

04835370

人工智能芯片设计导论

Fall 2023

Memory & Arithmetic

---

燕博南

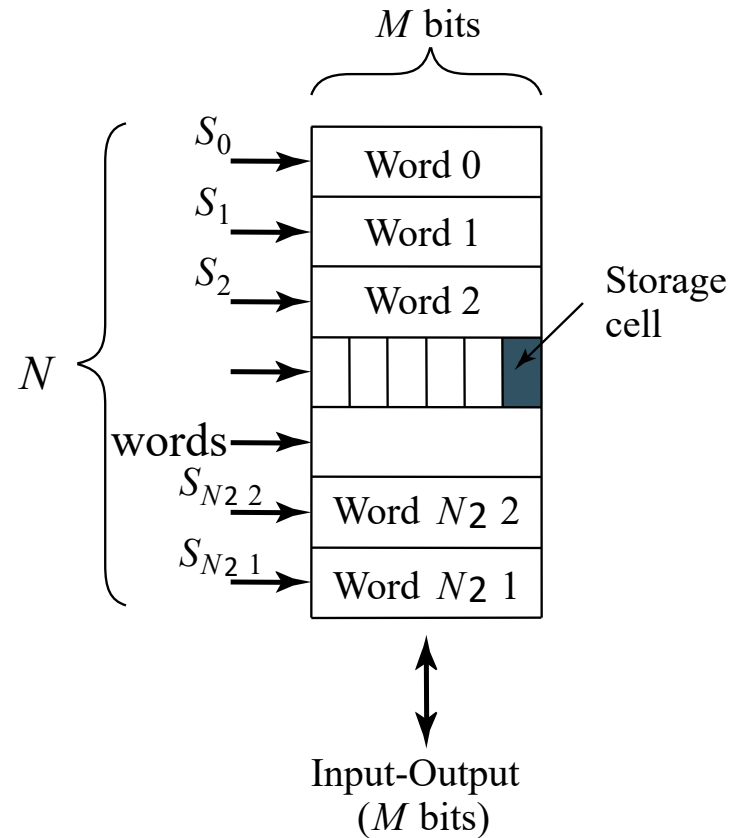
## Outline

- **Memory**
- Arithmetic Unit

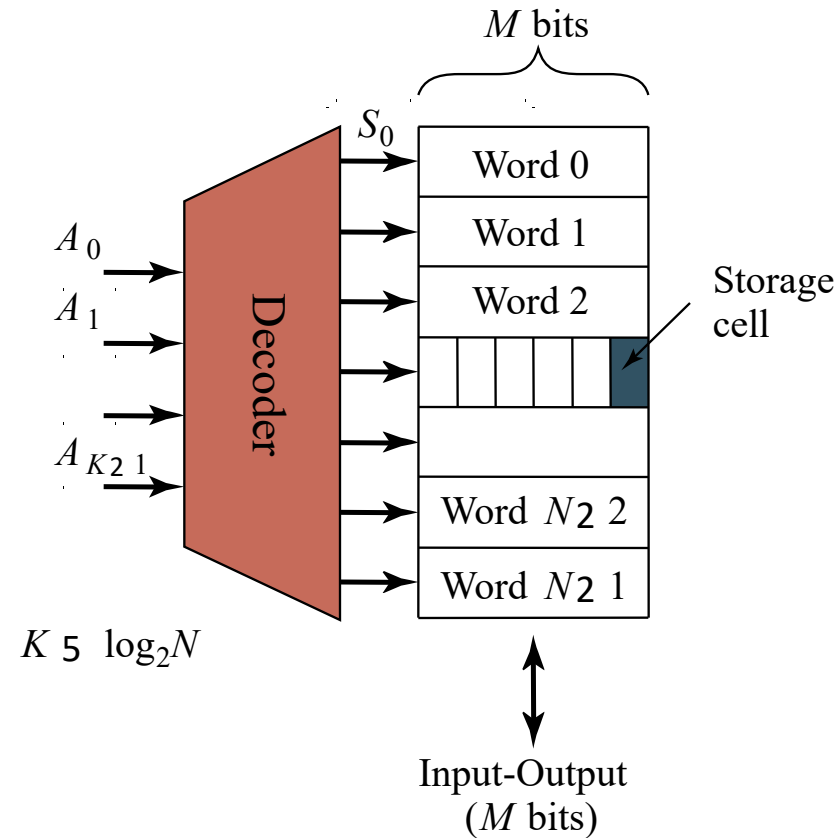
# Memory Types

Read-Write Memory		Non-Volatile Read-Write Memory	Read-Only Memory
Random Access	Non-Random Access	EPROM E <sup>2</sup> PROM  FLASH  RRAM <i>MRAM</i> <i>PCM</i>	Mask-Programmed Programmable (PROM)
SRAM  DRAM	FIFO  LIFO Shift Register  CAM		

# Memory Spatial Abstraction



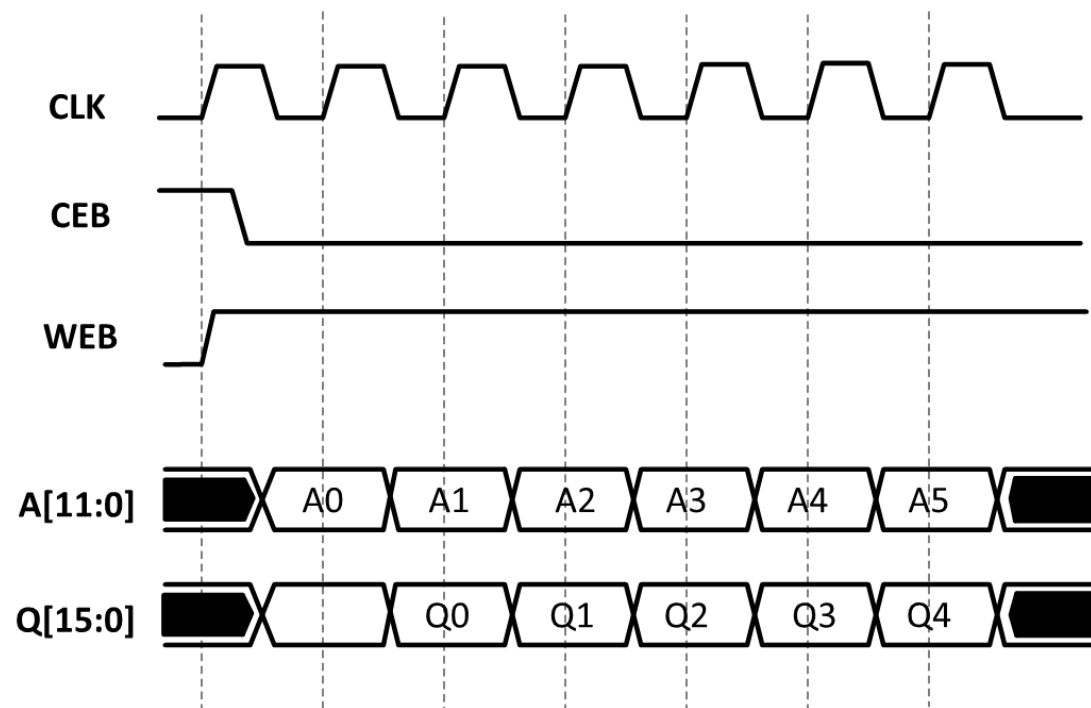
Intuitive architecture for  $N \times M$  memory  
Too many select signals:  
 $N$  words ==  $N$  select signals



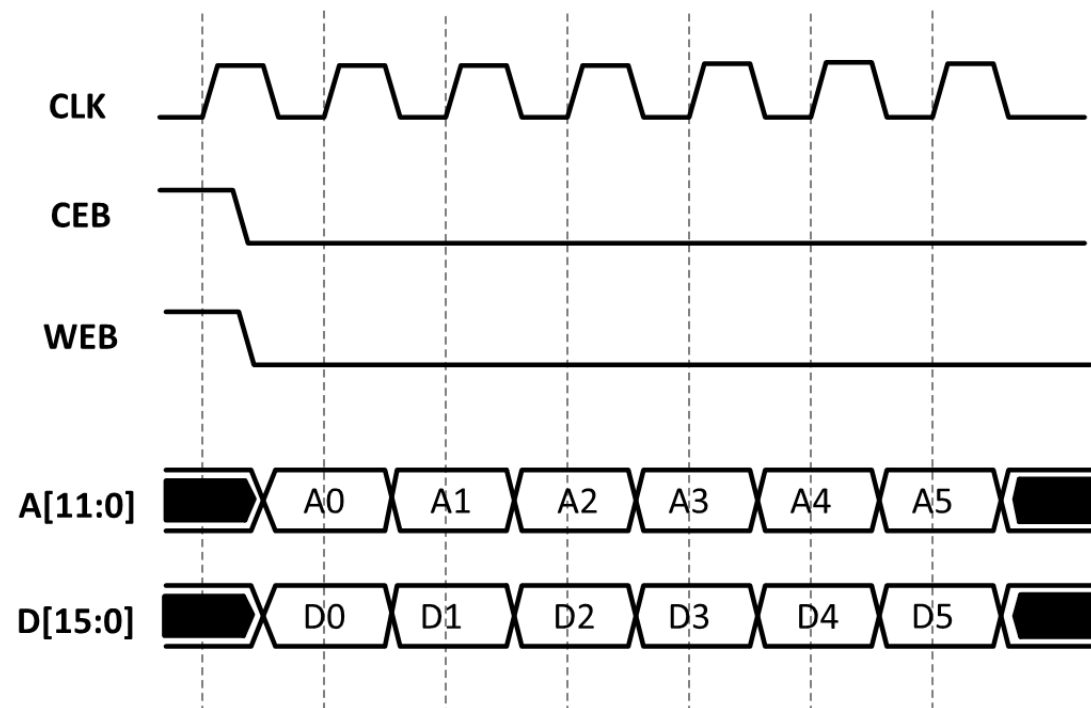
Decoder reduces the number of select signals  
 $K = \log_2 N$

