**ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN ĐIỆN TỬ

---------------o0o---------------

****

**BÁO CÁO BÀI TẬP LỚN**

**KỸ THUẬT SỐ NÂNG CAO**

**THIẾT KẾ MÁY TÍNH DẤU CHẤM ĐỘNG**

**IEEE 753, 32 BIT**

**GVHD:** TS. Trần Hoàng Linh

**SVTH:** 1. Nguyễn Hoàng Phụng - 1710264

2. Nguyễn Lâm Viên - 1713939

.

**TP.HỒ CHÍ MINH, THÁNG 12 NĂM 2020**

**LỜI CẢM ƠN**

Trên thực tế không có sự thành công nào mà không gắn liền với những sự hỗ  trợ, giúp đỡ  dù ít  hay  nhiều,  dù  trực tiếp  hay  gián  tiếp  của  người khác. Trong suốt thời gian từ khi bắt đầu học tập ở giảng đường đại học đến nay, em đã nhận được rất nhiều sự quan tâm, giúp đỡ của quý Thầy Cô, gia đình và bạn bè.

Với lòng biết ơn  sâu sắc  nhất, em xin gửi  đến quý  Thầy  Cô  ở Khoa Điện Điện tử– Trường Đại Học Bách Khoa TP.HCM đã cùng với tri thức và tâm huyết của mình để truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường.

Em xin chân thành cảm ơn TS. Trần Hoàng Linh và Anh Trịnh Vũ Đăng Nguyên đã hướng dẫn chúng em qua từng buổi học trên lớp cũng như những buổi nói chuyện,  thảo  luận về  lĩnh  vực  sáng  tạo trong  nghiên  cứu  khoa  học.  Nếu không có những lời hướng dẫn, dạy bảo của thầy thì em nghĩ bài tập lớn này của em rất khó có thể hoàn thiện được. Một lần nữa, em xin chân thành cảm ơn Thầy và Anh.

Bài báo cáo này không tránh khỏi những thiếu sót là điều chắc chắn, em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy Cô và các bạn để kiến thức của em trong lĩnh vực này được hoàn thiện hơn.

Trân trọng.

*Tp. Hồ Chí Minh, ngày 31 tháng 12 năm 2020 .*

**Sinh viên**

**TÓM TẮT BÀI TẬP LỚN**

Trong bài tập lớn này, nhóm đã tiến hành thiết kế một máy tính dấu chấm động thực hiện các phép toán (+, -, \*, /) giữa 2 số floating point 32 bit IEEE – 754 và một số phần bonus:

* Phép cộng/ trừ: 2 giải thuật Full Adđer và IHA, IFA
* Phép nhân: 2 giải thuật Shift cộng và Booth recording
* Phép chia: 1 giải thuật Shift trừ
* Căn bậc n
* Chuyển ngõ vào và ngõ ra thành dạng thập phân tương ứng.

**MỤC LỤC**

[1. Các bộ dùng chung: 1](#_Toc60607652)

[1.1. Normalize: 1](#_Toc60607653)

[1.2. Mux: 2](#_Toc60607654)

[1.3. Shift right: 2](#_Toc60607655)

[1.4. Shift left: 3](#_Toc60607656)

[1.5. Check special: 4](#_Toc60607657)

[1.6. Find1: 4](#_Toc60607658)

[2. Bộ cộng trừ 6](#_Toc60607659)

[2.1. Full adder: (Thuật toán 1) 6](#_Toc60607660)

[2.1.1. Full adder 1bit: 6](#_Toc60607661)

[2.1.2. Full adder n bit: 6](#_Toc60607662)

[2.2. IHA, IFA adder: (Thuật toán 2) 7](#_Toc60607663)

[2.3. Check sign of subtrahend: 8](#_Toc60607664)

[2.4. Compare exponent: 8](#_Toc60607665)

[2.5. Significand alignment: 9](#_Toc60607666)

[2.6. Calculate significand: 9](#_Toc60607667)

[2.7. Chuẩn hóa: 9](#_Toc60607668)

[2.8. Kết quả mô phỏng: 9](#_Toc60607669)

[2.8.1. Full adder: 9](#_Toc60607670)

[2.8.2. IHA, IFA: 9](#_Toc60607671)

[3. Bộ nhân: 11](#_Toc60607672)

[3.1. Shift cộng: (giải thuật 1) 11](#_Toc60607673)

[3.2. Bootler: (giải thuật 2) 11](#_Toc60607674)

[3.3. Kết quả mô phỏng: 13](#_Toc60607675)

[3.3.1. Shift cộng: 13](#_Toc60607676)

