ALGORITHMS FOR EXTRACTING SQUARE ROOTS AND CUBE ROOTS

by Hong Peng

Department of Mathematics, Northwest University,
Xi'an, People's Republic of China.

Abstract

This paper describes a kind of algorithms for fast extracting square roots and cube roots, their mathematical proofs, their revised algorithm formulae, and hardware implementation of the square root algorithm. These algorithms may be of no significance for large scale computer with fast division. But I am sure that it is effective and economical to apply these algorithms to the circuit designs of some mini- and micro-computers with general multiplication and division, such as nonrestoring division.

I. Introduction

Common used square root algorithms are based on the identity

$$n^2=1+3+5+\ldots+(2n-1)$$

or on the recurrence formula

$$y_{n+1}=\tfrac{1}{2}(y_n+\frac{A}{y_n}).$$

Common used cube root algorithms are based on the recurrence formula

$$y_{n+1}=\tfrac{1}{3}(2y_n+\frac{A}{y_n^2}).$$

These recurrence methods are fast for large scale computer with fast multiplication and division. But they are slow for common mini- and micro-computers with general multiplication and division, such as nonrestoring division, because computation of a square root by recurrence always spends several times of division time, and computation of a cube root by recurrence always spends several times of multiplication and division time. This paper describes a kind of algorithms for extracting square roots and cube roots, These algorithms are fast, economical and easily implemented for general mini- and micro-computers. The square root algorithm itself is very much like nonrestoring division algorithm. Its hardware implementation can be done by adding a little circuit to the nonrestoring division circuits. This is why we consider it economical and easily implemented. The time spent in extracting n-bit square root of a 2n-bit binary positive integer is n fixed-point addition periods. The time is the same as that spent in a nonrestoring division, in which dividend is 2n-bit and divisor is n-bit.This is why the algorithm is said to be fast. The cube root algorithm is much like the square root algorithm. The time spent in extracting n-bit cube root of a 3n-bit binary positive integer is 4n fixed-point addition periods. If we use above-mentioned time, both square roots and cube roots are accurate to within one unit in the last place. If we spend more additional fixed-point periods by one (or four) and the result is rounded, then the square roots (or respectively cube roots) may be accurate to within one half unit in the last place. So it seems to me that when some mini- or micro-computer with general nonrestoring division is going to be designed, we should at the same time add to them this kind of square root circuit (also cube root circuit, if necessary). Thus, without spending much money we can obtain a square root instruction, which is as fast as nonrestoring division instruction (also a cube root instruction, if necessary).

II. Square root algorithm

Let us suppose that we desire to extract the square root of a 2n-bit binary positive integer

$$A: a_1a_2a_3a_4\cdots a_{2n-3}a_{2n-2}a_{2n-1}a_{2n},$$

where a_i denotes 1 or 0 in i-th bit position. We divide it into n segments, each contains two bits:

$$A: a_1a_2,a_3a_4,\ldots,a_{2n-3}a_{2n-2},a_{2n-1}a_{2n}.$$

From A, we construct a sequence of positive integers

$$A_1,A_2,\ldots,A_n,$$

where

$$A_i: a_1a_2,a_3a_4,\ldots,a_{2i-1}a_{2i},\quad (i=1,2,\ldots,n).$$

In other words, A_i consists of left-most i segments of A. Obviously, $A_n=A$.

Our basic idea is as follows. First, we try to find the square root of A_1. Then we try to find several common rules such that whenever the square root R_i of some A_i $(i=1,2,\ldots,n-1)$ has been found, we can use

CH1630-3/81/0000/0121$00.75

R_i to find the square root R_{i+1} of next number A_{i+1}. Thus, beginning with A_1 and recurring, we can obtain the square root of A_n, i.e., the square root of A.

A_1 is a 2-bit binary integer, which has only the possible values 0,1,2,3. Therefore R_1 must be 0 or 1. And $R_1=0$ if and only if $A_1=0$. Otherwise, $R_1=1$. Thus we obtain

RULE 1. If $A_1-1\geq 0$, then $R_1=1$;
if $A_1-1<0$, then $R_1=0$.

This RULE 1 is easily implemented by logic circuits in one fixed-point addition period.

