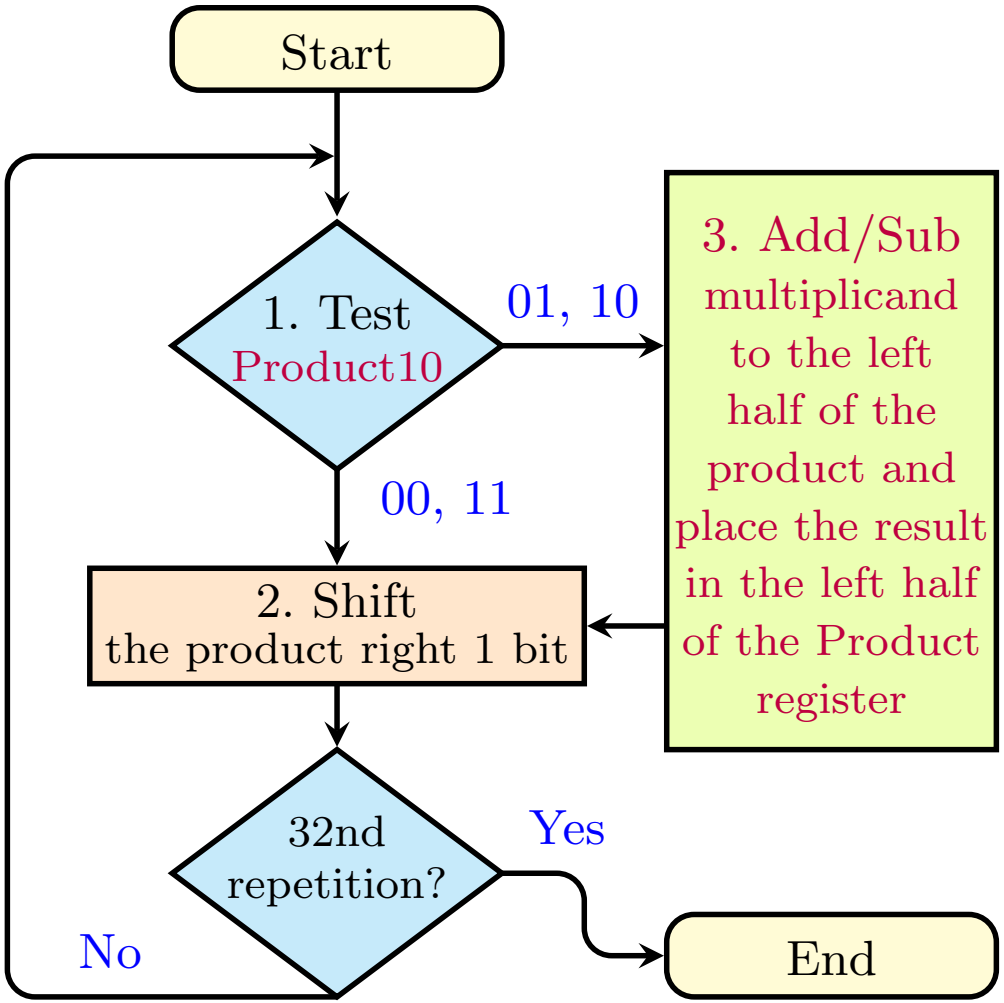
```
00:    No arithmetic operation

01:    Add the multiplicand to the left half of
       the product

10:    Subtract the multiplicand from the left
       half of the product

11:    No arithmetic operation
```

Booth's algorithm

Booth's algorithm was based on the fact that an ALU that could add or subtract could get the same result in more than one way. For example, we can multiply 0010_{two} by 0110_{two} as follows

	0010_{two}			0010_{two}	
$\times$	0110_{two}		$\times$	0110_{two}	
$+$	0000	shift	$+$	0000	shift
$+$	0010	add	$-$	0010	sub
$+$	0010	add	$+$	0000	shift
$+$	0000	shift	$+$	0010	add
	00001100_{two}			00001100_{two}	

Booth's algorithm



Demonstration ($1111\ 1010 = 0010 \times 1101$)	
Iter.	Product
0	0000 110 1 0
1	1110 11010 1111 011 0 1
2	0001 01101 0000 101 1 0
3	1110 10110 1111 010 1 1
4	1111 01011 1111 1010 1

