ALUs and Data Paths

Subtitle: How to design the data path of a processor.

Lecture overview

- General Data Path Design
- Design of a multifunction ALU

Design of ALUs and Data Paths

- Objective: Design a General Purpose Data Path such as the datapath found in a typical computer.
- A Data Path Contains:
  - Registers – general purpose, special purpose
  - Execution Units capable of multiple functions

ALU Operations

- Add (A+B)
- Add with Carry (A+B+Cin)
- Subtract (A-B)
- Subtract with Borrow (A-B-Cin)
- [Subract reverse (B-A)]
- [Subract reverse with Borrow (B-A-Cin)]
- Negative A (-A)
- Negative B (-B)
- Increment A (A+1)
- Increment B (B+1)
- Decrement A (A-1)
- Decrement B (B-1)
- Logical AND
- Logical OR
- Logical XOR
- Not A
- Not B
- A
- B
- Multiply Step or Multiply
- Divide Step or Divide
- Mask
- Conditional AND/OR (uses Mask)
- Shift
- Zero

A High Level Design

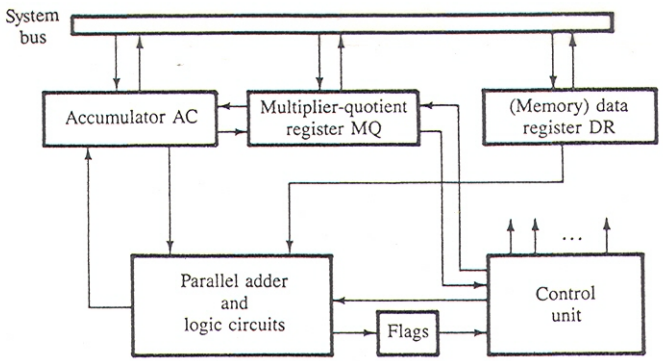


FIGURE 3.59
Structure of a basic fixed-point ALU.

The use typically made of these registers may be defined concisely as follows:

Addition:	$AC \leftarrow AC + DR$
Subtraction:	$AC \leftarrow AC - DR$
Multiplication:	$AC.MQ \leftarrow DR \times MQ$
Division:	$AC.MQ \leftarrow MQ \div DR$
AND:	$AC \leftarrow AC \wedge DR$
OR:	$AC \leftarrow AC \vee DR$
EXCLUSIVE-OR:	$AC \leftarrow AC \oplus DR$
NOT:	$AC \leftarrow \overline{AC}$

