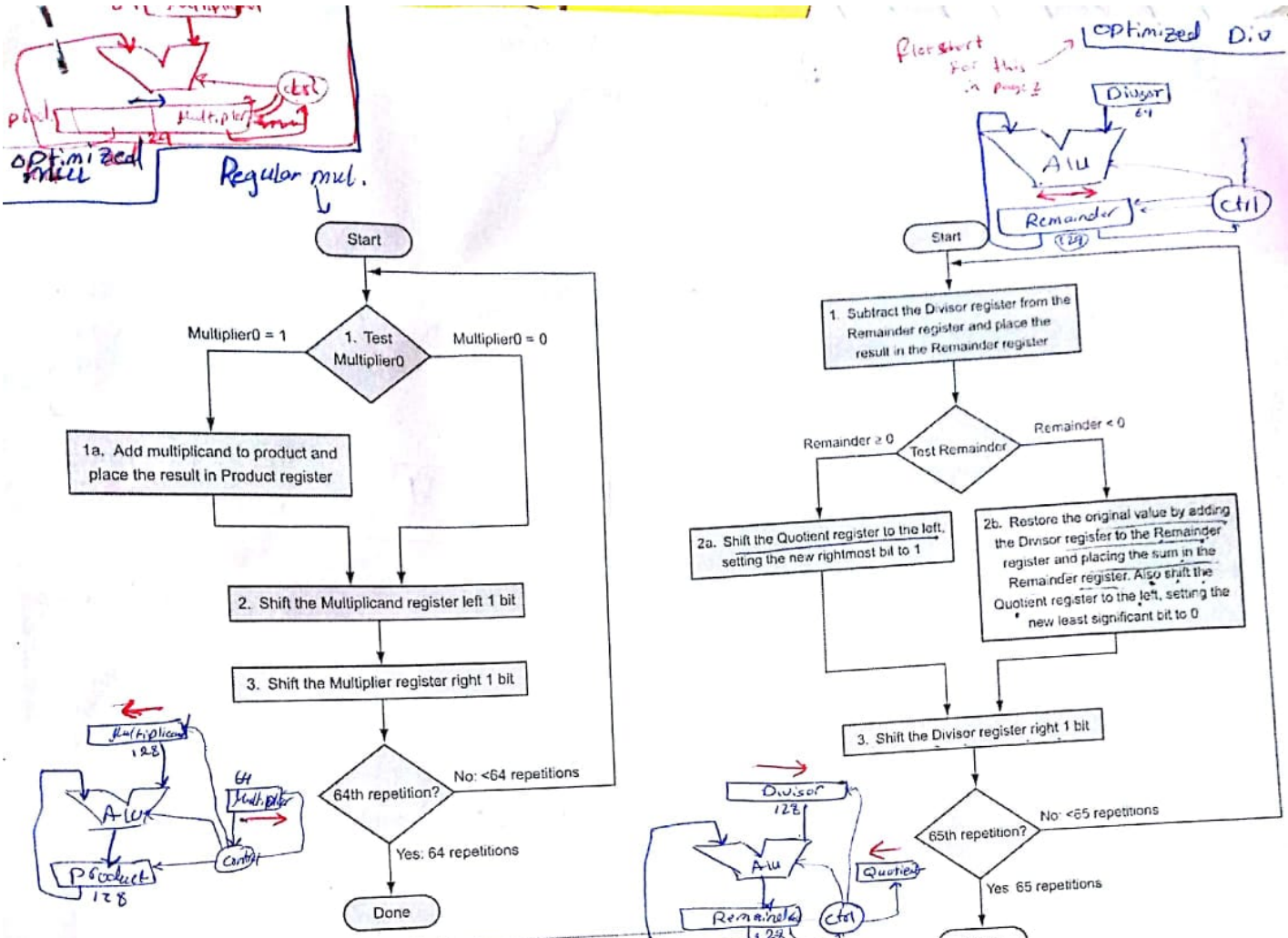


The first multiplication algorithm, using the hardware shown in Figure 3.3

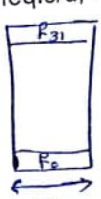
FIGURE 3.9 A division algorithm, using the hardware in Figure 3.8

Dividend = (Divisor x Quotient) + Remainder.

- OV = $Cout_n$ XOR $Cout_{n-1}$
- Real-Sign = OV XOR Sign-bit
- Remainder always have the same sign as the Dividend
- Approach 2 for signed multiplication:
  - Sign-Extend the partial products
  - Subtract the last partial product when multiplier is negative (last bit (MSB)) = 1
- Single precision accuracy: 6 decimal
- Double precision accuracy: 16 decimal
- Double precision accuracy: 16 decimal
- Sticky bit is used when both guard and round bits are exactly in the middle in binary 10
- Floating-point Instructions:
  - fadd.s/d, fsub.s/d, fmul.s/d, fdiv.s/d, fsqrt.s/d (rd, fs1, fs2)
  - flw, fsw, fld, fsd (f_D, offset(x_D)) Base address
  - feq.s/d, flt.s/d, fle.s/d (1 true, 0 false. Stored in integer register)

Approach 1: convert to positive num (63) iteration

Floating point Mul.
1) Add exponent
2) Multiply significant
3) Normalize.
4) Round
5) determine sign of result.



offset in 2dim array = (i x 32) + j) x 8

Const in FP should be loaded from memo

offset ... const

flw $f_{\square}$, const$_{\square}$(x_3)

global Pointer (x_3)

return → jalr x_0, 0(x_1)

feq.s x_5, f_4, f_6

beq x_0, x_0, Exit.

