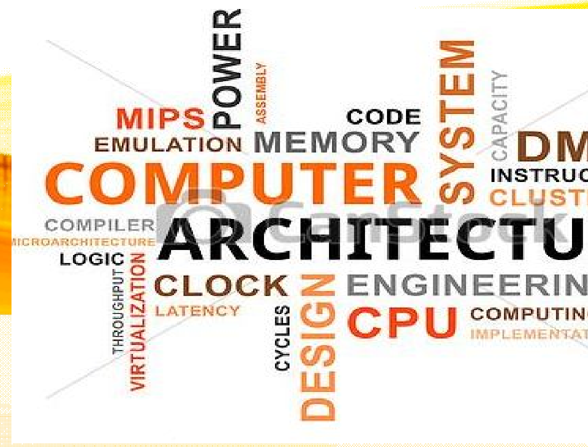
UNIT II
ARITHMETIC OPERATIONS

Addition and subtraction of signed numbers – Design of fast adders - Multiplication of positive numbers - Signed operand multiplication- fas multiplication – **Integer division** – Floating point numbers and operations



Recap the previous Class

Introduction

- Division is more complex than multiplication.
- *Example*: Typical values in Pentium-3 processor.

– Not easy to construct high-speed dividers.

- The ratios have not changed much in later processors.

Instruction	Latency	Cycles / Issue
Load / Store	3	1
Integer Multiply	4	1
Integer Divide	36	36
Floating-point Add	3	1
Floating-point Multiply	5	2
Floating-point Divide	38	38

- Latency:
  - Minimum delay after which the first result is obtained, starting from the time when the first se of inputs is applied.
- Cycles/Issue:
  - Whenever a new set of inputs is applied to a functional unit (e.g. adder), it is called an ***issue*.**
  - Pipelined implementation of arithmetic unit **reduces the number of clock cycles** between successive issues.
  - For non-pipelined arithmetic units (e.g. divider), the number of clock cycles between successive issues is much higher.

The Process of Integer Division

- In integer division, a ***divisor*** M and a ***dividend*** D are given.
- The objective is to find a third number Q, called the ***quotient***,

  such that **D = Q x M + R** where R is the ***remainder*** such that $0 \leq R < M$.
- The relationship D = Q x M suggests that there is a close correspondence between division and multiplication.
  - Dividend, quotient and divisor correspond to product, multiplicand and multiplier, respectively.

- One of the simplest division methods is the **sequential digit-by-digit algorithm** similar to that used in pencil-and-paper methods.

```
                       0 1 1 0           Quotient Q = Q0Q1Q2Q3
Divisor M    1 1 0 | 1 0 0 1 0 1         Dividend D = R0
             ------  1 1 0               Q0.M       (Does not go; Q0 = 0)
                    -------------
                     1 0 0 1 0 1         R1
                    -  1 1 0             Q1.2^-1.M  (Does go; Q1 = 1)
                    -------------
                       0 1 1 0 1         R2
                    -    1 1 0           Q2.2^-2.M  (Does go; Q2 = 1)
                    -------------
                         0 0 0 1         R3
                           1 1 0         Q3.2^-3.M  (Does not go; Q3 = 0)
                    -------------
                           0 0 1         R4 = Remainder R
```

$D = 37 = (1\,0\,0\,1\,0\,1)_2$

$M = 6 = (1\,1\,0)_2$

Quotient $Q = 6$

Remainder $R = 1$

- In the example, the quotient $Q = Q_0Q_1Q_2$... is computed one bit at a time.
  - At each step i, the divisor shifted i bits to the right (i.e. $2^{-i}.M$) is compared with the current partial remainder R_i.
  - The quotient bit Q_i is set to 0 (1) if $2^{-i}.M$ is greater than (less than) R_i,
  - The new partial remainder R_{i+1} is computed as:

$$R_{i+1} = R_i - Q_i.2^{-i}.M$$

- Machine implementation:
  - For hardware implementation, it is more convenient to shift the partial remainder to the left relative to a fixed divisor; thus

$$R_{i+1} = 2R_i - Q_i.M \quad (instead\ of\ R_{i+1} = R_i - Q_i.2^{-i}.M)$$

  - The final partial remainder is the required remainder shifted to the left, so that $R = 2^{-3}.R_4$