Now suppose that we have found the square root R_i of A_i, i.e.,

$$A_i=R_i^2+C_i \quad (i=1,2,\ldots,n-1), \tag{1}$$

where C_i is the remainder in the extraction. Then we must have

$$0\leq C_i\leq 2R_i. \tag{2}$$

Now we consider A_{i+1}. Obviously,

$$A_{i+1}=4A_i+J_{i+1}, \tag{3}$$

where J_{i+1} denotes the number constructed from the two binary bits of (i+1)-th segment J_{i+1}: $a_{2i+1}a_{2i+2}$. Obviously,

$$0\leq J_{i+1}\leq 3. \tag{4}$$

Substituting (1) into (3), we obtain

$$A_{i+1}=4R_i^2+(4C_i+J_{i+1}). \tag{5}$$

Adding 4 times of (2) to (4), we obtain

$$0\leq 4C_i+J_{i+1}\leq 8R_i+3.$$

Thus

$$4R_i^2\leq A_{i+1}\leq 4R_i^2+(8R_i+3)<4R_i^2+8R_i+4,$$

i.e.,

$$(2R_i)^2\leq A_{i+1}<(2R_i+2)^2.$$

Therefore R_{i+1} must be $2R_i$ or $2R_i+1$, so that we obtain

RULE 2. If the square root of A_i is R_i, then the square root R_{i+1} of A_{i+1} may be obtained by left-shifting R_i one bit and then adding 1 or 0 to its last bit.

Should we add 1 or add 0? Rewriting (5) into

$$A_{i+1}=(2R_i+1)^2+((4C_i+J_{i+1})-(4R_i+1)),$$

we see that

if $(4C_i+J_{i+1})-(4R_i+1)\geq 0$,
then $A_{i+1}\geq(2R_i+1)^2$, hence $R_{i+1}=2R_i+1$,
(There may be a remainder);
if $(4C_i+J_{i+1})-(4R_i+1)<0$,
then $A_{i+1}<(2R_i+1)^2$, hence $R_{i+1}=2R_i+0$,
(There may be a remainder).

Thus we obtain

RULE 3. Compute $D_{i+1}=(4C_i+J_{i+1})-(4R_i+1)$.
If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$;
if $D_{i+1}<0$, then $R_{i+1}=2R_i+0$.

This RULE 3 is easily implemented by logic circuits too. C_i is the remainder in extracting the square root of A_i. $4C_i+J_{i+1}$ can be obtained by left-shifting C_i two bits and then adding i-th segment J_{i+1} to its last two bit position. $4R_i+1$ can be obtained by left-shifting R_i two bits and adding 1 to its last bit position. Subtraction can be performed by adder. The left-shift operation and add 1 operation can simultaneously be performed in process of transfering data to the adder. Finally, $2R_i+1$ or $2R_i+0$ can be implemented by left-shifting R_i one bit and adding 1 or 0 to its last bit. We should add 1 or add 0 according to the sign of the output of the adder. All these operations can be completed in one fixed-point addition period. Thus we can find the square root R_{i+1} of A_{i+1} by using the square root R_i of A_i and the remainder C_i in one fixed-point addition period.

However, a problem has yet to be solved. We use the sign of the difference D_{i+1} to decide whether 1 or 0 should be added to $2R_i$. If $D_{i+1}\geq 0$, we have a root $2R_i+1$; in this case, D_{i+1} is the remainder C_{i+1} in extraction of A_{i+1}. If $D_{i+1}<0$, we have a root $2R_i+0$; in this case we can see from (5) that the remainder C_{i+1} is $4C_i+J_{i+1}$ instead of D_{i+1}. To obtain a true remainder, we must add $4R_i+1$ to D_{i+1}. That is, when $D_{i+1}<0$ and the square root of A_{i+1} is $2R_i+0$, we must first restore the remainder, and then we can decide how to obtain the square root in next step according to the same RULE 3. But restoring remainder has to spend one fixed-point addition period, this is not a good approach. We therefore consider whether we can use the difference D_{i+1} (instead of the remainder C_{i+1}) and R_i to obtain the square root in next step. For this purpose, we note Fig.1. In Fig.1 we substitute the expression of C_i into that of D_{i+1}, and note that $R_i=2R_{i-1}$, we have

$$\begin{aligned}D_{i+1}&=4(D_i+(4R_{i-1}+1))+J_{i+1}-(4R_i+1)\\&=(4D_i+J_{i+1})+(4R_i+3).\end{aligned}$$

