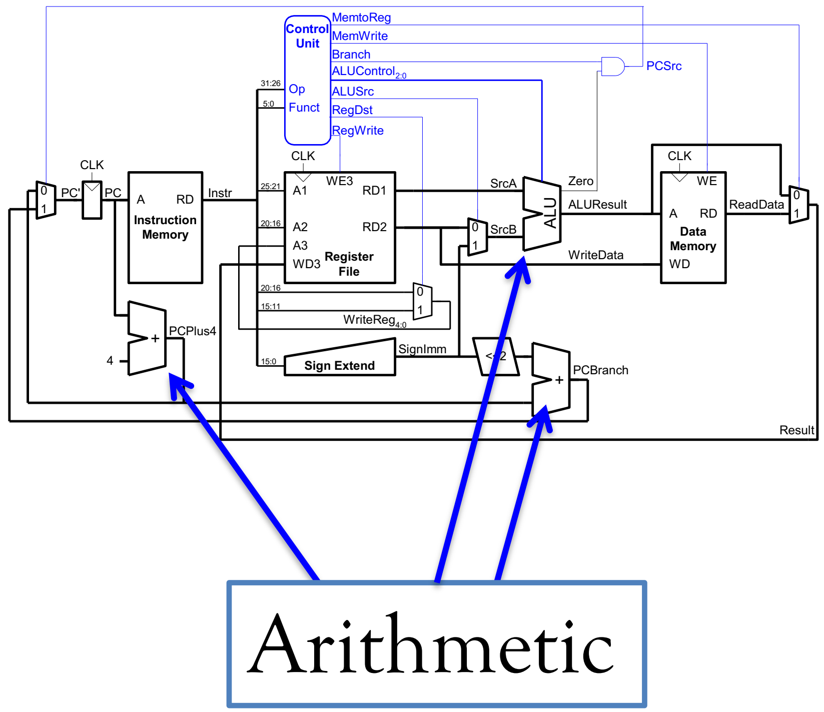
Arithmetic Circuits: Subtractors, Multipliers, ALU

Becker/Molitor, Chapter 9.2+9.3
Harris/Harris, Chapter 5.2

Jan Reineke
Universität des Saarlandes

Roadmap: Computer architecture



1. Combinatorial circuits: Boolean Algebra/Functions/Expressions/Synthesis
2. Number representations
3. Arithmetic Circuits: Addition, **Multiplication, Division, ALU**

4. Sequential circuits: Flip-Flops, Registers, SRAM, Moore and Mealy automata
5. Verilog

6. Instruction Set Architecture
7. Microarchitecture

8. Performance: RISC vs. CISC, Pipelining, Memory Hierarchy

Subtraction

As we have $-[b]=[\overline{b}]+1$ the difference $[a]-[b]$ is equal to the sum $[a]+[\overline{b}]+1$.

→ Derive subtractor circuit from adder circuit

→ combined adder/subtractor

Reminder: Two's complement

Lemma:

Let d be a fixed-point number and $\overline{d}$ the number obtained by flipping all bits ($0 \rightarrow 1$, $1 \rightarrow 0$) in d.

Then: $[\overline{d}]_2 + 1 = -[d]_2$.

Example: n = 2, k = 0:

d	000	001	010	011	100	101	110	111
$[d]_2$	0	1	2	3	-4	-3	-2	-1

Example: Subtraction

$[a]_2 = [0110]_2 = 6_{10}$

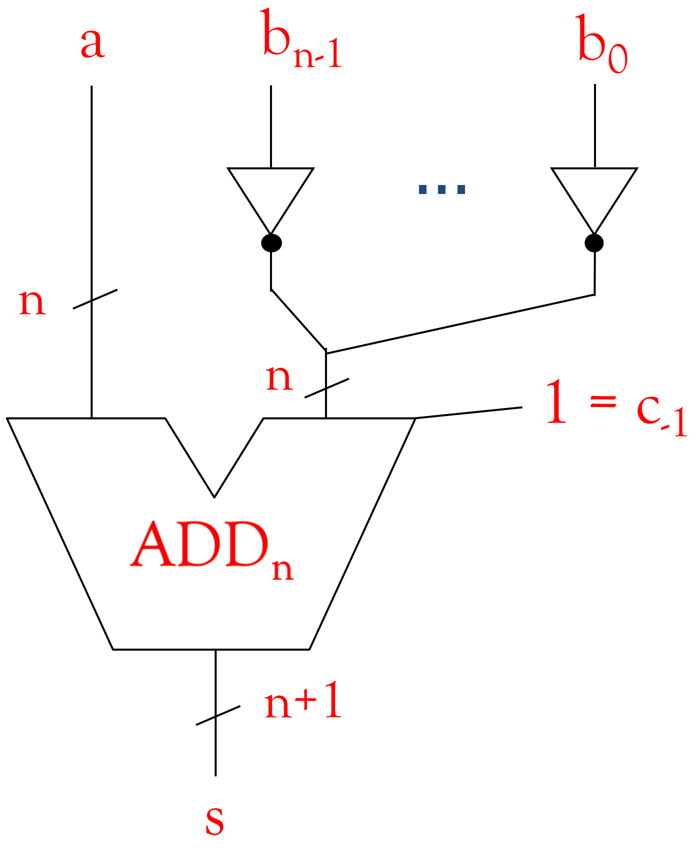
$[b]_2 = [0111]_2 = 7_{10}$

$[\overline{b}]_2 = [1000]_2 = -8_{10}$

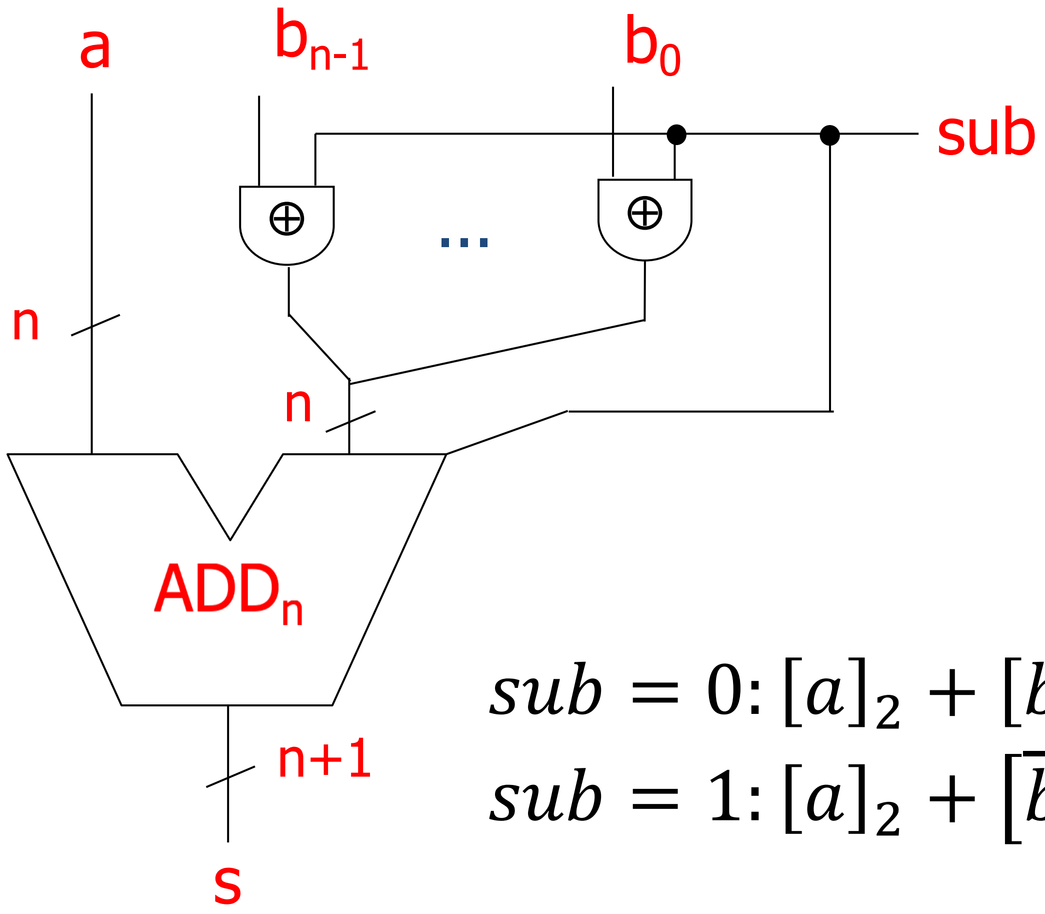
$$\begin{array}{cccc}
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
 & & & 1 \\
\hline
1 & 1 & 1 & 1
\end{array}$$

$$= (-1)_{10}$$

Schematic of a subtractor



Schematic of a combined adder/subtractor



$$b_i \oplus 0 = b_i$$
$$b_i \oplus 1 = \overline{b_i}$$

$$sub = 0: [a]_2 + [b]_2 + 0$$
$$sub = 1: [a]_2 + \left[\overline{b}\right]_2 + 1 = [a]_2 - [b]_2$$

Multiplier

Wanted: Circuit for the **multiplication** of two binary numbers $\langle a_{n-1}, \ldots, a_0 \rangle$, $\langle b_{n-1}, \ldots, b_0 \rangle$.

Example:

$(110) \cdot (101)$
$= 6_{10} \cdot 5_{10}$

$$\begin{array}{ccccc}
 & & 1 & 1 & 0 \\
 & 0 & 0 & 0 & \\
1 & 1 & 0 & & \\
\hline
1 & 1 & 1 & 1 & 0
\end{array}$$

$= 30_{10}$

Outputs of a multiplier

How many bits are required for the result?

$$\langle a\rangle \cdot \langle b\rangle \leq$$

$$\left(2^{n}-1\right)\cdot\left(2^{n}-1\right)=2^{2n}-2^{n+1}+1\leq 2^{2n}-1$$

Thus: **2n bits** are sufficient to represent the product of two n-bit binary numbers.

