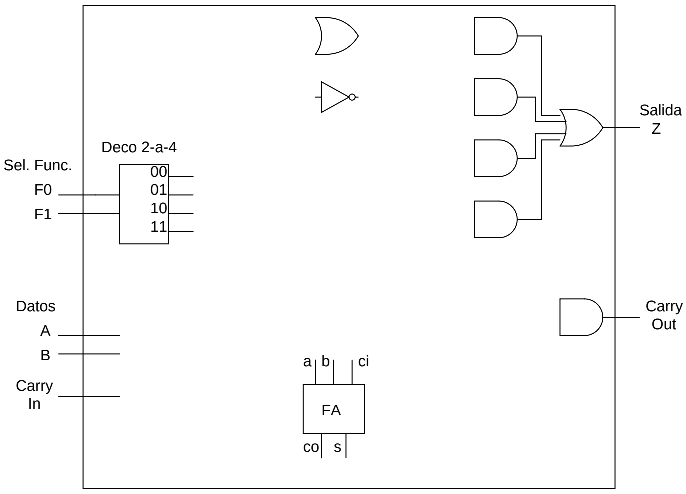
Organización del Computador
Práctico 5

B. Gonzalez Kriegel - N. Wolovick

29 de mayo de 2000

1. Diseñe una ALU de 1 bit utilizando el esquema parcial de la figura siguiente. Esta ALU realiza las operaciones de suma, AND, OR y NOT sobre las entradas de 1 bit A, B poniendo el resultado en la salida Z según las entradas F_0, F_1 como muestra la tabla.

   Solo interconecte los componentes dados, no agregue ningún tipo de dispositivo lógico.



F_0	F_1	Función
0	0	ADD(A,B)
0	1	AND(A,B)
1	0	OR(A,B)
1	1	NOT(A)

2. Diseñe una ALU (que sólo realiza operaciones lógicas) de 8 bits que tome dos entradas X, Y de 8 bits y produzca una salida Z también de 8 bits. Las entradas de control son C_1C_0 y operan de la siguiente forma:

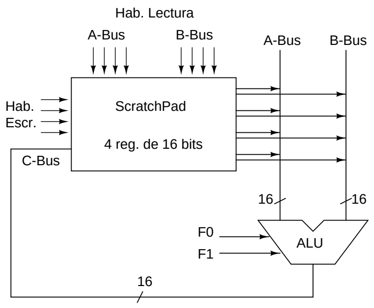
C_1	C_0	Función
0	0	AND(X,Y)
0	1	OR(X,Y)
1	0	NOR(X,Y)
1	1	XOR(X,Y)

   Realice un diseño *top-down*, mostrando primero el diagrama en bloques utilizando ALUs de 1 bit, luego cree la tabla de verdad de cada una de estas ALU y finalmente obtenga una implementación de la ALU de 1 bit utilizando un MUX 8-a-1.

3. Siguiendo el diagrama del *datapath* de una pequeñísima arquitectura que se muestra a continuación, obtenga la *secuencia de control* para realizar las operaciones

   (a) $r_0 \leftarrow r_0 - r_1$

   (b) $r_0 \leftarrow nxor(r_0, r_1)$



F_0	F_1	Función
0	0	ADD(A,B)
0	1	AND(A,B)
1	0	A
1	1	NOT(A)

La resta deberá ser utilizando complemento a 2's, y no podrá modificar ningún registro excepto r_0, r_1. Para la operación *nxor* necesitará un registro temporal. Suponga que inicialmente el registro r_3 contiene el valor 1.

A continuación se muestra el formato de la palabra de control que deberá utilizar.

Hab.Escr.				Hab.Bus A				Hab.Bus B					
0	1	2	3	0	1	2	3	0	1	2	3	F_0	F_1

Cuando no necesite realizar operaciones en un bus, puede poner todas sus líneas de selección en 0.

4. Escriba la forma binaria para las μinstrucciones mostradas. Utilize el formato del ejercicio 5 y rellene con 0's los bits cuyo valor resulta indistinto.

```
60:  R[temp0] ← NOR(R[0],R[temp0]); IF Z THEN GOTO 64;
61:  R[rd]     ← INC(R[rs1]);
```

5. Realice ahora el proceso inverso, es decir a partir de las palabras binarias de control que se dan, obtenga la versión nemotécnica utilizando el lenguaje μassembler como se mostró en el ejercicio 4.

	A		B		C	RW				
A	M_x	B	M_x	C	M_x	DR		ALU	Cond	DirSalto
100000	0	100001	0	100000	0	00100	0	11	0	00000000000
100101	0	100101	0	100101	0	00100	0	10	1	00000010000
000000	0	000000	0	000000	0	00010	1	10	0	00000001100

¿Que sección del microcódigo es?

6. ¿Cuántas μinstrucciones puede tener una instrucción ARC sin tener que cambiar el flujo de control a la parte baja del almacenamiento de control luego que la μoperación `Decode` produjo el salto a la parte alta correspondiente al opcode combinado?

