HUMBOLDT-UNIVERSITÄT ZU BERLIN
INSTITUT FÜR INFORMATIK

Lecture 11

ALU (2) - Integer Arithmetic

Sommersemester 2001

Leitung: Prof. Dr. Miroslaw Malek

ALU
ARITHMETIC-LOGIC UNIT

Arithmetic Units Classification

Number representations

Hardware/software continuum
and vertical migration

Integer Arithmetic
- **Addition/subtraction**
- **Multiplication**
  - **Booth's Algorithm**
  - **Bit Pair Algorithm**
- **Division**

Floating-point arithmetic
- Addition/subtraction
- Multiplication/division

Logic functions

ADDITION/SUBTRACTION OF POSITIVE AND NEGATIVE NUMBERS

- Sign & Magnitude (awkward +,-)

- 1's complement (better).
  Requires a correct cycle $S_{t+1} = S_t + C_n$

- 2's complement is awkward externally, but is the best hardware representation for addition and subtraction

a) 2's Complement Addition

(1) Modulo Arithmetic

Positive Numbers 7 + 4 = 11 no change

Negative Numbers -7 + (-4) = -11

2's complement (9 + 12) = 21
which is -11 in signed decimal notation

In general $(X + Y) \bmod 2^n$

2's complement (9+12) mod 16 = 5
which is also 11 in decimal

Add 2's complement using adder as in the figure. Ignore carry-out bit and you have a correct answer.

$$-2^{n-1} \leq N < +\left(2^{n-1}\right)$$

(2) Numbers outside the expected range cause an arithmetic overflow.

b) Subtraction

Subtraction is the same as performing addition of a complement.

7-4=3 is equivalent 7+12=19

in 2's complement which is 3 after ignoring the carry.

OVERFLOW

$c_n = 1,\ X + Y \geq 2^n$

Overflow is only possible when both operands have the same sign.

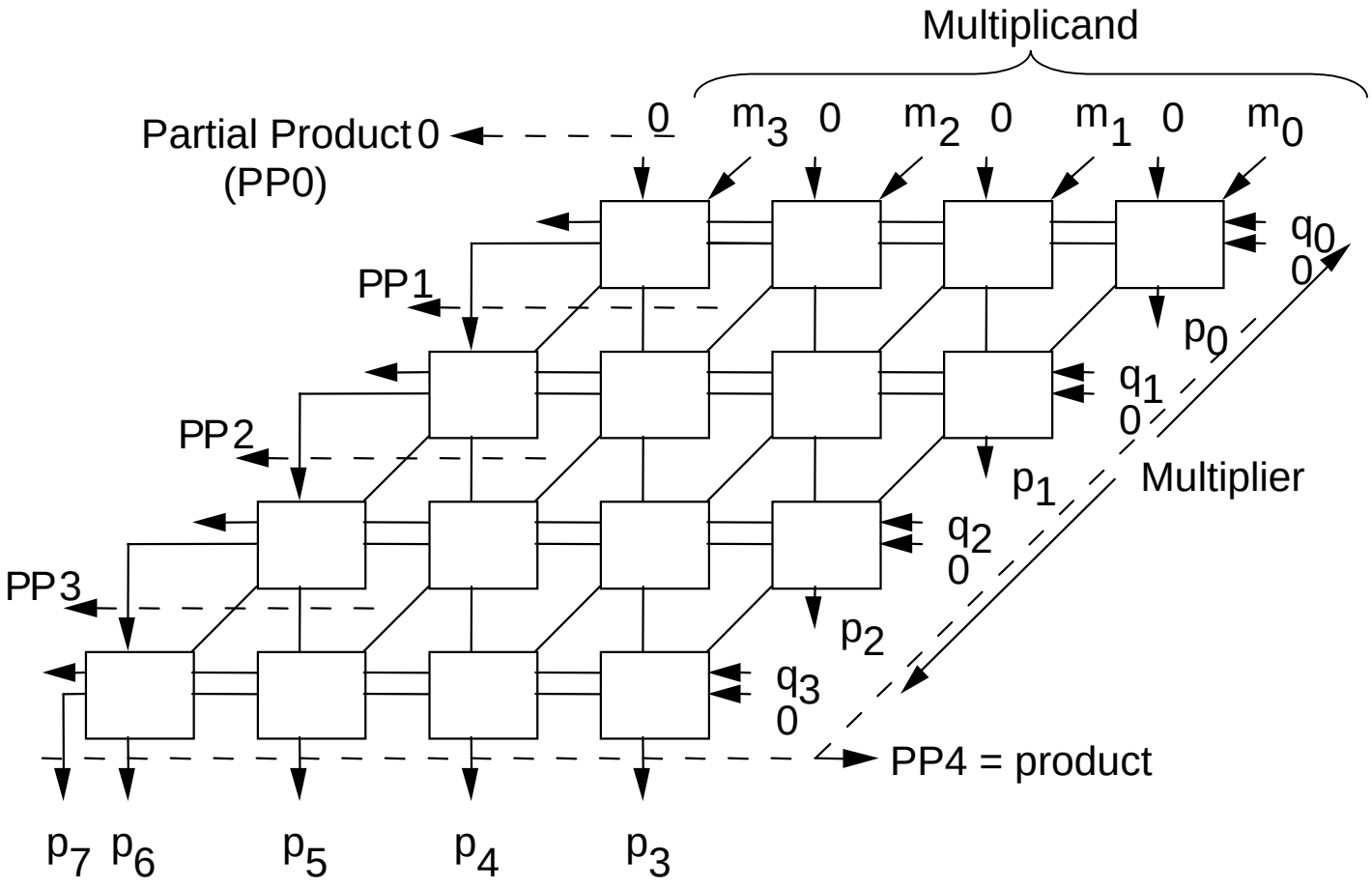
Overflow occurs when the sign of S does not agree with the signs of X and Y, i.e., when the signs of X and Y are the same.