Multiplier

Definition: An **n-bit multiplier** is a circuit that computes the following function:

mul_n: $\mathbf{B}^{2n} \rightarrow \mathbf{B}^{2n}$ with

$mul_n(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0) = (p_{2n-1}, \ldots, p_0)$ with

$<p_{2n-1}, \ldots, p_0> = <a> \cdot <b>$

$$<a> \cdot <b> \overset{\text{Def. <.>}}{=} <a> \cdot \sum_{i=0}^{n-1} b_i \cdot 2^i = \sum_{i=0}^{n-1} \underbrace{<a> \cdot b_i \cdot 2^i}_{\text{partial product}}$$

The multiplication matrix

n partial products, each with $2n$ bits

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & a_{n-1}b_0 & a_{n-2}b_0 & \dots & a_1b_0 & a_0b_0 \\ 0 & 0 & \dots & 0 & a_{n-1}b_1 & a_{n-2}b_1 & a_{n-3}b_1 & \dots & a_0b_1 & 0 \\ \vdots & & & & \vdots & \vdots & \vdots & & & \vdots \\ 0 & a_{n-1}b_{n-1} & \dots & a_2b_{n-1} & a_1b_{n-1} & a_0b_{n-1} & 0 & \dots & 0 & 0 \end{pmatrix}$$

carry?

Implementation of the mult. matrix?

→ with n^2 AND gates

(and n^2 constants 0).

Fast addition of partial products

Goal: Fast addition of **n partial products** of length $2n$.

First approach:
Use carry-lookahead adders (CLAs).

Cost: $O(n^2)$

Depth:
- $O(n \cdot \log n)$ if partial products are summed up one by one linearly
- $O(\log^2 n)$ $(= O((\log n)^2))$ for tree-shaped summation of partial products

Brainstorming: Addition of partial products:

Can we do better than adding up the partial products in **$O(\log^2 n)$**?

Fast addition of **n partial products**

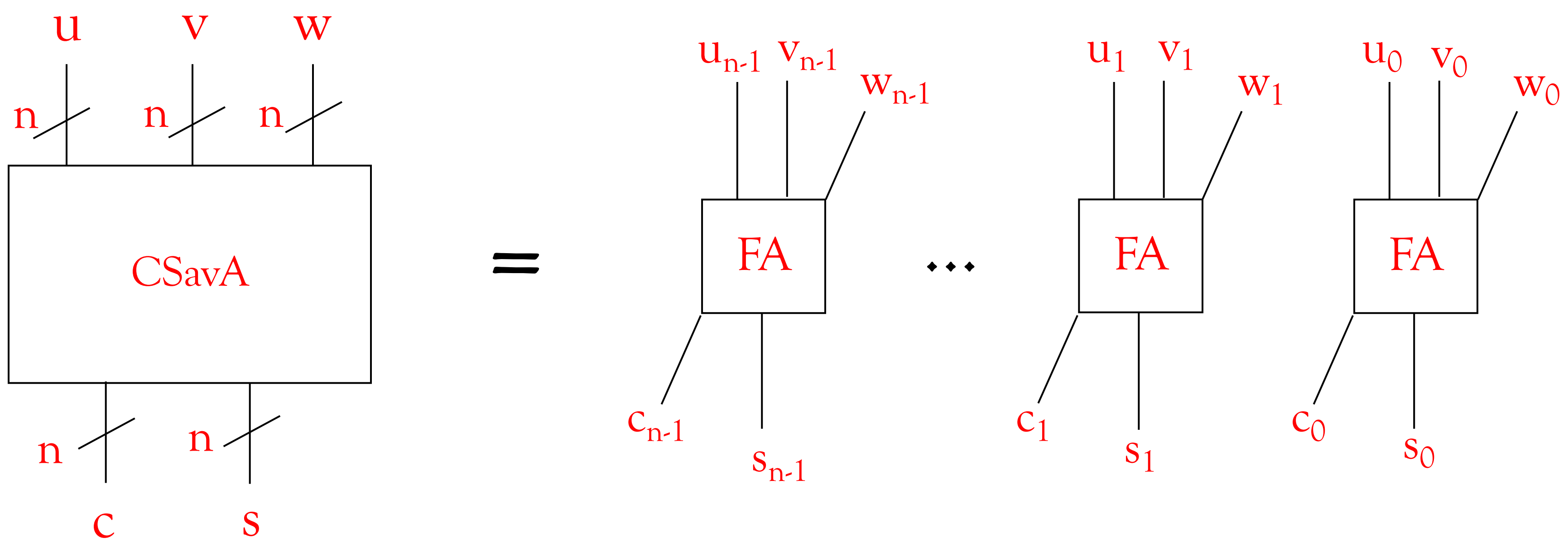
Use **carry-save adders.**

Reduction of three input values u, v, w into two output values s, c s.t. <u> + <v> + <w> = <s> + <c>.

$$\begin{array}{ccccccc}
 & u_{n-1} & u_{n-2} & \cdots & u_2 & u_1 & u_0 \\
 & v_{n-1} & v_{n-2} & & v_2 & v_1 & v_0 \\
 & w_{n-1} & w_{n-2} & & w_2 & w_1 & w_0 \\
\hline
c_{n-1} & c_{n-2} & c_{n-3} & \cdots & c_1 & c_0 & 0 \\
0 & s_{n-1} & s_{n-2} & & s_2 & s_1 & s_0
\end{array}$$

Solved by juxtaposition of independent full adders (**not** in a chain!)