[3.3.2. Booth recording: 14](#_Toc60607677)

[4. Bộ chia: 15](#_Toc60607678)

[4.1. Shift trừ: 15](#_Toc60607679)

[4.2. Kết quả mô phỏng: 15](#_Toc60607680)

[5. Căn bậc n: 16](#_Toc60607681)

[5.1. Giải thuật 16](#_Toc60607682)

[5.2. Kết quả mô phỏng: 16](#_Toc60607683)

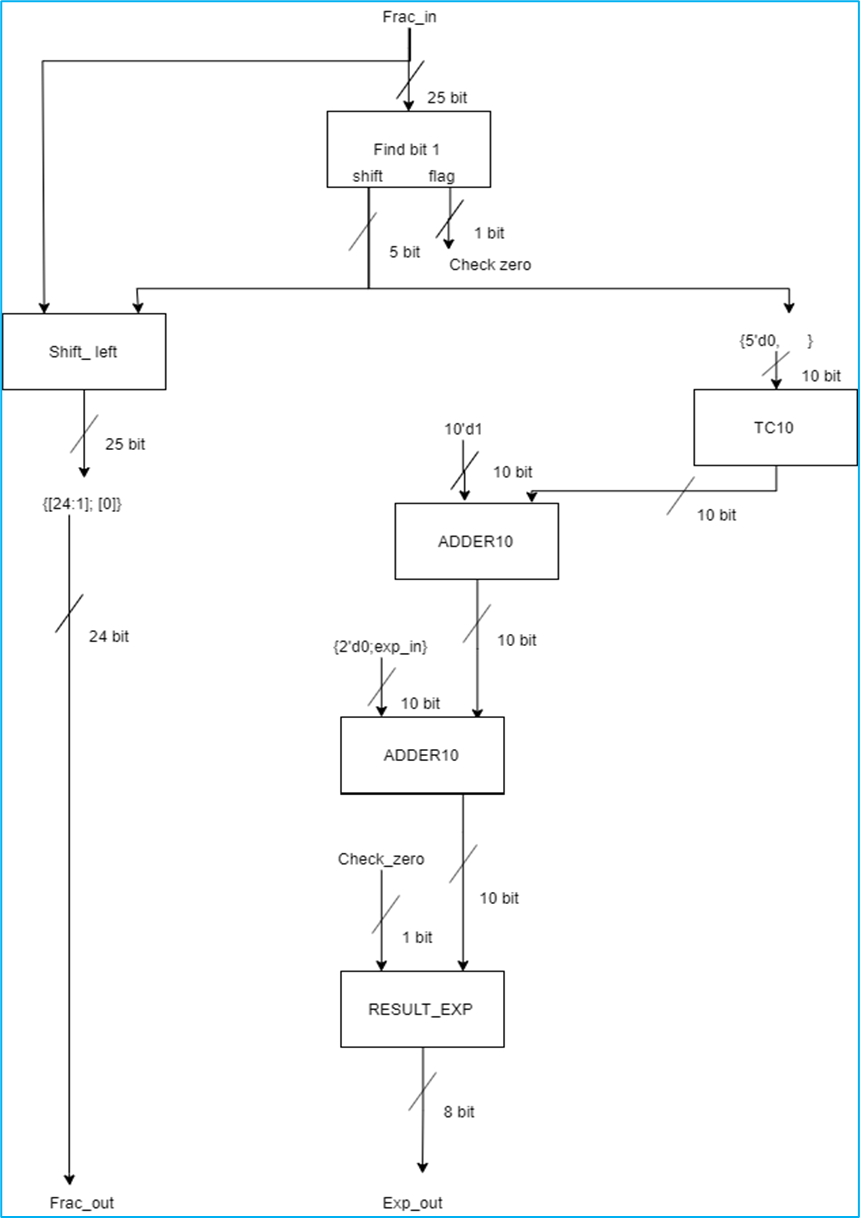
[6. Convert binary to decimal: 18](#_Toc60607684)

[6.1. Giải thuật: 18](#_Toc60607685)

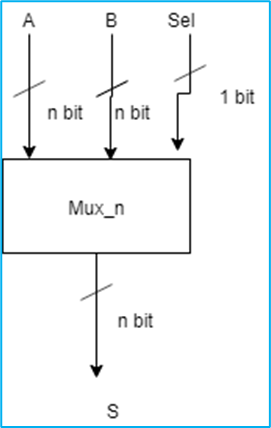
[6.2. Kết quả mô phỏng: 19](#_Toc60607686)