# Memory Timing Behavior

Normal Mode Read

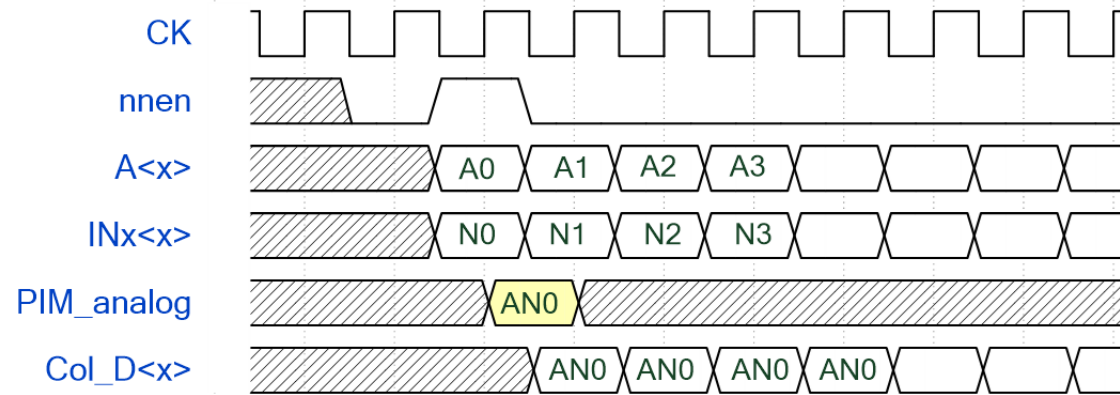


Normal Mode Write

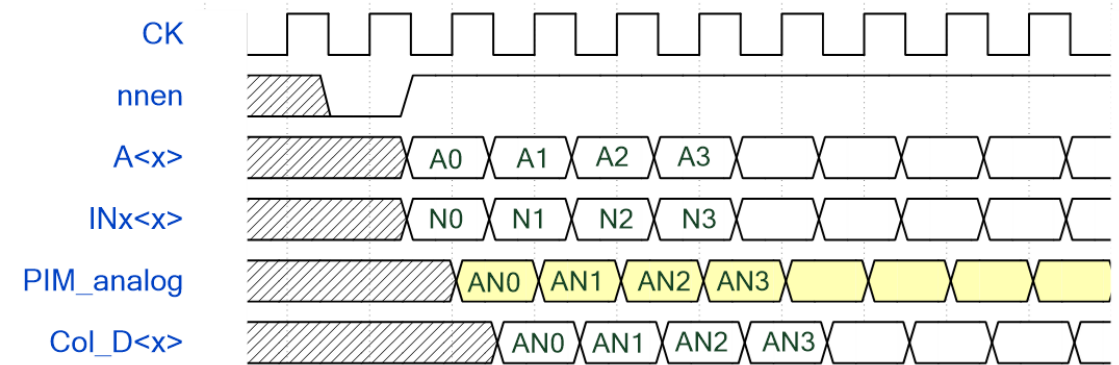


# Memory Timing Behavior / Compute-In-Memory

**Burst condition**

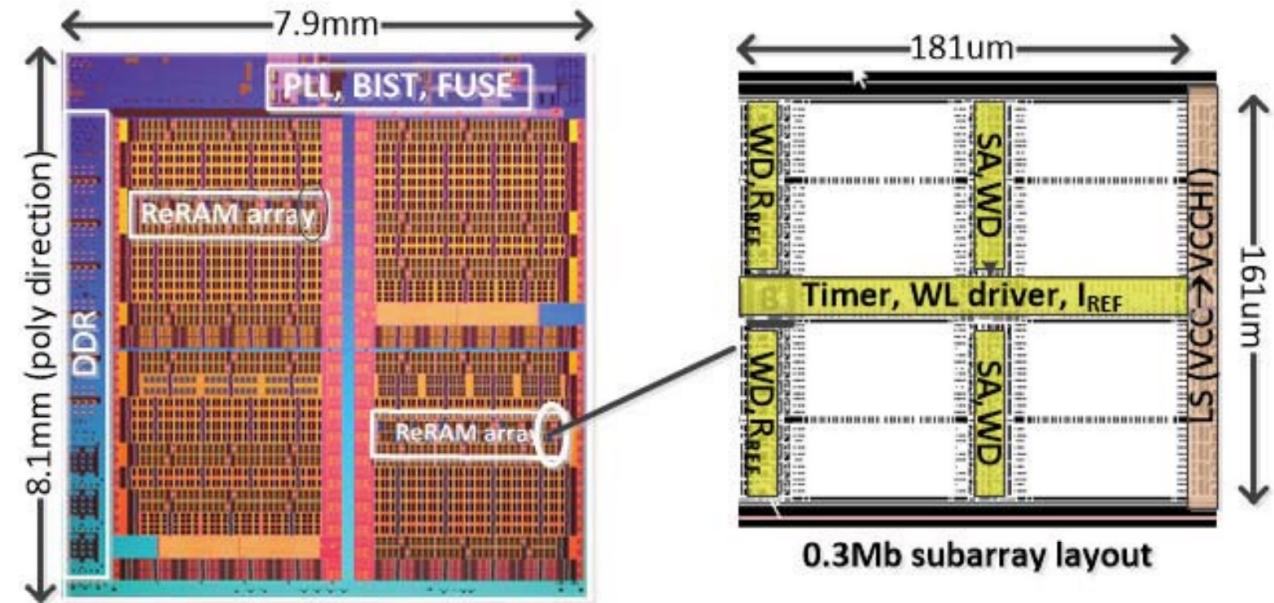
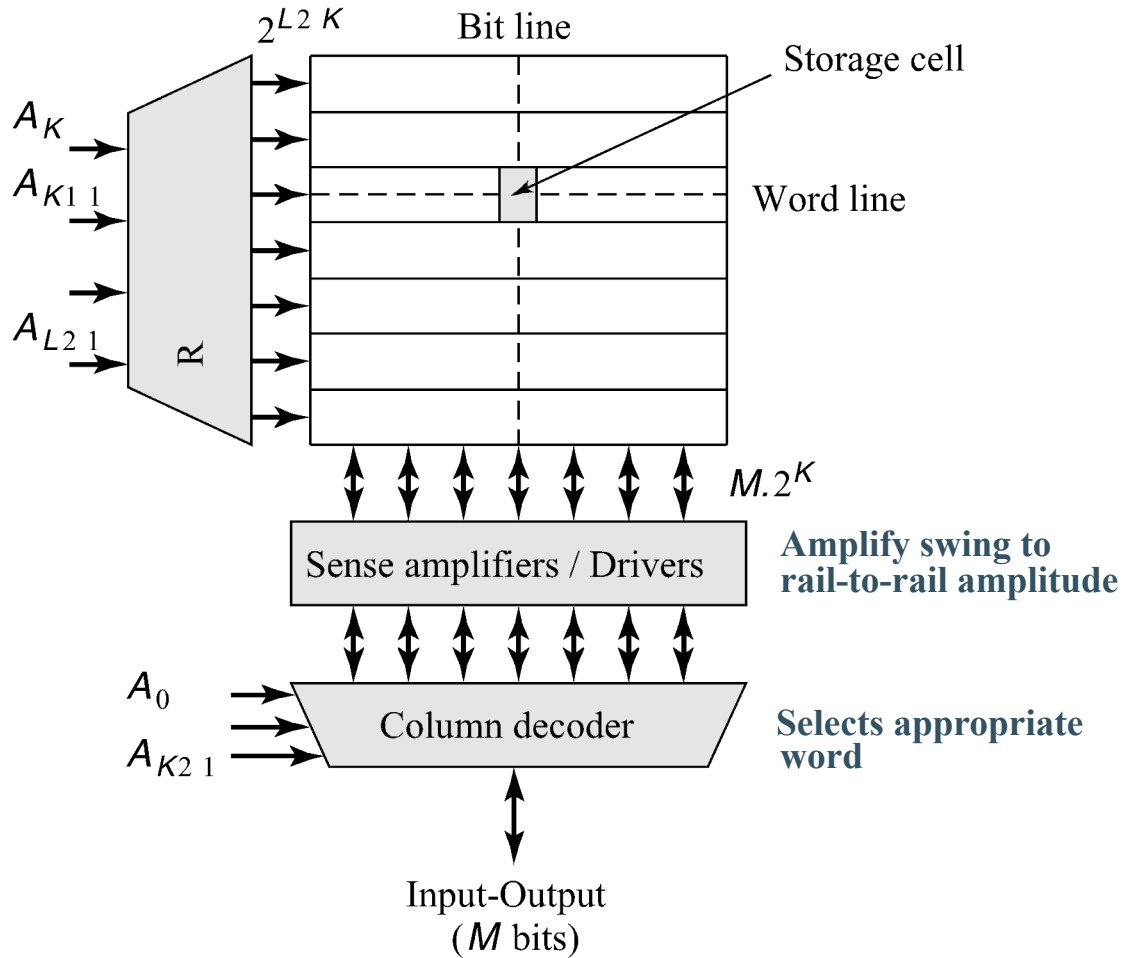


**Continuous condition**



- Add additional (compute mode) inputs
- Perhaps additional address bits

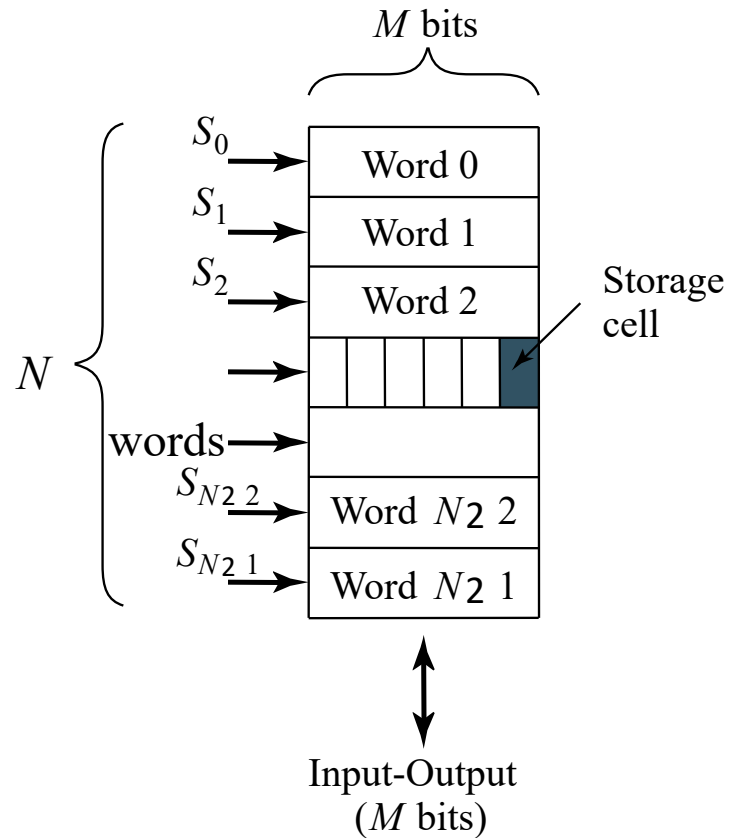
# Memory Architecture



How it can achieve large capacity?

Jain, Pulkit, et al. "13.2 A 3.6 Mb 10.1 Mb/mm<sup>2</sup> embedded non-volatile ReRAM macro in 22nm FinFET technology with adaptive forming/set/reset schemes yielding down to 0.5 V with sensing time of 5ns at 0.7 V." *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019.

# Memory Address



Each memory I/O bit width is 128bit

For 1Mb memory, what is the range of memory address?

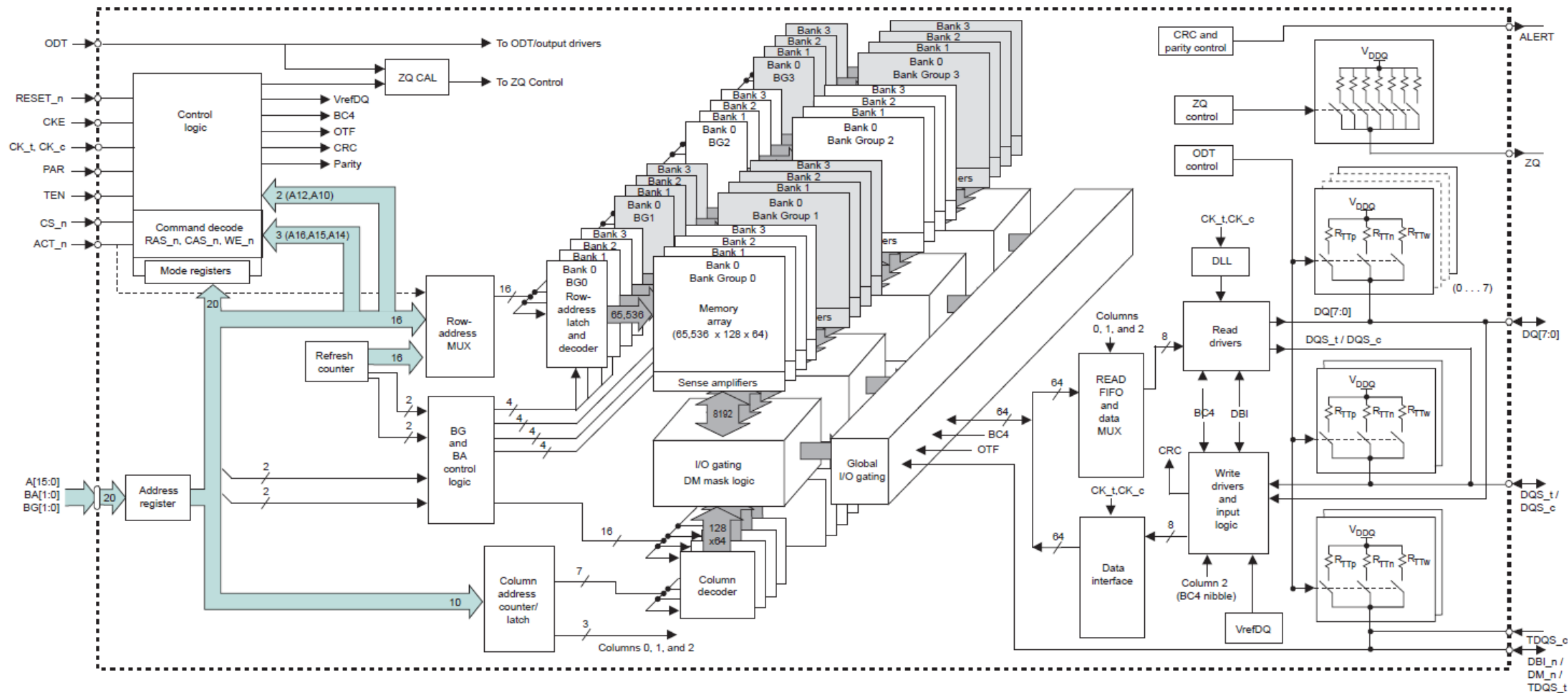
$$\begin{aligned} & 1\text{Mb}/128\text{b} \\ &= (2^{20} \text{ bit}) / (2^7 \text{ bit}) \\ &= 2^{13} \end{aligned}$$

13-wire address is necessary



# A Practical DRAM Product

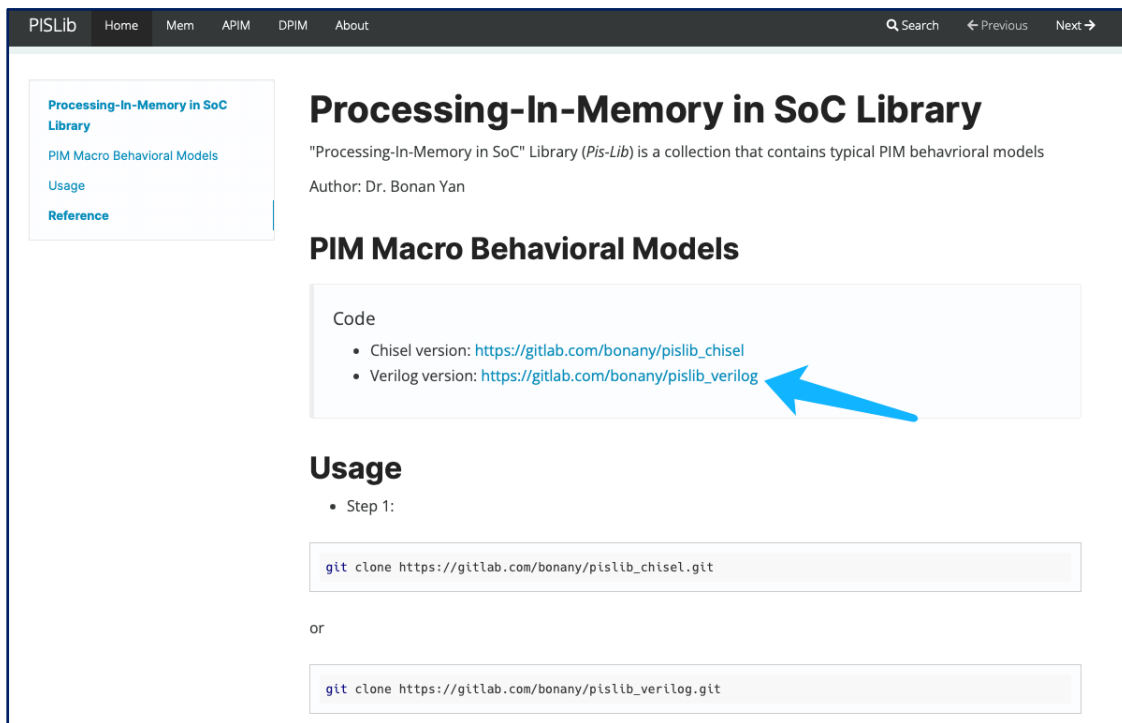
Figure 3: 1 Gig x 8 Functional Block Diagram



Source: Micron

# Memory Behavioral Model

<https://bonany.gitlab.io/pis/>



PISLib Home Mem APIM DPIM About Search Previous Next

Processing-In-Memory in SoC Library

PIM Macro Behavioral Models

Usage

Reference

## Processing-In-Memory in SoC Library

"Processing-In-Memory in SoC" Library (*Pis-Lib*) is a collection that contains typical PIM behavioral models

Author: Dr. Bonan Yan

### PIM Macro Behavioral Models

Code

- Chisel version: [https://gitlab.com/bonany/pislib\\_chisel](https://gitlab.com/bonany/pislib_chisel)
- Verilog version: [https://gitlab.com/bonany/pislib\\_verilog](https://gitlab.com/bonany/pislib_verilog)

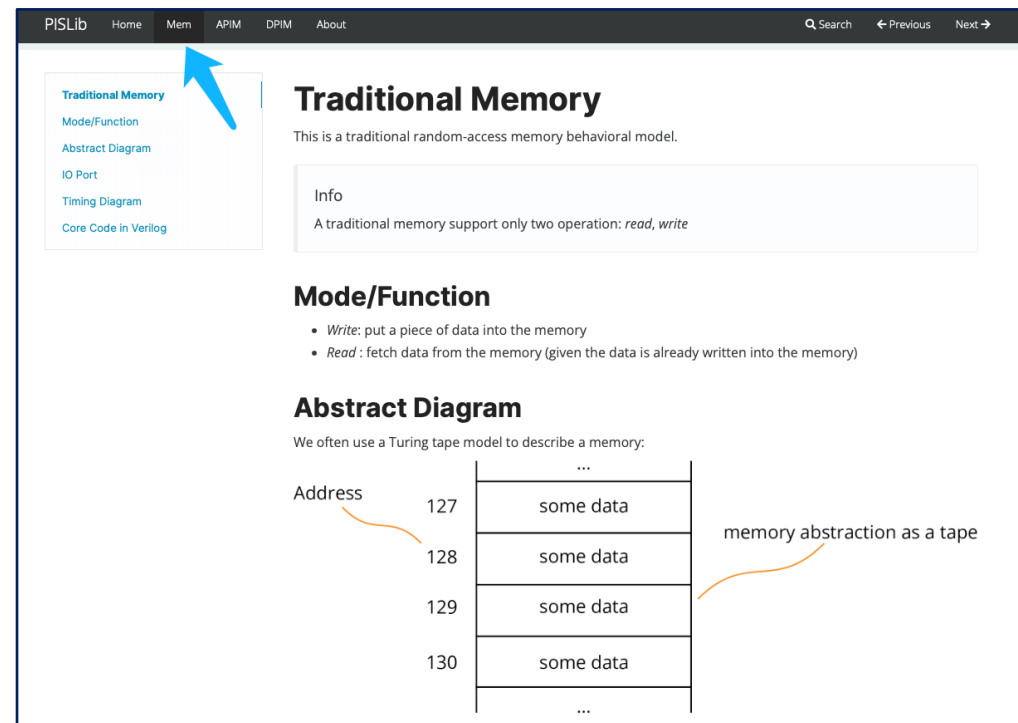
### Usage

- Step 1:

```
git clone https://gitlab.com/bonany/pislib_chisel.git
```

or

```
git clone https://gitlab.com/bonany/pislib_verilog.git
```



PISLib Home Mem APIM DPIM About Search Previous Next

Traditional Memory

Mode/Function

Abstract Diagram

IO Port

Timing Diagram

Core Code in Verilog

## Traditional Memory

This is a traditional random-access memory behavioral model.