$$Overflow = x_{n-1} y_{n-1} \bar{s}_{n-1} + \overline{\overline{x}_{n-1}} \overline{y}_{n-1} s_{n-1}$$

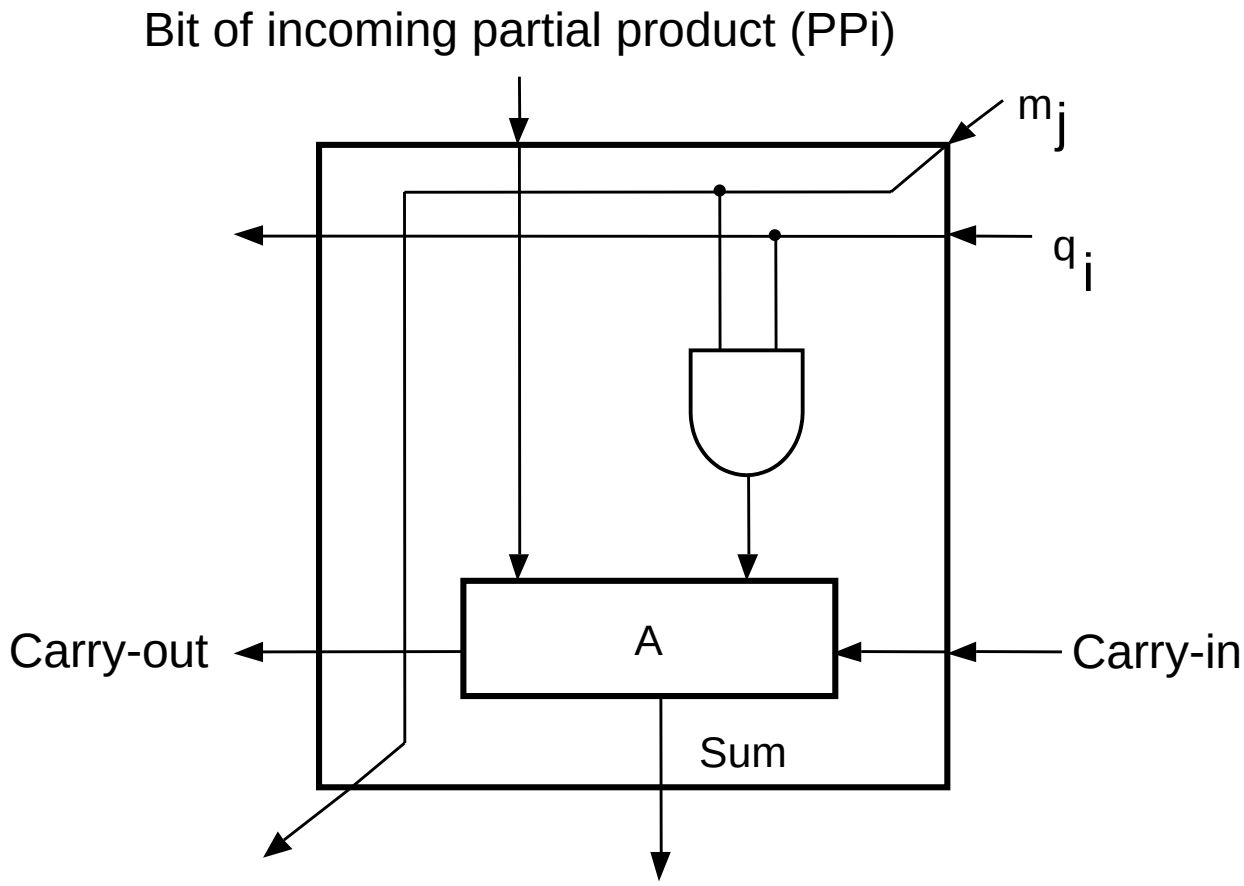
Detection method is to set a flag (V). Interrupts are sometimes allowed to occur on arithmetic overflow.

MULTIPLICATION
"PAPER AND PENCIL METHOD"