That is, if the last bit of R_i is 0, then the judge condition for finding R_{i+1} can be obtained by left-shifting D_i (which is a negative number and is not the remainder C_i) two bits and adding i-th segment J_{i+1} in its last two bit position, and finally adding $(4R_i+3)$ to it instead of subtracting $(4R_i+1)$

Number to be extracted	A_{i-1}	A_i	A_{i+1}
Judge condition		$D_i=(4C_{i-1}+J_i)-(4R_{i-1}+1)<0$	$D_{i+1}=(4C_i+J_{i+1})-(4R_i+1)$
Root	R_{i-1}	$R_i=2R_{i-1}+0$(which we suppose)	
Remainder	C_{i-1}	$C_i=D_i+(4R_{i-1}+1)$	

Fig.1

from it. Thus we obtain

RULE 4. Suppose that the last bit of the root R_i is 0.

Compute $D_{i+1}=(4D_i+J_{i+1})+(4R_i+3)$,

If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$;

if $D_{i+1}<0$, then $R_{i+1}=2R_i+0$.

When the last bit of the root R_i is 1, D_i is C_i and we still use the RULE 3.

The RULE 4 is also easily implemented by logic circuits in one fixed-point addition period. This is similar to that of RULE 3.

Combining the RULEs 1,2,3,4 together, we obtain the desired square root algorithm. Now we revise it as follows.

In order to extract the square root of a 2n-bit binary integer A, we must set up three registers:

D register, used for storing D_i;

A register, used for storing number A to be extracted;

R register, used for storing square root R_i.

These registers are not necessary to be new, some of them may be those used by multiplication and division.(See IV below).

The whole process is divided into n cycles, each of which spends a fixed-point addition time. The logic functions performed in various cycle are shown in Fig.2. Obviously, the whole process spends n fixed-point addition periods, and the root obtained is accurate to within one unit in the last place.

III. Cube root algorithm

Let us suppose that we desire to extract the cube root of a 3n-bit binary positive integer

$$A: a_1a_2a_3a_4a_5a_6\cdots a_{3n-2}a_{3n-1}a_{3n},$$

where a_i denotes 1 or 0 in i-th bit position. We divide it into n segments, each contains three bits:

$$A: a_1a_2a_3,a_4a_5a_6,\cdots,a_{3n-2}a_{3n-1}a_{3n}.$$

From A, we con-struct a sequence of positive integers

$$A_1,A_2,A_3,\cdots,A_n,$$

where

$$A_i: a_1a_2a_3,a_4a_5a_6,\cdots,a_{3i-2}a_{3i-1}a_{3i},\quad (i=1,2,\ldots,n).$$

In other words, A_i consists of left-most i segments of A. Obviously, $A_n=A$.

Our basic idea is as follows. First, we try to find the cube root R_1 of A_1. Then we try to find several common rules such that whenever the cube root R_i of some A_i (i=1, 2,...,n-1)has found, we can use R_i to find the cube root R_{i+1} of next number A_{i+1}. Thus, beginning with A_1 and recurring, we can obtain the cube root of A_n, i.e., the cube root of A.

A_1 is a 3-bit binary integer, which has only the possible values 0,1,...,7. Therefore R_1 must be 0 or 1. And $R_1=0$ if and only if $A_1=0$. Otherwise $R_1=1$.Thus we obtain