Info

A traditional memory support only two operation: *read*, *write*

### Mode/Function

- Write*: put a piece of data into the memory
- Read*: fetch data from the memory (given the data is already written into the memory)

### Abstract Diagram

We often use a Turing tape model to describe a memory:

Address	
127	some data
128	some data
129	some data
130	some data
...	...

memory abstraction as a tape

- Memory
- **Arithmetic Unit**
  - Number Systems
    - Integer
    - Fixed-Point
    - Floating-Point
  - Arithmetic
  - Circuits & Implementation

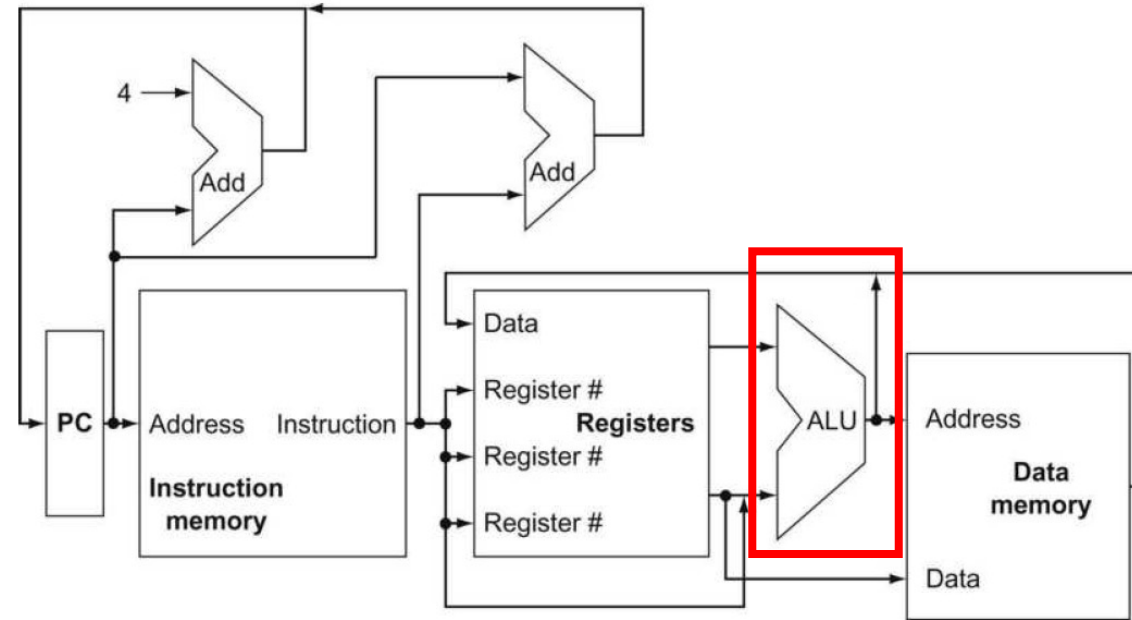


Numbers

Arithmetic

Circuits

# Why We Need to Introduce Arithmetic

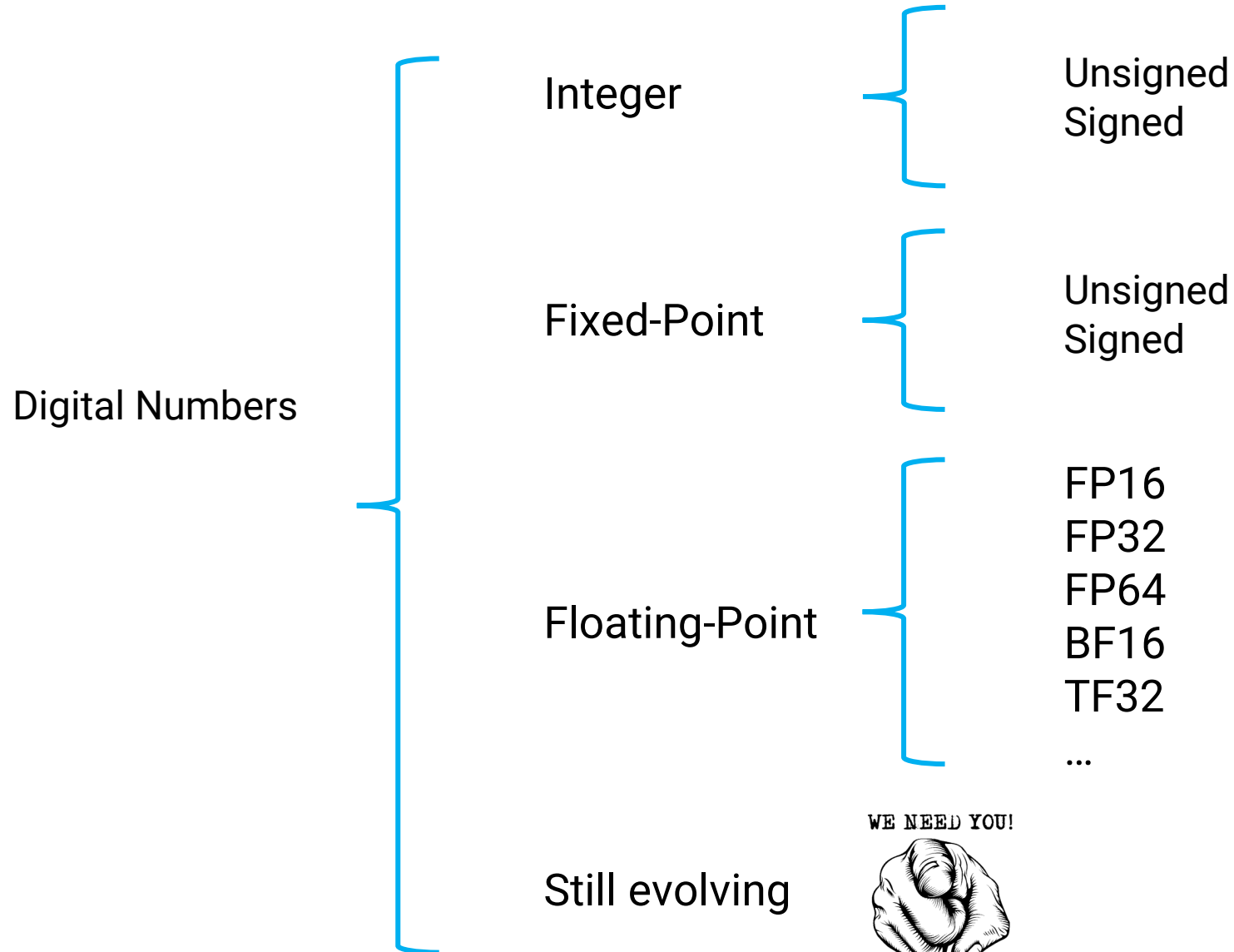


- Arithmetic Logic Unit (ALU): heart of von Neumann architecture
- Deal with various precisions: decimals, fractions, integers, ...

# Part 1

Unsigned Integers

# Number System of Digital Computers



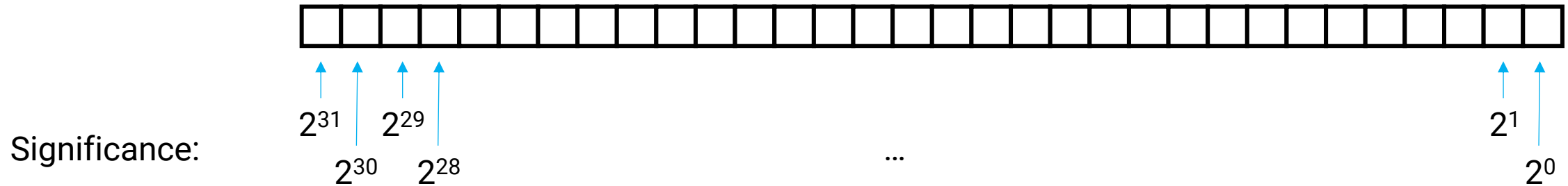
WE NEED YOU!



# Unsigned Integer



- Unsigned INT32



- INT16, INT8, ...

- Example:

32'd7 (=32'h0000\_0007)  
8'b1100\_1101 (=8'hCD)

Recommend tool:  
programmer's calculator

# Unsigned Integer Arithmetic – Add & Subtract

Numbers

Arithmetic

Circuits

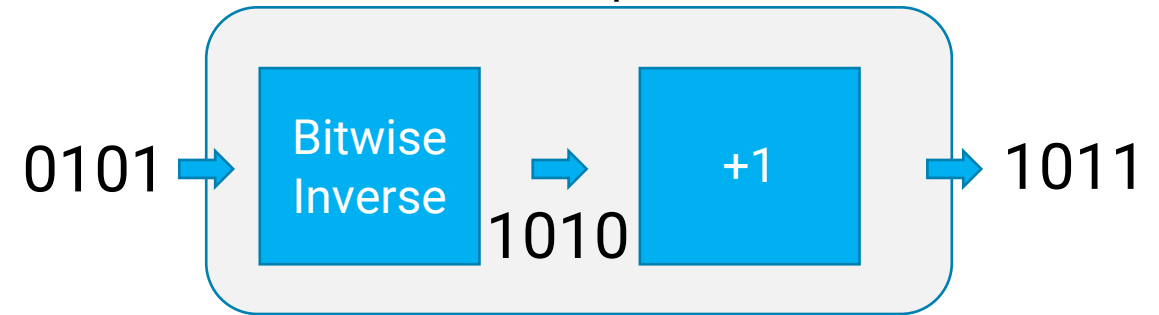
- INT4 as example:

Add:  $4'd7 + 4'd5 = 4'd12$

$$\begin{array}{r} 0111 \\ + 0101 \\ \hline 1100 \end{array}$$

Subtract:  $4'd12 - 4'd5 = 4'd7$

Two's Complement



$$\begin{array}{r} 0111 \\ - 0101 \\ \hline \end{array} \quad \rightarrow \quad \begin{array}{r} 1100 \\ + 1011 \\ \hline 0111 \end{array}$$



# Unsigned Integer Arithmetic – Multiply & Divide

Numbers

Arithmetic

Circuits

- INT4 as example:

Multiply:  $4'd3 * 4'd5 = 4'd15$

$$\begin{array}{r} \phantom{00}0011 \\ * \phantom{00}0101 \\ \hline \phantom{00}0011 \\ \phantom{00}0000 \\ \phantom{00}0011 \\ + \phantom{00}0000 \\ \hline \phantom{00}1111 \end{array}$$

Divide:  $4'd15 / 4'd3 = 4'd5$

$$\begin{array}{r} \phantom{00}101 \\ 11 \overline{) 1111} \\ \underline{- 11} \phantom{00} \\ \phantom{00}01 \\ \phantom{00}00 \\ \hline \phantom{00}11 \\ \phantom{00}11 \\ \hline \phantom{00}0 \end{array}$$

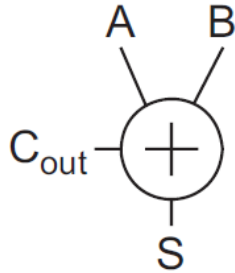
# Unsigned Integer Circuits – Adder

Numbers

Arithmetic

Circuits

Half-Adder

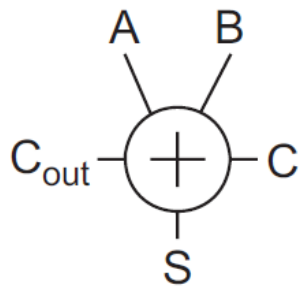


$A$	$B$	$C_{out}$	$S$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$

$$C_{out} = A \cdot B$$

Full-Adder



$A$	$B$	$C$	$G$	$P$	$K$	$C_{out}$	$S$
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

$$= (A \oplus B) \oplus C = P \oplus C$$

$$C_{out} = AB + AC + BC$$

$$= AB + C(A + B)$$

$$= \overline{\bar{A}\bar{B} + \bar{C}(\bar{A} + \bar{B})}$$

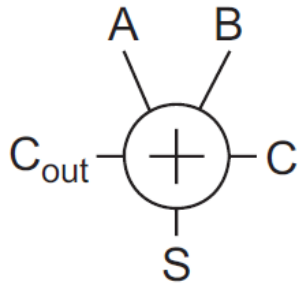
$$= \text{MAJ}(A, B, C)$$

# Unsigned Integer Circuits – Adder (Cont.)

Numbers

Arithmetic

Circuits



$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

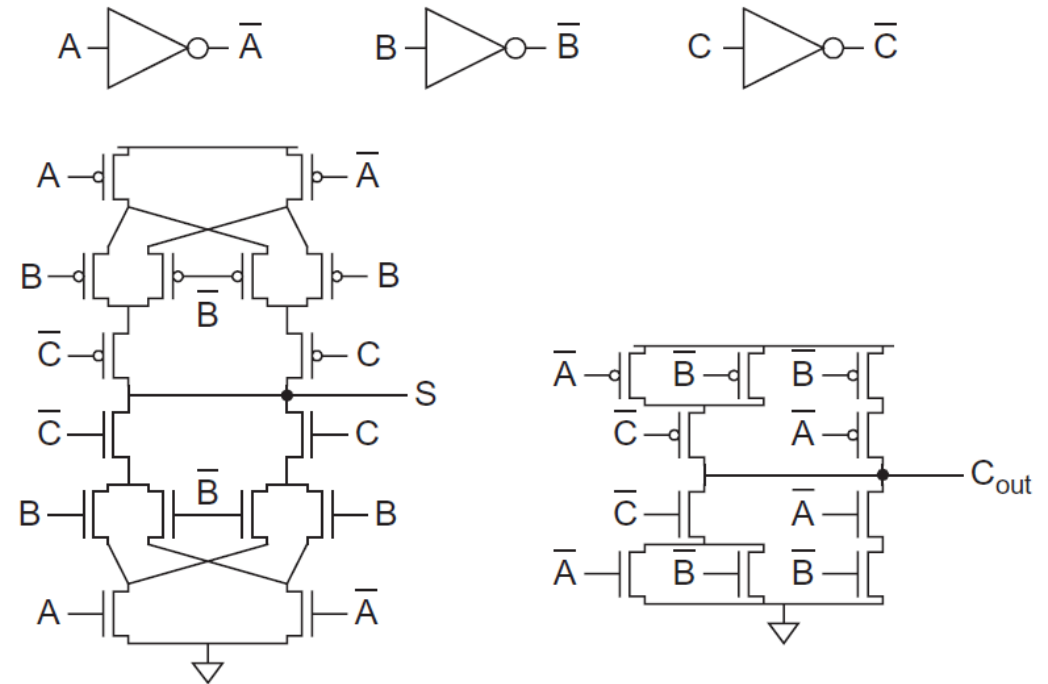
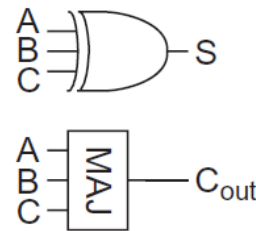
$$= (A \oplus B) \oplus C = P \oplus C$$

$$C_{\text{out}} = AB + AC + BC$$

$$= AB + C(A + B)$$

$$= \overline{\bar{A}\bar{B} + \bar{C}(\bar{A} + \bar{B})}$$

$$= \text{MAJ}(A, B, C)$$

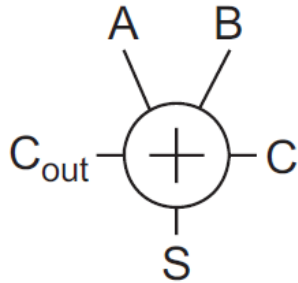


# Unsigned Integer Circuits – Adder (Cont.)

Numbers

Arithmetic

Circuits



$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

$$= (A \oplus B) \oplus C = P \oplus C$$

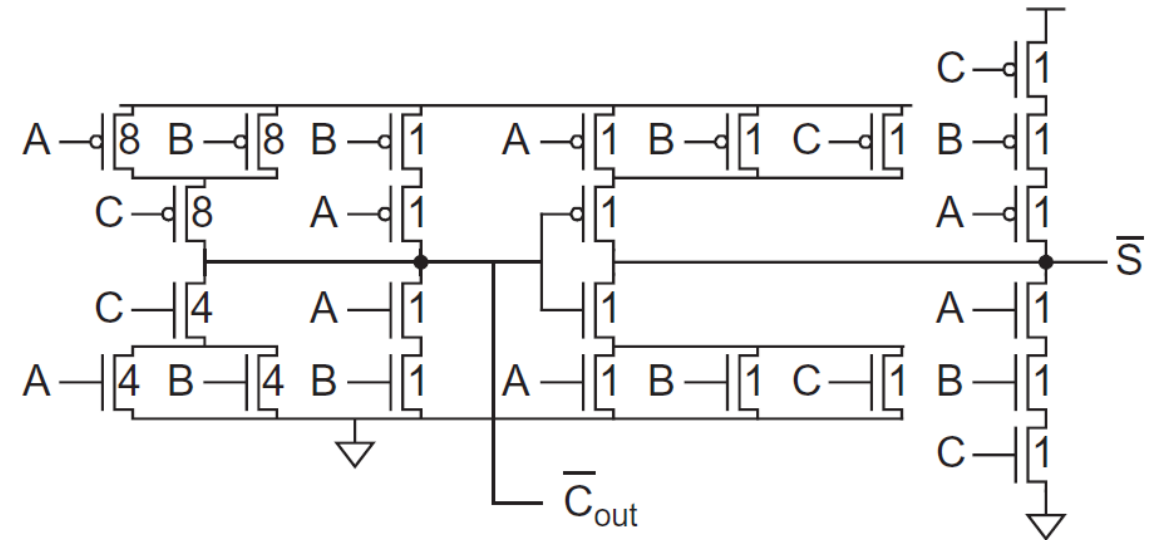
$$C_{\text{out}} = AB + AC + BC$$

$$= AB + C(A + B)$$

$$= \overline{\bar{A}\bar{B} + \bar{C}(\bar{A} + \bar{B})}$$

$$= \text{MAJ}(A, B, C)$$

Improved:



$$S = ABC + (A + B + C)\bar{C}_{\text{out}}$$

Idea behind:

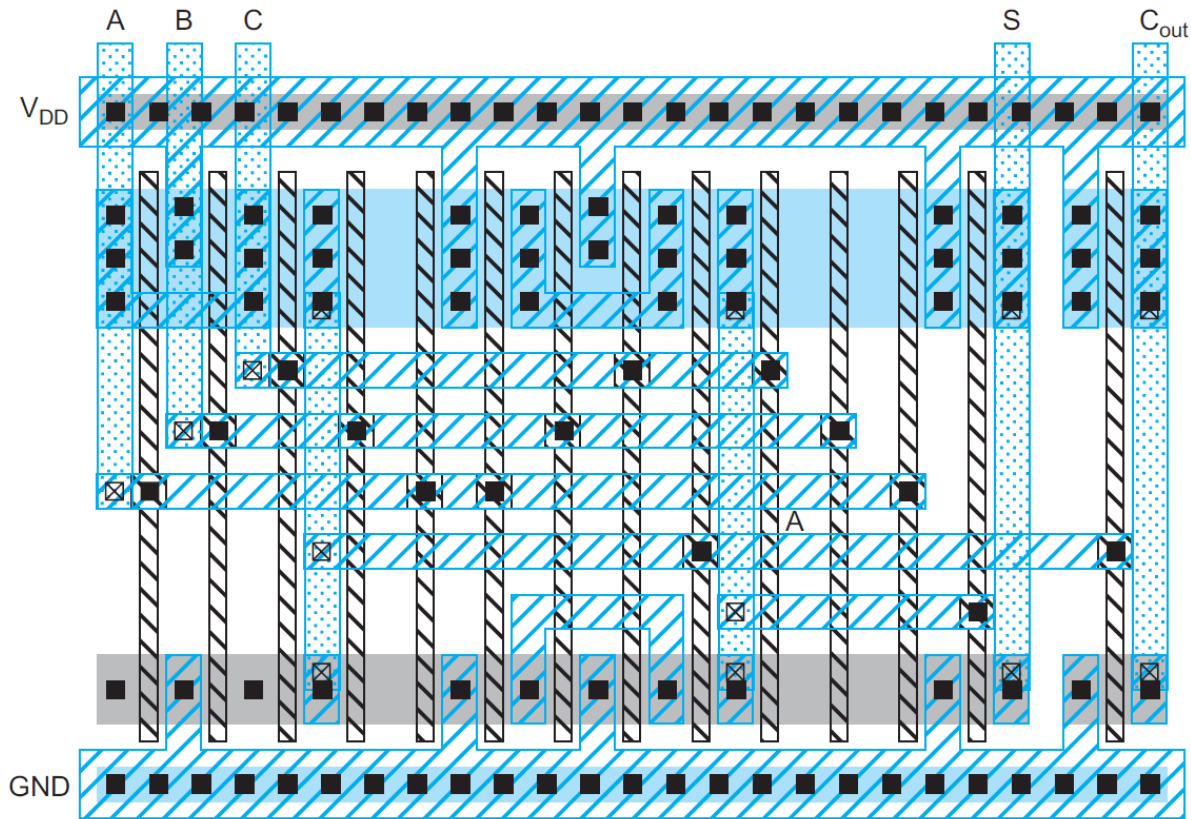
Reuse Cout compute circuits to obtain both S

# Unsigned Integer Circuits – Adder (Cont.)

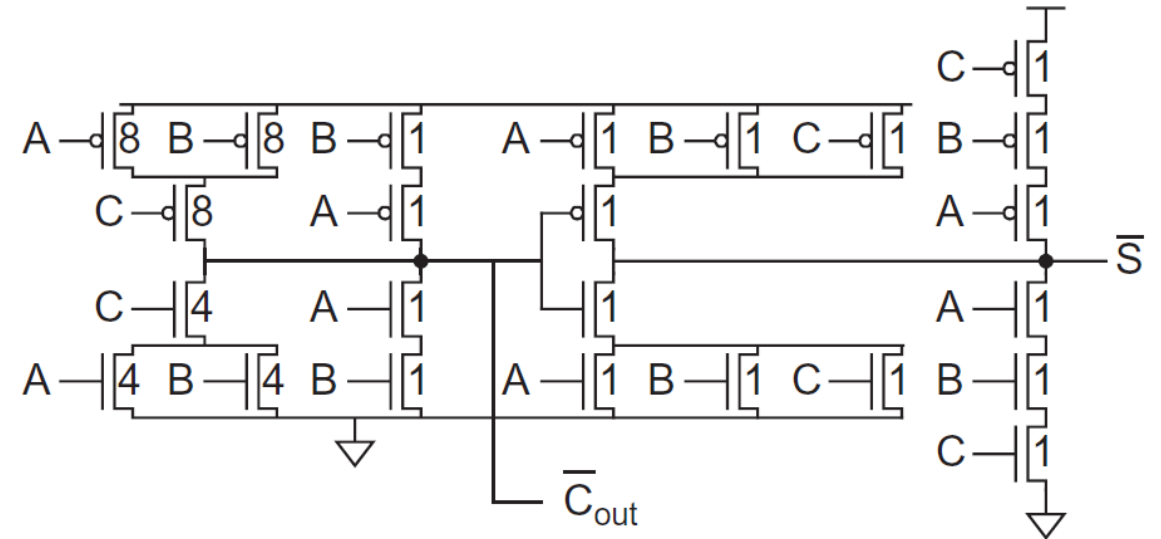
Numbers

Arithmetic

Circuits



Improved:



$$S = ABC + (A + B + C)\bar{C}_{out}$$

Idea behind:

Reuse Cout compute circuits to obtain both S

# Adder Family – Carry-Ripple Adder

Numbers

Arithmetic

Circuits

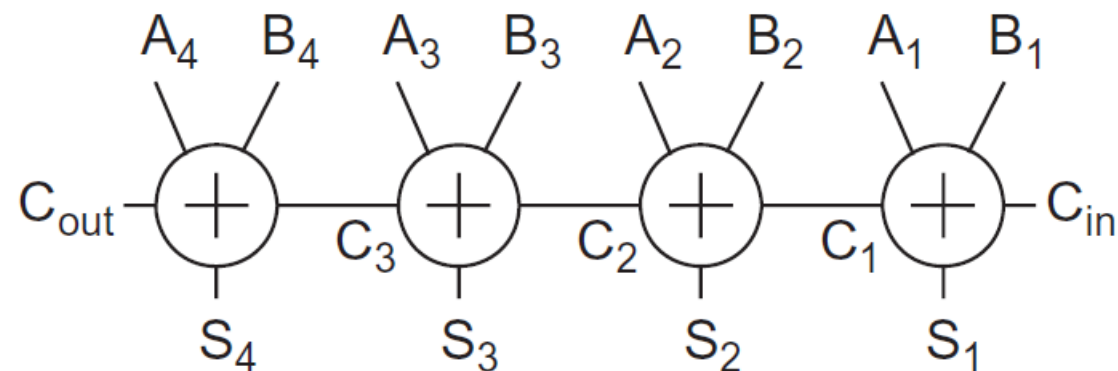
$$\begin{array}{r} 00000 \\ 1111 \\ +0000 \\ \hline 1111 \end{array}$$

$C_{out}$   $C_{in}$

$$\begin{array}{r} 11111 \\ 1111 \\ +0000 \\ \hline 0000 \end{array}$$

$C_{out}$   $C_{in}$

carries  
 $A_{4...1}$   
 $B_{4...1}$   
 $S_{4...1}$



Natural & Intuitive

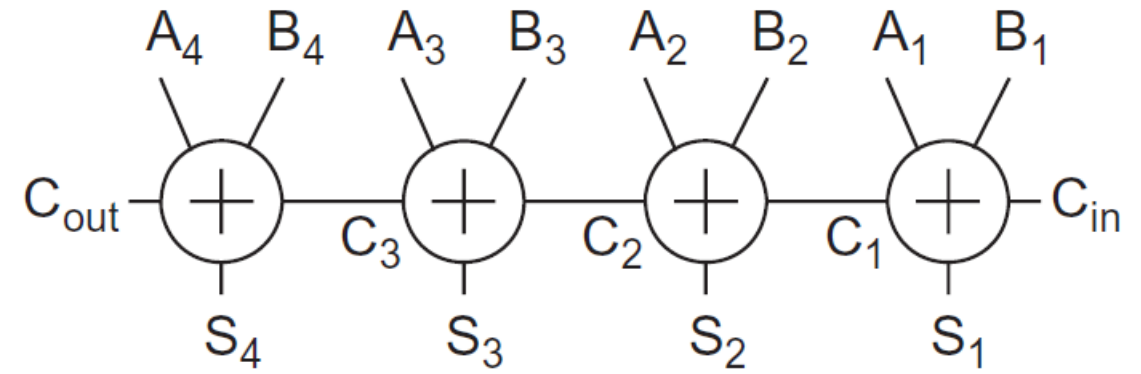
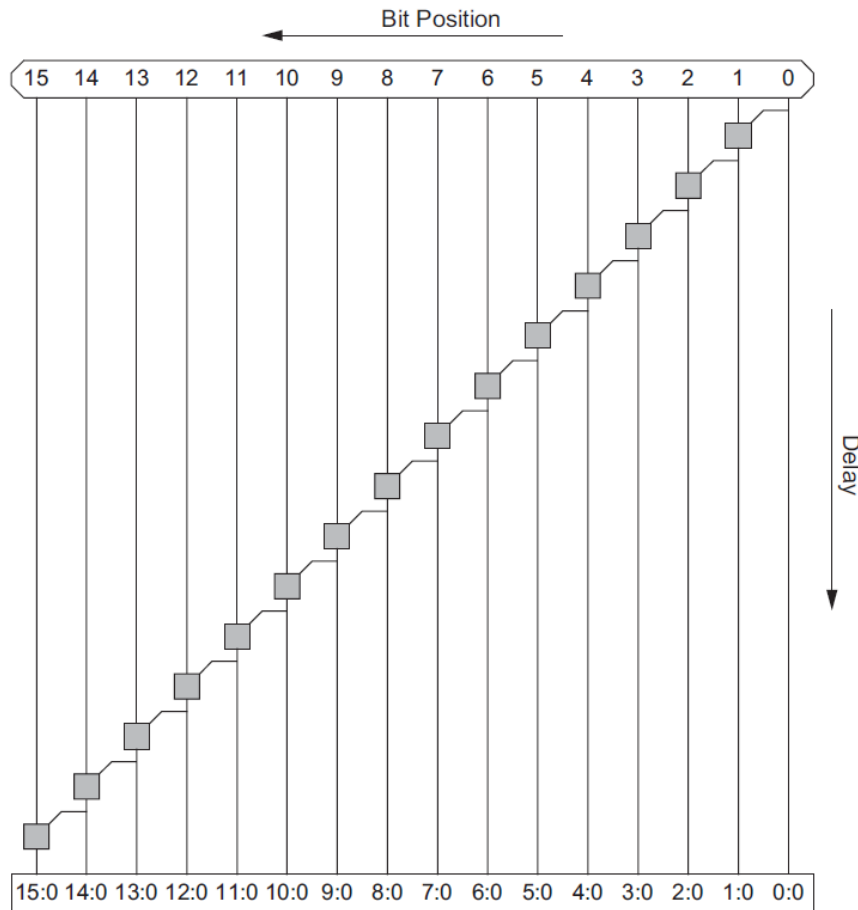
However, carry propagation path too long

# Adder Family – Carry-Ripple Adder

Numbers

Arithmetic

Circuits



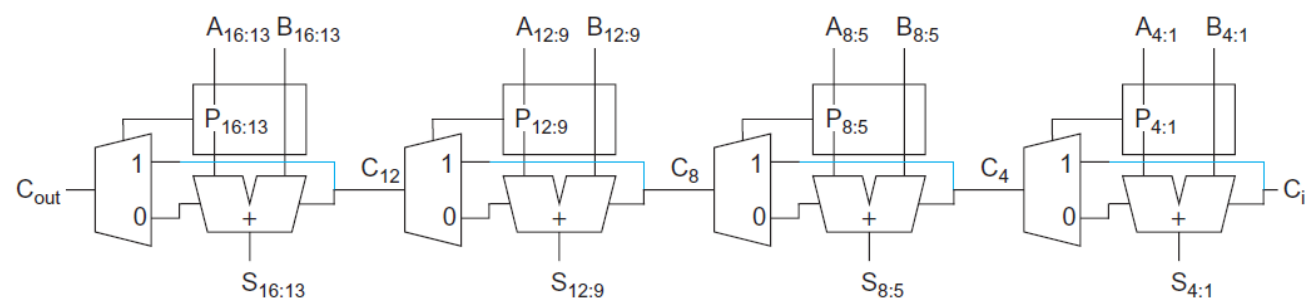
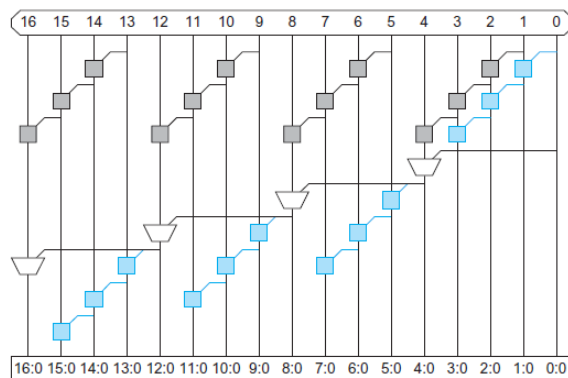
# Adder Family – Carry-Skip Adder

Numbers

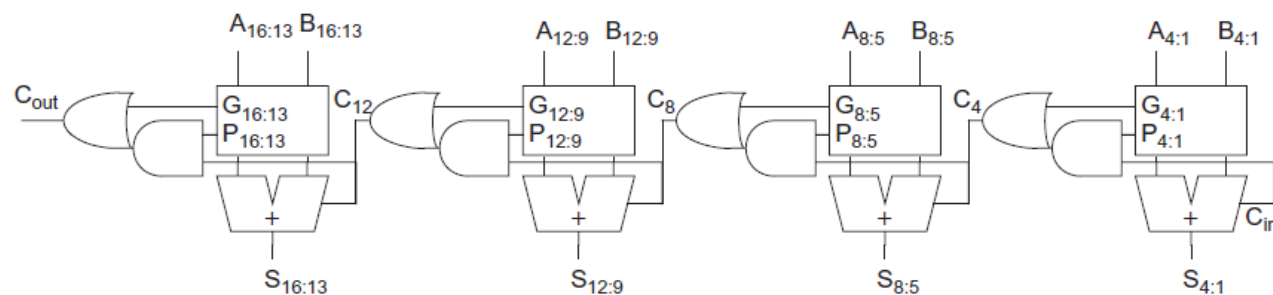
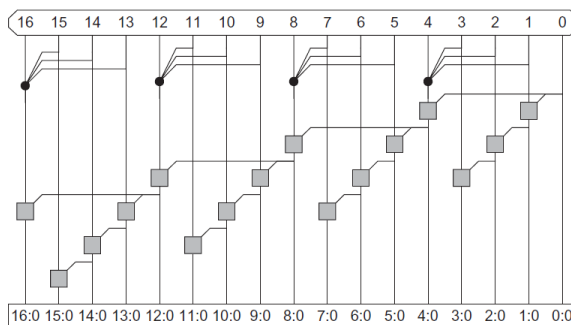
Arithmetic

Circuits

Carry-Skip  
Adder



Carry-Lookahead  
Adder



Idea Behind: Group and Divide!