The AMD 2901 Bit Slice ALU

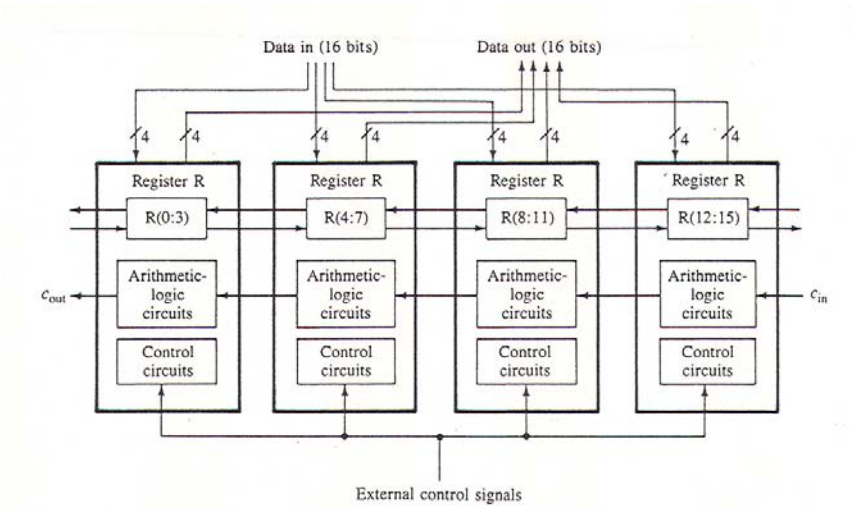
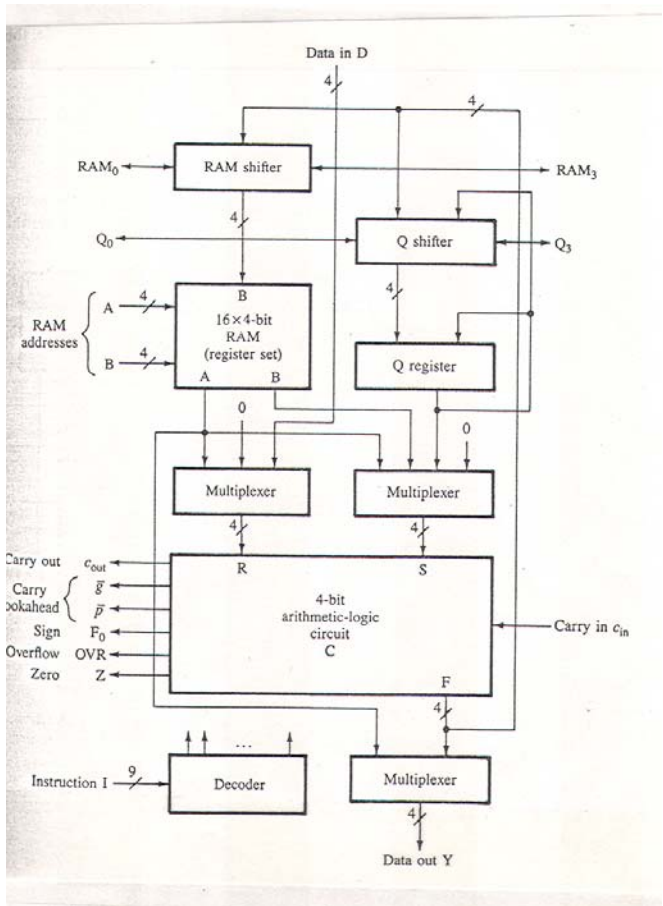


FIGURE 3.60
A 16-bit bit-sliced ALU composed of four 4-bit slices.



The Architecture

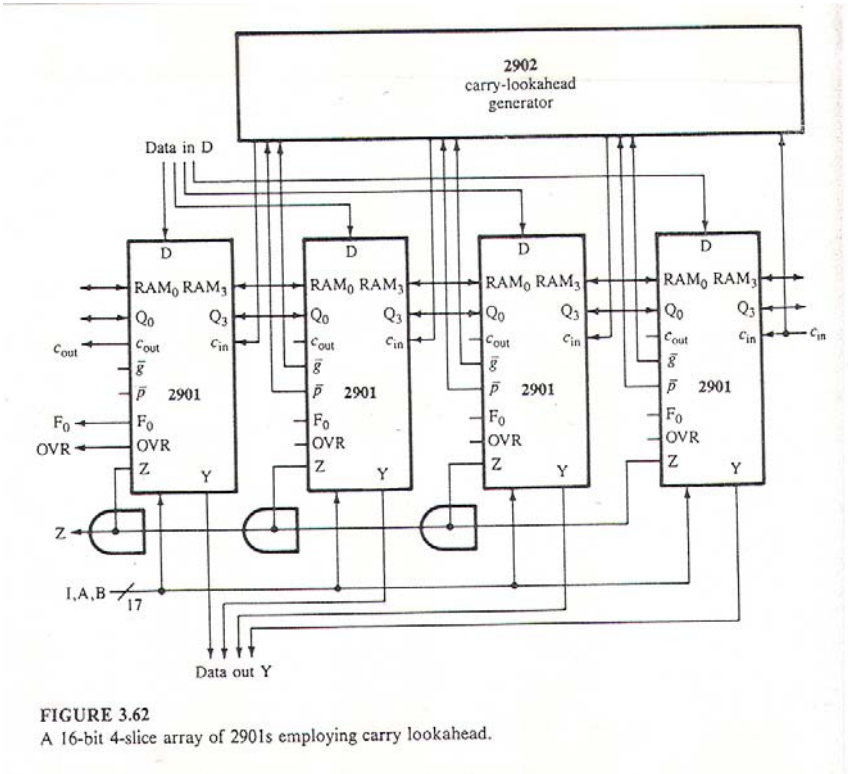


FIGURE 3.62
A 16-bit 4-slice array of 2901s employing carry lookahead.