# Các bộ dùng chung:

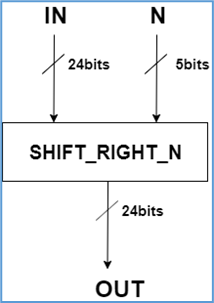
## Normalize:

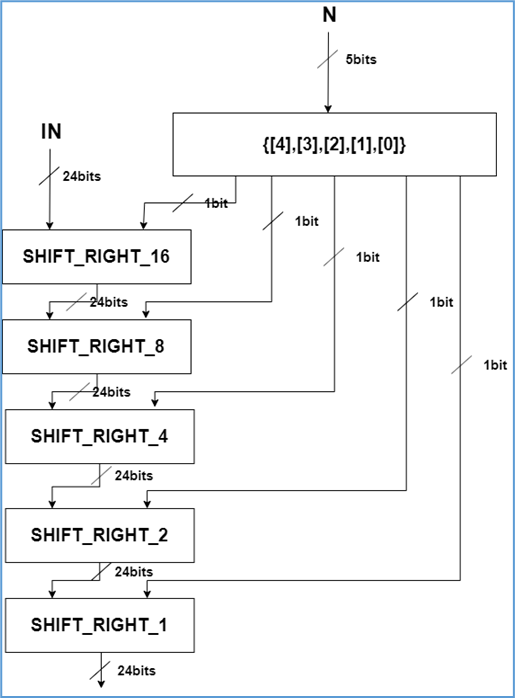


## Mux:

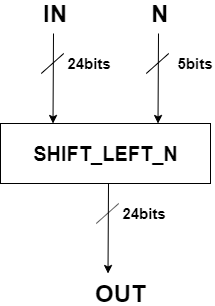


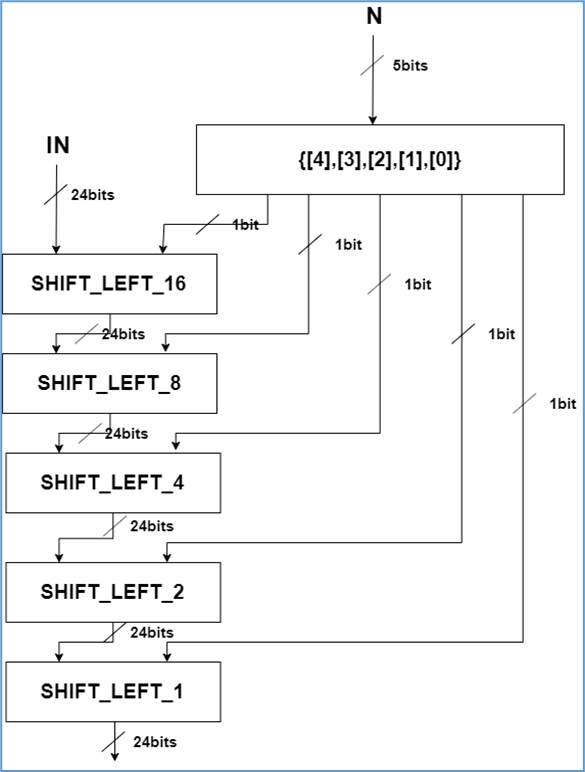
## Shift right:





## Shift left:

****



## Check special:

|  |  |  |
| --- | --- | --- |
| **Exponent** | **Fraction** | **Flag** |
| **00000000** | **0000000000000000000000000** | **FlagZero** |
| **11111111** | **1111111111111111111111111** | **FlagNaN** |
| **11111111** | **0000000000000000000000000** | **FlagInf** |