Divisor M			Quotient Q
1 1 0	1 0 0 1 0 1	Dividend = $2R_0$	
	1 1 0	$Q_0.M$	0
	---------------		
	1 0 0 1 0 1	R_1	
	1 0 0 1 0 1 0	$2R_1$	
	1 1 0	$Q_1.M$	0 1
	---------------		
	0 1 1 0 1 0	R_2	
	0 1 1 0 1 0 0	$2R_2$	
	1 1 0	$Q_2.M$	0 1 1
	---------------		
	0 0 0 1 0 0	R_3	
	0 0 0 1 0 0 0	$2R_3$	
	1 1 0	$Q_3.M$	0 1 1 0
	---------------		
	0 0 1 0 0 0	$R_4 = 2^3.R$	

Do not subtract

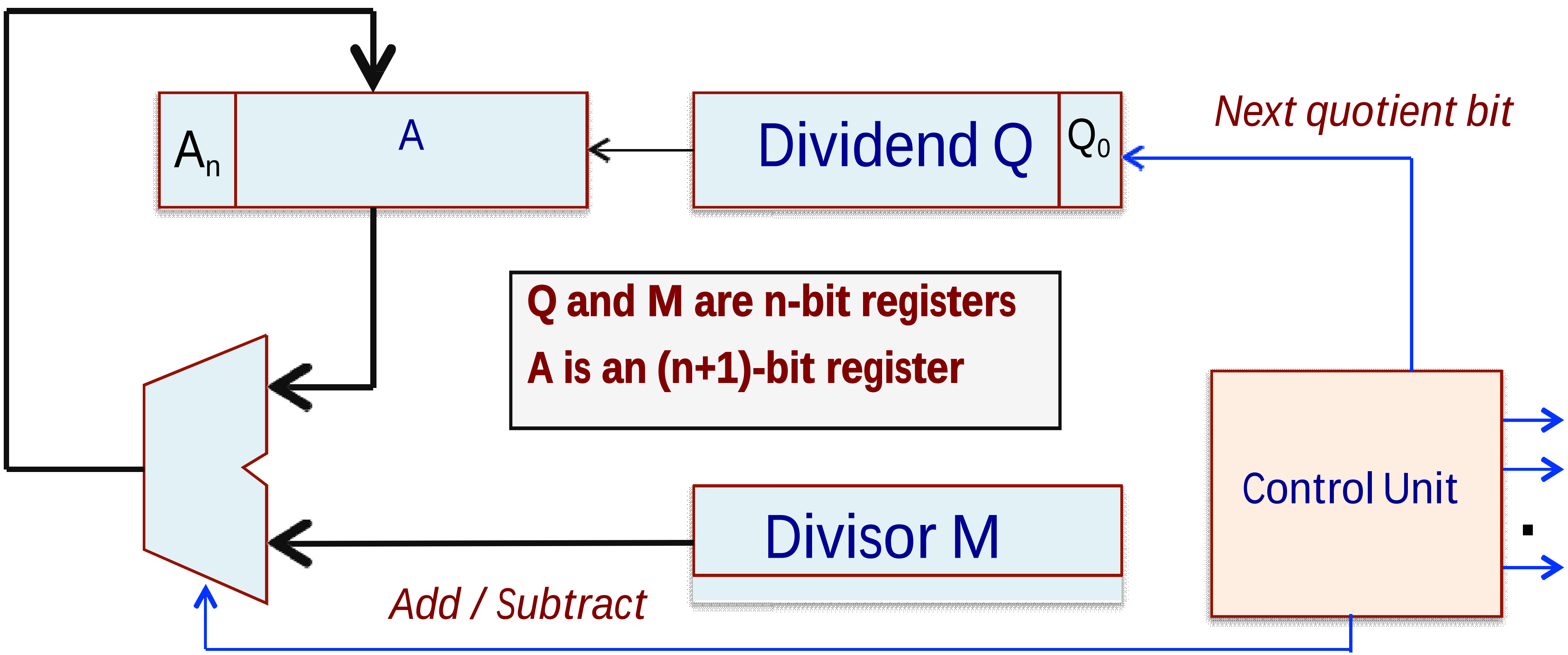
$D = 37 = (1\,0\,0\,1\,0\,1)_2$

$M = 6 = (1\,1\,0)_2$

Quotient $Q = 6$

Remainder $R = 1$

Restoring Division: The Data Path



Basic Steps

Repeat the following steps n times:

a) Shift the dividend one bit at a time starting into register A.