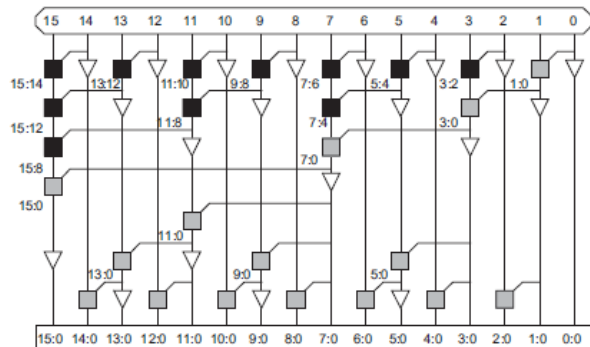
# Adder Family – Big Family!

Numbers

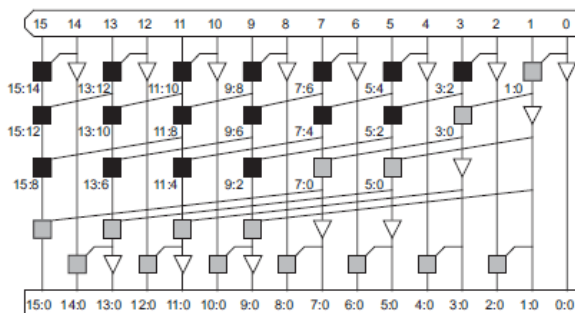
Arithmetic

Circuits

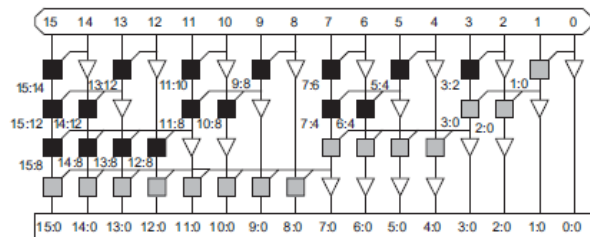
## Tree Adder Family



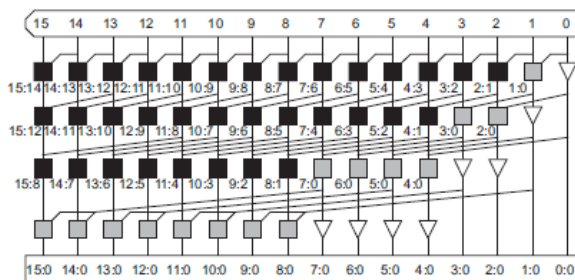
(a) Brent-Kung



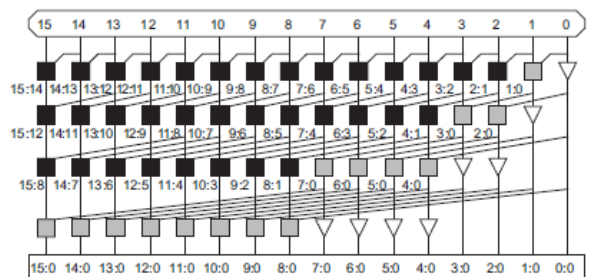
(d) Han-Carlson



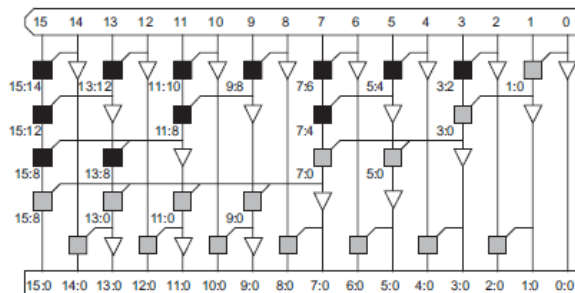
(b) Sklansky



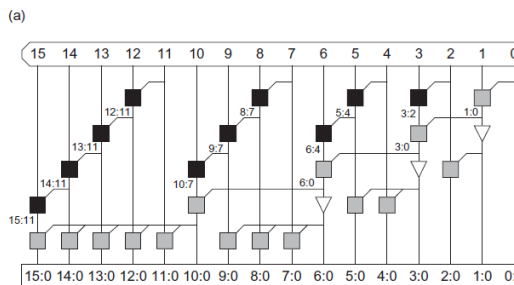
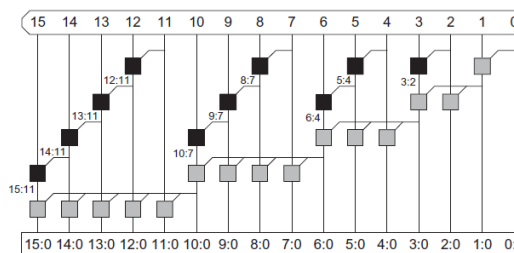
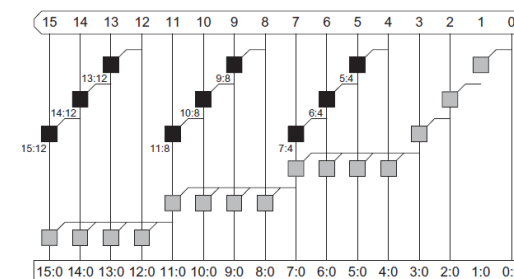
(e) Knowles [2,1,1,1]



(c) Kogge-Stone



(f) Ladner-Fischer



(b)

Carry-Incremental Adder

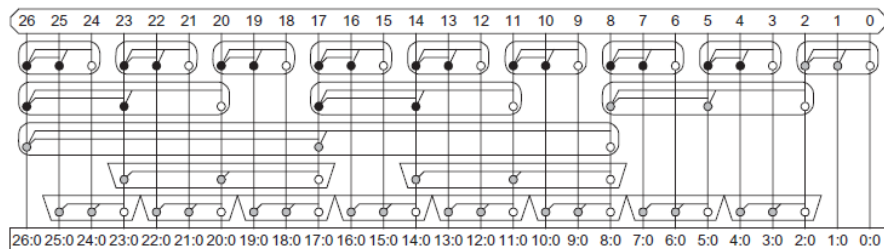
# Adder Family – Big Family! (Cont.)

## Sparse Tree Adders

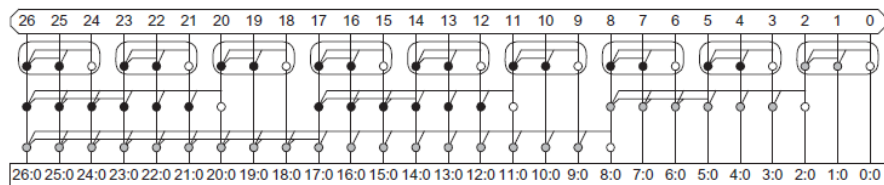
Numbers

Arithmetic

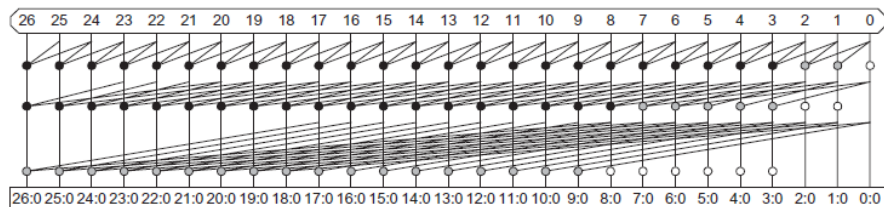
Circuits



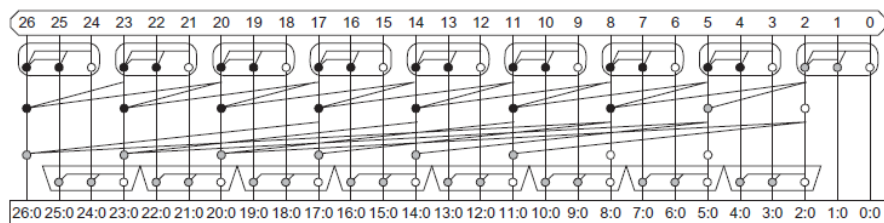
(a) Brent-Kung



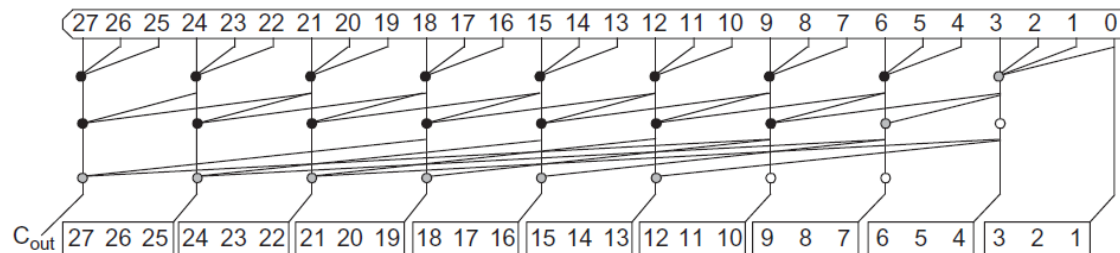
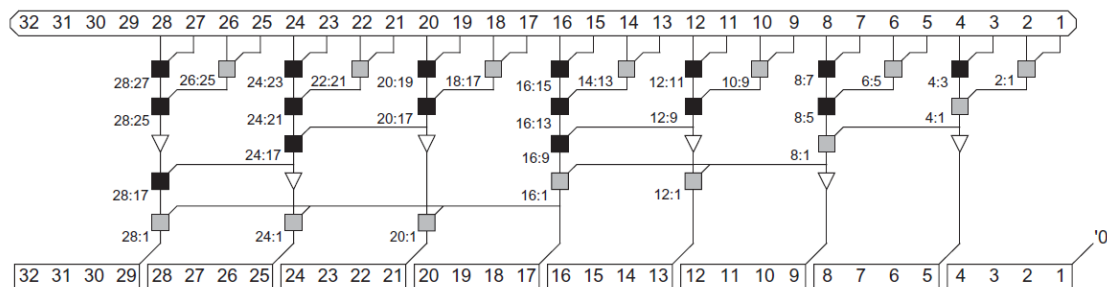
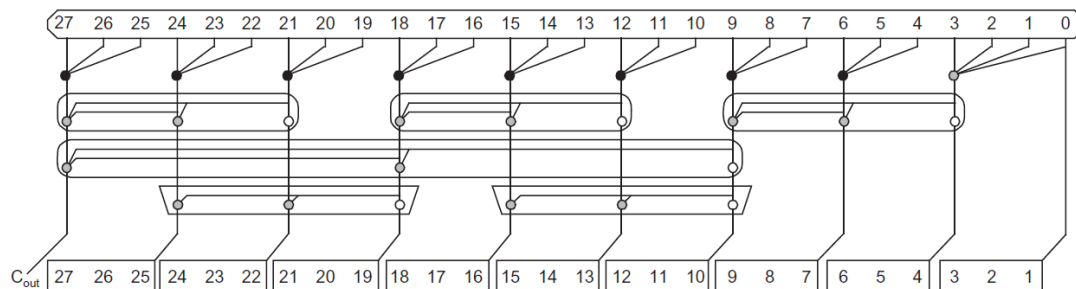
(b) Sklansky



(c) Kogge-Stone



(d) Han-Carlson

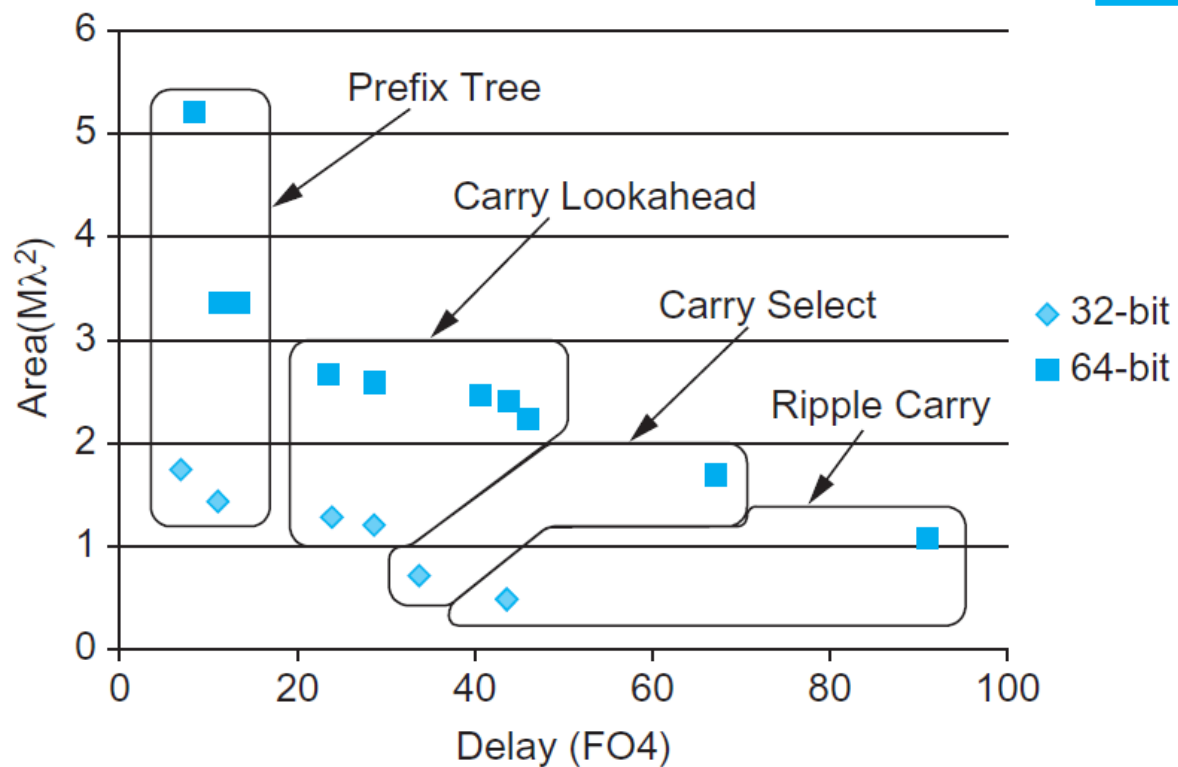


# Adder Family – Choose Wisely

Numbers

Arithmetic

Circuits



FO: Fan-Out

Everything has trade-off!