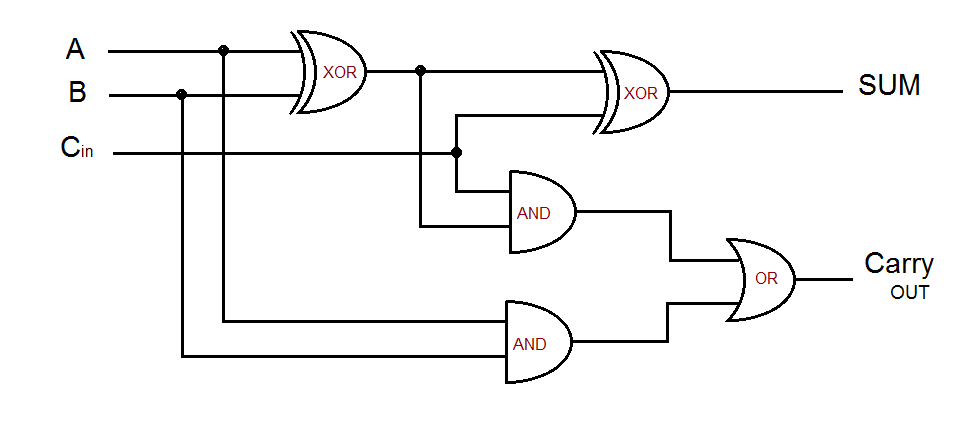
## Find1:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A24** | **A23** | **A22** | **A21** | **A20** | **A19** | **A18** |  | **S4** | **S3** | **S2** | **S1** | **S0** |
| **X** | **1** | **x** | **x** | **x** | **x** | **x** | **0** | **0** | **0** | **0** | **1** |
| **X** | **0** | **1** | **x** | **x** | **x** | **x** | **0** | **0** | **0** | **1** | **0** |
| **X** | **0** | **0** | **1** | **x** | **x** | **x** | **0** | **0** | **0** | **1** | **1** |
| **X** | **0** | **0** | **0** | **1** | **x** | **x** | **0** | **0** | **1** | **0** | **0** |
| **X** | **0** | **0** | **0** | **0** | **1** | **x** | **0** | **0** | **1** | **0** | **1** |
| **X** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **1** | **1** | **0** |
| **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** |

# Bộ cộng trừ

## Full adder: (Thuật toán 1)

### Full adder 1bit:



S = A^B^Cin

Cout = (a&b)|(a&cin)|(b&cin)

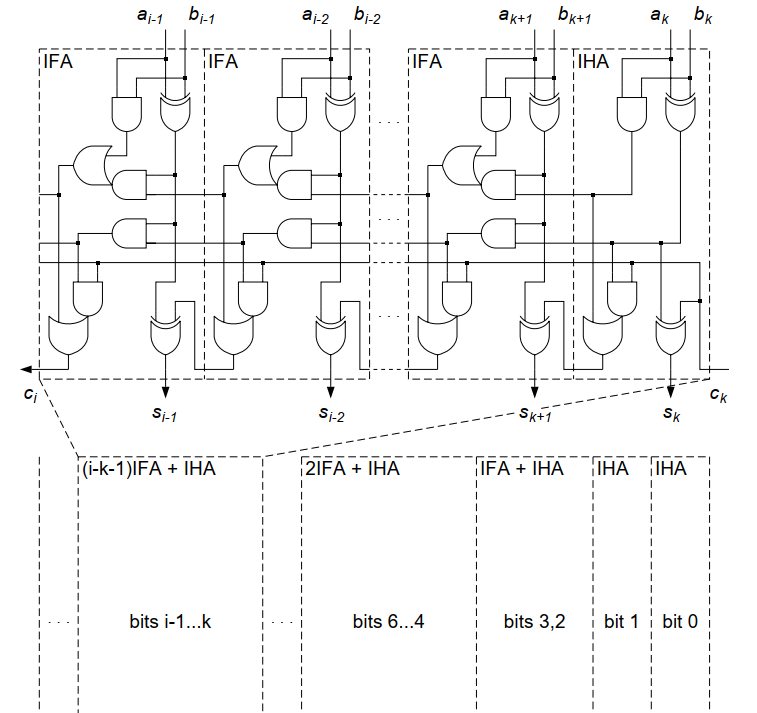
### Full adder n bit:

Gọi n lần khối full adder 1 bit, Carry out của khối trước là carry in của khối sau.

Ví dụ full adder 8 bit:

1. module adder\_8bit(in1, in2, S, Cout);
2. input [7:0]in1,in2;
3. output [7:0]S;
4. output Cout;
5. logic [7:1] temp\_c;
6. FA FA\_0(.a(in1[0]), .b(in2[0]), .cin(1'b0), .S(S[0]), .cout(temp\_c[1]));
7. FA FA\_1(.a(in1[1]), .b(in2[1]), .cin(temp\_c[1]), .S(S[1]), .cout(temp\_c[2]));
8. FA FA\_2(.a(in1[2]), .b(in2[2]), .cin(temp\_c[2]), .S(S[2]), .cout(temp\_c[3]));
9. FA FA\_3(.a(in1[3]), .b(in2[3]), .cin(temp\_c[3]), .S(S[3]), .cout(temp\_c[4]));
10. FA FA\_4(.a(in1[4]), .b(in2[4]), .cin(temp\_c[4]), .S(S[4]), .cout(temp\_c[5]));
11. FA FA\_5(.a(in1[5]), .b(in2[5]), .cin(temp\_c[5]), .S(S[5]), .cout(temp\_c[6]));
12. FA FA\_6(.a(in1[6]), .b(in2[6]), .cin(temp\_c[6]), .S(S[6]), .cout(temp\_c[7]));
13. FA FA\_7(.a(in1[7]), .b(in2[7]), .cin(temp\_c[7]), .S(S[7]), .cout(Cout));
14. Endmodule

## IHA, IFA adder: (Thuật toán 2)



Thay vì thực hiện bộ cộng bằng full adder, ta thực hiện bộ cộng bằng cách như trên giúp tiết kiệm thời gian thay vì phải đợi Cout của bộ full adđer trước để tính kết quả của các bit tiếp theo.

