0.7.3 Logical instructions

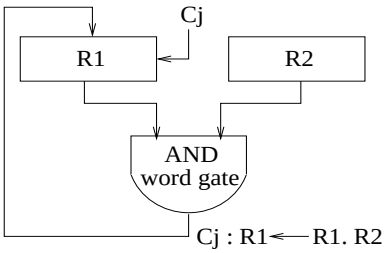

Figure 18: Logical micro-operation implementation

Implementation of logical microoperation is shown in Figure 18.

Figure 19 displays implementation of logical macro instructions of type Z $\leftarrow$ X op Y, where X, Y, Z are registers. Here, operators (op) are $and/or/xor/not$.

The not operates as Z $\leftarrow$ X$'$.

Control C1 and C2 of Figure 19(a) is received from instruction decoder, and

$$Z = C1'C2'\ (X.Y) + C1'C2\ (X+Y) + C1C2'\ (X \oplus Y) + C1C2\ X'.$$

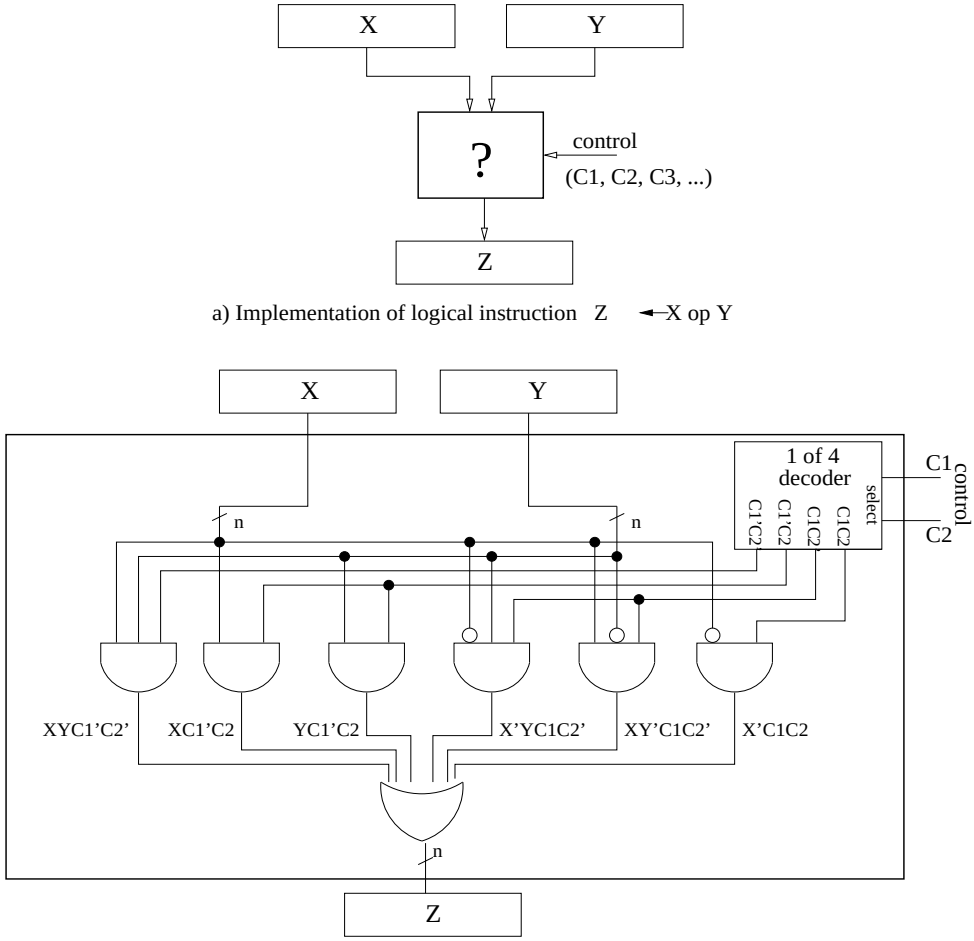

Figure 19: Logical macro-operation implementation

The AND and OR gates of Figure 19(b) are word gates.

0.7.4 Input/output instructions

Input/output instruction basically is a data movement operation (register-to-register transfer).

The very primitive input/output instructions are

IN xx
OUT yy

IN transfers data from input buffer to AC, whereas, OUT transfers data from AC to output buffer (Figure 22).

Activation of control signal C_{in} sets the path from input buffer to AC.

Similarly, C_{out} activates means a path is set from AC to output buffer.

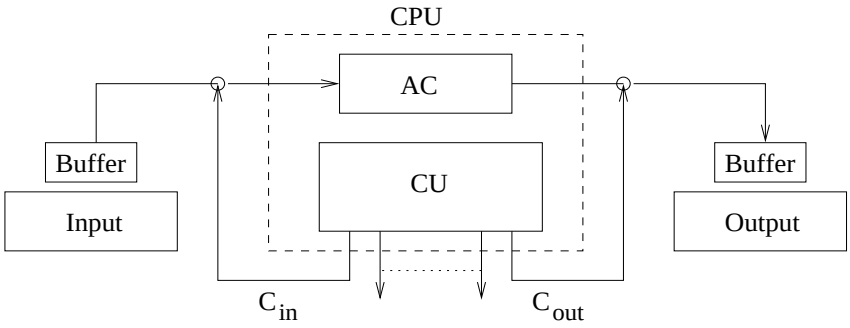

Figure 22: Input-Output instruction implementation

0.8 Arithmetic Instruction Implementations

Arithmetic instruction: a set of data movement and arithmetic micro-operations.

0.8.1 Addition and subtraction

In 2's complement, addition and subtraction micro-operations can be implemented with adder only (Figure 23).

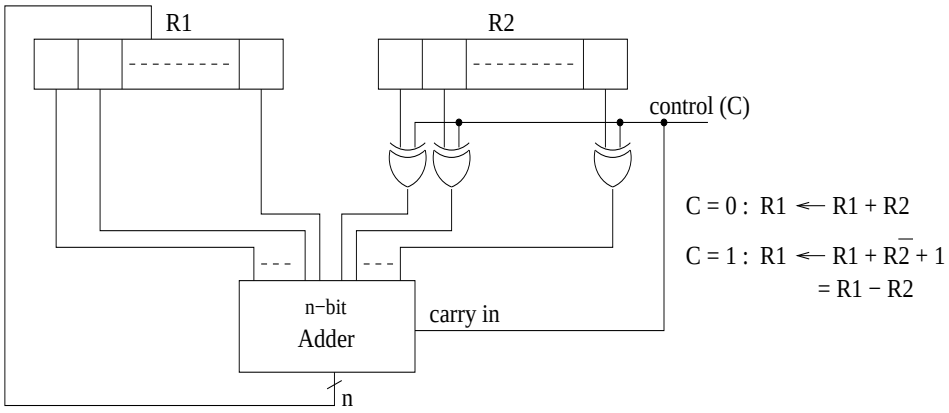

Figure 23: Adder/subtractor

Other arithmetic operations - increment, decrement, ... are implemented with multiple micro-operations and adder/subtractor. For example,

1. Increment R1 implies R2 ← 1 and then R1 ← R1 + R2 (C=0),
2. Decrement R1 implies R2 ← 1 and then R1 ← R1 - R2 (C=1),

Design adders

Binary adder Truth table of adder is formed and then logic optimization is done with Karnaugh-map (say)o. A binary adder (Figure 24) computes sum

$$z_i = x_i \oplus y_i \oplus c_{i-1}, \text{ and carry}$$

$$c_i = x_i\, y_i + x_i\, c_{i-1} + y_i\, c_{i-1}$$

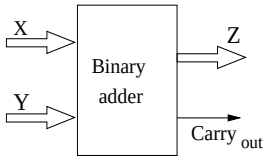

Figure 24: Binary adder

It is the fastest adder. However, number of gates required and the fan-in of each gate grow exponentially with n (number of bits) in the binary adder.

Serial adder Serial adder is shown in Figure 25. If d is the delay of a full adder and delay of F/F is D, then time to add two n-bit numbers is (d+D)n.

The hardware requirement in serial adder is independent of n.

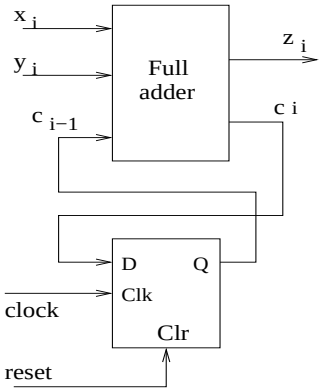

Figure 25: Serial adder

Parallel adder: is also referred to as ripple carry adder.

An n-bit parallel adders requires n full adders (Figure 26).

While adding two n-bit numbers, the carry propagares like ripples (carry generated from the i^{th} full adder (i^{th} stage) is input to the $(i+1)^{th}$ full adder).

If d denotes the delay of a full adder stage, the worst case addition time $n \times d$.

The hardware grows relative to n.

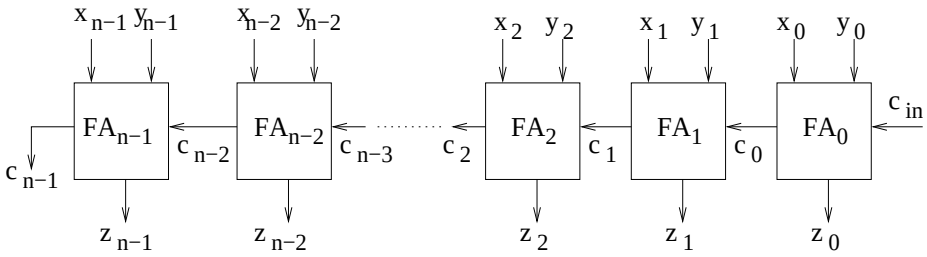

Figure 26: Parallel adder

Carry-Look-Ahead adder Strategy in to reduce the carry propagation delay.

The basic unit of a CLA is shown in Figure 27(a).

The ordinary carry c_i of ordinary full adder is replaced by two signals - carry propagare (p_i) and carry generate (g_i), where

$$p_i = x_i + y_i \text{ and}$$

$$g_i = x_i y_i.$$

The carry to be transmitted to stage (i+1) is

$$c_i = g_i + p_i c_{i-1} \text{ [as } c_i = x_i y_i + c_{i-1}(x_i + y_i)].$$

Similarly,

$c_{i-1} = g_{i-1} + p_{i-1} c_{i-2}$ -that is, $c_i = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-2}$.

By replacing c_{i-2} and so on, c_i can be expressed as SOPs of ps & gs and c_{in}.

Example: Carrys c_0, c_1, c_2 and c_3 are expressed as

$$c_0 = g_0 + p_0 c_{in}$$

$$c_1 = g_1 + p_1 c_0 = g_1 + p_1 g_0 + p_1 p_0 c_{in}$$

$$c_2 = g_2 + p_2 c_1 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{in}$$

$$c_3 = g_3 + p_3 c_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{in}$$

Figure 27(b) shows the n-bit CLA and the logic circuit for carry c_0.

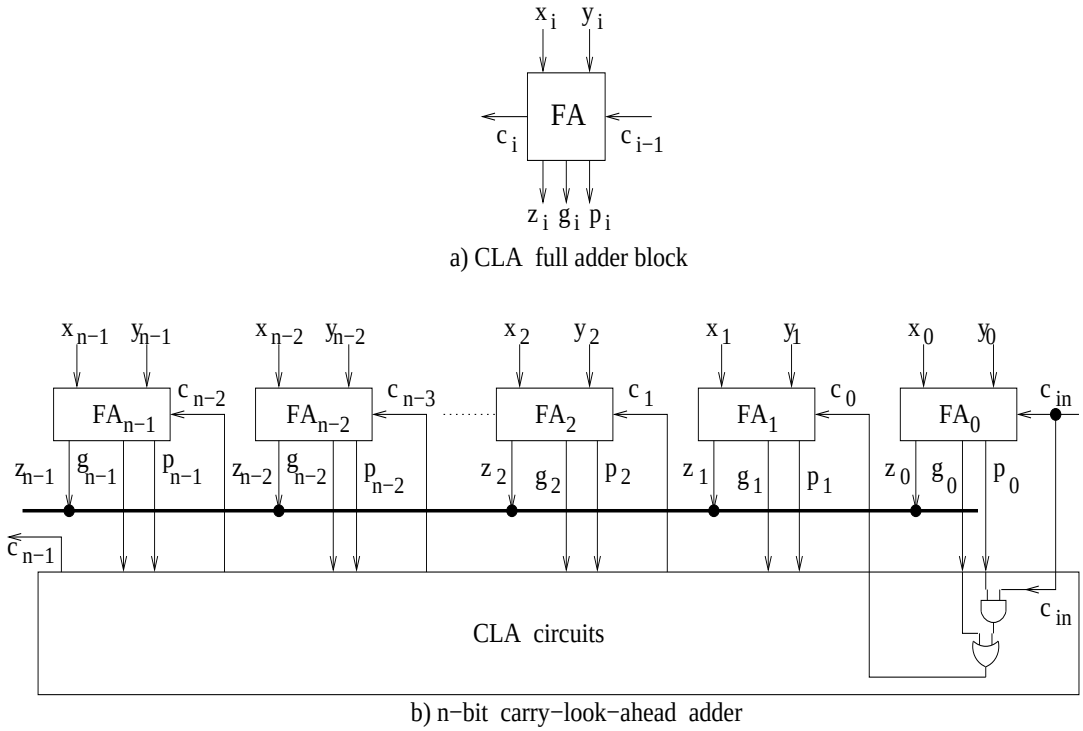

Figure 27: Carry look ahead adder

If d is the propagation delay of FA/..., the time required to compute sum of two n-bit numbers is (d+d+d) = 3d -that is, independent of n.

In first d, all the p_is and g_is are generated. During second d carrys (c_is) are calculated. In third d, zs (SUM) are computed.

Design complexity of a CLA adder increases with the value of n.

Fan-in/fan-out of AND/OR gates for computing carry become unrealistic.

This restricts the design of CLA for 16/32/64-bit adders.

In practice, 4-bit to 8-bit CLA adders are designed.

16-bit/32-bit/64-bit adder is implemented with cascade of 4/8-bit CLA adders.

16-bit adder using four 4-bit CLA adders:

16-bit adder of Figure 28 is designed with four 4-bit CLAs.

The 4-bit CLA modules are designed as in Figure 27(b).

Final carry generated from i^{th} CLA module is input to $(i+1)^{th}$ CLA module.

Time required to compute two 16-bit numbers in such an adder is

$$3d + d + d + d = 6d.$$

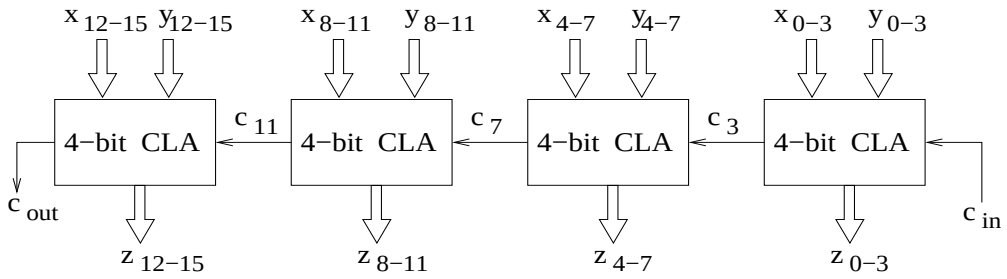

Figure 28: Cascaded carry look ahead adder