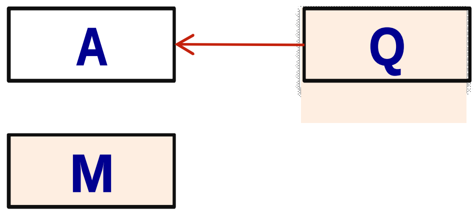
b) Subtract the divisor M from this register A (***trial subtraction***).

c) If the result is negative (***i.e. not going***):

- Add the divisor M back into the register A (***i.e. restoring back***).
- Record 0 as the next quotient bit.

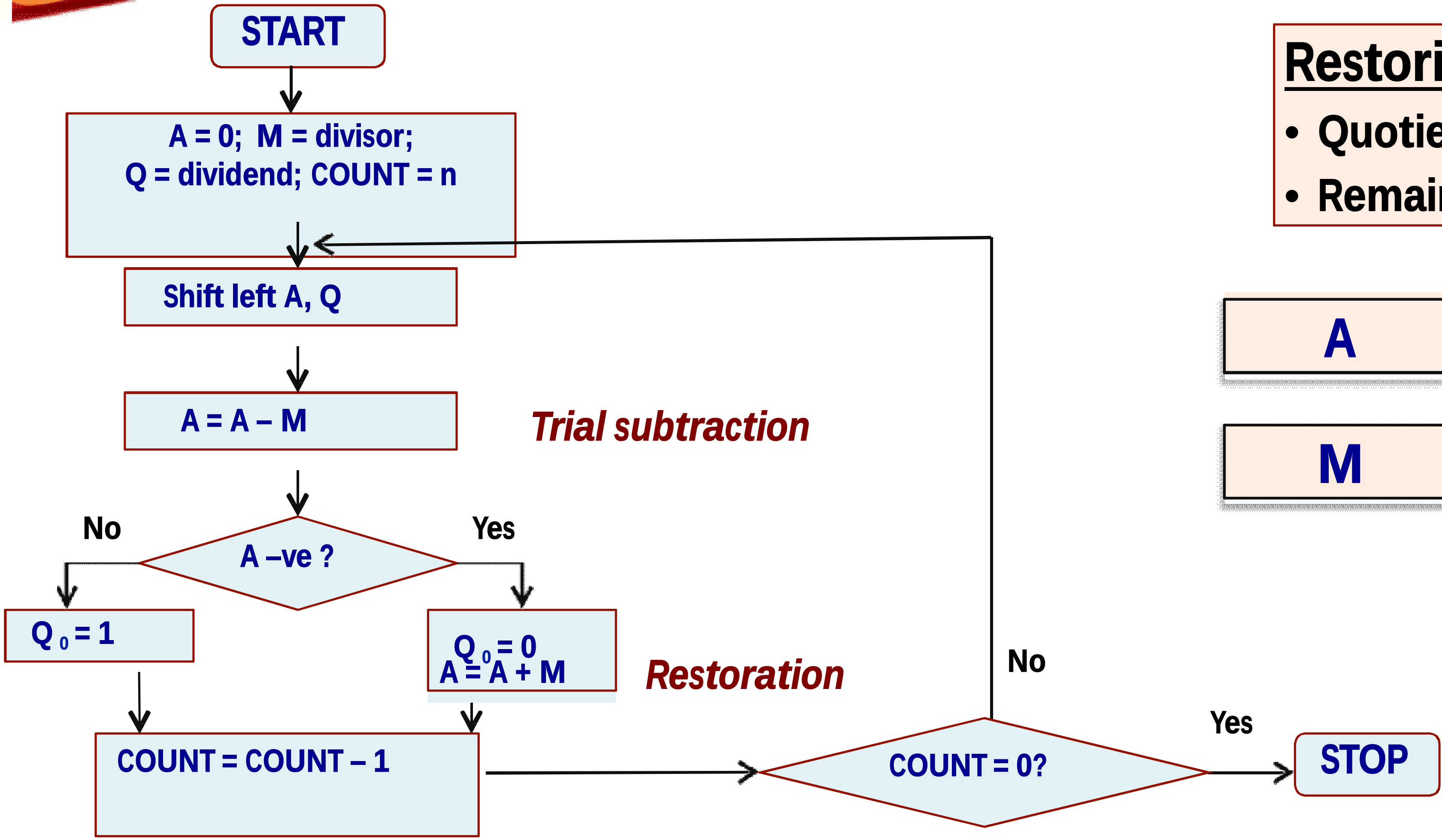
d) If the result is positive:

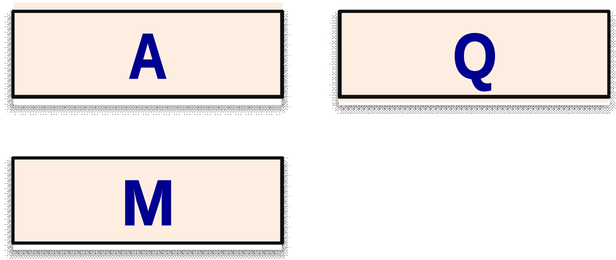
- Do not restore the intermediate result.
- Record 1 as the next quotient bit.



Restoring Division

- Quotient in Q
- Remainder in A





- **Analysis:**
  - For n-bit divisor and n-bit dividend, we iterate n times.
  - Number of trial subtractions: *n*
  - Number of restoring additions: *n/2* on the average
    - Best case: *0*
    - Worst case: *n*

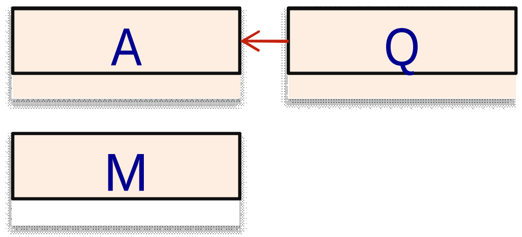
A Simple Example: 8/3 for 4-bit representation (n=4)

Initially:	0 0 0 0 0	1 0 0 0
	0 0 0 1 1	
Shift:	0 0 0 0 1	0 0 0 –
Subtract:		
Set Q_0:	1 1 1 1 0	
Restore:	0 0 0 1 1	
	0 0 0 0 1	0 0 0 0
Shift:	0 0 0 1 0	0 0 0 –
Subtract:		
Set Q_0:	1 1 1 1 1	
Restore:	0 0 0 1 1	
	0 0 0 1 0	0 0 0 0

Shift:	0 0 1 0 0	0 0 0 –
Subtract:		
Set Q_0:	0 0 0 0 1	
	0 0 0 0 0	0 0 0 1
Shift:	0 0 0 1 0	0 0 1 –
Subtract:		
Set Q_0:	1 1 1 1 1	
Restore:	0 0 0 1 1	
	0 0 0 1 0	0 0 1 0

Remainder	Quotient
00010 = 2	0010 = 2

Non-Restoring Division



The performance of restoring division algorithm can be improved by exploiting the following observation.

- In restoring division, what we do actually is:
  - – If A is positive, we shift it left and subtract M.
    - That is, we compute 2A – M.
  - – If A is negative, we restore is by doing A+M, shift it left, and then subtract M.
    - That is, we compute 2(A + M) – M = 2A + M.
- We can accordingly modify the basic division algorithm by eliminating the restoring step → ***NON-RESTORING DIVISION*.**

Shift left means multiplying by 2.

Basic steps in non-restoring division:

a)Start by initializing register A to 0, and repeat steps (b)-(d) *n* times.

b)If the value in register A is positive,

- Shift A and Q left by one bit position.
- Subtract M from A.