## Check sign of subtrahend:

3 input truyền vào bộ cộng là A, B và checkequation:

* A, B: là 2 số floating point 32 bit muốn tính toán, B là subtrahend.
* Checkequation: là bit dấu 0 là +, 1 là trừ.

Để mặc định là phép cộng ta thực hiện việc checksign đổi dấu B: B[31]^checkequation.

## Compare exponent:

So sánh 2 exponent với nhau, bằng cách trừ 2 exponent, expA – expB. Xét bit số 9 để biết kết quả, nếu expA > expB thì ekết quả là phép dương 🡪 bit số 9 =0 và ngược lại.

## Significand alignment:

Chọn ra 2 fraction: dịch và không dịch từ từ kết quả của khối compare exponent.

Dịch phải số fraction dịch nshift lần được tính từ khối compare exponent:

## Calculate significand:

Thực hiện cộng 2 fraction đã được alignment ở trên ta thu được fraction chưa được chuẩn hóa.

## Chuẩn hóa:

Tìm vị trí bit 1 đầu tiên trong 24 bit [23:0]

Dịch trái để cho ra dạng chuẩn 1,xxxxx hoặc dịch phải 1 bit nếu kết quả cho ra là 10,…..

Xuất kết quả ra màn hình, kiểm tra underflow, overflow, inf, NaN.

## Kết quả mô phỏng:

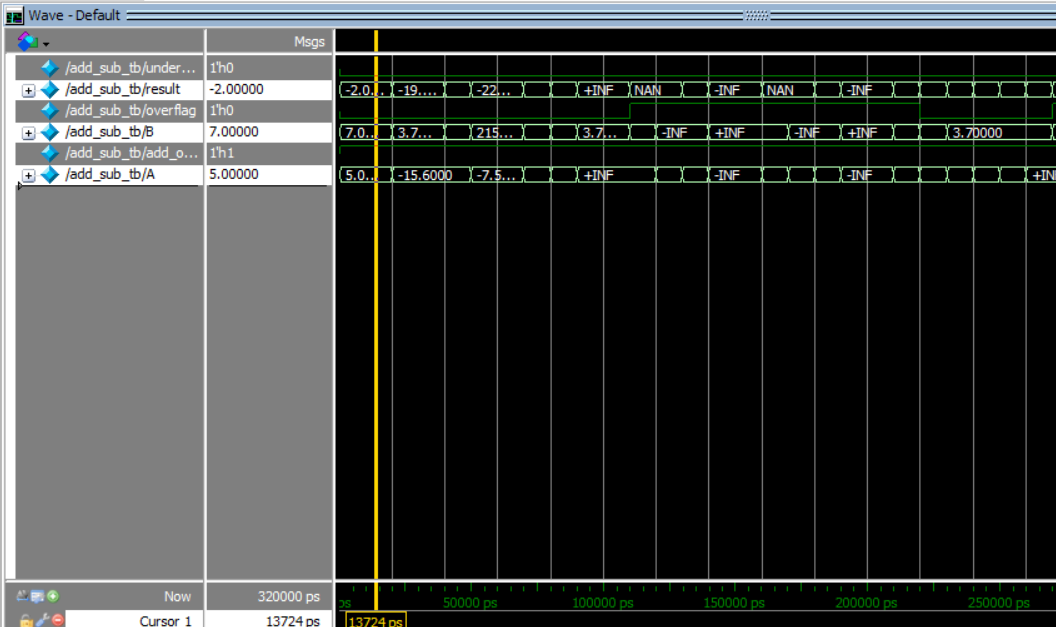
### Full adder:

Thực hiện phép trừ 2 số A và B



Kết quả cho ra chính xác.

### IHA, IFA:



Kết quả cho ra chính xác.

# Bộ nhân:

## Shift cộng: (giải thuật 1)

Thực hiện như phép nhân truyền thống, viết bộ nhân 24 bit x 24 bit để tính tích 2 fraction.