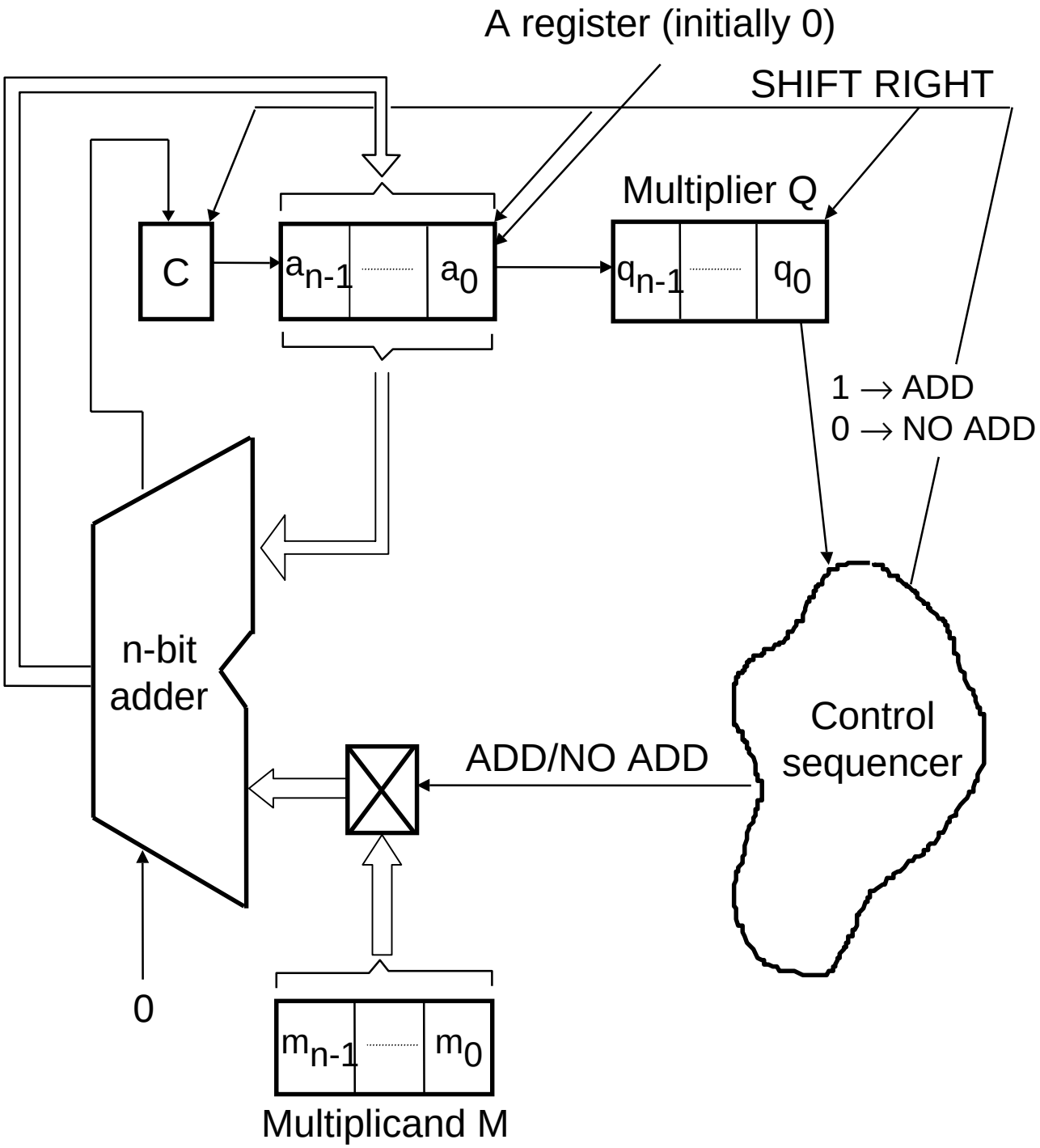
```
      1 1 0 1   (13) Multiplicand M
    x 1 0 1 1   (11) Multiplier Q
    ---------
      1 1 0 1
    1 1 0 1
  0 0 0 0
1 1 0 1
-----------------
1 0 0 0 1 1 1 1   (143) Product P
```