Cycle	Operation	Function
1	Compute $D_1=A_1-1$. If $D_1\geq 0$, left-shift root one bit and add 1 to its last bit; if $D_1<0$, left-shift root one bit and add 0 to its last bit.	Extract the first bit of square root
2 | n	1. When the last bit of root R_{i-1} is 1 (i=2,3,...,n), compute $D_i=(4D_{i-1}+J_i)-(4R_{i-1}+1)$. If $D_i\geq 0$, left-shift root one bit and add 1 to its last bit; if $D_i<0$, left-shift root one bit and add 0 to its last bit. 2. When the last bit of root R_{i-1} is 0 (i=2,3,...,n), compute $D_i=(4D_{i-1}+J_i)+(4R_{i-1}+3)$. If $D_i\geq 0$, left-shift root one bit and add 1 to its last bit; if $D_i<0$, left-shift root one bit and add 0 to its last bit.	Extract i-th bit of square root (i=2,3,...,n)

Fig. 2

RULE 1. If $A_1-1\geq 0$, then $R_1=1$;
if $A_1-1<0$, then $R_1=0$.

The RULE 1 is easily implemented by logic circuits in one fixed-point addition period.

Now suppose that we have found the cube root R_i of A_i, i.e.,

$$A_i=R_i^3+C_i \quad (i=1,2,\ldots,n-1) \tag{1}$$

where C_i is the remainder in the extraction. Then we must have

$$0\leq C_i\leq 3R_i^2+3R_i. \tag{2}$$

Now we consider A_{i+1}. Obviously,

$$A_{i+1}=8A_i+J_{i+1}, \tag{3}$$

where J_{i+1} denotes the number constructed from the three binary bits of (i+1)-th segment:

$$J_{i+1}: \ a_{3i+1}a_{3i+2}a_{3i+3}.$$

Obviously,

$$0\leq J_{i+1}\leq 7 \tag{4}$$

Substituting (1) into (3), we obtain

$$A_{i+1}=8R_i^3+(8C_i+J_{i+1}). \tag{5}$$

Adding 8 times of (2) to (4), we obtain

$$0\leq 8C_i+J_{i+1}\leq 24R_i^2+24R_i+7.$$

Thus

$$8R_i^3\leq A_{i+1}\leq 8R_i^3+24R_i^2+24R_i+7$$
$$<8R_i^3+24R_i^2+24R_i+8$$

i.e.,

$$(2R_i)^3\leq A_{i+1}<(2R_i+2)^3.$$

Therefore R_{i+1} must be $2R_i$ or $2R_i+1$. So that we obtain

RULE 2. If the cube root of A_i is R_i, then the cube root R_{i+1} of A_{i+1} may be obtained by left-shifting R_i one bit and then adding 1 or 0 to its last bit.

Should we add 1 or add 0? Rewriting (5) into

$$A_{i+1}=(2R_i+1)^3+((8C_i+J_{i+1})-(12R_i^2+6R_i+1)),$$

we see that

if $(8C_i+J_{i+1})-(12R_i^2+6R_i+1)\geq 0$,
then $A_{i+1}\geq(2R_i+1)^3$, hence $R_{i+1}=2R_i+1$,
(There may be a remainder);
if $(8C_i+J_{i+1})-(12R_i^2+6R_i+1)<0$,
then $A_{i+1}<(2R_i+1)^3$, hence $R_{i+1}=2R_i+0$,
(There may be a remainder).

Thus we can obtain a judge rule: We subtract

$$P_i: 12R_i^2+6R_i+1 \tag{6}$$

from $8C_i+J_{i+1}$, and then we use the sign of

$$D_{i+1}=(8C_i+J_{i+1})-P_i \tag{7}$$

to determine whether we should add 1 or 0 to the last bit of R_{i+1}. P_i is called a determinant.

The problem seems to be solved as in extraction of a square root. But in practice this way is not feasible, because when we find the value of P_i, we must compute R_i^2. Square operation generally takes a long time and we will lose "high-speed" meaning. Therefore the key of the problem is how to find rapidly the value of P_i without any square operation. For this purpose, we add a special register P to store the value of P_i and P_i. P_i will be introduced in the following.

There is yet a problem as in the case of extraction of a square root. If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$; in this case, D_{i+1} is the remainder C_{i+1} in extraction of A_{i+1}. If $D_{i+1}<0$, then $R_{i+1}=2R_i+0$; in this case, D_{i+1} is not the remainder in extraction of A_{i+1}. The true remainder should be $C_{i+1}=D_{i+1}+P_i$. That is, we must restore the remainder, this requires one more fixed-point addition period. As in the case of extraction of square roots, we expect to obtain a judge condition for finding the root R_{i+2} from the difference D_{i+1} instead of the remainder C_{i+1}. We will combine the two foregoing problems and solve them as follows.