Vì thực hiện phép nhân như truyền thống, kết quả phép nhân có thể là 48 bit và phải dịch trái lần lượt các kết quả nên ta phải mở rộng ra thành phép nhân 2 số 48 bit A,B có dạng: {24’b0,A}.

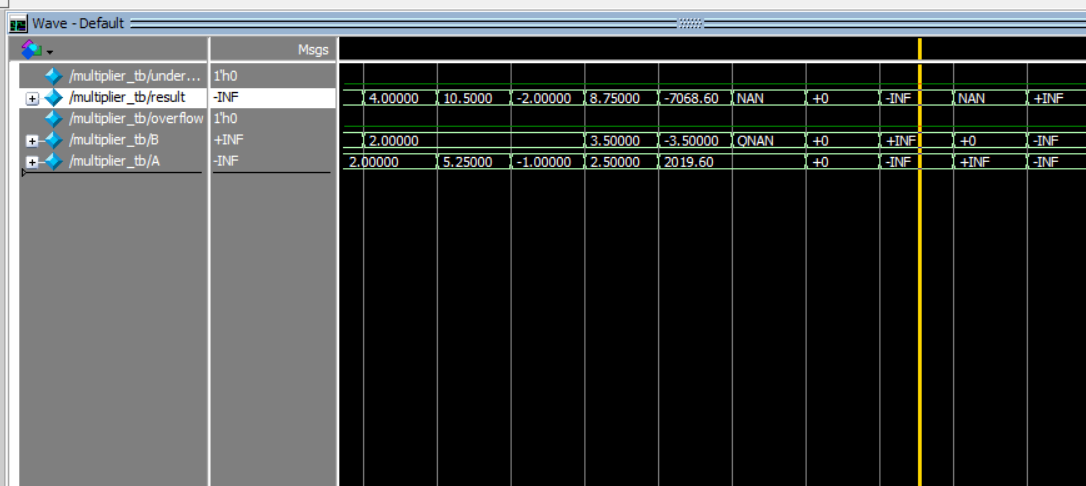
## Bootler: (giải thuật 2)



1. module unit\_mult24\_booth(A,M,Q,q0,A\_out,Q\_out,q0\_out);
2. input [24:0]A,M,Q;
3. input q0;
4. output [24:0]A\_out,Q\_out,q0\_out;
5. logic [24:0]temp, temp\_add, temp\_sub, M\_2;
6. logic [50:0]AQq0\_before\_shift,AQq0;
7. complement2\_25bit M\_complement2(.in(M),.out(M\_2));
8. adder\_25bit case1(.in1(A),.in2(M\_2),.S(temp\_sub),.Cout());
9. //FS\_24 case1(.a(A),.b(M),.cin(1'b0),.out(temp\_sub),.cout());
10. adder\_25bit case2(.in1(A),.in2(M),.S(temp\_add),.Cout());
11. always @(\*)
12. begin
13. case ({Q[0],q0})
14. 2'b10: temp = temp\_sub;
15. 2'b01: temp = temp\_add;
16. default: temp = A;
17. endcase
18. end
19. assign AQq0\_before\_shift = {temp,Q,q0};
20. //shift\_right49 shift1(AQq0\_before\_shift,6'b1,AQq0);
21. assign AQq0 = {AQq0\_before\_shift[50],AQq0\_before\_shift[50:1]};
22. assign A\_out = AQq0[50:26];
23. assign Q\_out = AQq0[25:1];
24. assign q0\_out = AQq0[0];
25. Endmodule