5

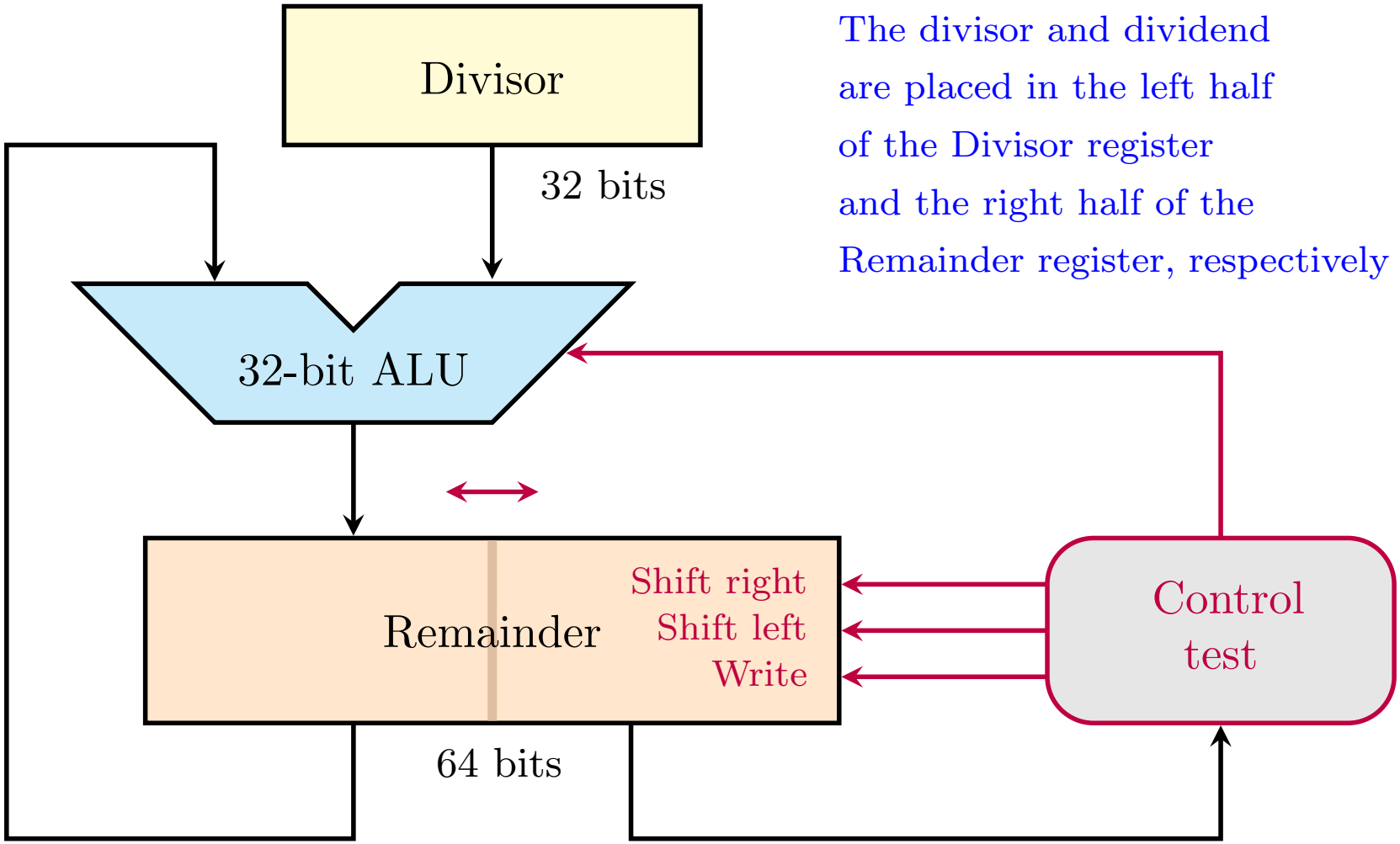
Division

Division

Let's start with an example of long division using decimal numbers to recall the grammar school division algorithm. Suppose that we divide 1001010_{ten} (dividend) by 1000_{ten} (divisor). Then, the division that yields 1001_{ten} (quotient) and 10_{ten} (remainder), is as follows