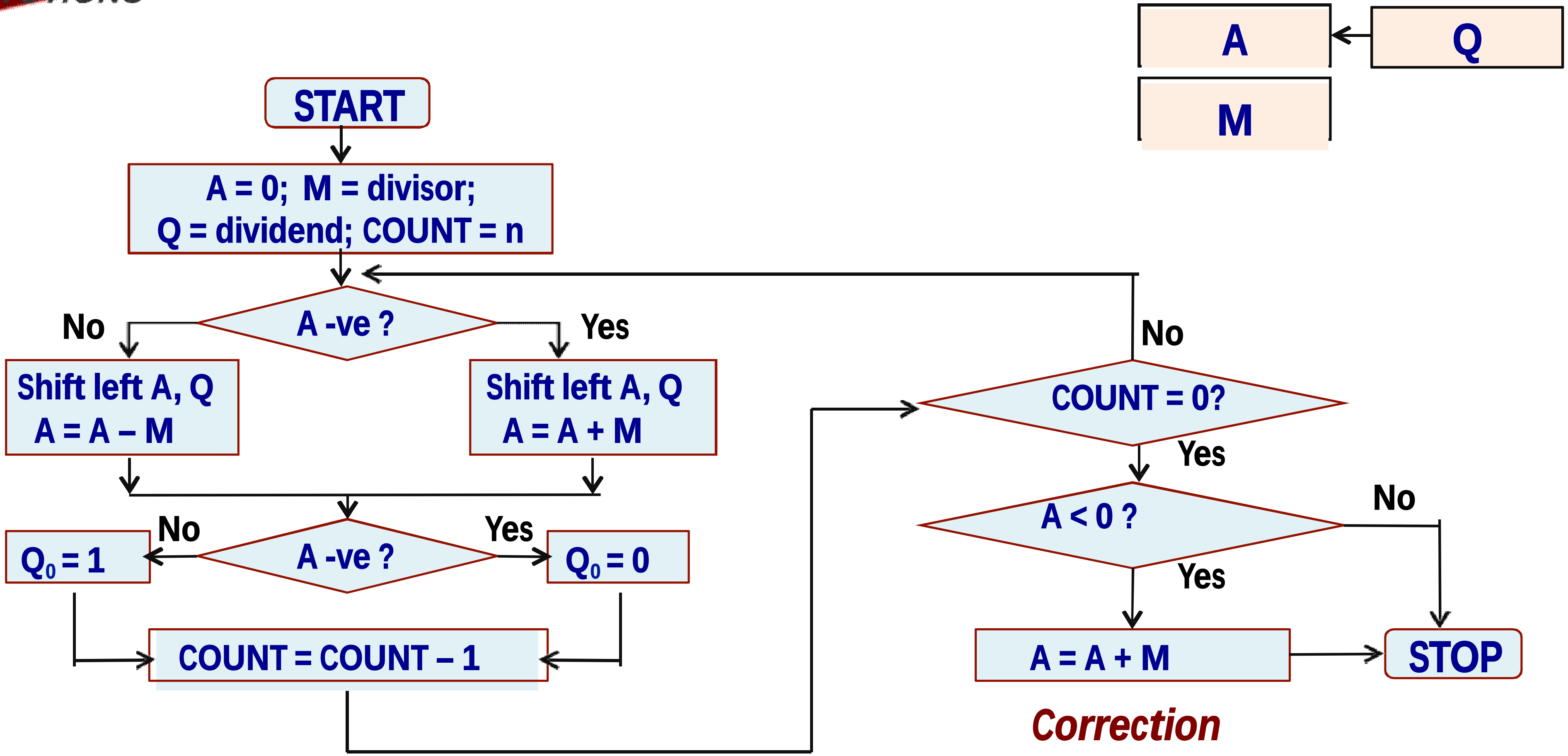
c) If the value in register A is negative,

- Shift A and Q left by one bit position.
- Add M to A.

c)If A is positive, set $Q_0 = 1$; else, set $Q_0 = 0$.

d)If A is negative, add M to A as a final corrective step.