ALU sources

$I_S = I_0I_1I_2$	R	S
000	RAM(A)	Q
001	RAM(A)	RAM(B)
010	0	Q
011	0	RAM(B)
100	0	RAM(A)
101	D	RAM(A)
110	D	Q
111	D	0

$I_F = I_3I_4I_5$	ALU function
000	$R + S + C_{in}$
001	$S - R - \overline{C}_{in}$
010	$R - S - \overline{C}_{in}$
011	$R \vee S$ (OR)
100	$R \wedge S$ (AND)
101	$\overline{R} \wedge S$ (COMPLEMENT-AND)
110	$R \oplus S$ (EXCLUSIVE-OR)
111	$\overline{R \oplus S}$ (EXCLUSIVE-NOR)

ALU destinations and shifts

$I_D = I_6I_7I_8$	Y	RAM(B)	Q
000	F	—	F
001	F	—	—
010	RAM(A)	F	—
011	F	F	—
100	F	F ÷ 2	Q ÷ 2
101	F	F ÷ 2	—
110	F	2 × F	2 × Q
111	F	2 × F	—

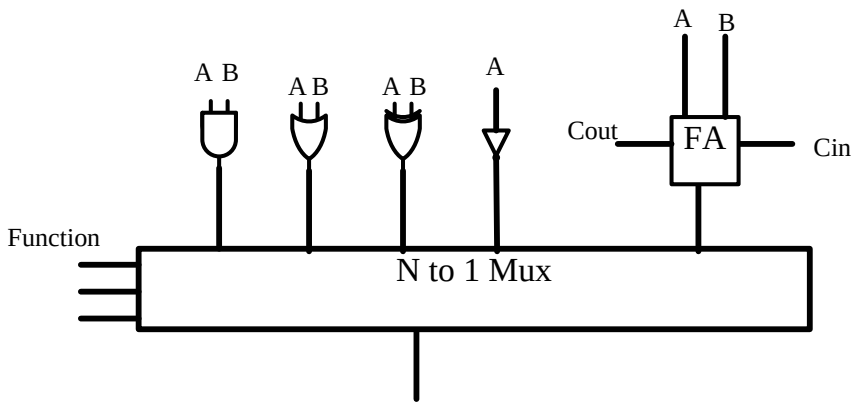
Arithmetic Logic Circuits

- The Brute Force Approach

- A more modern approach

Arithmetic Logic Circuits

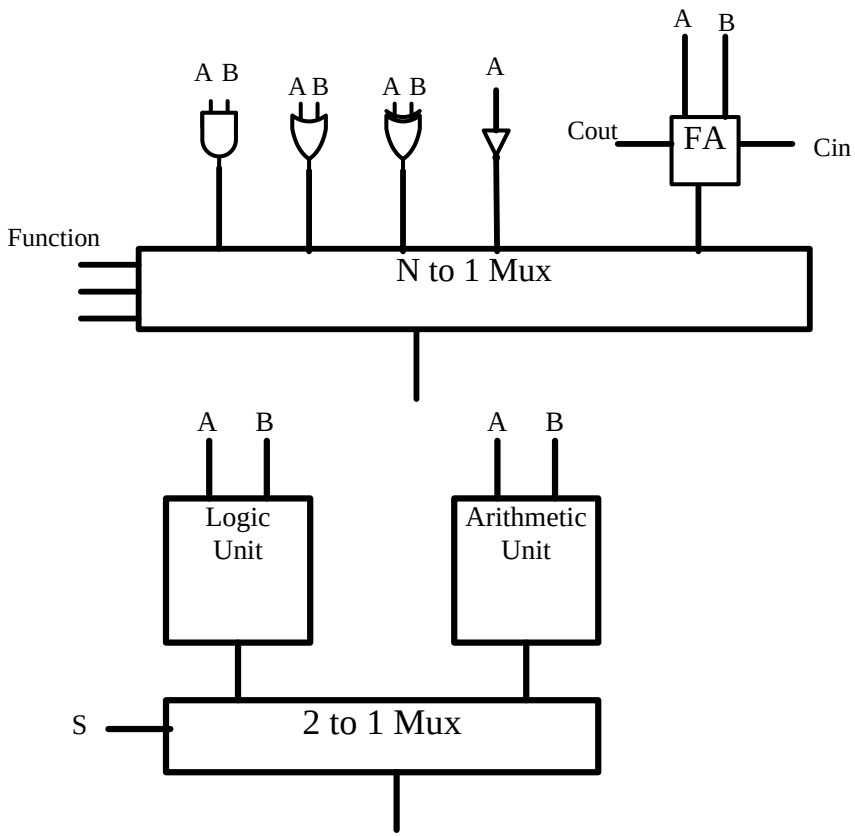
- The Brute Force Approach



- A more modern approach

Arithmetic Logic Circuits

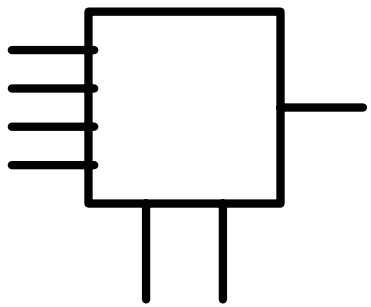
- The Brute Force Approach



- A more modern approach

A Generic Function Unit

- Desire a generic functional unit that can perform many functions