7. Extienda el juego de instrucciones de ARC para incluir una nueva instrucción, modificando el μprograma. La nueva instrucción es `andn`, realiza la operación lógica *nand* entre sus operandos y no modifica los códigos de condición. La instrucción estará entre las de formato artimético ($op = 10$) y tendrá $op3 = 010100$. Deberá interpretar adecuadamente el bit 13 de direccionamiento inmediato.

8. En muchas porciones del μcódigo se utilizan instrucciones del tipo `ADD(a,a)` para obtener desplazamientos aritméticos a izquierda. Demuestre formalmente a partir de las especificaciones que `s←a+a` $\equiv$ `s←LSHIFT(a)`.

   $\{a = (a_{n-1}, \ldots, a_0)\}$
   `s←a+b`
   $\{s = (s_{n-1}, \ldots, s_0) \wedge s_0 = a_0 \oplus b_0 \wedge (\forall i : i \geq 1 : s_i = a_i \oplus b_i \oplus c_i)$
   $[c_0 = 0 \wedge (\forall i : i \geq 0 : c_{i+1} = a_i b_i + a_i c_i + b_i c_i)]\}$

   $\{a = (a_{n-1}, \ldots, a_0)\}$
   `s←LSHIFT(a)`
   $\{s = (s_{n-1}, \ldots, s_0) \wedge s_0 = 0 \wedge (\forall i : i \geq 0 : s_{i+1} = a_i)\}$

9. Comparar la cantidad pasos que toma ejecutar con control μprogramado y control cableado:

   (a) La instrucción `ld` con y sin operador inmediato.

   (b) El salto por signo negativo `bneg` suponiendo que $N = 1$ o sea el salto se realiza.

   En el control μprogramado la cuenta está dada por la *cantidad de μinstrucciones* que se ejecutan desde el comienzo en la dirección 0 hasta que se vuelve a ella. Para el control cableado la cuenta es la *cantidad de estados* visitados desde el inicio en el estado 0 hasta que la secuencia regresa a él.

   En ambos casos escriba la secuencia de ejecución. En un tipo de contol será una secuencia de μdirecciones y en el otro de estados.

   Compare ambos acercamientos con los elementos obtenidos.

10. La línea 2047 del μprograma ARC contiene una μinstrucción `GOTO 0`. ¿Cambiará el comportamiento del μprograma si cambiamos esta μinstrucción por un `skip`?

11. En la *μprogramación horizontal*, las μpalabras son anchas, mientras que en la *μprogramación vertical* resultan angostas. En general la diferencia reside en que para la μprogramación horizontal las señales de selección de los buses se encuentran decodificadas, mientras que en la vertical es necesario incluir decodificadores n en 2^n, perdiendo velocidad pero reduciendo los requerimientos de la memoria de control [1].

    Si hiciéramos más anchas las μpalabras expandiendo A, B y C a 38 bits para contener un único bit de selección por cada uno de los 38 registros:

    (a) Calcular el ancho de la nueva μpalabra.

    (b) ¿En qué procentaje se incrementará el μalmacenamiento?

12. Se quiere llevar la idea de la nanoprogramación un paso más allá y crear una picomemoria que sería referenciada por la nanomemoria donde esta ahora contendría picodirecciones. ¿Tiene sentido este planteamiento?

[1] *Trade-off* entre almacenamiento y velocidad.

13. La ALU de ARC se implementa con una tabla de búsqueda (*lookup table - LUT*). Muestre la tabla LUT_0 y LUT_x $x > 0$ para la operación `INC(A)` ($F_3F_2F_1F_0 = 1101$).

14. ¿Habrá que cambiar el μprograma de la ARC si se modifica la ALU para que trabaje en cpl1's en vez de cpl2's? Donde suponemos que siempre tomamos la representación adecuada de los números con signo en nuestros programas.

15. *[2] En el práctico 4a se utilizó mucho la siguiente construcción en la ISA de ARC para compilar la expresión booleana $a \geq b$.

```
subcc %r1, %r2, %r0
bpos  label
```

Sin embargo esto produce problemas si, por ejemplo el primer argumento es positivo y el segundo negativo. Para números de 8 bits con signo en complemento a 2's, este test dice que $\neg(2 \geq -128)$.

La solución está en verificar en conjunto los bits de estado N y V.

(a) Encuentre la expresión lógica entre N y V que si resulte equivalente a $a \geq b$ luego de hacer la resta.

(b) Modifique el μprograma de ARC para incluir la instrucción de salto `bge` (*branch greater equal*) con lo realizado en el punto anterior.

(c) Muestre como, a partir de esta instrucción, podemos compilar guardas de la forma $a > b$, $a \leq b$ y $a < b$.

[2] La primera persona que entregue el ejercicio correcto y completo, sumará 1 punto más en la nota del examen final escrito.