Carry-save adder (also: 3:2 adder)

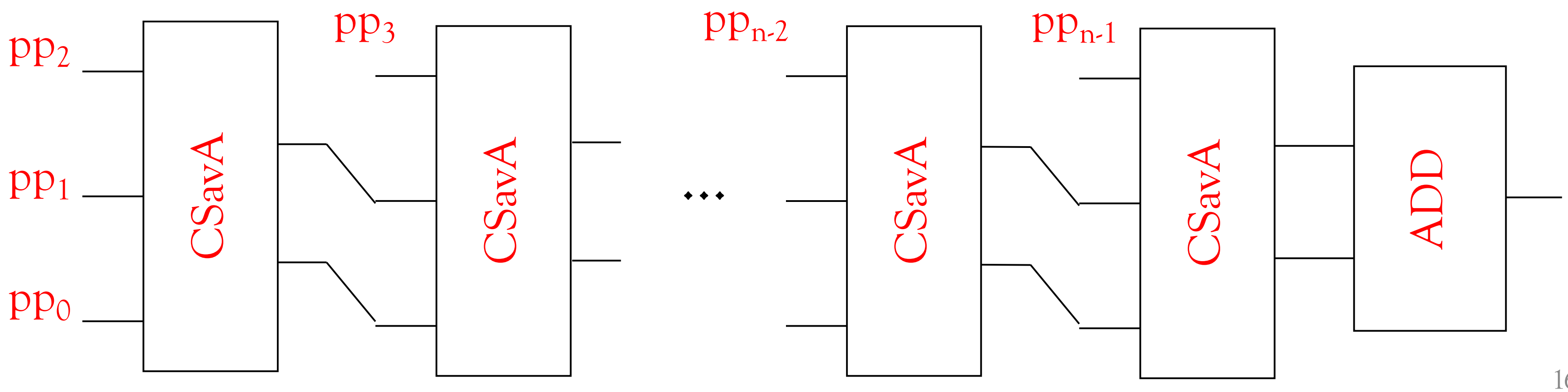


Cost: $C(CSavA^n) = n \cdot C(\mathit{FA}) = 5 \cdot n$
Depth: $\text{depth}(CSavA^n) = \text{depth}(\mathit{FA}) = 3$

1. Serial solution

- Cascade connection of n-2 CSavAs of length *2n*:

  → Combine n partial products into two 2n-bit words
- Add the last two 2n-bit words using a CLA
- *Cost*: $O(n^2)$, *Depth*: $O(n)$