Non-Restoring Division



A Simple Example: 8/3 for n=4

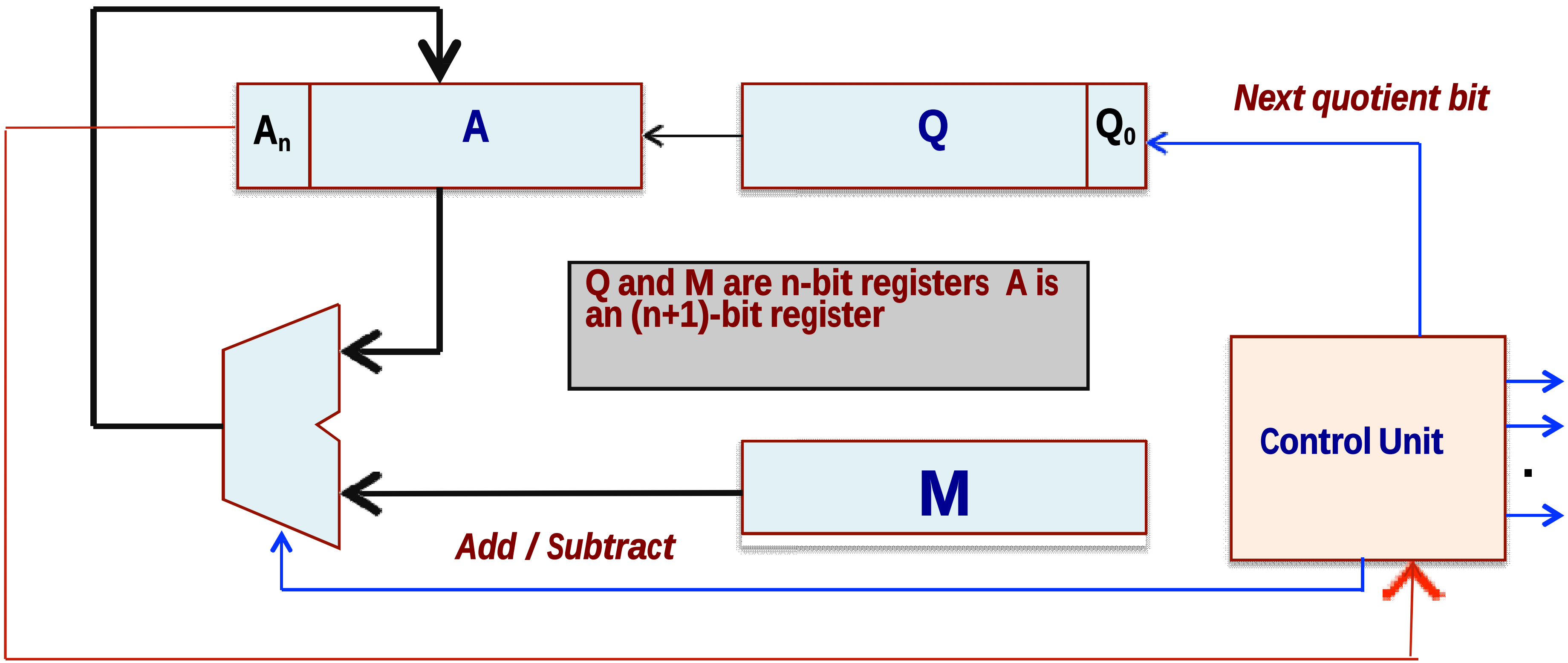
Initially:	0 0 0 0 0	1 0 0 0
Shift:	0 0 0 0 1	0 0 0 –
Subtract:	– 0 0 1 1	
Set Q_0:	(1) 1 1 1 0	0 0 0 (0)
Shift:	1 1 1 0 0	0 0 0 –
Add:	0 0 1 1	
Set Q_0:	(1) 1 1 1 1	0 0 0 (0)
Shift:	1 1 1 1 0	0 0 0 –
Add:	0 0 1 1	
Set Q_0:	(0) 0 0 0 1	0 0 0 (1)

Shift:	0 0 0 1 0	0 0 1 –
Subtract:	– 0 0 1 1	
Set Q_0:	(1) 1 1 1 1	0 0 1 (0)
Correction Add:		
	1 1 1 1 1	
	0 0 0 1 1	
	0 0 0 1 0	

Quotient
0010 = 2

Remainder
00010 = 2

Data Path for Non-Restoring Division



High Speed Dividers

- Some of the methods used to increase the speed of multiplication can also be modified to speed up division.
  - High-speed addition and subtraction.
  - High-speed shifting.
  - Combinational array divider (implementing restoring division).
- The main difficulty is that it is very difficult to implement division in a pipeline to improve the performance.
  - Unlike multiplication, where carry-save Wallace tree multipliers can be used for pipeline implementation.

TEXT BOOK

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, McGraw-Hill, 6th Edition 2012.

REFERENCES

1. David A. Patterson and John L. Hennessey, “Computer organization and design”, MorganKauffman ,Elsevier, 5th edition, 2014.
2. William Stallings, “Computer Organization and Architecture designing for Performance”, Pearson Education 8th Edition, 2010
3. John P.Hayes, “Computer Architecture and Organization”, McGraw Hill, 3rd Edition, 2002
4. M. Morris R. Mano “Computer System Architecture” 3rd Edition 2007
5. David A. Patterson “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann; 5th edition 2011

THANK YOU