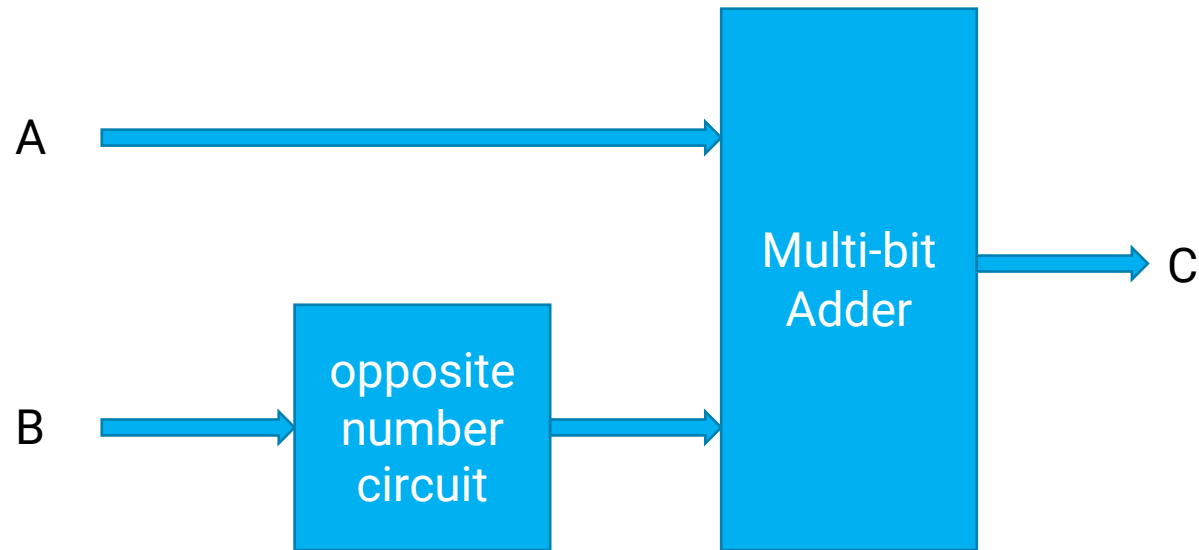
# Unsigned Integer Circuits – Subtractor

Numbers

Arithmetic

Circuits

$$C = A - B = A + (-B)$$



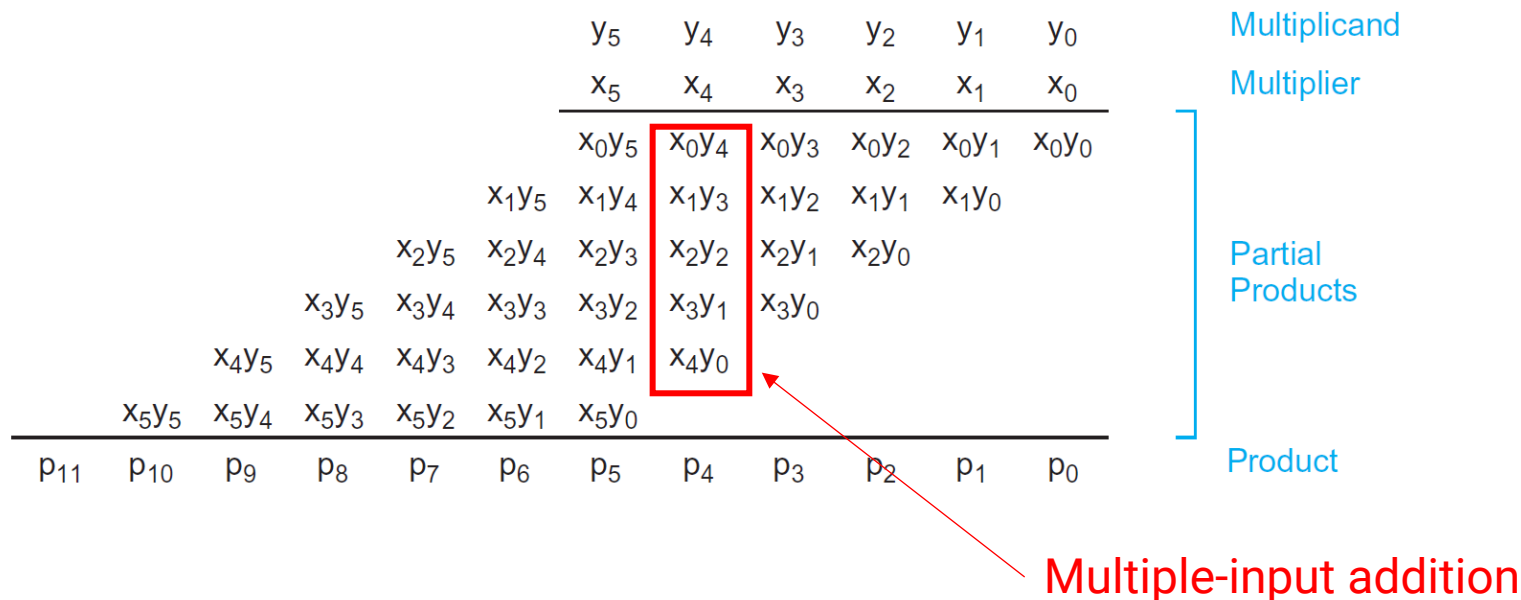
What is “opposite number circuit” though?  
[Save for a moment later]

# Unsigned Integer Circuits – Multiplier

Numbers

Arithmetic

Circuits



How do you implement it?

**Yes! Adders!**

# Unsigned Integer Circuits – Multiplier

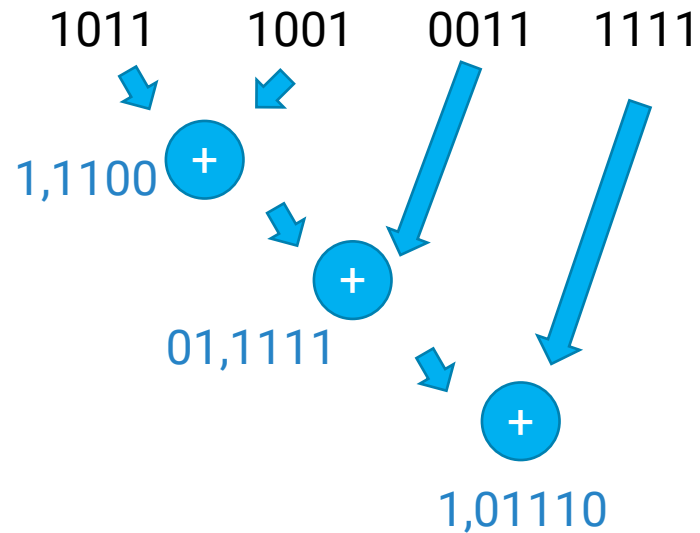
Numbers

Arithmetic

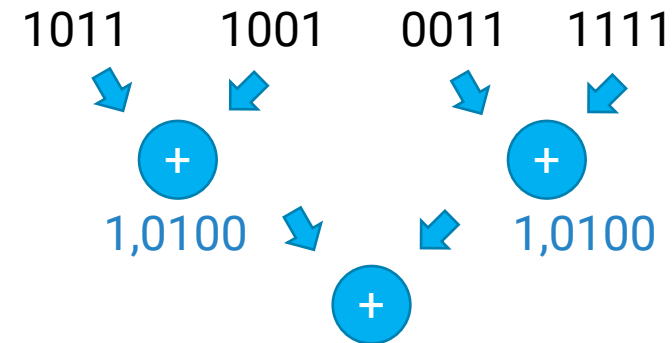
Circuits

Carry-Save Adder (CSA) & “*carry-save redundant format*”

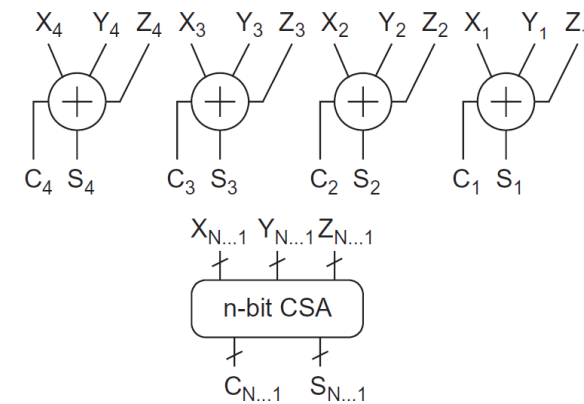
Example: Sum of 1011 1001 0011 1111 ?



( Carry, Sum )



Carry add together,  
Sum add together  
> Final sum

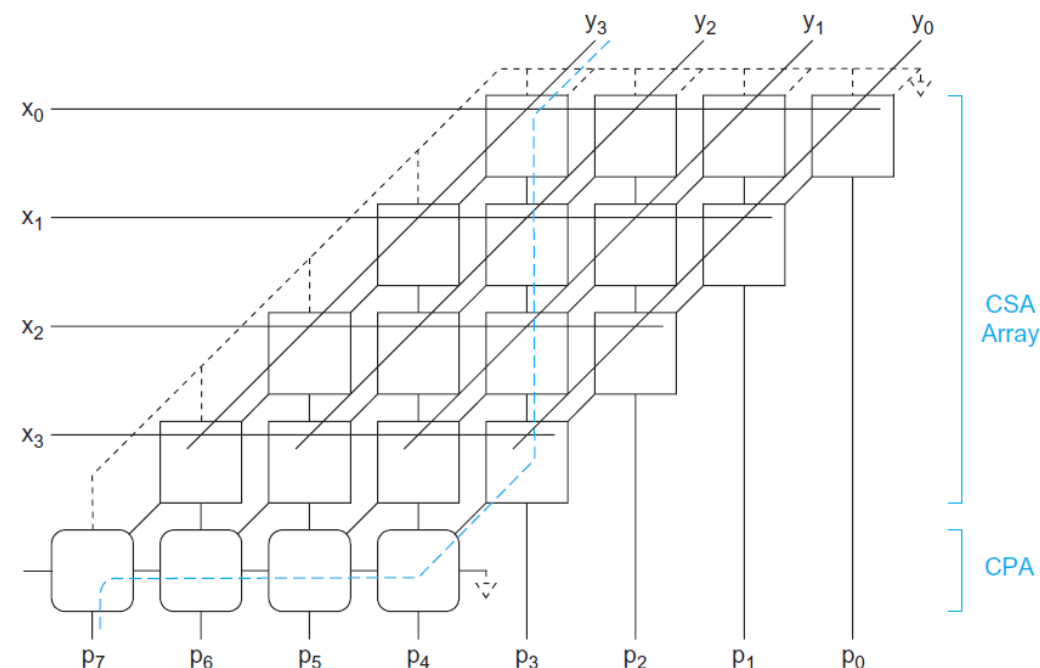
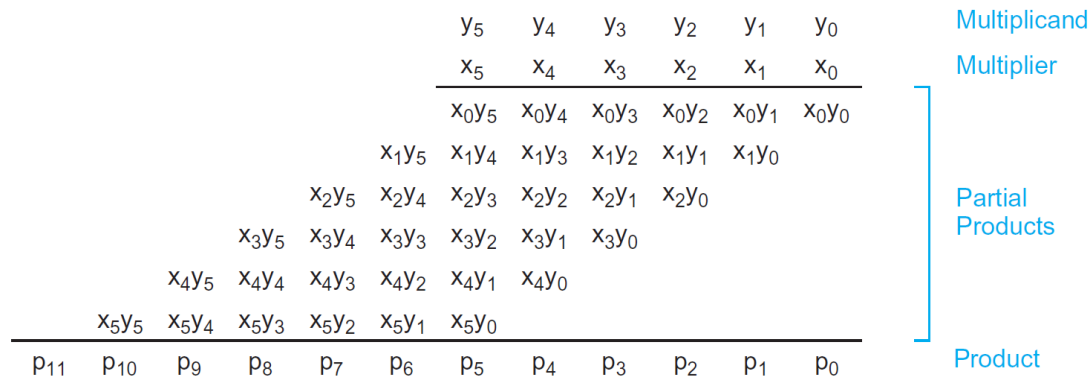


# Unsigned Integer Circuits – Multiplier

Numbers

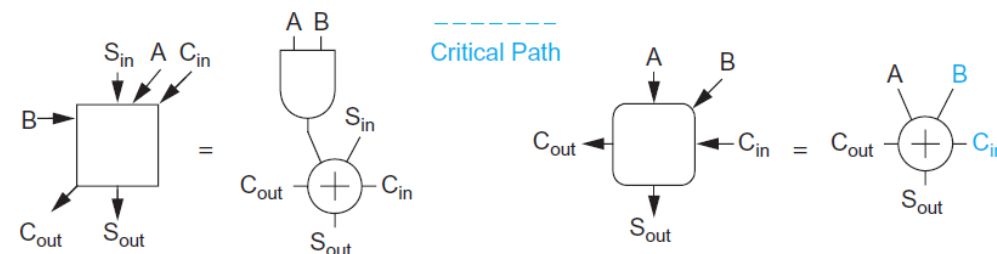
Arithmetic

Circuits



Partial Product:  
Single-bit multiplication is equivalent to “AND”

x	y	Partial product
0	0	0
0	1	0
1	0	0
1	1	1

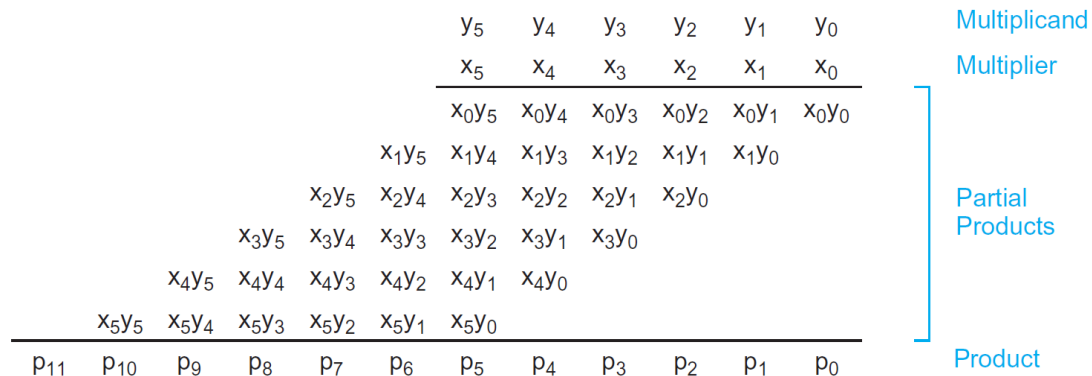


# Unsigned Integer Circuits – Multiplier

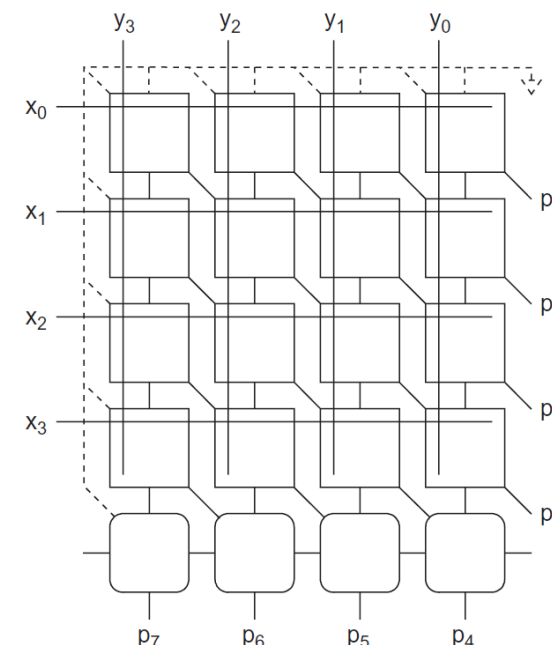
Numbers

Arithmetic

Circuits



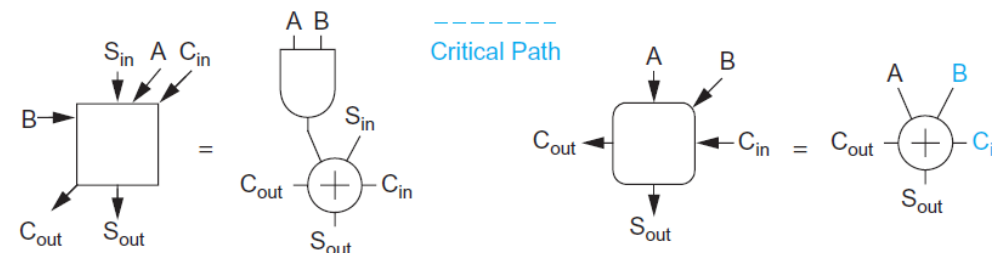
Reshape to Rectangular:



Partial Product:

Single-bit multiplication is equivalent to “AND”

x	y	Partial product
0	0	0
0	0	0
1	0	0
1	1	1





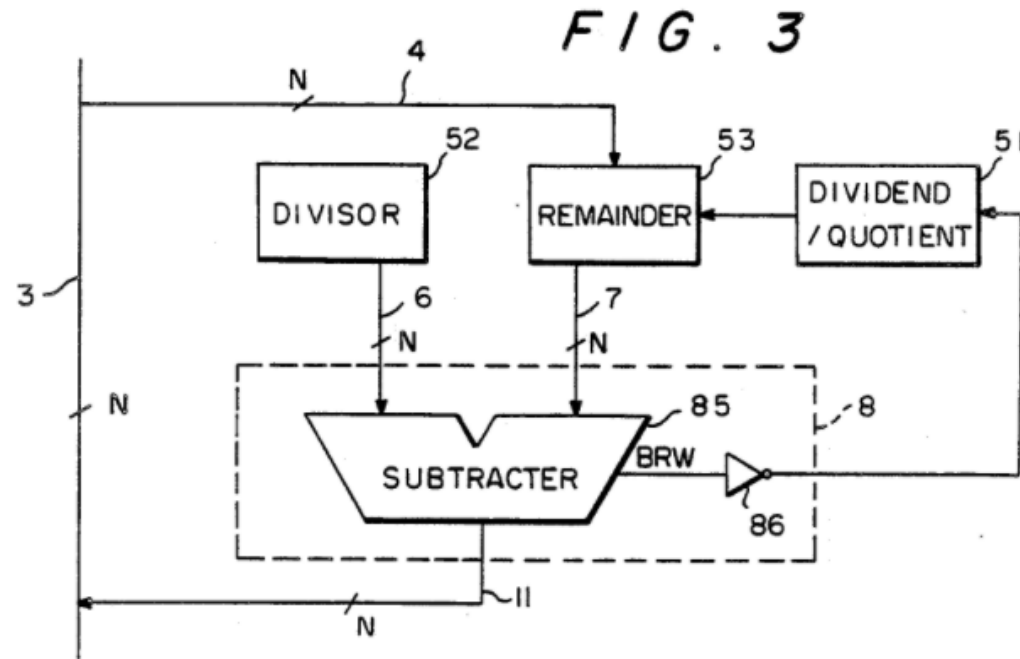
# Unsigned Integer Circuits – Divider

quotient

$$\begin{array}{r} 101 \\ 11 \overline{) 1111} \\ \underline{- 11} \phantom{00} \\ 0100 \\ \underline{- 11} \phantom{00} \\ 11 \\ \underline{- 11} \\ 0 \end{array}$$

divisor 11      dividend 1111

remainder 0



Yamahata, Hitoshi. "Integer division circuit provided with a overflow detector circuit." U.S. Patent No. 4,992,969. 12 Feb. 1991.

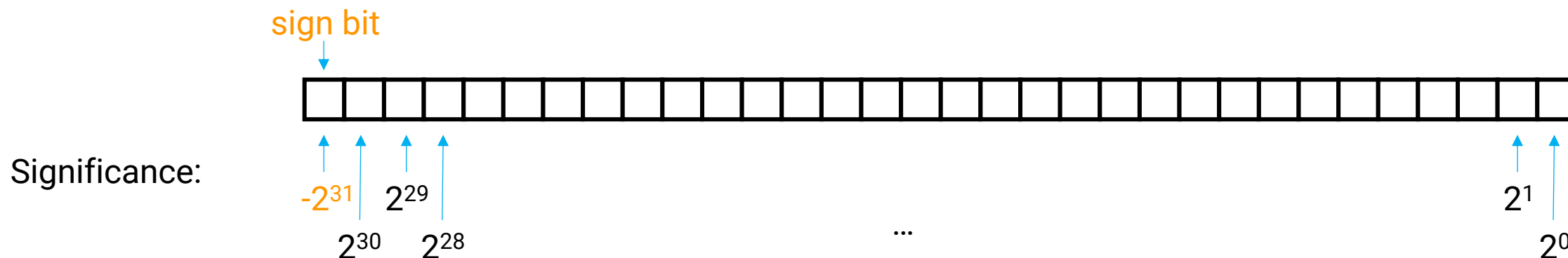
# Part 2

Signed Integers

# Signed Integer



- Signed INT32



- INT16, INT8, ...

- Example:

-32'd7 (=32'hff\_ff\_ff\_f9)  
8'b0100\_1101 (=8'hCD)

Signed Bit	Meaning
0	Positive
1	Negative

# Signed Integer

- Problem: What is the range of signed vs. unsigned integers?

For  $\text{UINT}_n$ :  $0 \sim 2^n - 1$

For  $\text{INT}_n$ :  $-2^{n-1} \sim 2^{n-1} - 1$

Numbers

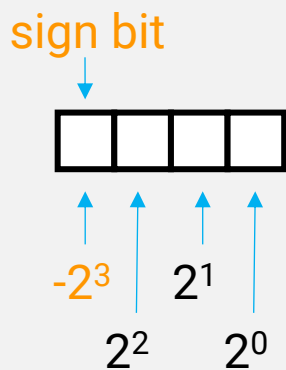
Arithmetic

Circuits

Here is the answer of “opposite number circuits”!



Example of Signed  $\text{INT}_4$



Binary Codeword	Unsigned $\text{INT}_4$	Signed $\text{INT}_4$
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Binary Codeword	Unsigned $\text{INT}_4$	Signed $\text{INT}_4$
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

# Signed Integer – Two Types of Shift

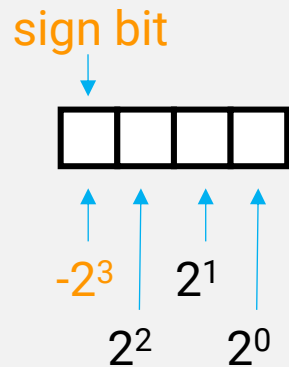
- Verilog HDL supports 2 types of shift:
- Logic Shift Operators ( $\ll$ ,  $\gg$ )
- Arithmetic Shift Operators ( $\lll$ ,  $\ggg$ )

Numbers

Arithmetic

Circuits

Example of Signed INT4



- **Logic shift**  $\gg$   $\ll$ : filling with zeros
- $3'b100 \gg 1'd1$  gives  $3'b010$
- $3'b101 \gg 1'd1$  gives  $3'b010$
- $3'b101 \ll 1'd2$  gives  $3'b100$

Not really stable rule:  $\ll 1$  : multiply 2 ;  $\gg 1$ : divided by 2

- **Arithmetic shift:**
  - $\lll$ : Shift left specified number of bits, filling with zero.
  - $\ggg$ : Shift right specified number of bits, fill with value of sign bit if expression is signed, otherwise fill with zero.

# Signed Integer – Two Types of Shift



- Verilog HDL supports 2 types of shift:
- Logic Shift Operators (<<, >>)
- Arithmetic Shift Operators (<<<, >>>)

Binary Codeword	Unsigned INT4	Signed INT4
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

4'b1110 >>> 1'd1 gives 4'b1111  
4'b0110 >>> 1'd1 gives 4'b0011

- **Logic shift** >> <<: filling with zeros
- 3'b100 >> 1'd1 gives 3'b010
- 3'b101 >> 1'd1 gives 3'b010
- 3'b101 << 1'd2 gives 3'b100

Not really stable rule: <<1 : multiply 2 ; >>1: divided by 2

- **Arithmetic shift:**
  - <<<: Shift left specified number of bits, filling with zero.
  - >>>: Shift right specified number of bits, fill with value of sign bit if expression is signed, otherwise fill with zero.

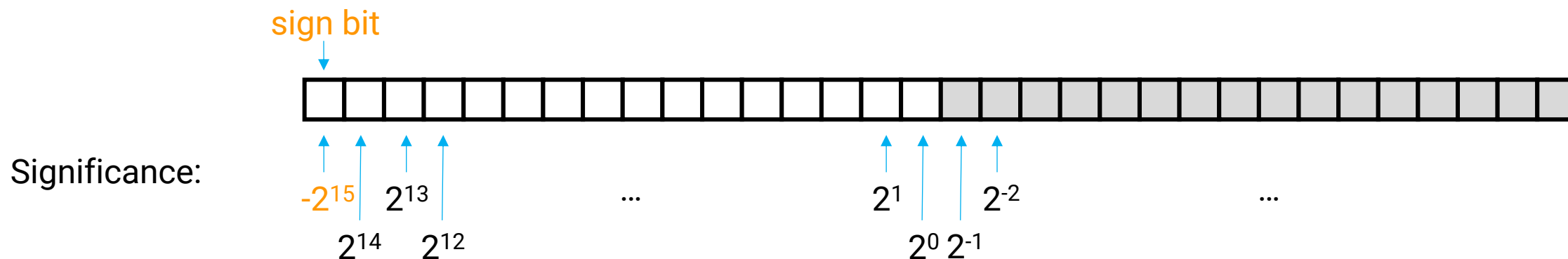
# Part 3 About Fraction

Fixed-Point

# Fixed-Point Number



- Fixed32



- Example:

$$(1.10)_2 = (1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2})_{10} = (1.50)_{10}$$

Signed Bit	Meaning
0	Positive
1	Negative



# Fixed-Point Number



- Arithmetic Just Works the Same Way!

Integer Arithmetic ↔ Fixed-Point Arithmetic

- Verilog HDL does not support fixed-point natively

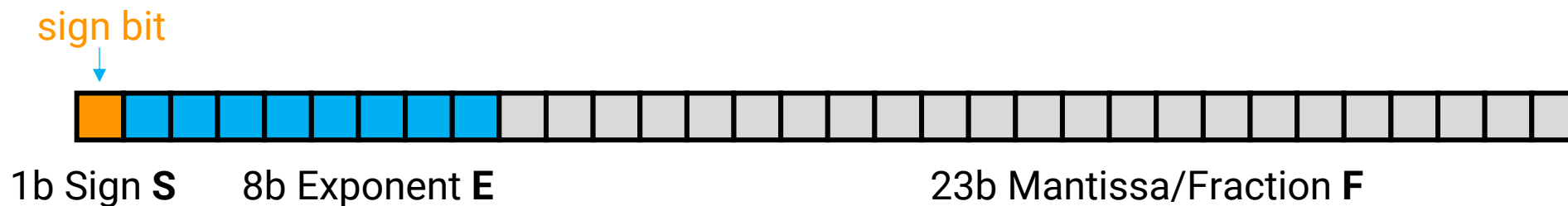
# Part 4

Floating-Point

# Floating-Point Number



- FP32

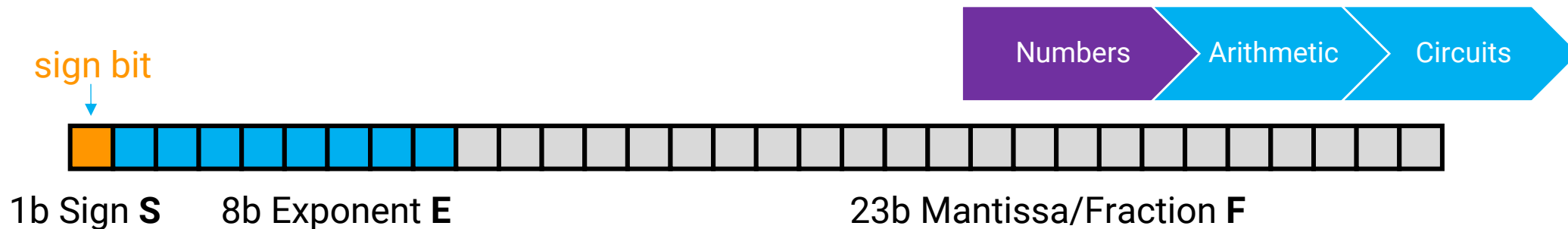


- Meaning:

Exponent	Fraction	Object	Value
0	0	0	
0	Nonzero	Denormalized number	$(-1)^S \times (0.F) \times 2^{E-B}$
Nonzero	Anything	Floating-point number	$(-1)^S \times (1.F) \times 2^{E-B}$
All "1"	0	infinity	
All "1"	Nonzero	NaN (not a number)	

# Floating-Point Number

- FP32



Exponent	Fraction	Object	Value
0	0	0	--
0	Nonzero	Denormalized number	$(-1)^S \times (0.F) \times 2^{E-B}$
Nonzero	Anything	Floating-point number	$(-1)^S \times (1.F) \times 2^{E-B}$
All "1"	0	infinity	--
All "1"	Nonzero	NaN (not a number)	--

S=1, E=0, F=0, what is the value?    -0=0

# Floating-Point Number

- FP32

Numbers

Arithmetic

Circuits

Exponent	Fraction	Object	Value
0	0	0	--
0	Nonzero	Denormalized number	$(-1)^S \times (0.F) \times 2^{1-B}$
Nonzero	Anything	Floating-point number	$(-1)^S \times (1.F) \times 2^{E-B}$
All "1"	0	infinity	--
All "1"	Nonzero	NaN (not a number)	--

Type	Sign	Exponent	Exponent bias	significand	total
Half (IEEE 754-2008)	1	5	15	10	16
Single	1	8	<b>B</b> 127	23	32
Double	1	11	1023	52	64
Quad	1	15	16383	112	128

$S=0, E=0,$

$F=23'b10000\_00000\_00000\_00000\_000$

what is its decimal value?

$$(0.1)_2 \times 2^{-126} = 0.5 \times 2^{-126}$$

# Floating-Point Number

- FP32

Numbers

Arithmetic

Circuits

Exponent	Fraction	Object	Value
0	0	0	--
0	Nonzero	Denormalized number	$(-1)^S \times (0.F) \times 2^{1-B}$
Nonzero	Anything	Floating-point number	$(-1)^S \times (1.F) \times 2^{E-B}$
All "1"	0	infinity	--
All "1"	Nonzero	NaN (not a number)	--

Type	Sign	Exponent	Exponent bias	significand	total
Half (IEEE 754-2008)	1	5	15	10	16
Single	1	8	<b>B</b> 127	23	32
Double	1	11	1023	52	64
Quad	1	15	16383	112	128

$S=0, E=8'b127,$

$F=23'b10000\_00000\_00000\_00000\_000$

what is its decimal value?

$$(1.1)_2 \times 2^{127-127} = 1.5$$

# Floating-Point Number

- Range



Exponent	Fraction	Object	Value
0	0	0	--
0	Nonzero	Denormalized number	$(-1)^S \times (0.F) \times 2^{1-B}$
Nonzero	Anything	Floating-point number	$(-1)^S \times (1.F) \times 2^{E-B}$
All "1"	0	infinity	--
All "1"	Nonzero	NaN (not a number)	--

Absolute Max: S=0, E=8'b1111\_1110, F=23'h7FFFFFFF:  $(1.11111111111111111111111111111111)_2 \times 2^{8'b1111_1110-8'b127}$   
=3.40282346639e+38