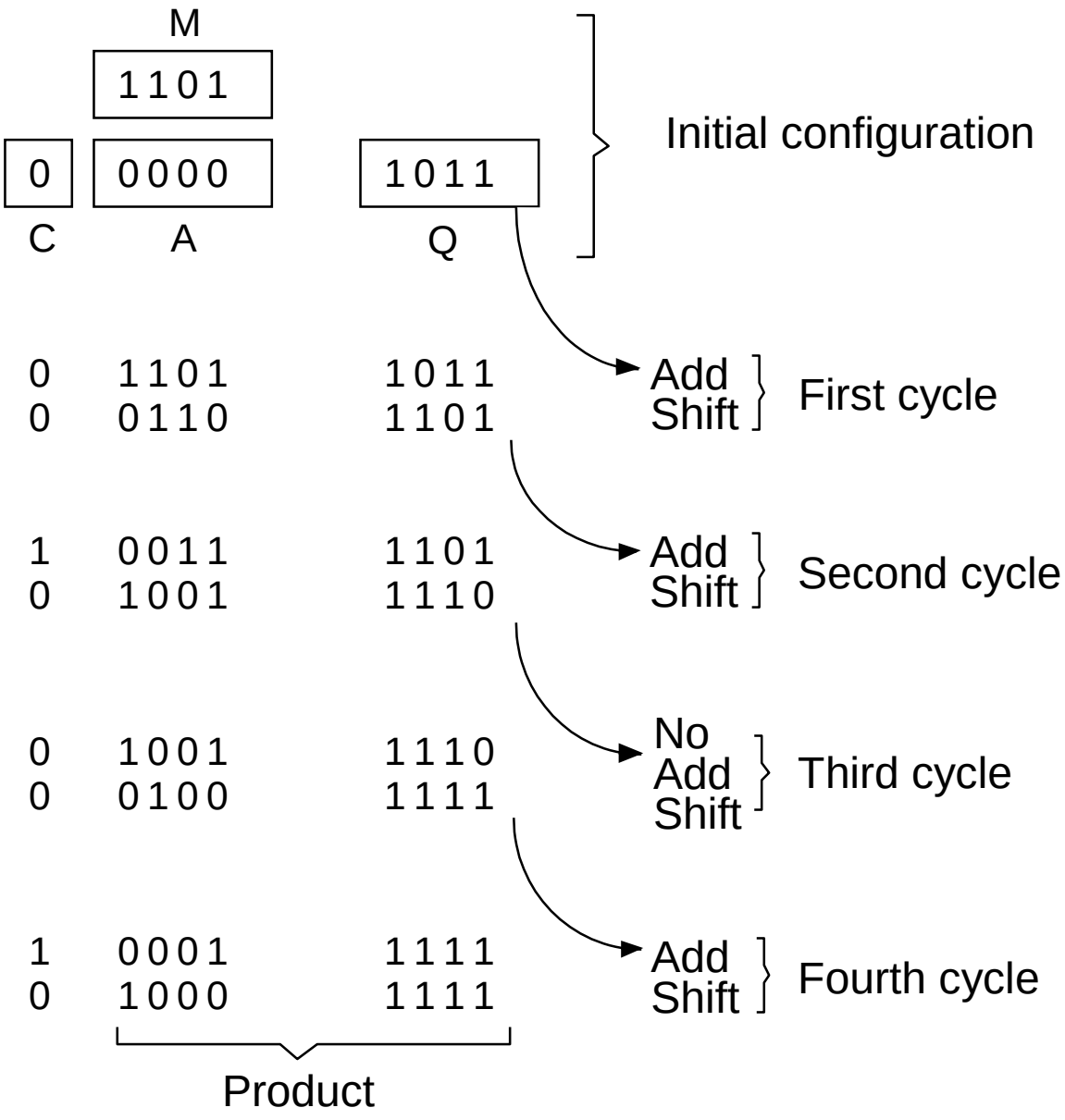
TYPICAL CELL OF MULTIPLICATION ARRAY IMPLEMENTATION



REGISTER CONFIGURATION FOR SEQUENTIAL CIRCUIT BINARY MULTIPLIER

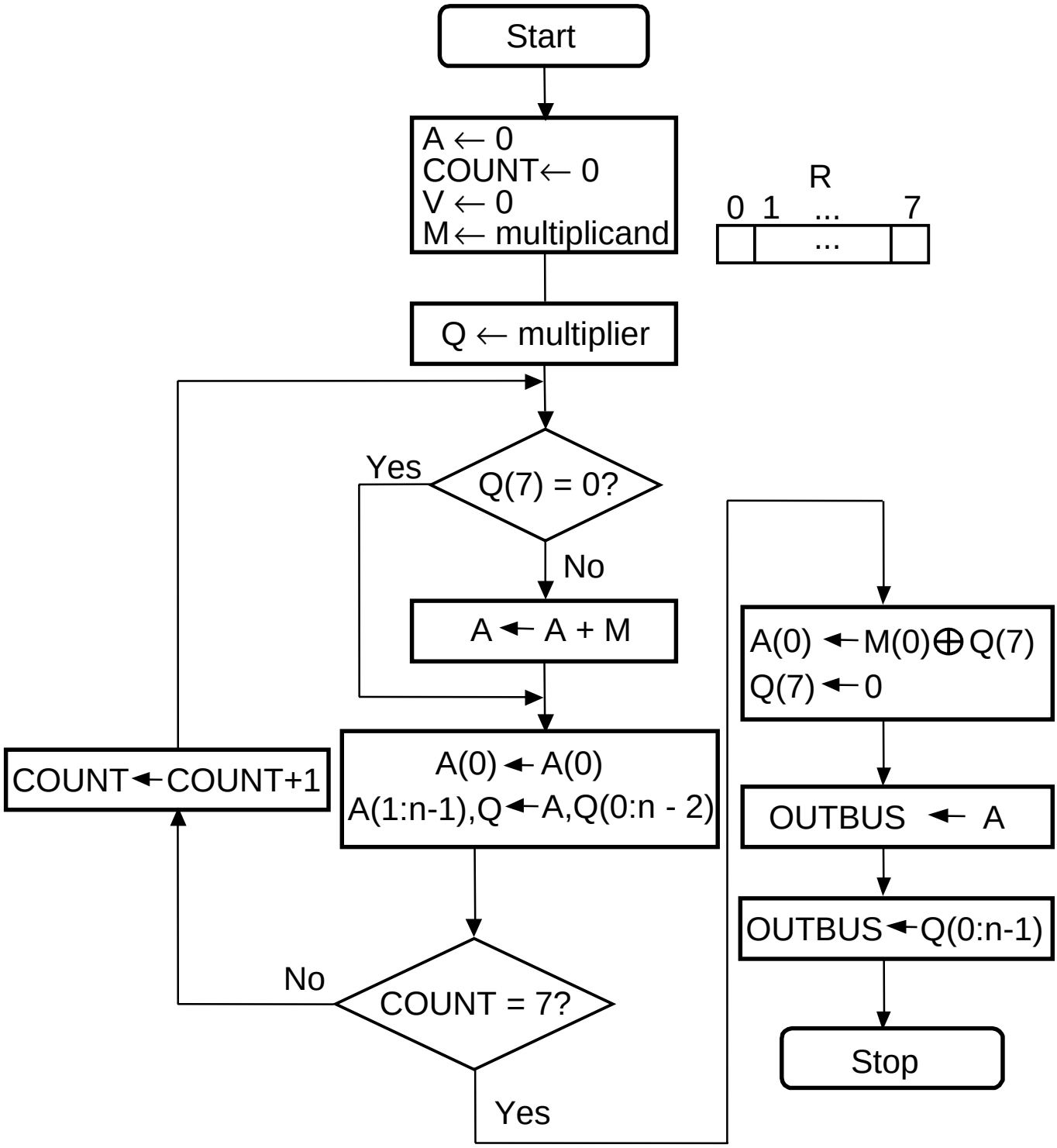


MULTIPLICATION EXAMPLE

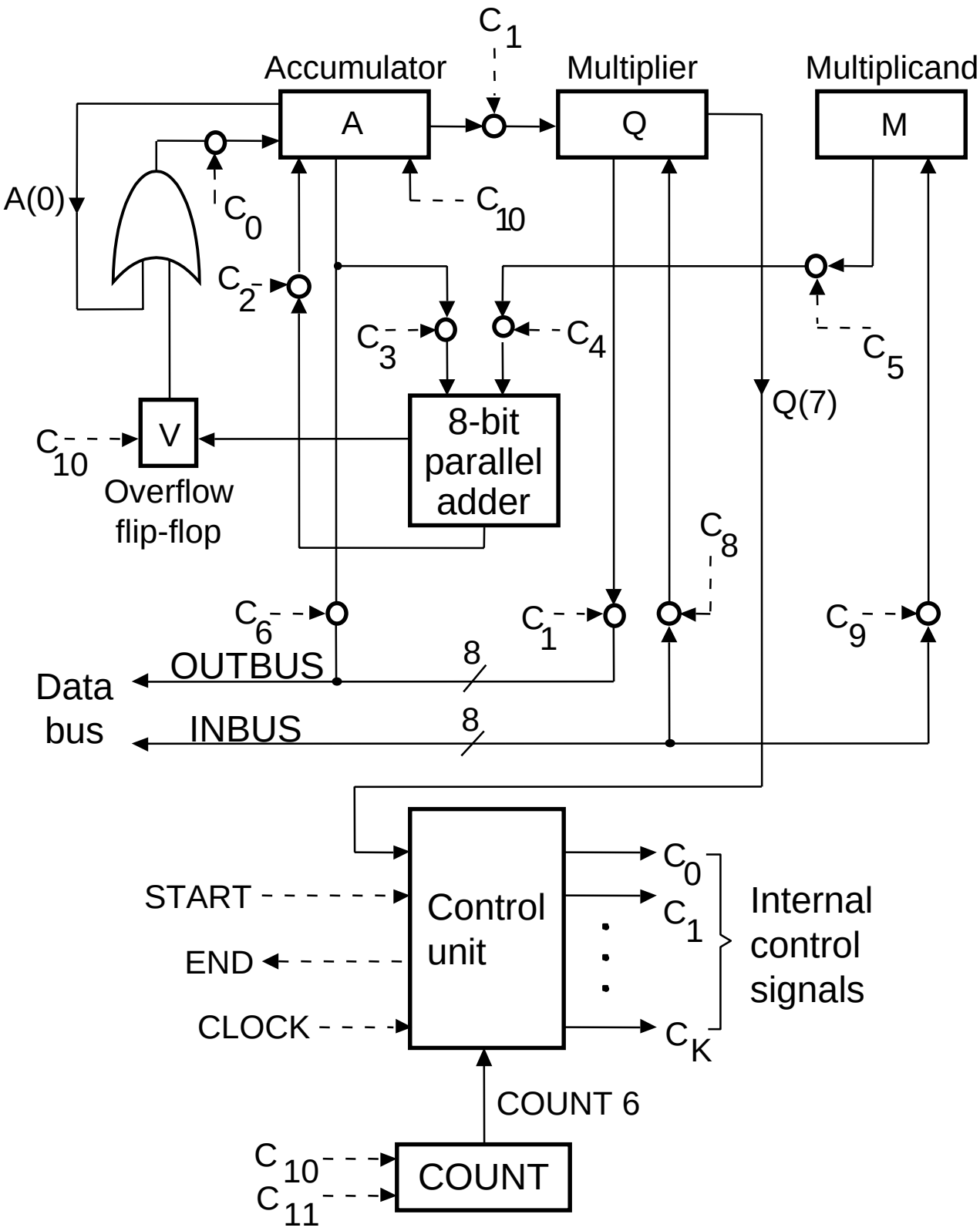


PP_0 is in A after the first ADD

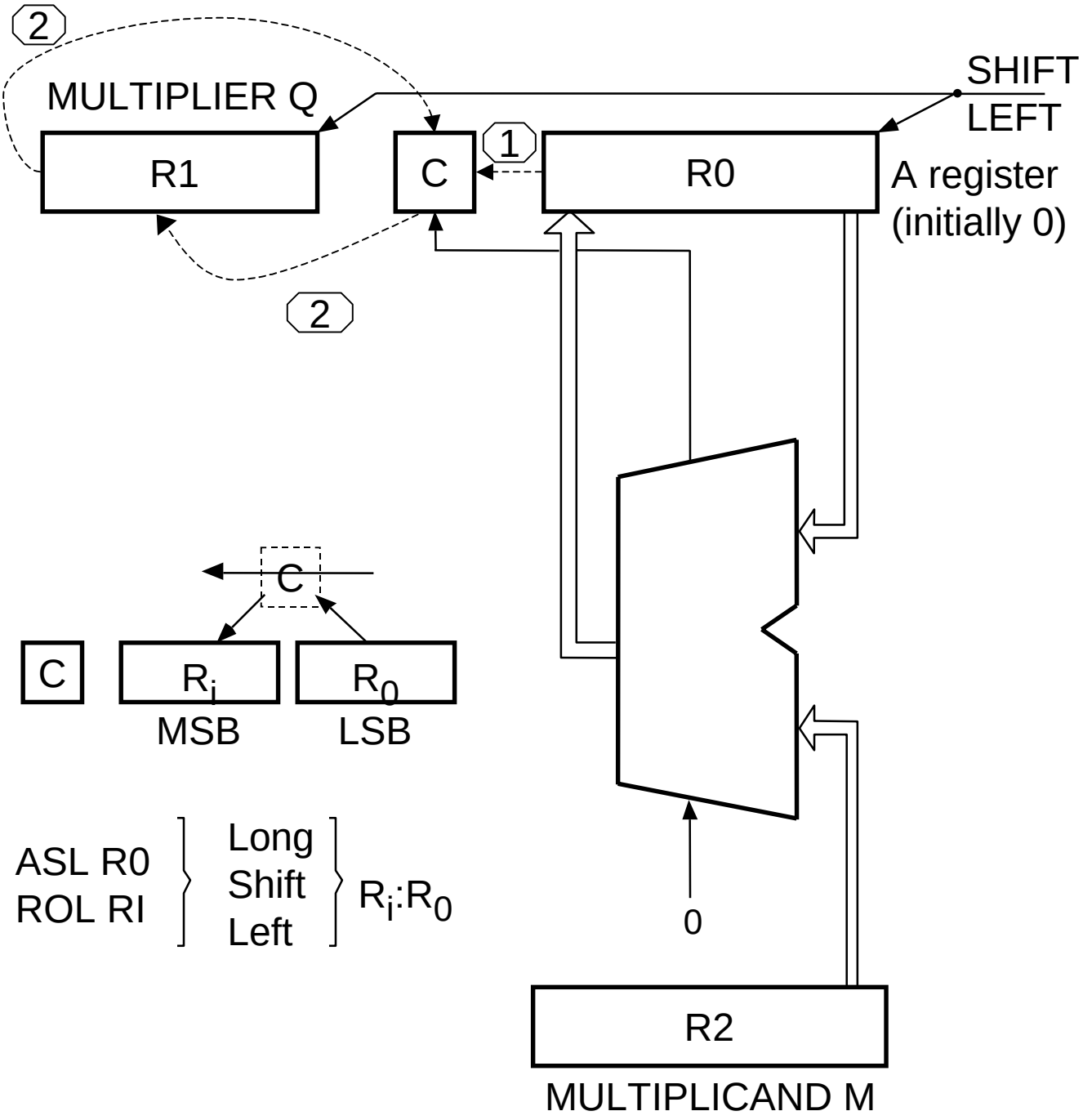
FLOWCHART FOR 2'S-COMPLEMENT MULTIPLICATION



BLOCK DIAGRAM OF 2'S-COMPLEMENT FIXED-POINT MULTIPLIER



PDP-11 REGISTER ARRANGEMENT FOR SOFTWARE MULTIPLY



SOFTWARE MULTIPLICATION

CLR	**R0**	Clear register R0
MOV	**#-16.,R3**	Set register R3 to -16 → use as cycle counter
mloop:		
ASL	**R0**	ASL (Arithmetic Shift Left) instruction shifts contents
ROL	**R1**	of R0 (low-order half of partial product) left one bit. MSB of R0 is shifted into C bit. LSB of R0 is set to 0
		ROL (Rotate Left) instruction rotates contents of R1 and the C bit by one bit position. Old contents of C is moved into LSB of R1, old contents of MSB of R1 is moved into C.
		The combination of ASL and ROL have thus shifted the (R1, R0) pair left one bit position, placing the old MSB of R1 into C.
BCC	**noadd**	The multiplier bit to be checked (first q_{n-1}, then q_{n-}, etc.) is now in C, and the instruction BCC causes a branch around the add operation if C is 0.
ADD	**R2,R0**	If C=1, multiplicand in R2 is added into the low-order
