Diseño Lógico

CPU de cuatro instrucciones

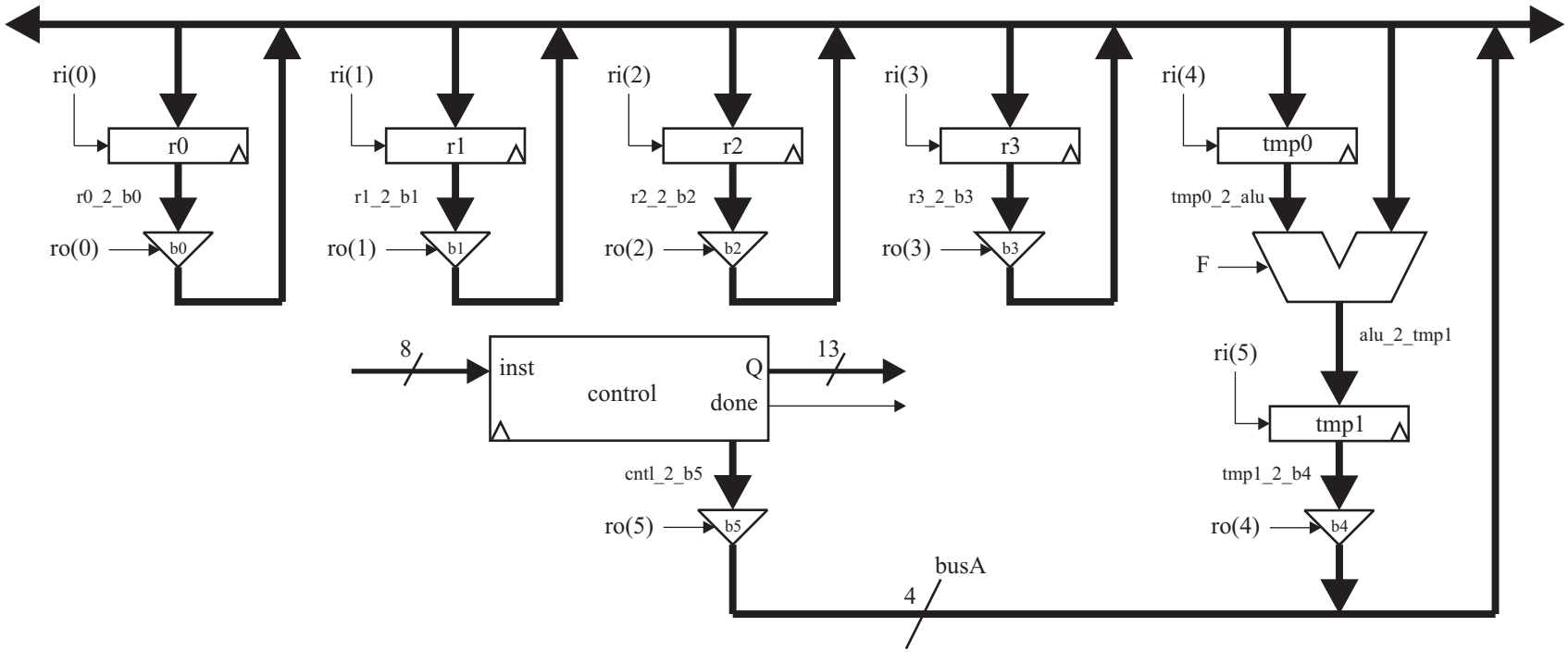
Oscar Alvarado Nava

oan@azc.uam.mx
Departamento de Electrónica
División de Ciencias Básicas e Ingeniería
Universidad Autónoma Metropolitana, Unidad Azcapotzalco

17-Primavera, junio de 2017

Contenido

Organización



Registros

- Cuatro registros (`%r0-%r3`) de propósito general de 4 bits
  - Los registros son direccionables por el programa
- Dos registros (`%tmp0`, `%tmp1`) de propósito específico de 4 bits
  - No direccionables por el programa