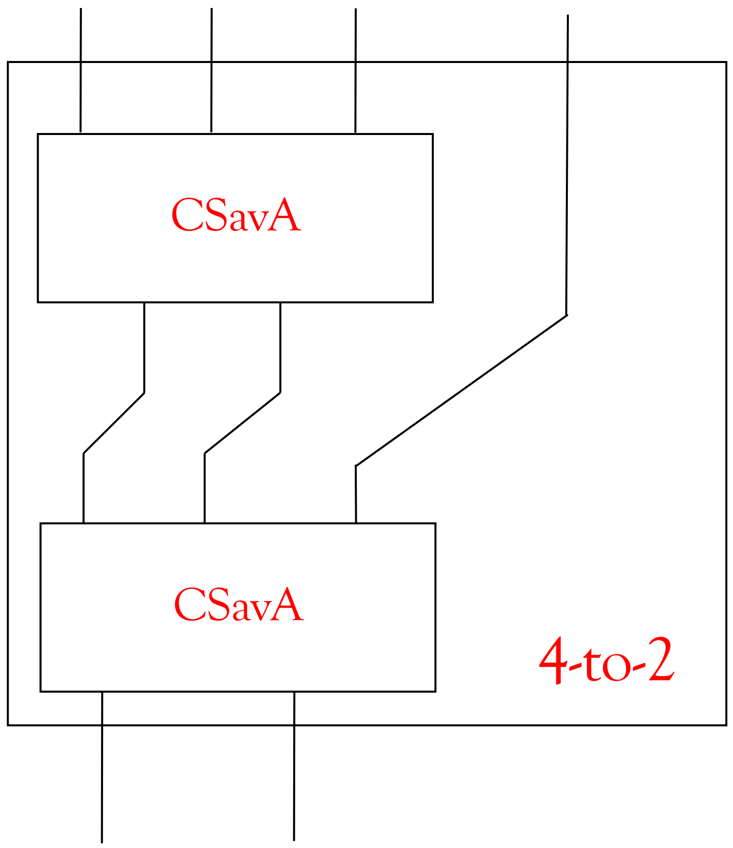


2. Tree-shaped solution

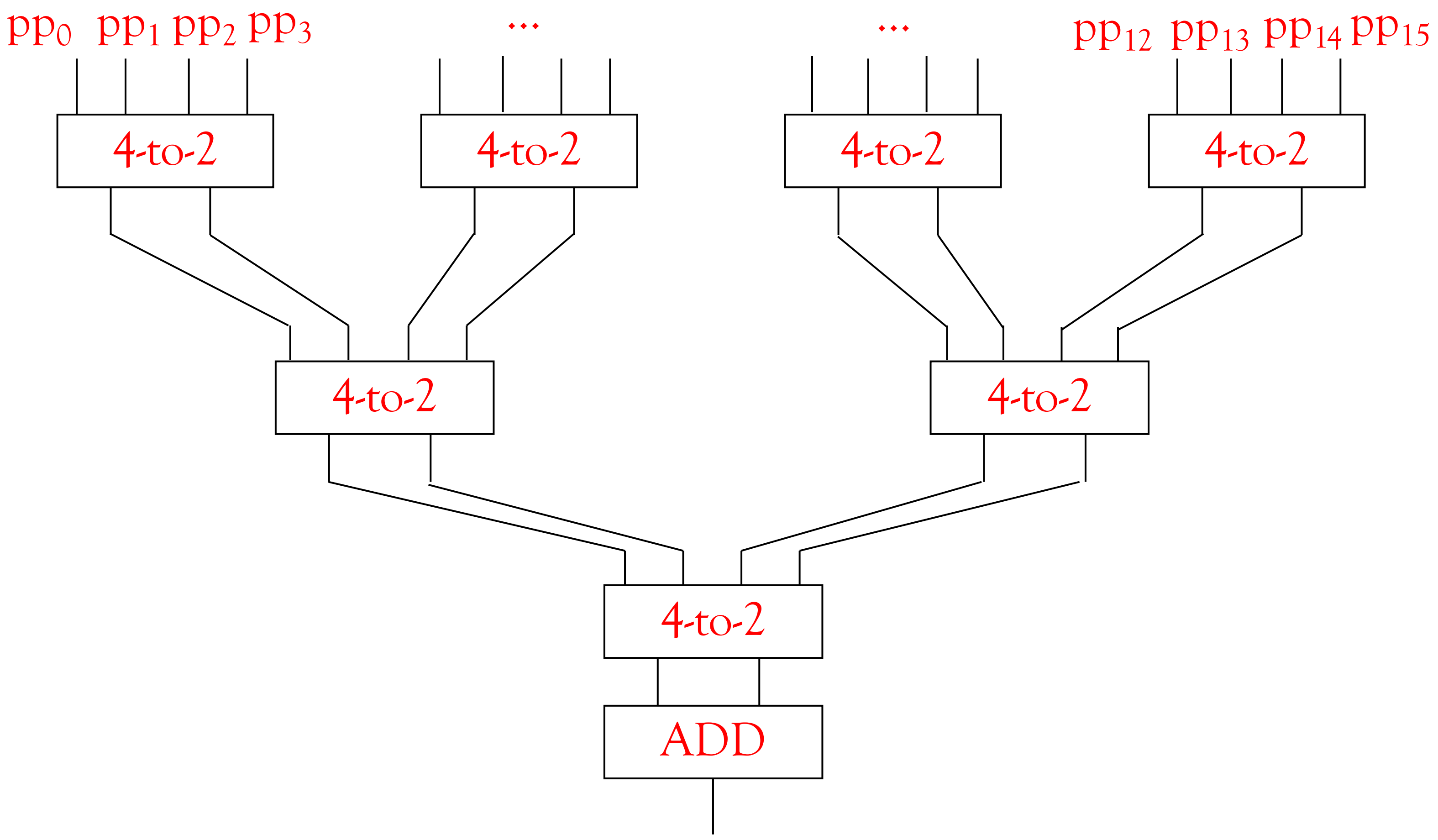
- Combine two carry-save adders to reduce four 2n-bit input words into two output words: 4-to-2 reducer
- Balanced binary tree of 4-to-2 reducers to summarize the n partial products using two 2n-bit words
- Addition of the two final 2n-bit words using a CLA

Cost: O(n^2)
Depth: O(log n)

4-to-2 reducer



Adder stage of the log-time multiplier for 16 bits



pp_0 pp_1 pp_2 pp_3
…
…
pp_{12} pp_{13} pp_{14} pp_{15}
4-to-2
4-to-2
4-to-2
4-to-2
4-to-2
4-to-2
4-to-2
ADD