Array Mul. (eliminate nested array)

memAddress = offset + Base Address

slli $x_{\square}$, $x_{\square}$, 5

add $x_{\square}$, $x_{\square}$, $x_{\square}$

slli $x_{\square}$, $x_{\square}$, 3

add $x_{\square}$, $x_{\square}$, $x_{\square}$ Base Address

fld f_0, 0($x_{\square}$)

mul, add, fadd, blt, ...

fsd f_0, 0($x_{\square}$)

4p

Significand

1.00 ... 0 ... 1.111 ... 1 = 2

Relative precision

$n = \lfloor \log_2 M \rfloor$, $M = 2^{23} = 2$

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$ Single Precision: Bias 127; min: -126, max: 127. Double Precision: Bias 1023; min: -1022, max: 1023

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

S	Exponent	Fraction (Mantissa)

$c1 = g0 + (p0 \cdot c0)$
$c2 = g1 + (p1 \cdot g0) + (p1 \cdot p0 \cdot c0)$
$c3 = g2 + (p2 \cdot g1) + (p2 \cdot p1 \cdot g0) + (p2 \cdot p1 \cdot p0 \cdot c0)$
$c4 = g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) + (p3 \cdot p2 \cdot p1 \cdot p0 \cdot c0)$

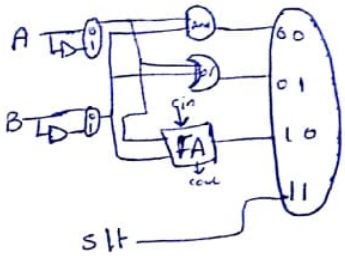
$C1 = G0 + (P0 \cdot c0)$
$C2 = G1 + (P1 \cdot G0) + (P1 \cdot P0 \cdot c0)$
$C3 = G2 + (P2 \cdot G1) + (P2 \cdot P1 \cdot G0) + (P2 \cdot P1 \cdot P0 \cdot c0)$
$C4 = G3 + (P3 \cdot G2) + (P3 \cdot P2 \cdot G1) + (P3 \cdot P2 \cdot P1 \cdot G0) + (P3 \cdot P2 \cdot P1 \cdot P0 \cdot c0)$

Iteration	Step	Multiplicand	Product
0	Initialization	0010	0 0000 0011
1	Prod (MS half) = Prod (MS half)+ Multiplicand *Then* SRL Product	0010	0 0010 0011 0 0001 0001
2	Prod (MS half) = Prod (MS half)+ Multiplicand *Then* SRL Product	0010	0 0011 0001 0 0001 1000
3	SRL Product	0010	0 0000 1100
4	SRL Product	0010	0 0000 0110

mul rd,rs1,rs2 → Gives lower 64 bit of product
mulh, mulhu → " upper 64 " " "
mulhsu → " upper 64 bit " "
Signed
unsigned
Zero-extend

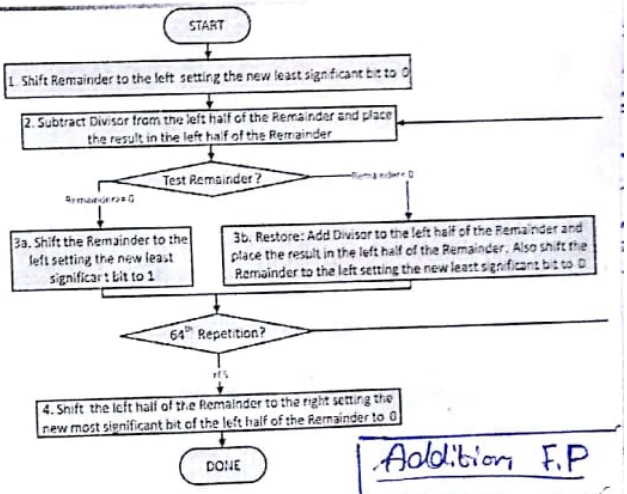
RISC-V Multiplication

ALU Block



$gi = ai \cdot bi$
$pi = ai + bi$
$c_{i+1} = gi + pi \cdot ci$

$P0 = p3 \cdot p2 \cdot p1 \cdot p0$
$P1 = p7 \cdot p6 \cdot p5 \cdot p4$
$P2 = p11 \cdot p10 \cdot p9 \cdot p8$
$P3 = p15 \cdot p14 \cdot p13 \cdot p12$
$G0 = g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0)$
$G1 = g7 + (p7 \cdot g6) + (p7 \cdot p6 \cdot g5) + (p7 \cdot p6 \cdot p5 \cdot g4)$
$G2 = g11 + (p11 \cdot g10) + (p11 \cdot p10 \cdot g9) + (p11 \cdot p10 \cdot p9 \cdot g8)$
$G3 = g15 + (p15 \cdot g14) + (p15 \cdot p14 \cdot g13) + (p15 \cdot p14 \cdot p13 \cdot g12)$



div / rem rd, rs1, rs2.
divu / remu

RISC-V Division

DivideBy Zero:
ex: beq x4, x0, Exit
div x6, x5, x4

OverflowProp in Div:
$\frac{-2^{63}}{-1}$ → overflow

ALU control lines	Function
0000	AND
0001	OR
0010	add → +(load/store)
0110	subtract → -(branch)

Instruction	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

opcode/Main

Name (Field Size)	Field					
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type load	immediate[11:0]		rs1	funct3	rd	opcode
S-type Store	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode

ALU

Addition F.P

1. Align
2. Add significand
3. Normalize
4. Rounding

And Renormlize