Unidad aritmética

- La unidad aritmética es capaz de realizar las operaciones
  - Suma `f=0`
  - Resta `f=1`

Datos

- Los datos son representados en 4 bits
- Los valores son números signados en complemento a dos
- En el lenguaje ensamblador, los datos se pueden expresar en decimal o en hexadecimal

Conjunto de instrucciones **ISA**

- El conjunto instrucciones está compuesto de cuatro instrucciones
- Todas las instrucciones están conformadas de 8 bits

Conjunto de instrucciones

Instrucción	Función
`load`	Carga dato inmendiato a un registro
`move`	Mueve dato entre registros
`add`	Suma dos datos
`sub`	Resta dos datos

Carga de datos

- `load` carga un dato a un registro
- Modo de direccionamiento inmediato

```
load 0xA,%r1
load -6,%r2
```

Movimiento de datos

- `move` mueve (copia) un dato a un registro a otro registro

```
move %r1,%r2
```

Suma de datos

- `add` suma dos datos almacenados en registros y coloca el resultado en un tercer registro

```
add %r1, %r2, %r3
```

Resta de datos

- `sub` resta dos datos almacenados en registros y coloca el resultado en un tercer registro

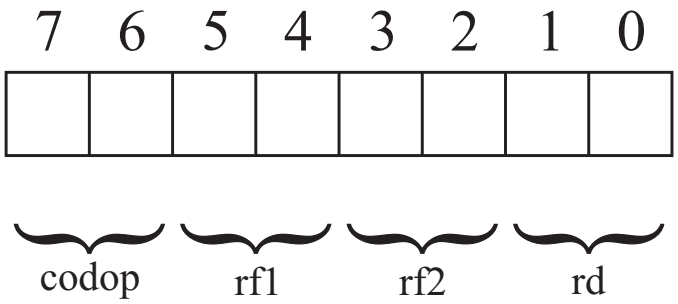
```
sub %r1, %r2, %r3
```

Programa en lenguaje ensamblador

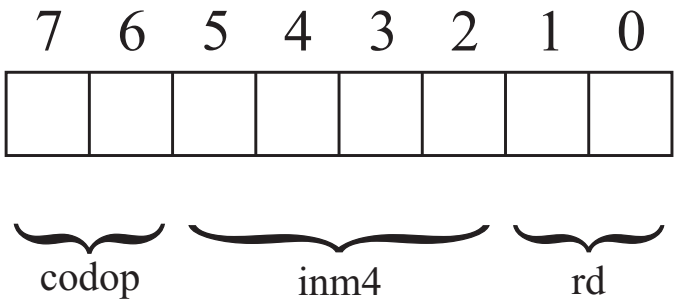
```
1                                                   !programa demo
2   .begin                                          !inicia ensamblado
3           load    0xA,    %r0             !carga el valor a r0
4           load    3,      %r1             !carga el valor a r1
5           add     %r0,    %r1,    %r2     !suma el contenido de r0 y r1
6           move    %r2,    %r3             !mueve el contenido de r2 a r3
7           sub     %r3,    %r0,    %r2     !resta
8   .end                                            !fin de ensamblado
```

Formatos de instrucción

- Cuando la instrucción tiene tres operandos



- Cuando la instrucción tiene dos operandos



Codigos de operación

codop	Instrucción
00	load
01	move
10	add
11	sub

Programa en lenguaje ensamblador

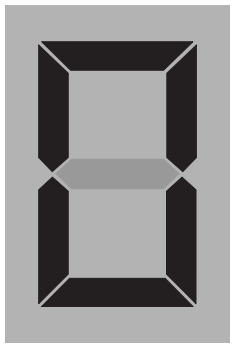
```
1
2   .begin
3           load    0xA,    %r0
4           load    3,      %r1
5           add     %r0,    %r1,    %r2
6           move    %r2,    %r3
7           sub     %r3,    %r0,    %r2
8   .end
```

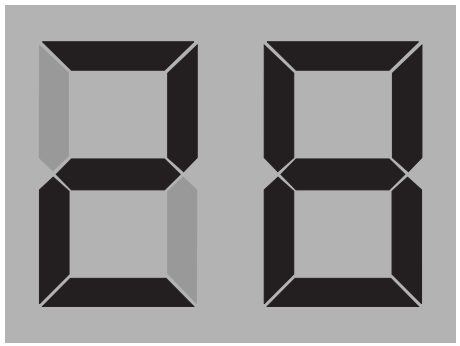
```
1
2
3       00    10    10    00
4       00    00    11    01
5       10    00    01    10
6       01    10    00    11
7       11    11    00    10
8
```

Ejecución

```
1
2    .begin
3              load    0xA,    %r0
4              load    3,      %r1
5              add     %r0,    %r1,    %r2
6              move    %r2,    %r3
7              sub     %r3,    %r0,    %r2
8    .end
```

```
1
2
3        00    10    10    00
4        00    00    11    01
5        10    00    01    10
6        01    10    00    11
7        11    11    00    10
8
```





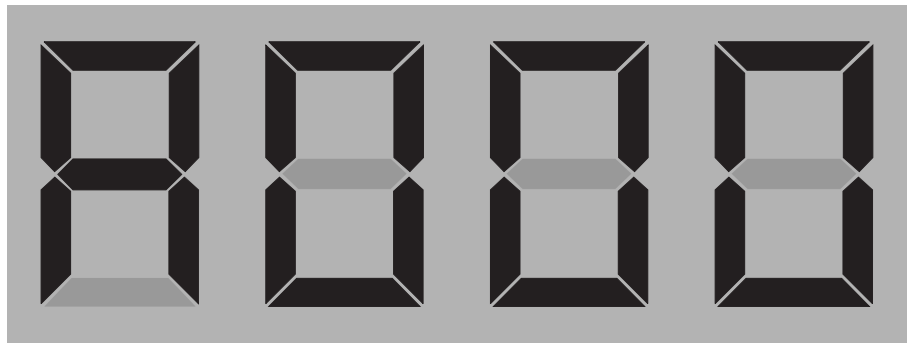
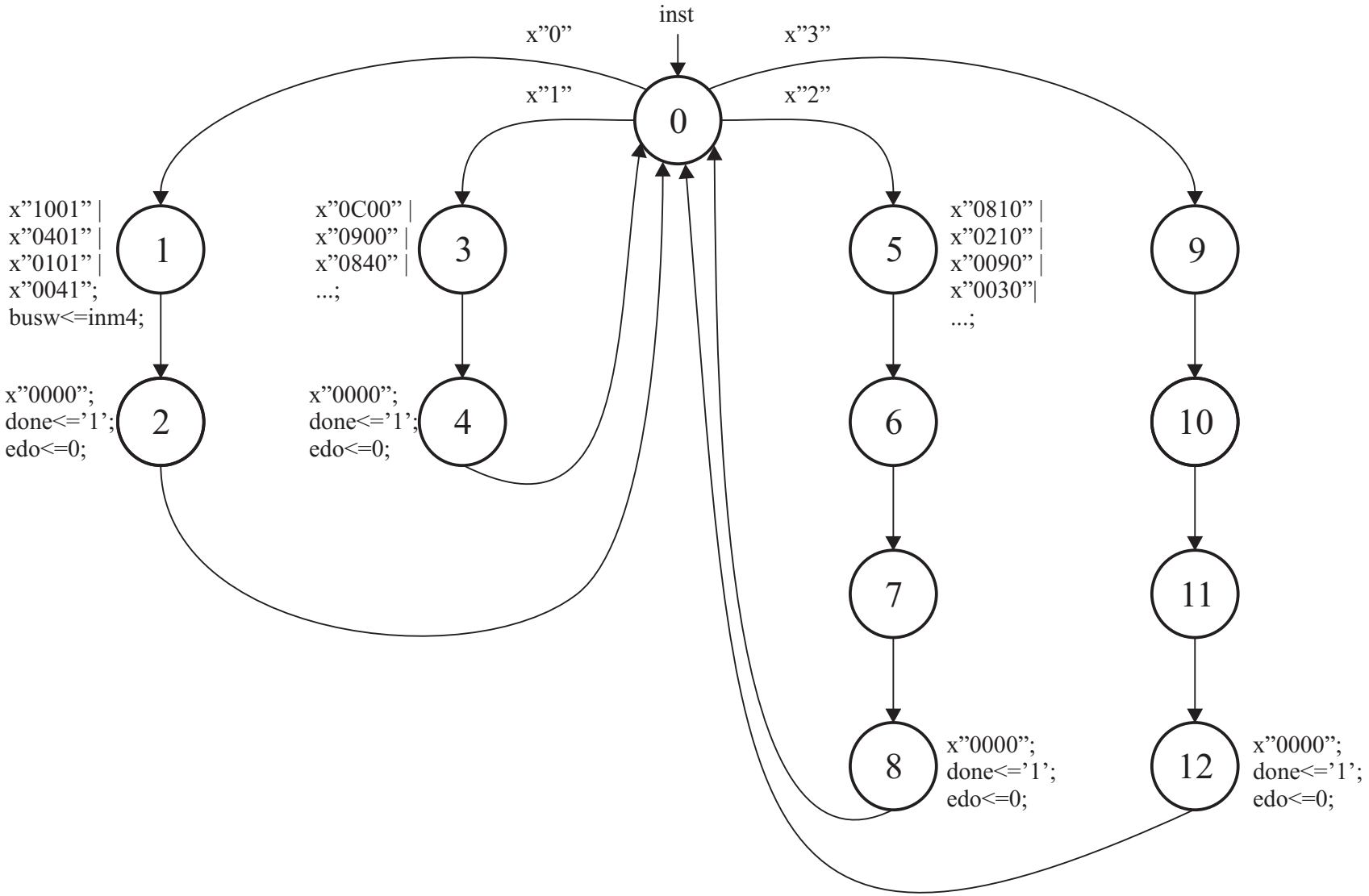


Diagrama de estados del control



Palabra de control `q(12 downto 0)`

12	11	10	9	8	7	6	5	4	3	2	1	0
r0		**r1**		**r2**		**r3**		**t0**	**t1**		**f**	**ctrl**
r	w	r	w	r	w	r	w	r	r	w	a/s	w

Entidad

```
1  --Arquitectura de Computadoras, Oscar Alvarado Nava
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use work.ac_bib.all;
5
6  entity cpu4inst is
7     port(
8        clk:in  std_logic;
9       done:out std_logic;
10       --rst:in std_logic;
11      inst:in  std_logic_vector (7 downto 0)
12     );
13 end entity cpu4inst;
```

Decodificación

```
57        case ep is
58 ------------------------------------------------------------------
59           when 0 =>  --decodificacion
60             dn <='0';
61             qint <= "0000000000000";
62                case codop is
63                   when "00" => --load del edo 1 al 2
64                      es <= 1;
65                   when "01" => --move del edo 3 al 4
66                      es <= 3;
67                   when "10" | "11" => --add o sub del edo 5 al 8
68                      es <= 5;
69                   when others => --
70                      es <= 0;
71                end case;
72 ------------------------------------------------------------------
```

Instrucción load

```
72 ------------------------------------------------------------------
73       when 1 =>  --load
74          case rd is --selecciona registro destino
75             when "00" =>
76                qint <= "1000000000001"; --0x1001
77             when "01" =>
78                qint <= "0010000000001"; --0x0401
79             when "10" =>
80                qint <= "0000100000001"; --0x0101
81             when "11" =>
82                qint <= "0000001000001"; --0x0041
83             when others =>
84          end case;
85          busw<=inm4;
86          es <= 2;
87 ------------------------------------------------------------------
88       when 2 =>
89          dn <='1';
90          es <= 0;
91 ------------------------------------------------------------------
```

Instrucción `move`

Instrucción add

Instrucción `sub`