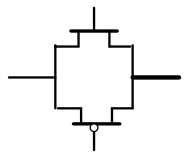


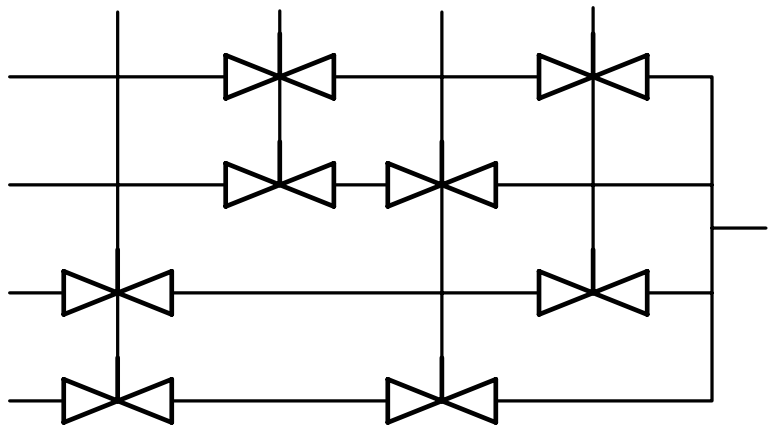
- A 4-to-1 mux will perform all basic logic functions of 2 inputs

Low level implementation

At the implementation
Level the design can be
With transmission gates
Very important for VLSI
implementation

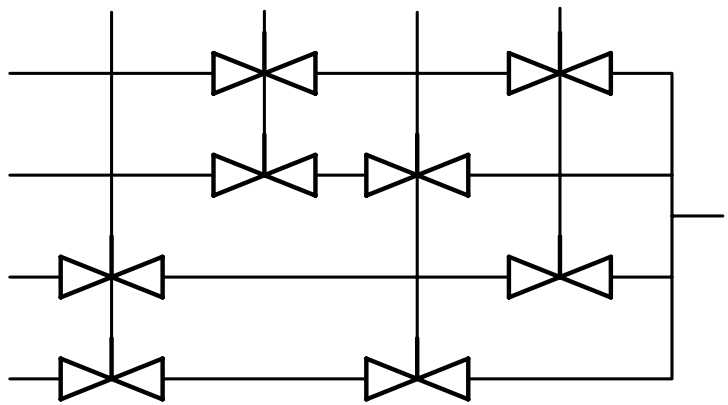
Implementation has a total
Of 16 transistors.





Low level implementation

- An implementation in pass gates (CMOS)
- When the control signal is a ‘1’ the value will be transmitted
- Otherwise it is an open switch



A	B	A’	B’	G(A,B)	AB	A+B	AxorB
0	0	1	1	G0	0	0	0
0	1	1	0	G1	0	1	1
1	0	0	1	G2	0	1	1
1	1	0	0	G3	1	1	0

Lets look at Binary Addition

- We can use this generic function unit construct a generic ALU.
- For Binary Addition consider the following:
  - SUMi = Ai xor Bi xor Ci
  - Ci+1 = Ai Bi + Ai Ci + Bi Ci

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

An Alternative - Define two signals

- Equations for P and K
- Pi = Ai xor Bi
- Ki = A' B'