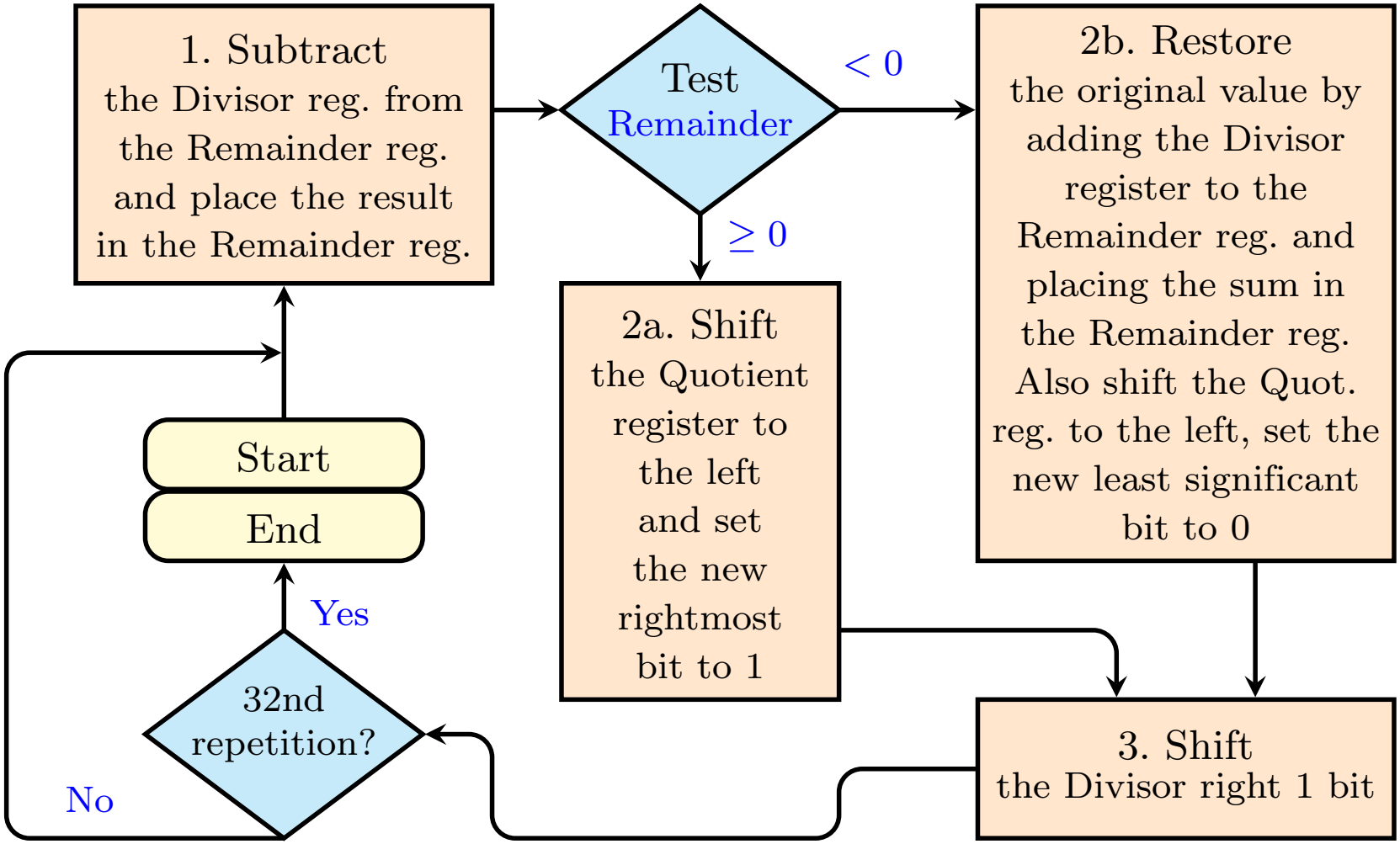
Dividend	1001010_{ten}	1000_{ten}	Divisor
	-1000	1001_{ten}	Quotient
	10		
	101		
	1010		
	-1000		
	10_{ten}		Remainder

The division hardware



The divisor and dividend are placed in the left half of the Divisor register and the right half of the Remainder register, respectively

The division algorithm



An example: divide $0000\,0111_{two}$ by 0010_{two}

Iter.	Step	Quo. (Q)	Divisor (D)	Remainder (R)
0	Initial values	0000	0010 0000	0000 0111
1	1: $R = R - D$	0000	0010 0000	①110 0111
	2b: $R = R + D; \ll Q; Q_0 = 0$	0000	0010 0000	0000 0111
	3: $\gg D$	0000	0001 0000	0000 0111
2	1: $R = R - D$	0000	0001 0000	①111 0111
	2b: $R = R + D; \ll Q; Q_0 = 0$	0000	0001 0000	0000 0111
	3: $\gg D$	0000	0000 1000	0000 0111
3	1: $R = R - D$	0000	0000 1000	①111 1111
	2b: $R = R + D; \ll Q; Q_0 = 0$	0000	0000 1000	0000 0111
	3: $\gg D$	0000	0000 0100	0000 0111
4	1: $R = R - D$	0000	0000 0100	⓪000 0011
	2a: $\ll Q; Q_0 = 1$	0001	0000 0100	0000 0011
	3: $\gg D$	0001	0000 0010	0000 0011
5	1: $R = R - D$	0001	0000 0010	⓪000 0001
	2a: $\ll Q; Q_0 = 1$	0011	0000 0010	0000 0001
	3: $\gg D$	0011	0000 0001	0000 0001

Explore and try to run

MIPS instructions covered

in Sections 3.1 to 3.4,

Chapter 3 of the textbook!

Thank you very much for your attention!