## Kết quả mô phỏng:

### Shift cộng:



### Booth recording:



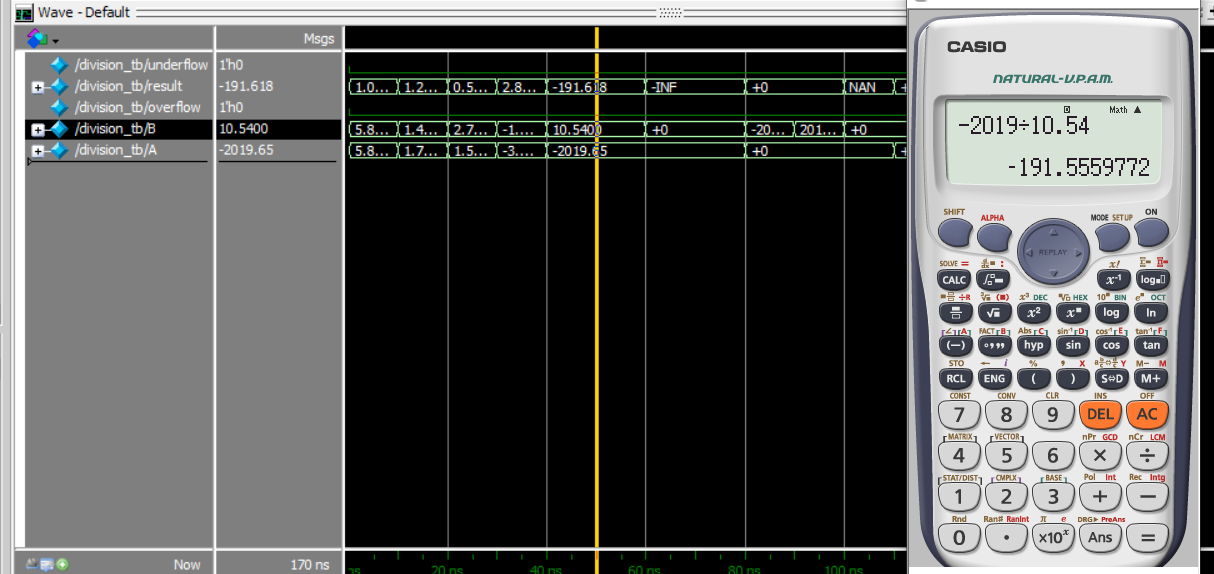
# Bộ chia:

## Shift trừ:

1. if D = 0 then error(DivisionByZeroException) end
2. Q := 0 -- Initialize quotient and remainder to zero
3. R := 0
4. for i := n − 1 .. 0 do -- Where n is number of bits in N
5. R := R << 1 -- Left-shift R by 1 bit
6. R(0) := N(i) -- Set the least-significant bit of R equal to bit i of the numerator
7. if R ≥ D then
8. R := R − D
9. Q(i) := 1
10. end
11. end

Sử dụng Shift trừ để tính toán chia 2 significand 24 bit (1 +23 bit fraction)

## Kết quả mô phỏng:



# Căn bậc n:

## Giải thuật

Tính toán với là 2 số floating point.

Sử dụng khai triển taylor để tính toán:

// base ^ exp = e ^ (exp \* ln(base))

// = e ^ (exp \* ln( (2^ (base\_exp\_real)) \* base\_frac) )

// = e ^ (exp \* (base\_exp\_real\*ln(2) + ln(base\_frac))

Base\_frac nằm trong đoạn [1;2] nên middle point là 1.5. Khi khai triển taylor thì ta dùng công thức 0<x<2a nên 2a lúc này là 3. Khoảng cách lớn dẫn đến sai số nhiều do đó ta scale lại thành 0<x<0.75.

// = e ^ (exp \* (base\_exp\_real\*ln(2) + ln(base\_frac/4) + 2\*ln(2)))

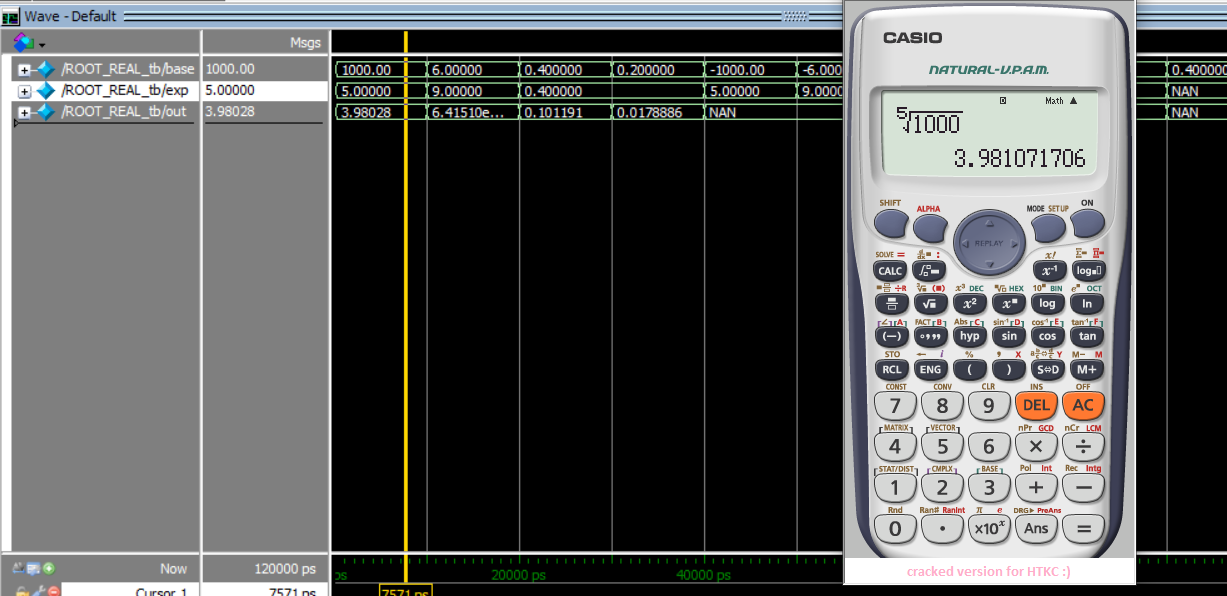
// = e ^ (exp \* ((base\_exp - 127)\*ln(2) + ln(base\_frac/4) + 2\*ln(2)))