(I) When the last bit of R_i is 1, how do we find the judge condition for finding the last bit of root R_{i+1}? We note Fig.3. In Fig. 3 we substitute the expression of R_i into that of P_i and note the expression of P_{i-1}, we have

$$P_i=4(12R_{i-1}^2+6R_{i-1}+1)+18(2R_{i-1}+1)-3$$
$$=4P_{i-1}+18R_i-3, \tag{8}$$

and the judge condition for finding the last bit of R_{i+1} is

$$D_{i+1}=(8C_i+J_{i+1})-P_i=(8D_i+J_{i+1})-P_i. \tag{9}$$

From this we see that the determinant P_i may be obtained by simple addition and subtraction from P_{i-1} of the last cycle and R_i of this cycle instead of being obtained by square operation from R_i. This saves greatly calculating time.

The formulae (8) and (9) can easily be implemented by logical circuits in four steps. In step 1, we find $4P_{i-1}+16R_i$. In step 2, we add $2R_i$ to $(4P_{i-1}+16R_i)$ to obtain $(4P_{i-1}+18R_i)$. In step 3, we subtract 3 from $(4P_{i-1}+18R_i)$ to obtain P_i. Finally, in step 4, we find D_{i+1} and obtain the cube root R_{i+1} according to the sign of the adder output. Each foregoing step takes one fixed-point addition period. Therefore we use altogether four fixed-point addition periods to obtain R_{i+1}.

(II) When the last bit of R_i is 0, how do we find the judge condition for finding the last bit of root R_{i+1}? We note Fig.4.

Number to be extracted	A_{i-1}	A_i	A_{i+1}
Judge condition		$D_i=(8C_{i-1}+J_i)-P_{i-1}\geq 0$	$D_{i+1}=(8C_i+J_{i+1})-P_i$
Root	R_{i-1}	$R_i=2R_{i-1}+1$	
Remainder	C_{i-1}	$C_i=D_i$	
Determinant	$P_{i-1}=12R_{i-1}^2+6R_{i-1}+1$	$P_i=12R_i^2+6R_i+1$	

Fig.3

Number to be extracted	A_{i-1}	A_i	A_{i+1}
Judge condition		$D_i=(8C_{i-1}+J_i)-P_{i-1}<0$	$D_{i+1}=(8C_i+J_{i+1})-P_i$
Root	R_{i-1}	$R_i=2R_{i-1}+0$	
Remainder	C_{i-1}	$C_i=D_i+P_{i-1}$	
Determinant	$P_{i-1}=12R_{i-1}^2+6R_{i-1}+1$	$P_i=12R_i^2+6R_i+1$	

Fig.4

In Fig.4, substituting the expression of R_i into that of P_i and note the expression of P_{i-1}, we have

$$P_i=4P_{i-1}-6R_i-3. \tag{10}$$

And substituting (10) and the expression of C_i into that of D_{i+1}, we have

$$D_{i+1}=(8D_i+J_{i+1})+(4P_{i-1}+6R_i+3).$$

Let

$$P_i'=4P_{i-1}+6R_i+3, \tag{11}$$

then

$$D_{i+1}=(8D_i+J_{i+1})+P_i'. \tag{12}$$

Thus we see that D_{i+1} can be directly obtained from the difference D_i instead of the remainder C_i. This saves the time for restoring remainder. However, instead of subtracting P_i from $(8D_i+J_{i+1})$, we add another formula P_i' to $(8D_i+J_{i+1})$.

Only three fixed-point addition periods are enough to find P_i'. In step 1 we compute $4P_{i-1}+4R_i$. In step 2 we add $2R_i$ to $(4P_{i-1}+4R_i)$ to obtain $(4P_{i-1}+6R_i)$. In step 3 we add 3 to $(4P_{i-1}+6R_i)$ to obtain P_i'. P_i' is still stored in the register P. In step 4 we find D_{i+1} and R_{i+1}. Thus we also use only four fixed-point addition periods to obtain R_{i+1}.