ADC	**R1**	end of the partial product in R0. Any carry-out that goes into C must be added to the high-order half of the partial product. This is done by the ADC which adds contents of C to the LSB position of R1. Note that this process will never damage the unused portion of the multiplier in R1 because the very first time that (R1, R0) is shifted left, a = is placed in the LSB position of R1, creating enough space for any later encroachment by the partial product.
noadd:		
INC	**R3**	Counter is incremented and BNE (Branch if Not
BNE	**mloop**	Equal to 0) instruction does branch back to *mloop*
HALT		as long 16 cycles are not completed.

Labels

SIGNED OPERAND MULTIPLICATION

1. Positive Multiplier, Negative Multiplicand(M)
   - Extend the sign bit 1..."1" to the left just as we assume we extend the "0" sign bit. Proceed to execute the multiply algorithm as for a positive number.
2. Negative Multiplier, Positive Multiplicand(M)
   - Complement both the Multiplier and Multiplicand and proceed to multiply with sign extension.
3. Negative Multiplier, Negative Multiplicand (M)
   - Complement both the multiplier and multiplicand and proceed to multiply. Note: this is the same procedure as when the multiplier is detected to be negative.

BOOTH´S ALGORITHM

- 1. Booth's algorithm is a powerful direct algorithm to perform signed-number multiplication. The algorithm is based on the fact that any binary number can be represented by the sum and difference of other binary numbers. Using a signed binary notation we can represent a multiplier in a unique scheme with the possibility of fewer add cycles for a given multiplier.
- 2.a. Examples of the scheme are shown (a positive Multiplier and a negative Multiplier). The first example shows how zero's may be skipped over for faster implementation.
- b. Figure illustrates how the Booth recordings are accomplished and next figure illustrates the transition recording table.
- c. Also a flow chart and a table show, respectively, the use of the algorithm for multiplication of 2's complement numbers.
- d. The speed of the algorithm depends upon the bit savings if any that the Booth algorithm will generate. Figure illustrates the worst, normal and good case for a 16 bit number.
- e. The algorithm accomplishes the following
  - Uniform treatment of positive and negative numbers. Achieves efficiency in the number of summands in some cases (data dependent).
- 3. It would be desirable to use the Booth technique in some way that removed some of the data dependence.

BOOTH'S REPRESENTATION (1)

- Any binary number can be represented by the sum or difference of other binary numbers.
- For example, 30_{10} (0011110_2) can be represented by 32_{10} (0100000_2) minus 2_{10} (0000010_2). See the example below.

