Organización del Computador
Práctico 4: Circuitos Aritméticos*

B. Gonzalez Kriegel - N. Wolovick

16 de abril de 2001

1. Para cada una de las siguientes representaciones binarias de 5 bits: $(10110, 10111)$, $(01011, 10101)$, $(11110, 11101)$ y $(11111, 01111)$:
   (a) obtenga los 5 bits de la suma $s_4s_3s_2s_1s_0$ y los acarreos n y $n-1$ c_5 y c_4.
   (b) Calcule los bits de estado: "valor a 0", "valor negativo", "acarreo" y "desbordamiento" z, n, c, v.
   (c) Interprete los resultados $zncv$ $s_4s_3s_2s_1s_0$ para las funciones de representación $(a_4a_3a_2a_1a_0)_2$ y $(a_4a_3a_2a_1a_0)_{CPL2}$.
2. * El bit de estado v que se utiliza para marcar desbordamiento en la suma *CPL2* y se define como $v = c_n \oplus c_{n-1}$, es decir que el *overflow* ocurre cuando el acarreo de entrada del *MSB* difiere del acarreo de salida.
   (a) Dar ejemplos de pares de enteros que en la suma generen las cuatro posibilidades de c_nc_{n-1}. Verifique la regla de desbordamiento.
   (b) Explique porque esta regla siempre funciona.
3. ! Se tiene una computadora que no detecta *overflow* en hardware. Muestre como detectar desbordamiento de una suma de enteros *CPL2* en software.
4. Efectúe la suma $1011101 + 0111011$ pensando estos números como *CPL1* y como *CPL2*.
5. * *End Around Carry* en suma *CPL1*
   (a) Demuestre

$$\forall n \exists a_{n-1} \ldots a_0, b_{n-1} \ldots b_0 \cdot (adder(a_{n-1} \ldots a_0, b_{n-1} \ldots b_0))_{CPL1} \neq (a_{n-1} \ldots a_0)_{CPL1} + (b_{n-1} \ldots b_0)_{CPL1}$$

   (b) Explique en términos de la definición

$$(a_{n-1} \ldots a_0)_{CPL1} = -a_{n-1}(2^n - 1) + \sum_{i=0}^{n-2} a_i 2^i$$

   porque la regla *End Around Carry* soluciona desigualdad.

*Los ejercicios marcados con **!** son un poco más complicados, mientras que los marcados con ***** sirven para ganarse un punto por aterisco en los parciales. No se puede acumular más de 2 puntos por persona por parcial.

6. Un *Carry Lookahead Adder* puro de 4 bits CLA_4 es un sumador que computa más rápidamente que un *Ripple Carry Adder* de 4 bits RCA_4.

   (a) Escriba las expresiones lógicas que definen un sumador completo con entradas a_i, b_i y c_i y salidas s_i, p_i, g_i y c_{i+1}. Reescriba c_{i+1} en función de p_i, g_i y c_i.

   (b) A partir de la expresión obtenida en el punto anterior desenrolle las definiciones de c_0, c_1 y c_2 a fin de obtener SOPs de p_is, g_is y c_is.

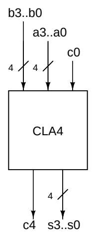
   (c) Obtenga la expresión para c_4 continuando con lo hecho en el punto anterior y obtenga $G_{0,2}$ y $P_{0,2}$ de tal forma que $c_4 = G_{0,2} + P_{0,2}c_0$. Interpretar las expresiones $G_{i,j}$ (*Group Carry Generate*) y $P_{i,j}$ (*Group Carry Propagate*) de bloque $[i, j]$.

7. Comparación del RCA y CLA puro.

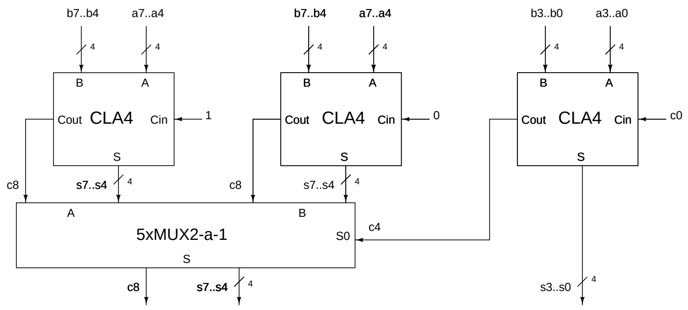
   (a) Compare RCA_4 y CLA_4 puro en función de las medidas *GCount*, *Depth* y *fanin* donde sólo podemos construir circuitos con compuertas AND, OR y NOT[1].

   (b) ! Idem al punto anterior sólo que para RCA_n y CLA_n puro. También escriba las 3 funciones en notación "O grande" ($\mathcal{O}(1)$, $\mathcal{O}(log\ n)$, $\mathcal{O}(n)$, etc.).

8. Diseñar a partir de CLA_4 un sumador híbrido RCA-CLA de 16 bits donde se van a utilizar 4 bloques como el que se muestra abajo conectando sus acarreos en cascada. ¿Cuál es la profundidad máxima y mínima del circuito? ¿Sobre que entradas y salidas se producen?



9. Un *Carry Select Adder* ($CSeA$) se basa en el principio de efectuar 2 sumas en paralelo, una suponiendo $c_{in} = 0$ y la otra $c_{in} = 1$. Cuando el acarreo de entrada se conoce, un multiplexor selecciona la suma apropiada. El siguiente es un diagrama básico para un $CSeA$ de 8 bits dividido en 2 grupos de 4 bits, donde cada grupo está siendo operado con CLA_4.



Obtenga las medidas del circuito *GCount*, *Depth* y *fanin*, y comparelas con un RCA_8 y CLA_8 puro, todas sobre la base $\{AND_n, OR_n, NOT\}$.

[1] es decir que las medidas de complejidad son sobre la base completa (infinita) $\{AND_n, OR_n, NOT\}$.

10. Dado $a = a_{n-1} \ldots a_0$, demuestre que $a + a = a << 1$, donde $a_{n-1} \ldots a_0 << 1 = a_{n-2} \ldots a_0 0$. Esta operación se la conoce como *sll* (*shift left logical*).

11. Muestre el proceso de la multiplicación serial sin signo entre el multiplicando 1010 y el multiplicador 0101. Utilice el esquema propuesto por el libro *"Principles of Computer Architecture"*.

12. Muestre el proceso de división serial sin signo entre el dividendo 1010 y el divisor 0100. Aquí también siga el esquema propuesto por el libro citado en el punto anterior.

13. Multiplique 010011 (multiplicando) por 011011 (multiplicador) utilizando el algoritmo de Booth ($Booth-2$) y el algoritmo de Booth modificado ($Booth-4$).

14. El algoritmo de Booth maneja de manera directa enteros en representación *CPL2*. Efectúe la multiplicación -8×-8 utilizando este método.

15. ! El algoritmo de Booth-4 examina conjuntos de 3 bits superpuestos $a_{i+1}a_i a_{i-1}$ con $i \in \{0, 2, 4, \ldots\}$ y determina el factor que le corresponde a cada par de bits. Explique porque no resulta conveniente extender este sistema a 4 o más bits (Booth-8, Booth-16, etc.).

16. * Para el algoritmo de Booth ordinario (*Booth radix-2*), el *Booth recorded multiplier* i-ésimo está dado por $f_2(a_i, a_{i-1}) = a_{i-1} - a_i$ (¡Compruébelo!). Encuentre la función $f_4(a_{i+1}, a_i, a_{i-1})$ para los *bit pair recorded multiplier* del algoritmo del algoritmo Booth modificado (*Booth radix-4*).

17. Diseñe una ALU de 4 bits con 2 puertos de entrada A y B y una salida C que efectúa las siguientes operaciones según los bits de control $F = f_1 f_0$. La ALU no dispone de entradas o salidas adicionales.

$f_1 f_0$	Función
00	$NAND(A, B)$
01	$SLL(A, 1)$
10	$ADD(A, B)$
11	$SUB(A, B)$

18. ! Diseñe una ALU genérica de 2 puertos de entrada A y B y uno de salida C, todos de n bits, que efectúa las siguientes operaciones según los bits de control $F = f_1 f_0$

$f_1 f_0$	Función
00	$AND(A, B)$
01	$NOT(A)$
10	$INC(A)$
11	$ADD(A, B)$

La ALU posee además la entrada c_{in} y las salidas $c_{out}zcnv$. Utilice sumadores completos para las operaciones aritméticas, compuertas para las operaciones lógicas y cualquier tipo de bloque constructivo (*MUX*es, *DECODER*s, etc.) o conjunto de compuertas para la lógica de selección.

19. Muestre la manera más sencilla de multiplicar por 2 un número *IEEE754* de precisión simple. No considere casos denormalizados o número especiales.

20. ! Escriba en pseudocódigo imperativo algoritmos para comparar y sumar dos números de punto flotante *IEEE754* $f_1 = m_1 \cdot 2^{e_1}$ y $f_2 = m_2 \cdot 2^{e_2}$, sin tener en cuenta casos denormalizados ni números especiales (el 0 si se incluye).

21. ! ¿Podría suceder en *IEEE754* que $-x \neq 0 - x$?