The relation between P_i and P_i' can be obtained from (10) and (11):

$$P_i=P_i'-12R_i-6. \tag{13}$$

Combining the foregoing (I) and (II), we have

RULE 3. (1) When the last bit of the root R_i is 1, compute

$$D_{i+1}=(8D_i+J_{i+1})-(4P_{i-1}+18R_i-3).$$

If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$;

if $D_{i+1}<0$, then $R_{i+1}=2R_i+0$.

(2) When the last bit of the root R_i is 0, compute

$$D_{i+1}=(8D_i+J_{i+1})+(4P_{i-1}+6R_i+3).$$

If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$;

if $D_{i+1}<0$, then $R_{i+1}=2R_i+0$.

But a problem has yet to be solved. We must use P_{i-1} to find D_{i+1}, whether in case of (1) or in case of (2). Sometimes there is not , however, a ready-made P_{i-1}. For example, when we found D_i in the last cycle, if the last bit of R_{i-1} is 0, then

$$D_i=(8D_{i-1}+J_i)+P_{i-1}' \text{ (according to (12))}.$$

At that time the content of the register P is P_{i-1}' instead of P_{i-1}. In this case, we must find a formula for deriving D_{i+1} from P_{i-1}'. From (13), we obtain

$$P_{i-1}=P_{i-1}'-12R_{i-1}-6. \tag{13}'$$

Substituting (13)' into (8) and noticing that $R_i=2R_{i-1}+1$, we have

$$P_i=4P_{i-1}'-6R_i-3. \tag{14}$$

Substituting (13)' into (11) and noticing that $R_i=2R_{i-1}$, we have

$$P_i'=4P_{i-1}'-18R_i-21. \tag{15}$$

Thus we obtain

RULE 4. (1) When the last bit of R_{i-1} is 0 and the last bit of R_i is 1, compute

$$D_{i+1}=(8D_i+J_{i+1})-(4P_{i-1}'-6R_i-3).$$

If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$;

if $D_{i+1}<0$, then $R_{i+1}=2R_i+0$.

(2) When the last bit of R_{i-1} is 0 and the last bit of R_i is 0, compute

$$D_{i+1}=(8D_i+J_{i+1})+(4P_{i-1}'-18R_i-21).$$

If $D_{i+1}\geq 0$, then $R_{i+1}=2R_i+1$;

if $D_{i+1}<0$, then $R_{i+1}=2R_i+0$.

Combining the RULEs 1,2,3,4 together, we obtain the desired cube root algorithm. Now we revise it as follows.

To extract the cube root of a 3n-bit binary integer A, we must set up four registers:

D register, used for storing D_i;

A register, used for storing the number A to be extracted;

R register, used for storing the cube root R_i;(Let its last two bits be $r_{n-1}r_n$)

P register, used for storing P_i or P_i'.

Note that these registers are not necessary to be new, some of them may be those used by multiplication, division and extraction of square roots.

Whole process is divided into n cycles, each cycle is subdivided into 4 steps. Each step takes a fixed-point addition period. The logical functions performed in various steps are shown in Fig.5. Obviously, the whole process takes 4n fixed-point addition periods, and the cube root obtained is accurate to within one unit in the last place.

