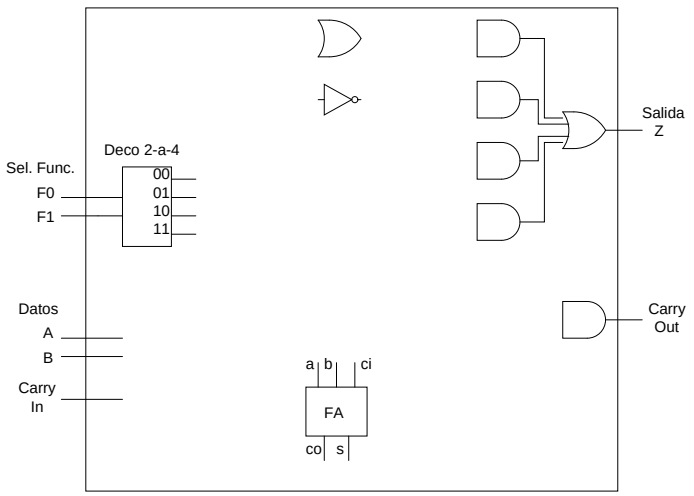
Organización del Computador
Práctico 6: Microarquitectura*

B. Gonzalez Kriegel - N. Wolovick

30 de mayo de 2001

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 barra de desplazamiento (*barrel shifter*) de 8 bits que pueda hacer corrimientos *aritméticos* a la izquierda de 0 a 3 bits.

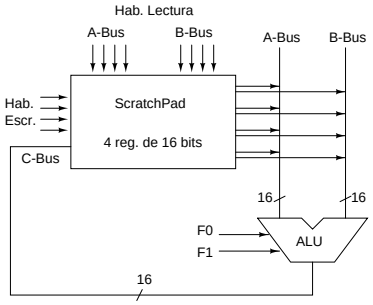
3. Siguiendo el diagrama del *datapath* de una pequeñísima arquitectura horizontal[1] que se muestra a continuación, obtenga la *secuencia de control* para realizar las operaciones

   (a) $r_0 \leftarrow r_0 - r_1$

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

*Los ejercicios marcados con ! son un poco más complicados, los marcados con * sirven para ganarse un punto por aterisco en los parciales y los marcados con + son para ganarse un punto por cruz en el examen final. No se puede acumular más de 2 puntos por persona por parcial o por final.

[1] sus instrucciones son anchas.



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. μInstrucciones a palabras de control

   (a) 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]);
```

   (b) ¿Cómo son las palabras de control para el siguiente μprograma?

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

5. palabras de control a μinstrucciones

   (a) Desensamble la siguiente secuencia de palabras de control:

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

   (b) ¿Que sección del microcódigo es?

6. Salto múltiple con `DECODE`

   (a) Forme la dirección de salto dentro del almacenamiento de control para las instrucciones `srl` y `ld`.

   (b) ¿Porqué el μprograma de `st` ocupa los segmentos $[1808, 1811]$ y $[40, 44]$ en vez de ocupar uno solo como el $[1808, 1816]$?

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. Comparación de la performance entre control μprogramado y cableado.

   Podemos pensar que la cantidad de μinstrucciones que toma una instrucción en ejecutarse es una medida del tiempo que tardará. Definimos esta cantidad como la longitud de la secuencia de ejecución del μprograma que va desde el *instruction fetching* (dirección 0) al incremento del `pc` (dirección 2047). La misma medida puede ser tomada para el control cableado, pero sobre la secuencia de estados que va del 0 (*instruction fetching*) al 20 (incremento del `pc`).

   Dar la secuencia de ejecución para el control μprogramado y cableado en los siguientes casos. Comparar la performance.

   (a) Instrucción `ld` con y sin operando inmediato.

   (b) Instrucción `bneg` suponiendo `psw.n=1`, o sea que el salto se realiza.

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

10. ! ¿Porqué la ALU tiene operaciones como `LSHIFT2(A)` y `LSHIFT10(A)` si también se cuenta con una barra de desplazamiento (*shift barrel*) a su salida?

11. En la *μprogramación horizontal*, las μpalabras son anchas, mientras que en la *μprogramación vertical* resultan angostas. La diferencia reside que en la μprogramación horizontal las señales de selección de los buses se encuentran decodificadas, evitando los decodificadores $n - 2^n$, ganando en velocidad pero aumentando los requerimientos de la memoria de control [2].

    Transformamos ARC a μprogramación horizontal, expandiendo los campos A, B y C a 38 bits para contener un único bit de selección por cada uno de los registros. Calcular el ancho de la nueva μpalabra. ¿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?

13. ¿Habrá que cambiar el μprograma de la ARC si se modifica la ALU para que trabaje en CPL_1's en vez de CPL_2's? Donde suponemos que siempre tomamos la representación adecuada de los números con signo en nuestros programas.

14. ** Uno de los diseñadores de ARC dijo que había encontrado un método muy práctico para codificar el μprograma de la nueva instrucción `swap` sin haber hecho uso de ningún registro temporal. Intente obtener este μprograma.

15. La ALU de ARC se implementa con una tabla de búsqueda (*lookup table* - LUT).

    (a) Muestre las tablas LUT_x $x \in [0, 32)$ para la operación `INC(A)` $F_{3:0} = 1101$[3].

    (b) ! Idem pero para las operaciones `INCPC(A)` $F_{3:0} = 1110$ y `SEXT13(A)` $F_{3:0} = 1100$.

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

[3] Hint: separe en LUT_0 y $\text{LUT}_x, x > 0$.

16. ! La semántica "à la Patterson" de la instrucción `ba` es la siguiente:

`ba disp22`	`R[pc]←ADD(R[pc],LSHIFT2(SEXT22(disp22)))`

Muestre que la implementación del control μprogramado no sigue esta semántica y explique como cambiaría el μcódigo para solucionarlo.

17. + En el práctico de Arquitectura del Conjunto de Instrucciones se utilizó mucho la siguiente construcción de *ARCISA* 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. Demuestre que su expresión es correcta.

(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 generar esquemas de compilación de guardas de la forma $a > b$, $a \leq b$ y $a < b$.