Absolute Min: S=0, E=8'b0000\_0001, F=23'h0000\_0001:  $(1.000000000000000000000001)_2 \times 2^{1-127}$

# Floating-Point Number

- Range



Exponent	Fraction	Object	Value
0	0	0	--
0	Nonzero	Denormalized number	$(-1)^S \times (0.F) \times 2^{1-B}$
Nonzero	Anything	Floating-point number	$(-1)^S \times (1.F) \times 2^{E-B}$
All "1"	0	infinity	--
All "1"	Nonzero	NaN (not a number)	--

Absolute Max: S=0, E=8'b1111\_1110, F=23'h7FFFFFFF:  $(1.111111111111111111111111)_2 \times 2^{8'b1111\_1110-8'b127}$   
=3.40282346639e+38

Absolute Min: S=0, E=8'b0000\_0001, F=23'h0000\_0000:  $(1.000000000000000000000000)_2 \times 2^{1-127}$   
=1.17549435082e-38

Denormalized:

Absolute Min: S=0, E=8'b0000\_0000, F=23'h0000\_0001:  $(0.000000000000000000000001)_2 \times 2^{1-127}$   
=1.40129846432e-45



# Floating-Point Number

Numbers

Arithmetic

Circuits

## • Add

```
123456.7 = 1.234567 * 10^5
101.7654 = 1.017654 * 10^2 = 0.001017654 * 10^5
```

Hence:

```
123456.7 + 101.7654 = (1.234567 * 10^5) + (1.017654 * 10^2)
                    = (1.234567 * 10^5) + (0.001017654 * 10^5)
                    = (1.234567 + 0.001017654) * 10^5
                    = 1.235584654 * 10^5
```

```
E=5;  F=1.234567      (123456.7)
+ E=2;  F=1.017654      (101.7654)
-----
E=5;  F=1.234567
+ E=5;  F=0.001017654 (after shifting)
-----
E=5;  F=1.235584654 (true sum: 123558.4654)
```

Round-off error

Actually, result is:  $e=5; s=1.235585$  (final sum: 123558.5)

Try by yourself:

$(E=5, F=1.234567) + (E=-3, F=9.876543) = ??$

```
E=5;  F=1.234567
+ E=-3; F=9.876543
```

```
E=5;  F=1.234567
+ E=5;  F=0.00000009876543 (after shifting)
```

```
-----
E=5;  F=1.23456709876543 (true sum)
E=5;  F=1.234567          (after rounding/normalization)
```

# Floating-Point Number

Numbers

Arithmetic

Circuits

- Subtract

Try by yourself:

$$(E=5, F=1.234571) - (E=5, 1.234567) = ??$$

```
E=5;  F=1.234571
- E=5;  F=1.234567
-----
E=5;  F=0.000004
E=-1; F=4.000000 (after rounding/normalization)
```

Change to normalized form of FP numbers

# Floating-Point Number



- Multiply:

```
| E=3;   F=4.734612
x E=5;   F=5.417242
-----
| E=8;   F=25.648538980104 (true product)
| E=8;   F=25.64854         (after rounding)
| E=9;   F=2.564854         (after normalization)
```

Exponent: Sum Operation

Mantissa: Multiply Operation

Don't forget normalization

- Divide:

Exponent: **Subtract** Operation

Mantissa: **Divide** Operation

Don't forget normalization

Q: What if normalized number multiplies  
denormalized number?

# Floating-Point Number

Numbers

Arithmetic

Circuits

## Incompleteness of Floating-Point Arithmetic

- May not associative:

$$1234.567 + 45.67844 = 1280.245$$

$$1280.245 + 0.0004 = 1280.245$$

but

$$45.67840 + 0.00004 = 45.67844$$

$$45.67844 + 1234.567 = 1280.246$$

- May not distributive:

$$1234.567 \times 3.333333 = 4115.223$$

$$1.234567 \times 3.333333 = 4.115223$$

$$4115.223 + 4.115223 = 4119.338$$

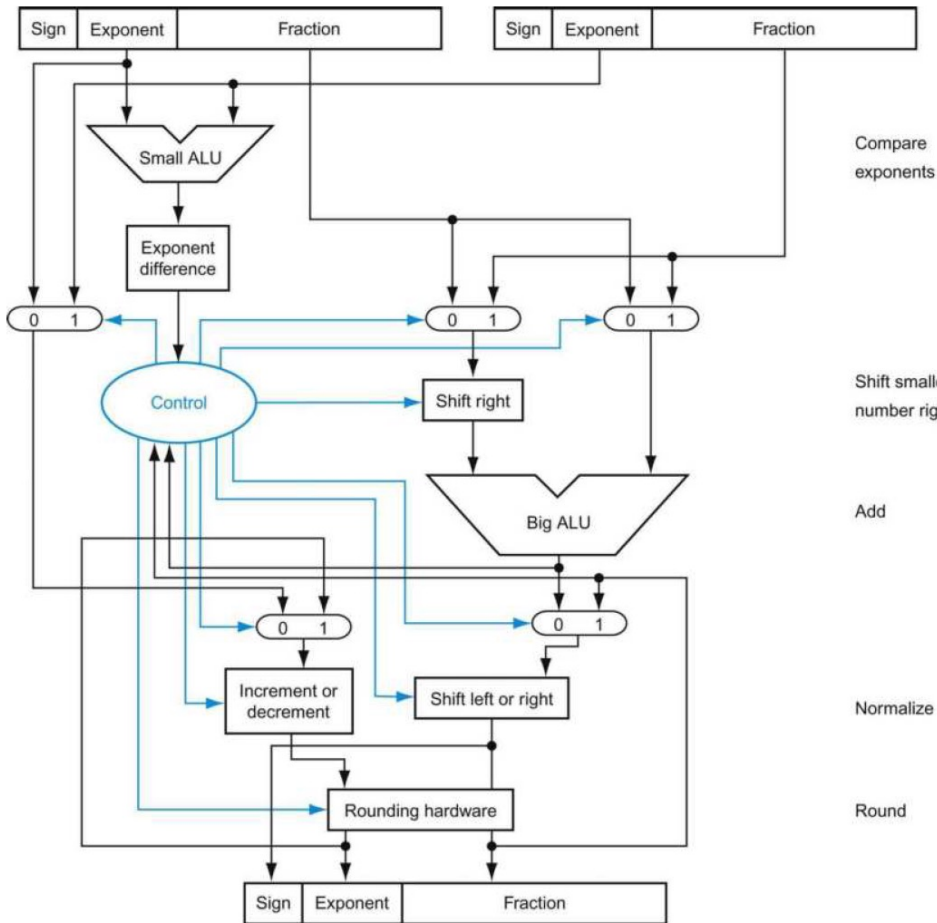
but

$$1234.567 + 1.234567 = 1235.802$$

$$1235.802 \times 3.333333 = 4119.340$$

# Floating-Point Number

## Adder:



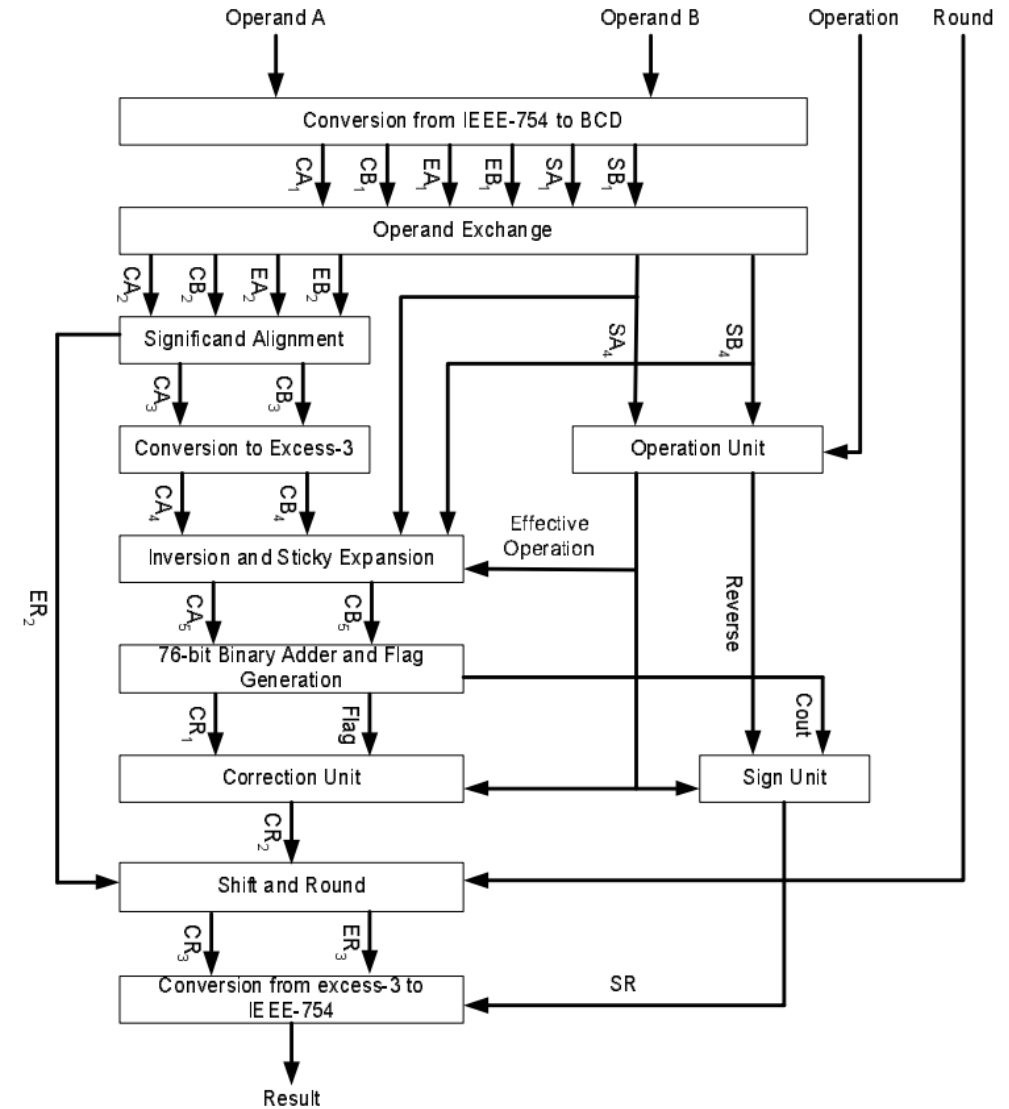
Numbers

Arithmetic

Circuits



北京大学  
PEKING UNIVERSITY



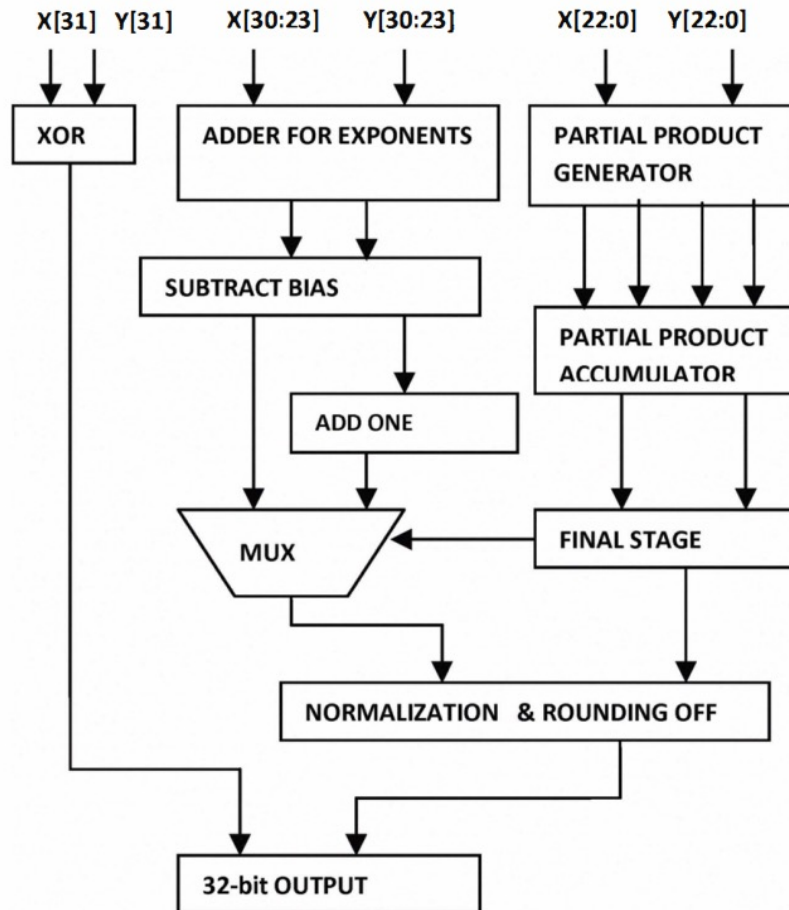
# Floating-Point Number

Numbers

Arithmetic

Circuits

**Multiply:**



Sign: XOR

Exponent: Add

Mantissa: Multiply

# Floating-Point Number

Numbers

Arithmetic

Circuits

Floating Point ALU supports +-\*/\*

**RISC-V floating-point assembly language**

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	fadd.s f0, f1, f2	$f0 = f1 + f2$	FP add (single precision)
	FP subtract single	fsub.s f0, f1, f2	$f0 = f1 - f2$	FP subtract (single precision)
	FP multiply single	fmul.s f0, f1, f2	$f0 = f1 * f2$	FP multiply (single precision)
	FP divide single	fdiv.s f0, f1, f2	$f0 = f1 / f2$	FP divide (single precision)
	FP square root single	fsqrt.s f0, f1	$f0 = \sqrt{f1}$	FP square root (single precision)
	FP add double	fadd.d f0, f1, f2	$f0 = f1 + f2$	FP add (double precision)
	FP subtract double	fsub.d f0, f1, f2	$f0 = f1 - f2$	FP subtract (double precision)
	FP multiply double	fmul.d f0, f1, f2	$f0 = f1 * f2$	FP multiply (double precision)
	FP divide double	fdiv.d f0, f1, f2	$f0 = f1 / f2$	FP divide (double precision)
	FP square root double	fsqrt.d f0, f1	$f0 = \sqrt{f1}$	FP square root (double precision)
Comparison	FP equality single	feq.s x5, f0, f1	$x5 = 1$ if $f0 == f1$ , else 0	FP comparison (single precision)
	FP less than single	flt.s x5, f0, f1	$x5 = 1$ if $f0 < f1$ , else 0	FP comparison (single precision)
	FP less than or equals single	fle.s x5, f0, f1	$x5 = 1$ if $f0 \leq f1$ , else 0	FP comparison (single precision)
	FP equality double	feq.d x5, f0, f1	$x5 = 1$ if $f0 == f1$ , else 0	FP comparison (double precision)
	FP less than double	flt.d x5, f0, f1	$x5 = 1$ if $f0 < f1$ , else 0	FP comparison (double precision)
	FP less than or equals double	fle.d x5, f0, f1	$x5 = 1$ if $f0 \leq f1$ , else 0	FP comparison (double precision)
Data transfer	FP load word	flw f0, 4(x5)	$f0 = \text{Memory}[x5 + 4]$	Load single-precision from memory
	FP load doubleword	fld f0, 8(x5)	$f0 = \text{Memory}[x5 + 8]$	Load double-precision from memory
	FP store word	fsw f0, 4(x5)	$\text{Memory}[x5 + 4] = f0$	Store single-precision from memory
	FP store doubleword	fsd f0, 8(x5)	$\text{Memory}[x5 + 8] = f0$	Store double-precision from memory

# Summary

- Memory
- Arithmetic Unit
  - Number Systems
    - Integer
    - Fixed-Point
    - Floating-Point
  - Arithmetic
  - Circuits & Implementation