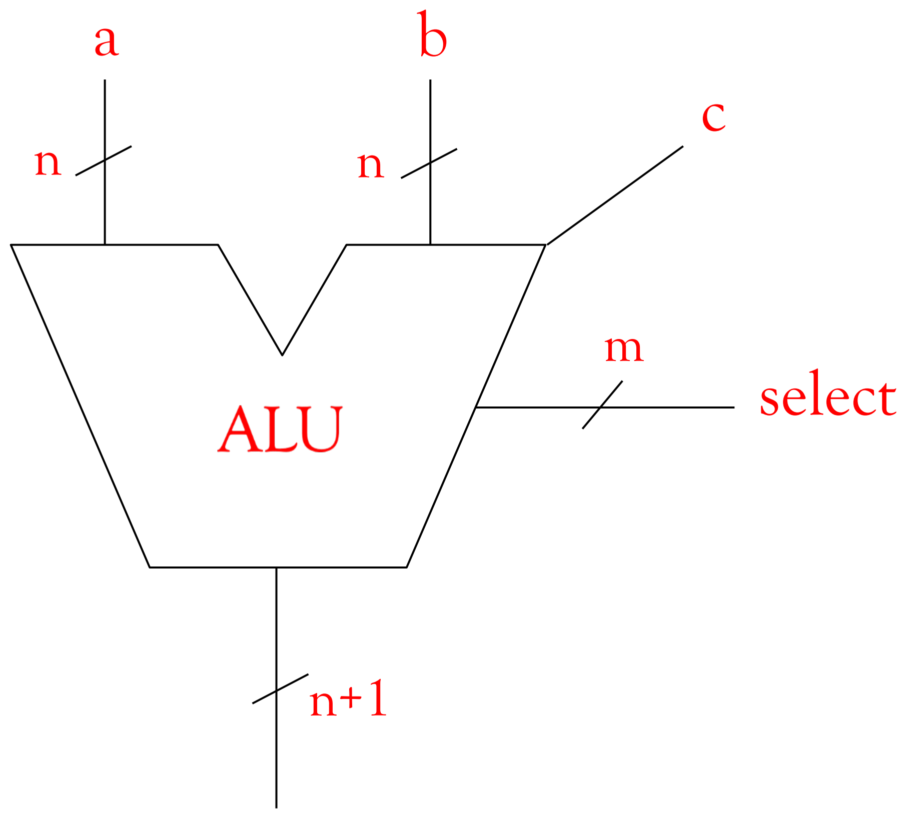
Construction of an ALU

ALU = arithmetic logic unit for
performing basic arithmetic and logic operations

Here: n-bit-ALU with:

- 2 n-bit operands a, b, carry-in c
- m-bit select input, which selects the function to execute
- (n+1)-bit output

Schematic of an n-bit ALU



Example ALU specification

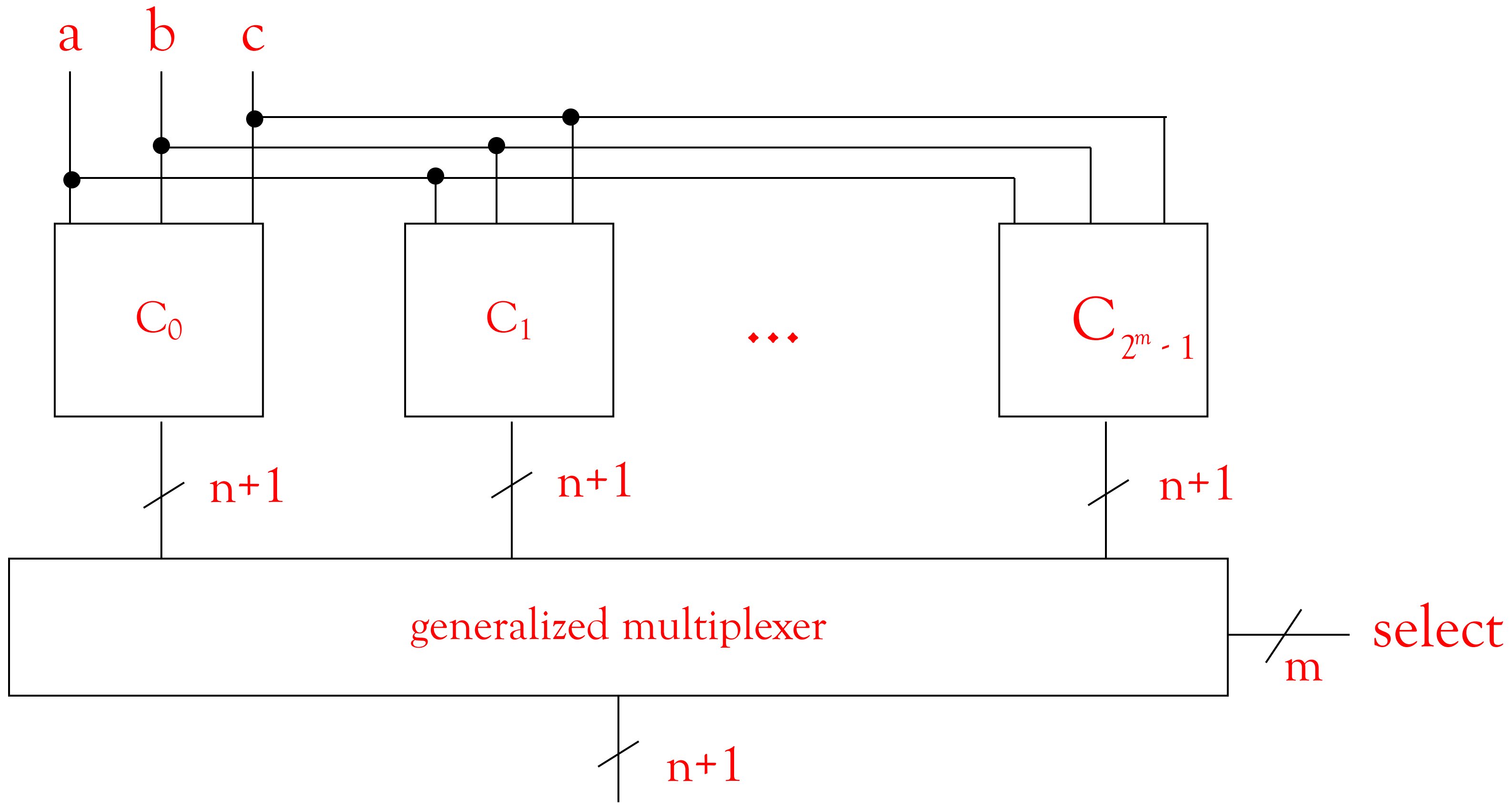
Here: 8 functions, i.e. 3-bit select input