// = e ^ (exp \* (base\_exp\*ln(2) + ln(base\_frac/4) - 125\*ln(2)))

Sử dụng khai triển taylor để tính ln(base\_frac/4)

Sử dụng khai triển taylor để tính kết quả cuối cùng:

## Kết quả mô phỏng:



Kết quả cho ra khá chính xác, có xuất hiện sai số do các phép tính trong bộ ALU không chính xác và việc ta tính xấp xỉ bằng chuỗi taylor cũng dẫn đến sai số.

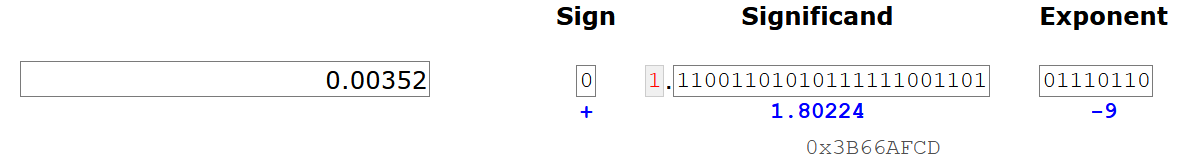
# Convert binary to decimal:

## Giải thuật:

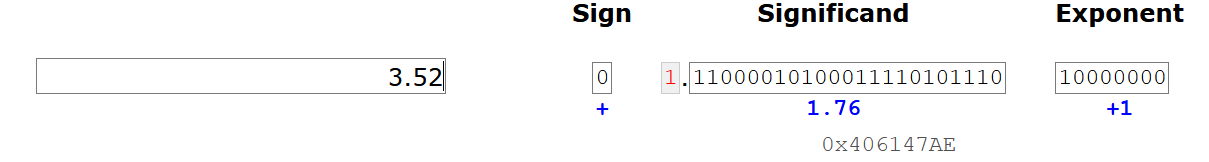


Giả sử với trường hợp exponent <127

Ví dụ số

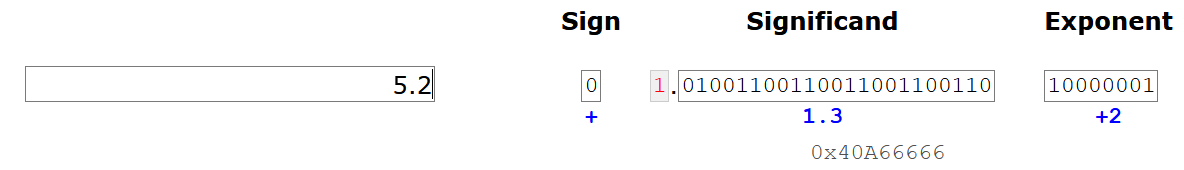


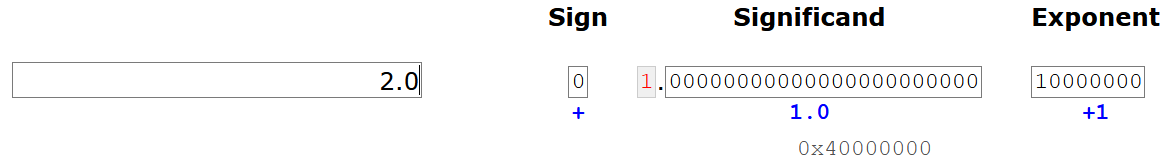
Ta thấy exponent < 127 nên ta lấy số đó nhân lên 10 cho đến khi exponent >=127 🡪 Số lần nhân với 10 chính là bù 2 của số mũ (3) 🡪 số mũ là -3



Dịch phải 24 bit significand (bit 1 + 23 bit fraction) 22 bit (23 – (exponent – 127)) ta thu được phần nguyên là 3.

Lấy 3.52 – 3 thu được 0.52. Thực hiện việc như tính toán để lấy phần nguyên khoảng 5 lần để thu được 5 chữ số thập phân sau dấu ,





## Kết quả mô phỏng:

