

Arquitectura: diseño lógico

95.57/75.03 Organización del computador

Docentes: Patricio Moreno y Adeodato Simó

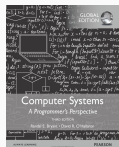
1.^{er} cuatrimestre de 2020

Última modificación: Mon Jul 27 13:03:15 2020 -0300

Facultad de Ingeniería (UBA)

Créditos

Para armar las presentaciones del curso nos basamos en:



R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition. Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2015.



D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2018.



J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2019.

Tabla de contenidos

1. Hardware Control Language

2. Señales

3. Diseño lógico

Overview

Requisitos de hardware

- Comunicación ¿cómo llegan los datos de un lugar a otro?
- Cálculos
- Almacenamiento

Bits

- Todo se expresa en términos de ceros y unos
- Comunicación: analógica
 - Valor “alto” o “bajo” en un cable
- Cálculos: funciones booleanas
- Almacenamiento: de a bits
 - un byte son 8 bits,
 - un registro son 8 bytes → 64 bits,
 - un *register file* son 16 registros → 1024 bits,
 - etc.

Tabla de contenidos

1. Hardware Control Language

2. Señales

3. Diseño lógico

Hardware Control Language

- Lenguaje de descripción de hardware (HDL) muy simple
 - Sintaxis similar a las operaciones lógicas en C para las operaciones booleanas
- Diseñado por los docentes del curso de la CMU
- Permite expresar muy pocos aspectos del diseño
 - Lo mínimo que vamos a analizar
- Se utiliza para describir la lógica del procesador diseñado

Hardware Control Language: tipos

Tipos de datos

- `bool`: booleano
 - `a`, `b`, `c`, ...
- `int`: enteros
 - `A`, `B`, `C`, ...
 - No especifica el tamaño de la palabra—bytes, palabras de 64 bits, ...

Declaraciones

- `bool a = expresión-booleana`
- `int A = expresión-entera`

Hardware Control Language: operaciones

Expresiones booleanas

- Operaciones lógicas
 - `a && b`, `a || b`, `!a`
- Comparaciones
 - `A == B`, `A != B`, `A < B`, `A <= B`, `A > B`, `A >= B`
- Operaciones con conjuntos
 - `A in {B, C, D}`
 - Equivalente a `(A == B || A == C || A == D)`

Expresiones con palabras

- `case`: `[a : A; b : B; c : C]`
 - Evalúa las expresiones `a`, `b`, `c`, ... en esa secuencia
 - Retorna la expresión `A`, `B`, `C`, ... para la primera expresión (de las anteriores) que retorna verdadero

Clasifica los tipos de las expresiones en función del tipo de retorno

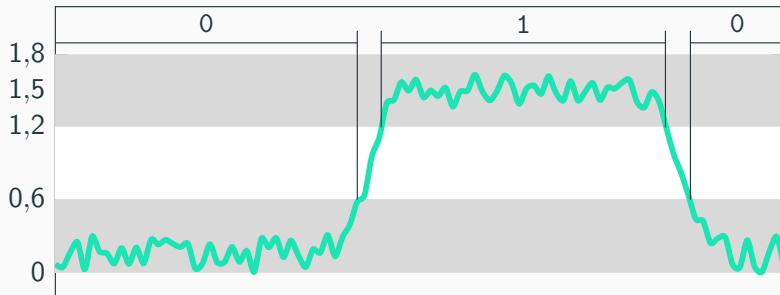
Tabla de contenidos

1. Hardware Control Language

2. Señales

3. Diseño lógico

Señales digitales



- Se utilizan umbrales para discretizar la señal
- Representación más simple: señales binarias
 - Se dividen en valor alto, "1", y valor bajo, "0".
 - A cada una se le da un umbral rango de tensiones y se separan (simplifica la electrónica, filtra oscilaciones de alta frecuencia, etc.)
- No son afectadas (fuertemente) por ruido o componentes de baja calidad
 - Se puede diseñar circuitos simples, rápidos y pequeños

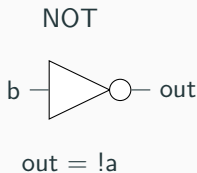
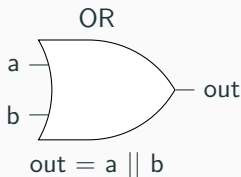
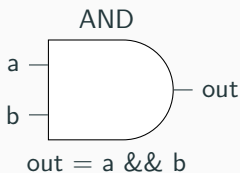
Tabla de contenidos

1. Hardware Control Language

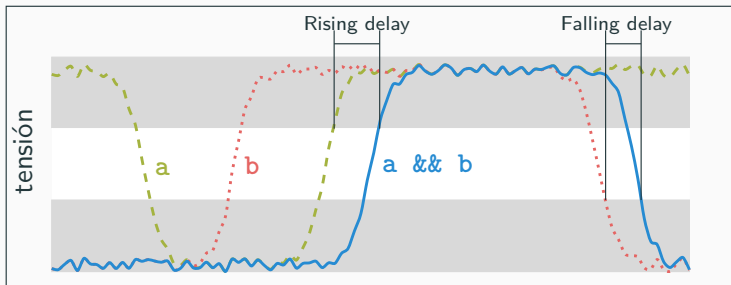
2. Señales

3. Diseño lógico

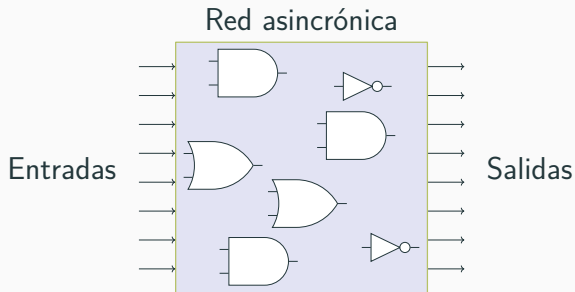
Compuertas lógicas



- Salidas booleanas en función de las entradas (doh!)
- Respuestas **continuas** a cambios en las entradas
 - con pequeños delays dados por la electrónica



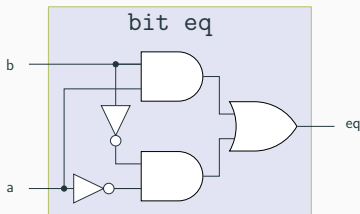
Circuitos combinacionales



- Son asincrónicos
- Continúa respondiendo a cambios en las entradas
- Al cabo de un cierto delay, las salidas son funciones booleanas de las entradas
- Permiten construir bloques más complejos

Igualdad de bits

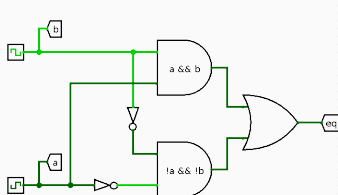
Circuito lógico



Esquemático

Expresión HCL

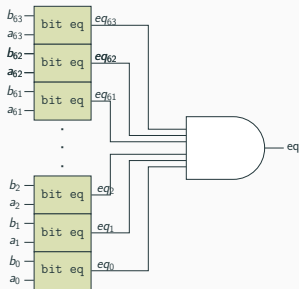
```
bool eq = (a && b) || (!a && !b)
```



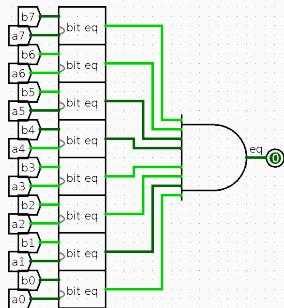
Logisim

Genera un 1 (bit) si a y b son iguales

Igualdad de palabras



Esquemático



Logisim (8 bits)

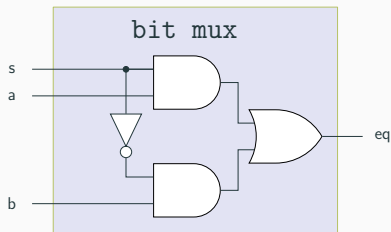
Expresión HCL

```
bool eq = (A == B)
```

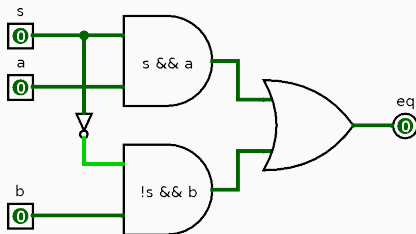
- Utiliza palabras de 64 bits
- Expresión en HCL: operador de igualdad → genera un valor booleano

Multiplexores: 1 bit

- Señal de control s
- Señales de datos en a y b
- s selecciona entre a y b



Esquemático

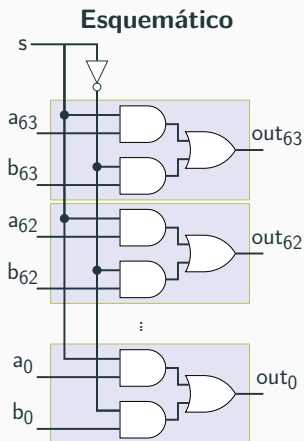


Logisim

Expresión HCL

```
bool out = (s && a) || (!s && b)
```


Multiplexores: palabras



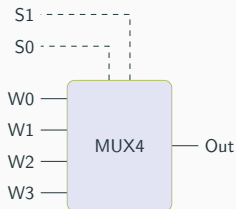
Expresión HCL

```
int Out = [  
    s : A;  
    1 : B;  
];
```

- Es similar a un case
- Las pruebas pueden ser complejas
- El 1 final funciona como un default o else

Multiplexores: ejemplos

■ Multiplexor de 4 vías



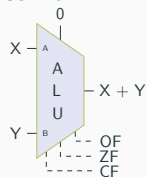
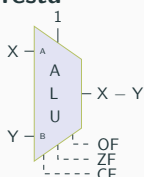
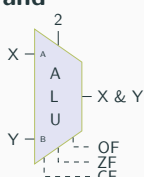
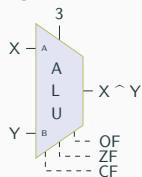
```
int Out = [  
    !s1 && !s0 : D0;  
    !s1        : D1;  
    !s0        : D2;  
    1          : D3;  
];
```

■ Mínimo entre 3



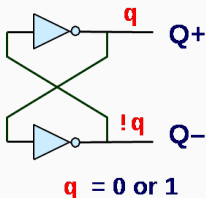
```
int Out = [  
    A < B && A < C : A;  
    B < C           : B;  
    1              : C;  
];
```

Unidad aritmética lógica (ALU)

suma**resta****and****xor**

- **Lógica combinacional**
- **La señal de control selecciona la operación**
 - En correspondencia con las 4 operaciones de Y86-64
- También modifica los códigos de condición

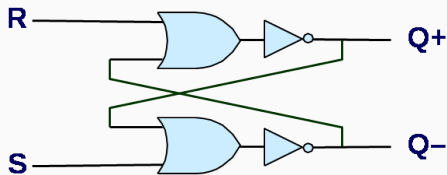
Almacenando bits



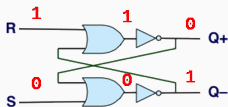
Elemento biestable

- Es la base para la construcción de circuitos más complejos
- Tiene un 0 y un 1 estables

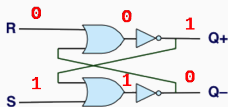
Almacenamiento y acceso a un bit: RS-Latch



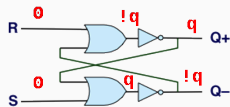
Reset



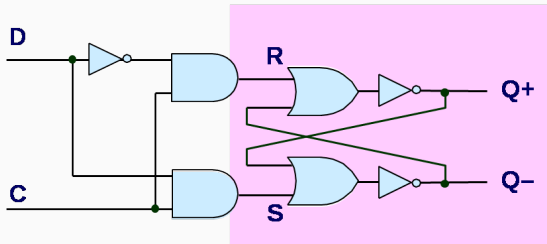
Set



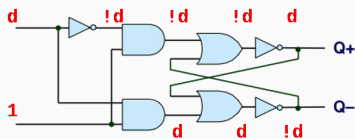
Almacenado



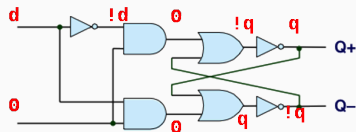
Almacenamiento y acceso a un bit: D-Latch



Latching



Almacenar



Almacenamiento y acceso a un bit: D-Latch

Latching

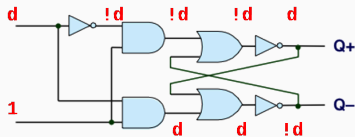
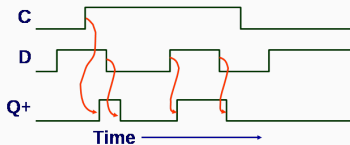


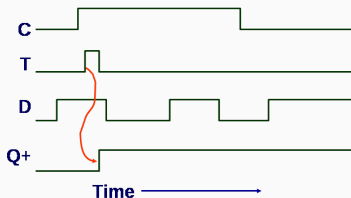
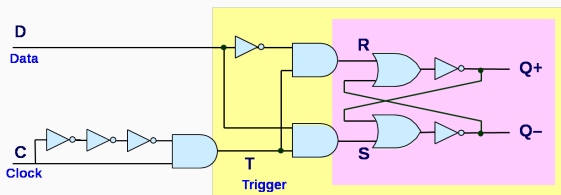
Diagrama de tiempos



- En modo *latching*, los valores para Q+ y Q- se propagan desde D.
- El valor final depende del valor de D cuando C *desciende*

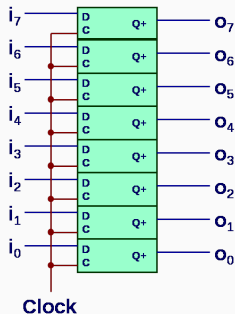
Almacenamiento y acceso a un bit: D-Latch

D-Latch disparado por flanco



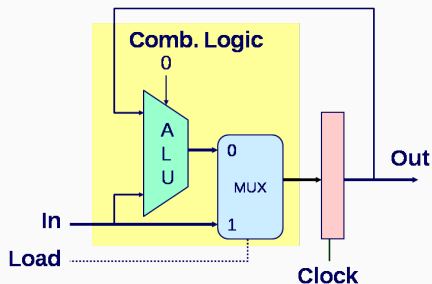
- En modo *latching* ($T = 1$) durante un breve período de tiempo
 - En el clock ascendente
- El valor final depende del valor de D cuando C *asciende*

Registros

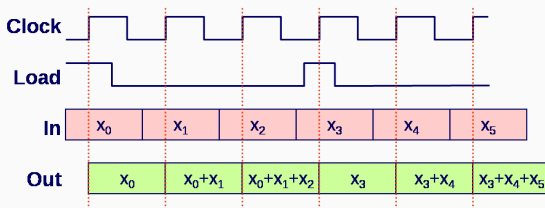


- Almacenan palabras
 - No son lo mismo que los registros de assembly (`%rax`, `%rdi`, etc.)
- Es un conjunto de *latches* disparados por flanco
- Cargan los datos en el clock ascendente

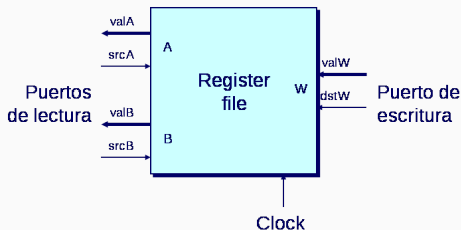
Registro: ejemplo con una máquina de estados



- Circuito acumulador
- Carga o acumula en cada ciclo del clock

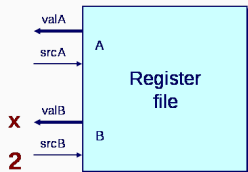


Register File: load/store



- Almacena múltiples palabras
 - Las entradas de direcciones indican qué palabra se va a leer o escribir
- *Register file*
 - Guarda los valores de los registros (`%rax`, `%rdi`, etc.)
 - Los identificadores de registros sirven de direcciones
 - El id 15 (0xf) implica que no se lleva a cabo una lectura ni escritura
- Posee múltiples puertos
 - Puede leer y/o escribir múltiples palabras en un ciclo
 - Tiene varias entradas y salidas (para direcciones y datos) separadas

Register File: timing

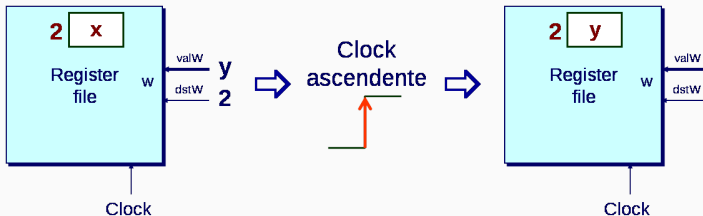


Lectura

- Es lógica combinacional
- La salida cambia al hacerlo la entrada

Escritura

- Funciona como un registro
- El cambio se produce en el flanco del clock



Licencia del estilo de beamer

Obtén el código de este estilo y la presentación demo en

`github.com/pamoreno/mtheme`

El estilo *en sí* está licenciado bajo la Creative Commons Attribution-ShareAlike 4.0 International License. El estilo es una modificación del creado por Matthias Vogelgesang, disponible en

`github.com/matze/mtheme`