	0	1	0	0	0	0	0	(32)
-	0	0	0	0	0	1	0	(2)
	0	0	1	1	1	1	0	(30)
	0	**+1**	**0**	**0**	**0**	**-1**	**0**	**booth**

- Using a signed binary notation 30_{10} can be represented as (0 + 1 0 0 0 - 1 0).
- Note that in scheme transitions determine multipliers.
  - 0 <---- 1 *(1) Multiplier
  - 1 <---- 0 *(-1) Multiplier
  - Repetitions *(0) Multiplier
- Scan Right to Left

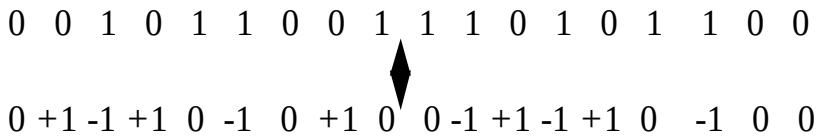
					1	0	0	1	1			(-13)			
				x	0	1	0	1	1			(+11)			
1	1	1	1	1	1	0	0	1	1						
1	1	1	1	1	0	0	1	1							
0	0	0	0	0	0	0	0								
1	1	1	0	0	1	1									
0	0	0	0	0	0										
1	1	0	1	1	1	0	0	0	1		(-143)				

Sign extension of negative multiplicand

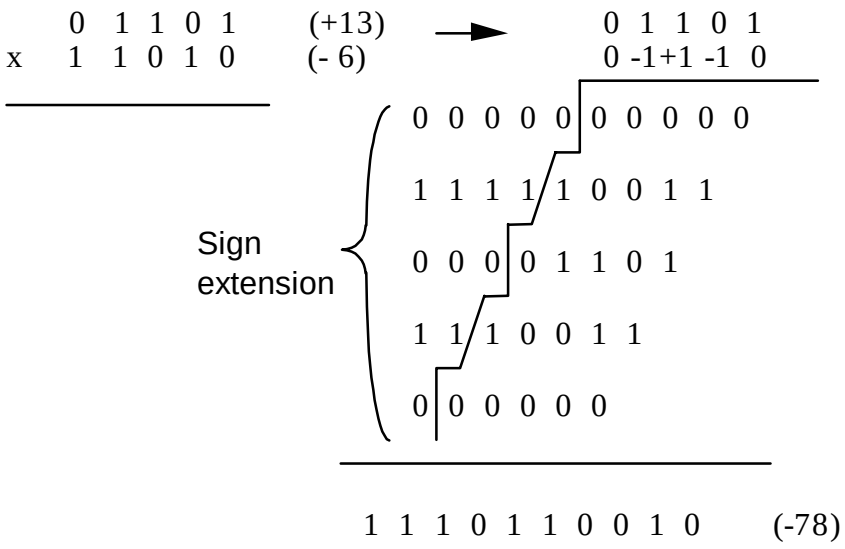
							0	1	0	1	1	0	1	(+45)	
							0	0	+1	+1	+1	+1	0	(+30)	
							0	0	0	0	0	0	0		
						0	1	0	1	1	0	1			
					0	1	0	1	1	0	1				
				0	1	0	1	1	0	1					
			0	1	0	1	1	0	1						
		0	0	0	0	0	0	0							
	0	0	0	0	0	0	0								
0	0	0	1	0	1	0	1	0	0	0	1	1	0	(+1350)	

							0	1	0	1	1	0	1		
							0	+1	0	0	0	-1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0		
1	1	1	1	1	1	1	0	1	0	0	1	1	(2's compl.		
0	.....												0	of multiplicand)	
0	.....												0		
0	.....												0		
0	0	0	1	0	1	1	0	1							
0	0	0	0	0	0	0	0								
0	0	0	1	0	1	0	1	0	0	0	1	1	0		

Normal and Booth multiplication schemes.



Booth recoding of multiplier.



Booth multiplication with a negative multiplier

Multiplier Bit i	Bit i-1	Version of multiplicand selected by bit i
0	0	0 x M
0	1	+1 x M
1	0	-1 x M
1	1	0 x M

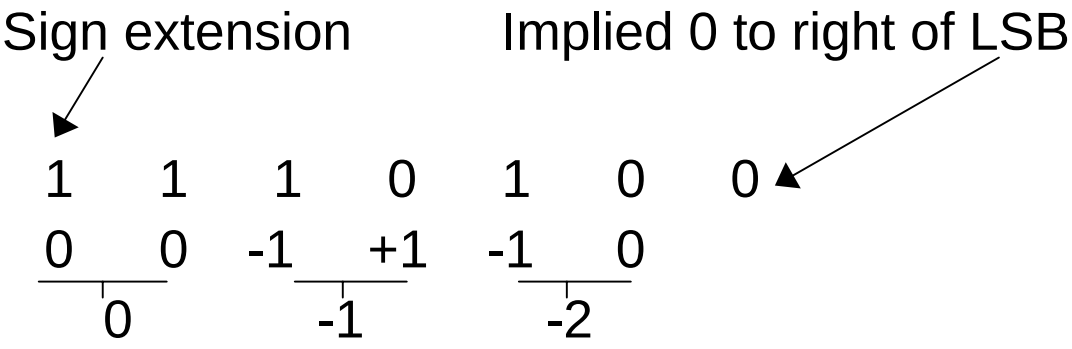
Booth multiplier recording table.