P	K	A	B	Cin	Sum	Cout
0	1	0	0	0	0	0
0	1	0	0	1	1	0
1	0	0	1	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	1	0
1	0	1	0	1	0	1
0	0	1	1	0	0	1
0	0	1	1	1	1	1

- Now can reform equations into functions of P and K

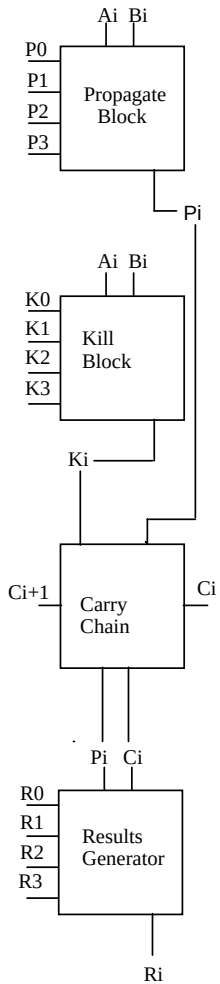
New functions

- Using these definitions of P and K
- SUMi = Pi xor Ci
- Ci+1 = Pi Ci + Pi' Ki'
- = Pi Ci + AB

- You can use the generic functional blocks to generate P and K and then select the correct function for final output

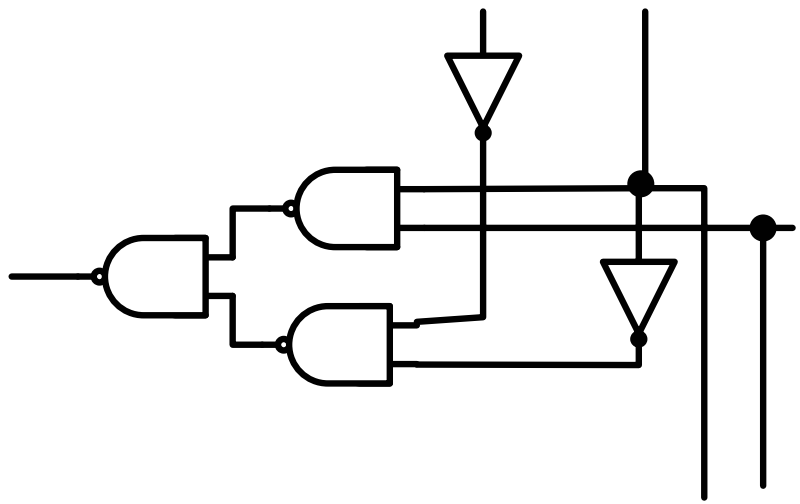
A bit slice of the ALU

- Slice starts out with a generic unit which can produce any function of inputs Ai and Bi to produce P
- Need another to produce K
- And a 3rd to generate the result
- And also need a dedicated unit to compute the carry out, the Ci+1 term



Generation of the carry out

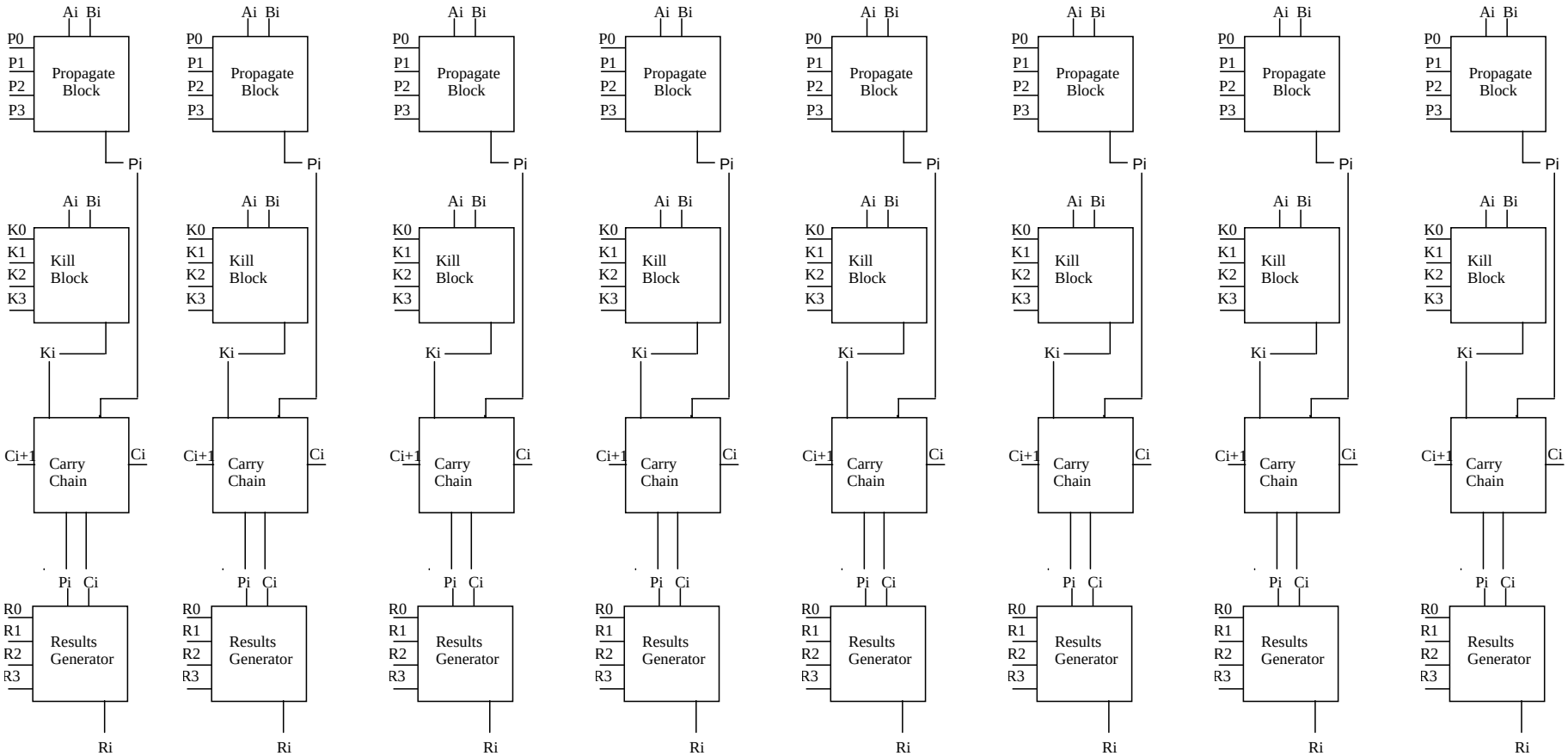
- The carry chain is the critical path for arithmetic operations.
- A simple ripple carry circuit is shown here for the slice
- Actual implementation depend on the technology in which implemented.



Carry chain implementation

- CMOS – Manchester carry chain using precharge pulldown logic works well.
- ECL – Carry look-ahead circuitry works well as ECL allows for large fan-in wired OR gates.

Multibit implementation



Function codes

A	B	A'	B'	G	0	NOR	A'B	A'	AB'	B'	XOR	NAND	AND	XNOR	B	A'+B	A	A+B'	OR	1
0	0	1	1	G_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	G_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	1	G_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	G_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

The G values for the various logic functions
By setting the value of the G inputs the output is the
Corresponding logic function of the data which comes in
On the select inputs.

Binary Subtraction

- Much like addition but now choose
  - P = A xnor B
  - K = A B'
- Diff
  - D = A xor B xor Bin
  - = A xnor B xnor Bin
- Borrow Out
  - Bout=P Bin + P' K'

P	K	A	B	Bin	Diff	Bout
1	0	0	0	0	0	0
1	0	0	0	1	1	1
0	0	0	1	0	1	1
0	0	0	1	1	0	1
0	1	1	0	0	1	0
0	1	1	0	1	0	0
1	0	1	1	0	0	0
1	0	1	1	1	1	1

The codes to have the ALU work

- Consider as we go to the sliced ALU
- Logic function done in the P generic unit
- Math used all the blocks

Function	P	K	R	Cin
A	12	---	12	---
A and B	8	--	12	---
OR	14	---	12	---
A + B + Cin	6	1	6	Cin
A+B	6	1	6	0
Incr A	12	3	6	1
A – B	9	4	9	0 or Bin

Multibit implementation