Function number			ALU function	
s_2	s_1	s_0		
0	0	0	0 … 0	
0	0	1	[b] – [a]	
0	1	0	[a] – [b]	“Arithmetic”
0	1	1	[a] + [b] + c	
1	0	0	$a \oplus b = (a_{n-1} \oplus b_{n-1}, \ldots, a_0 \oplus b_0)$	
1	0	1	$a \vee b = (a_{n-1} \vee b_{n-1}, \ldots, a_0 \vee b_0)$	“Logic”
1	1	0	$a \wedge b = (a_{n-1} \wedge b_{n-1}, \ldots, a_0 \wedge b_0)$	
1	1	1	1 … 1	

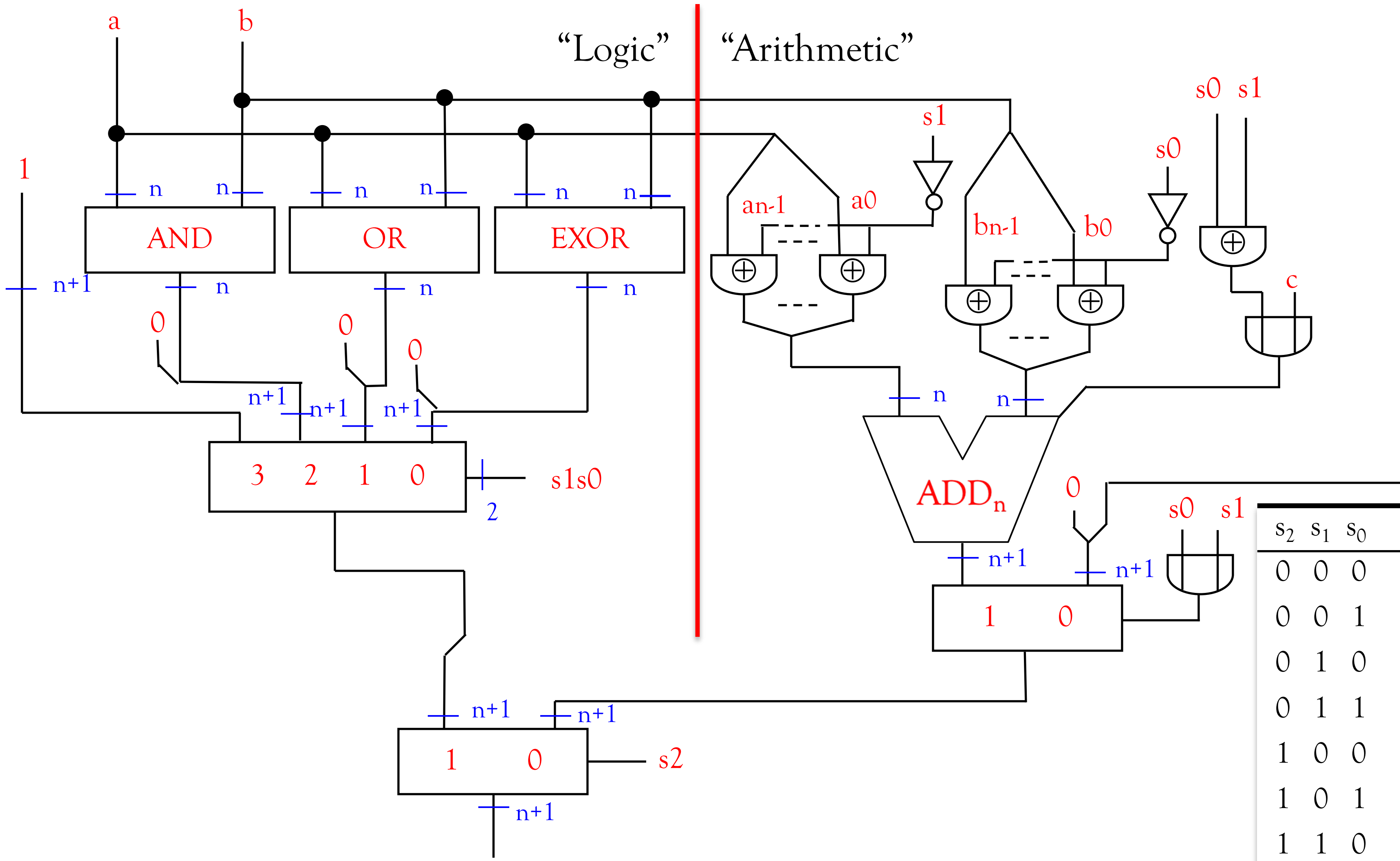
Possible implementations of an ALU

1. **Possibility:**
   Implement f_0, ..., $f_{2\wedge m-1}$ separately via circuits C_i for f_i; then select correct output via generalized multiplexer
   (see upcoming figure)