Cycle	Step	Operation	Function
1	1	Empty	
	2	Empty	
	3	Set that (P)=1, i.e., let P_0=1.	
	4	Compute $D_1=A_1-P_0$. If $D_1\geq 0$, left-shift root one bit and set r_n=1; if $D_1<0$, left-shift root one bit and set r_n=0.	Find the first bit of cube root
2	1	(1) When r_n=1, compute $P_1^*=4P_0+16R_1$. (2) When r_n=0, compute $P_1^*=4P_0+4R_1$.	Find the value of P_1 or P_1'
	2	Compute $P_1^{**}=P_1^*+2R_1$.	
	3	(1) When r_n=1, compute $P_1^{***}=P_1^{**}-3$. (2) When r_n=0, compute $P_1^{***}=P_1^{**}+3$	
	4	(1) When r_n=1, compute $D_2=(8D_1+J_2)-P_1$. (2) When r_n=0, compute $D_2=(8D_1+J_2)+P_1'$ If $D_2\geq 0$, left-shift root one bit and set r_n=1; if $D_2<0$, left-shift root one bit and set r_n=0.	Find the second bit of cube root
3 ... n	1	(1) When $r_{n-1}r_n$=11, compute $P_i^*=4P_{i-1}+16R_i$. (2) When $r_{n-1}r_n$=10, compute $P_i^*=4P_{i-1}+4R_i$. (3) When $r_{n-1}r_n$=01, compute $P_i^*=4P_{i-1}-4R_i$. (4) When $r_{n-1}r_n$=00, compute $P_i^*=4P_{i-1}-16R_i$.	Find the values of P_i or P_i' (i=2,3,...,n-1)
	2	(1) When r_{n-1}=1, compute $P_i^{**}=P_i^*+2R_i$. (2) When r_{n-1}=0, compute $P_i^{**}=P_i^*-2R_i$.	
	3	(1) When r_n=1, compute $P_i^{***}=P_i^{**}-3$. (2) When $r_{n-1}r_n$=10, compute $P_i^{***}=P_i^{**}+3$. (3) When $r_{n-1}r_n$=00, compute $P_i^{***}=P_i^{**}-21$.	
	4	(1) When r_n=1, compute $D_{i+1}=(8D_i+J_{i+1})-P_i$. (2) When r_n=0, compute $D_{i+1}=(8D_i+J_{i+1})+P_i'$ If $D_{i+1}\geq 0$, left-shift root one bit and set r_n=1 if $D_{i+1}<0$, left-shift root one bit and set r_n=0	Find the (i+1)-th bit of cube root

Fig.5

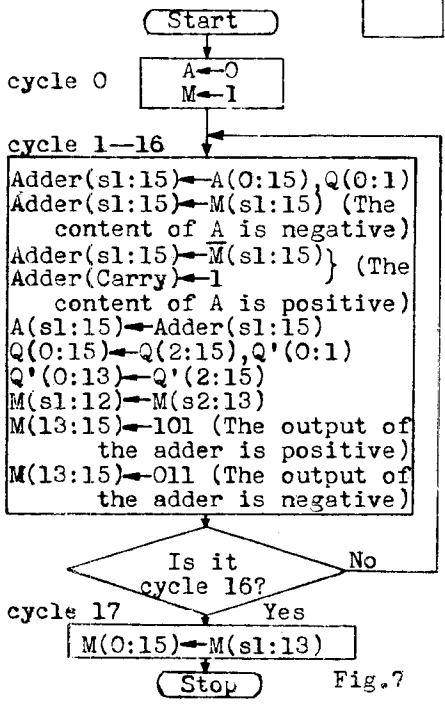

Fig.7

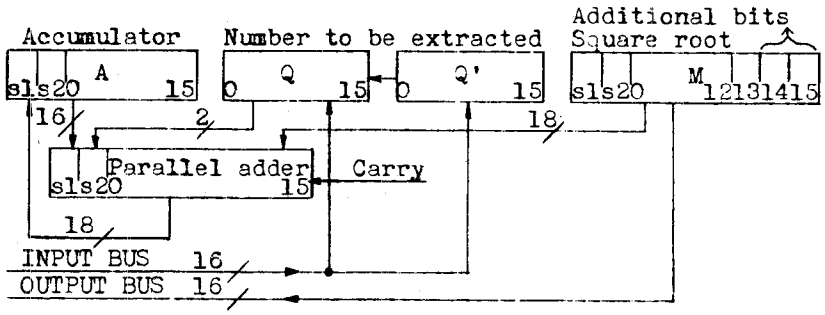

Fig.6

IV. Hardware implementation of the square root algorithm

In 1977, a "Fast Processing Unit" was designed and manufactured at the Department of Mathematics, Northwest Univercity, People's Republic of China. It is used as a special peripheral equipment of the (Chinese) DJS-130 minicomputer. Its square root circuit is designed according to the principle of this paper. Fig.6 and Fig.7 are its square root block diagram and flowchart, respectively. Note that Fig.6 is simultaneously used for multiplication, division and square root,(with suitable change).