BIT-PAIR

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ⇒
1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 0 ⇒
0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 ⇒

+1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 bad
0 -1 0 0 +1 -1 +1 0 -1 +1 0 0 0 -1 0 0 normal
0 0 0 +1 0 0 0 0 -1 0 0 0 +1 0 0 -1 good

- Booth-recorded multipliers

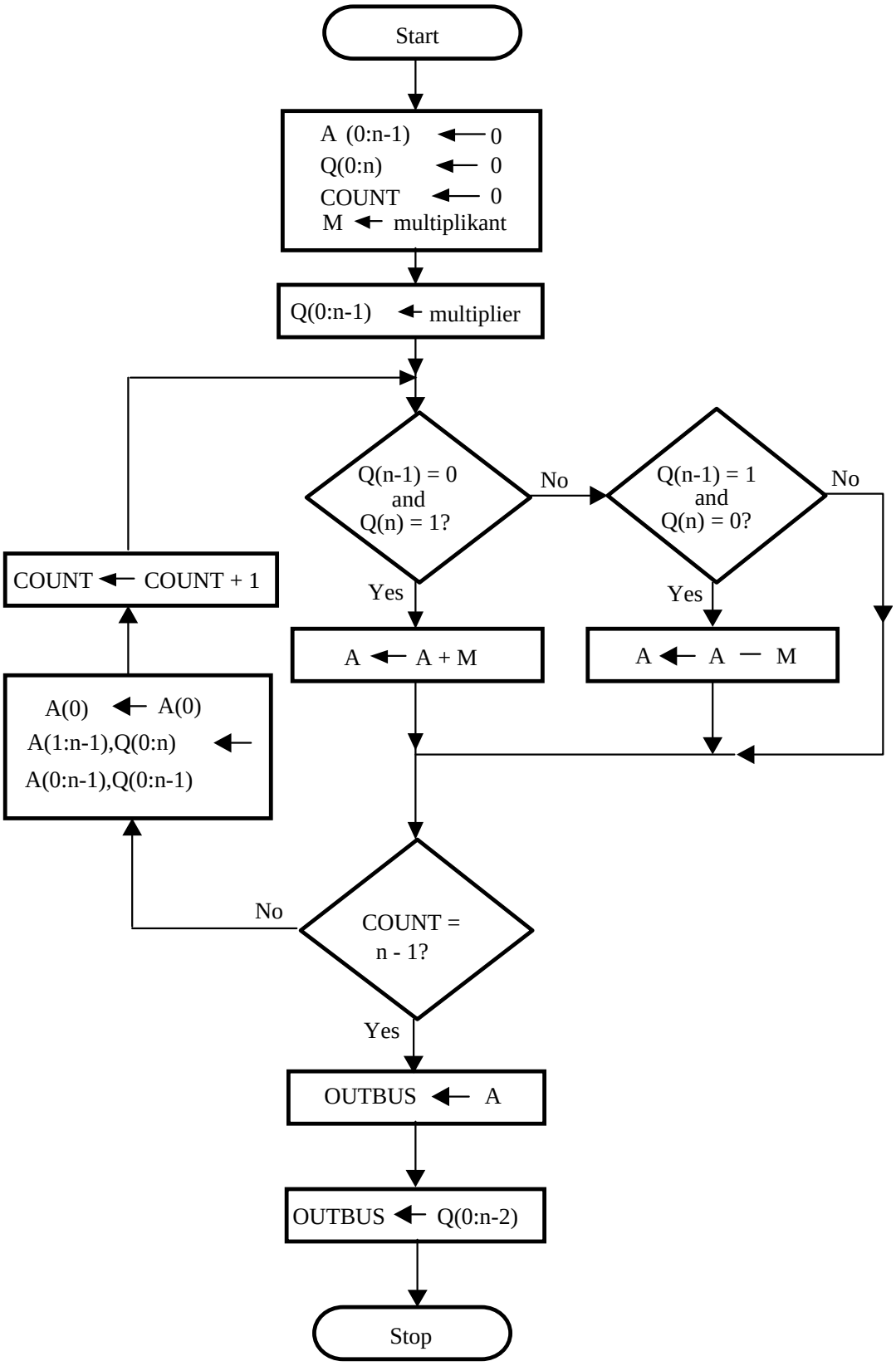


- Example of bit pair recording derived from Booth recording

Multi-plier i+1	bit-pair i	Multiplier bit on the right i-1	Multiplicand selected at Spi
0	0	0	0 x M
0	0	1	+1 x M
0	1	0	+1 x M
0	1	1	+2 x M
1	0	0	-2 x M
1	0	1	-1 x M
1	1	0	-1 x M
1	1	1	0 x M

- Table of multiplicand selection decisions

Booth´s Algorithm - a Flowchart



M	A	Q	Comments
0.010	0.000	10110	
	0.010		Subtract M
	1.110	1.110	
	1.111	01011	Shift A,Q
			No addition nor subtraction
	1.111	10101	Shift A,Q
	0.010		Add M
	0.001	10101	
	0.000	11010	Shift A,Q
	0.010		Subtract M
	1.110	11010	

--- Product P ---------

- Example of 2´s-complement multiplication using Booth's algorithm.

Theoretical Lower Bounds (Fastest Times) For Addition And Multiplication

R Denotes each logic element fan-In

N Denotes number of bits in each operand

T Represents time (in gate delays)

	THEORY	PRACTICE
Addition	$T \geq \lceil LOG_R(N-2) \rceil$	$4\lceil LOG_R N \rceil$ Carry Lookahead
Multiplikation	$T \geq \lceil LOG_R 2N \rceil$	$2\lceil LOG_{3/2}(N) \rceil + 2\lceil LOG_R N \rceil$

- Example : R = 4, N = 16

	THEORY	PRACTICE
Addition	2 Gate Delays	8 Gate Delays
Multiplikation	3 Gate Delays	18 Gate Delays

SPEED OF SINGLE CHIP MULTIPLIERS

		Bits
AMD (Advanced Micro Devices) 25S05	40 ns	2x4
Monolithic Memories Inc. 67516	800 ns	16x16
Motorola	100 ns	32x32
TRW MP4 16	230 ns	32x32
TJ 74S274	70 ns	4x4
MIPS R3010	40 ns	
TI 8847	30 ns	