2. **Possibility:**
   Shared circuit for similar functions
   (see upcoming figure)

1. Possible implementation

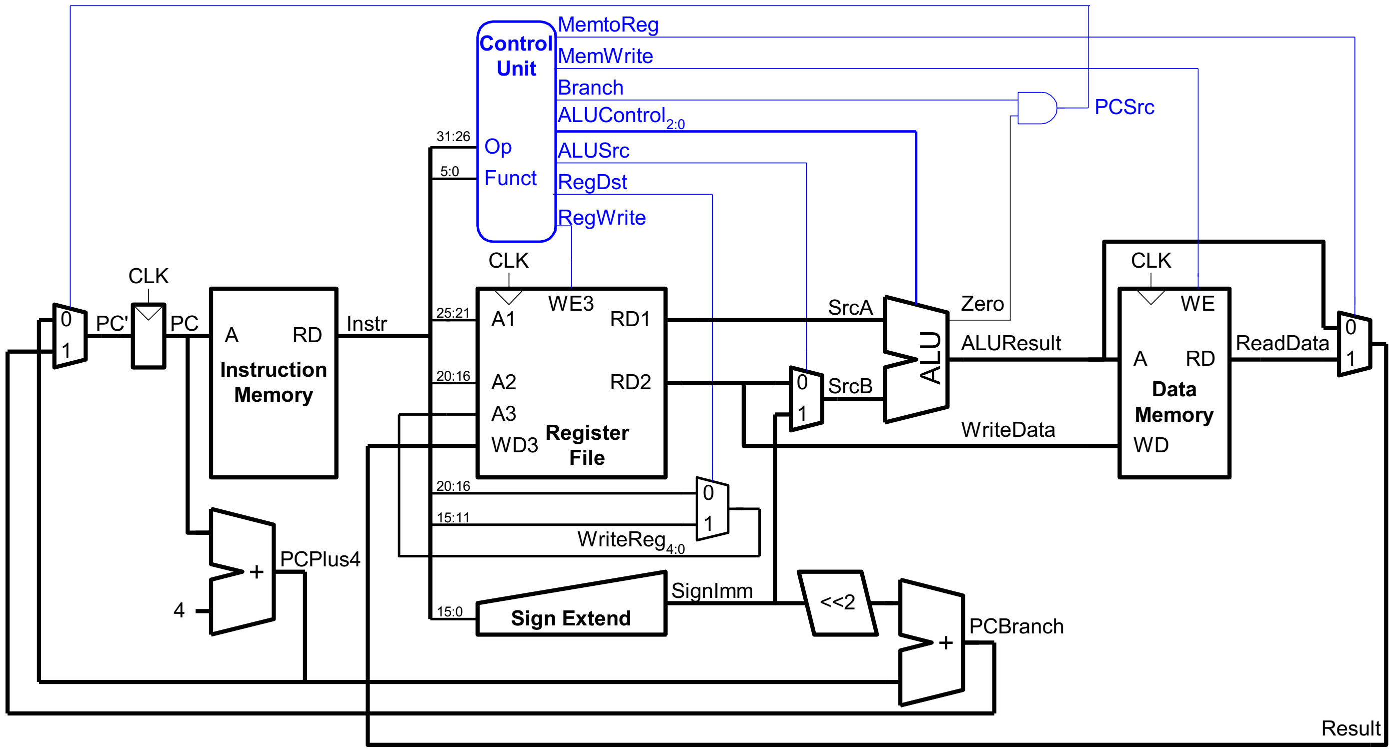


2. Possible implementation



s_2	s_1	s_0	Function
0	0	0	0 … 0
0	0	1	[b] – [a]
0	1	0	[a] – [b]
0	1	1	[a] + [b] + c
1	0	0	$a \oplus b$
1	0	1	$a \vee b$
1	1	0	$a \wedge b$
1	1	1	1 … 1

Outlook: Datapath and instruction execution



Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
PCSrc
31:26
5:0
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
A
RD
Data Memory
WD
ReadData
20:16
15:11
WriteReg4:0
15:0
Sign Extend
SignImm
<<2
PCBranch
PCPlus4
4
+
Result
0
1