Procesador Simple de 16 bits: Programación - Especificación

- Diseñar un programa en el ensamblador de nuestro procesador simple, que dado un vector de números enteros calcule el valor máximo almacenado en el vector
- Especificación:
  - En resultado (máximo) se tiene que almacenar en la posición 0 de memoria de datos
  - El tamaño del vector se encuentra almacenado en la posición 1 de memoria de datos
  - El vector de datos comienza en la posición 2 de memoria de datos

Ejemplo

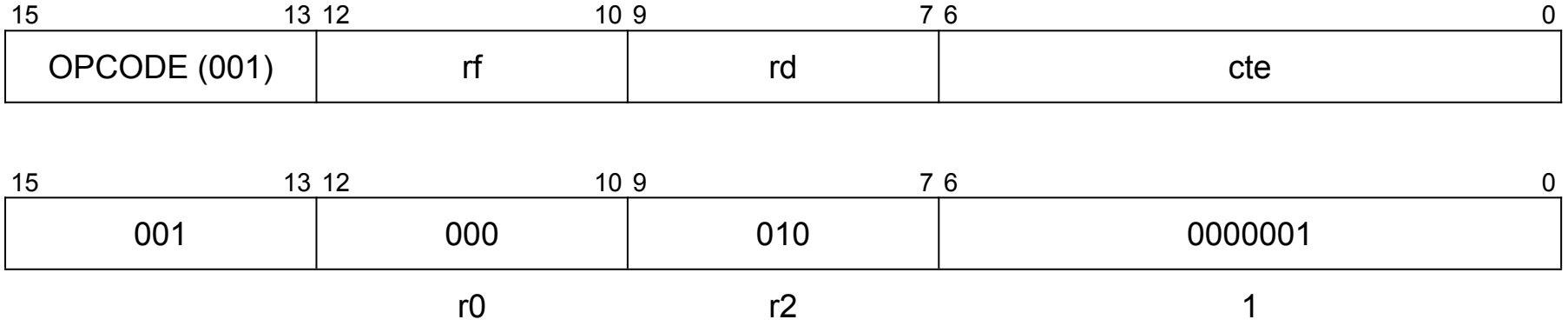
Pos	MEMDAT	
0	0	Resultado
1	5	Tam. vector
2	3	Vector[0]
3	4	Vector[1]
4	7	Vector[2]
5	1	Vector[3]
6	3	Vector[4]

Procesador Simple de 16 bits: Programación - Programa

0	`addi r2, r0, 1`	// inicializamos r2 a 1 (para leer tamaño vector)
1	`ldr r1, [r2]`	// leemos tamaño vector en r1 (r1 contador del bucle)
2	`addi r2, r2, 1`	// r2 = 2, dirección de comienzo del vector (vector[0])
3	`ldr r3, [r2]`	// leemos primera posición de vector en r3 (r3 es max. temporal)
4	`loop: beq r1, r0, exit(+7)`	// si r1 = 0 saltamos fuera del bucle (desplazamiento +7 instrucciones)
5	`ldr r4, [r2]`	// cargamos valor del vector en r4 (posición indicada por r2)
6	`addi r2, r2, 1`	// avanzamos siguiente posición del vector
7	`addi r1, r1, -1`	// restamos 1 al contador del bucle
8	`blt r4, r3, loop(-4)`	// si valor leído(r4) menor que máx.temporal(r3), volvemos bucle (desplz. -4 inst.)
9	`addi r3, r4, 0`	// si no, cargamos como max.temporal(r3) el valor leído(r4)
10	`b loop(-6)`	// volvemos a cabecera de bucle (desplazamiento -6 instrucciones)
11	`exit: str r3, [r0]`	// almacenamos max.temporal(r3) en posición 0 de memoria
12	`end: b end(+0)`	// bucle infinito para que no siga ejecutando pos. siguientes (desplz. +0 instruc.)

Procesador Simple de 16 bits: Programación - Codificación

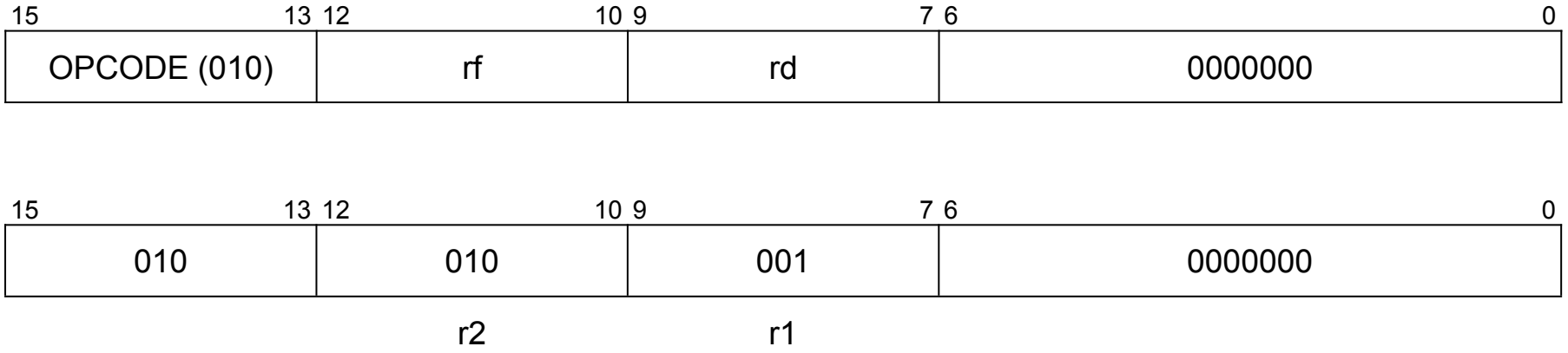
- Codificación de instrucción: `addi r2, r0, 1`
- Formato instrucción: **ADDI rd, rf, cte**
  - **rd → r2**
  - **rf → r0**
  - **cte → 1**



- **Codificación: 0010000100000001 → 0x2101**

Procesador Simple de 16 bits: Programación - Codificación

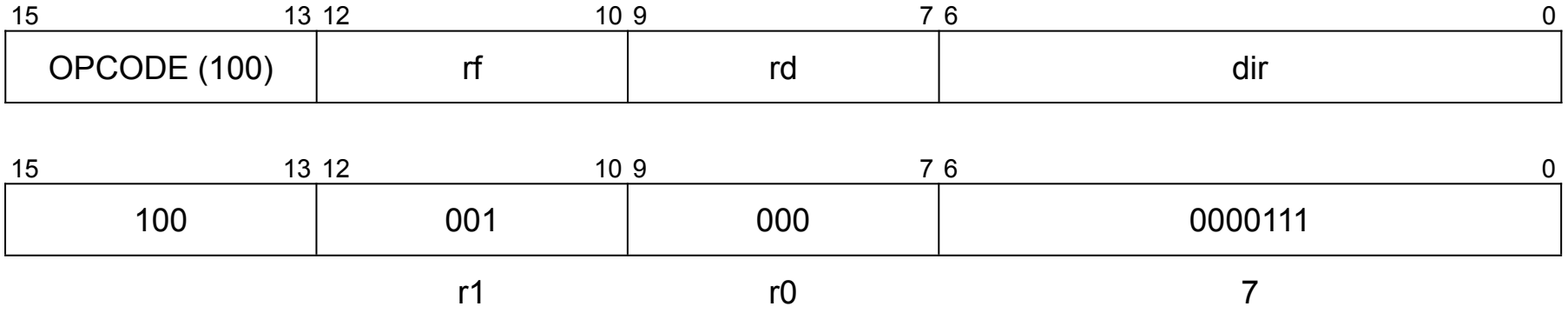
- Codificación de instrucción: `ldr r1, [r2]`
- Formato instrucción: **LDR rd, [rf]**
  - **rd → r1**
  - **rf → r2**



- **Codificación: 0100100010000000 → 0x4880**

Procesador Simple de 16 bits: Programación - Codificación

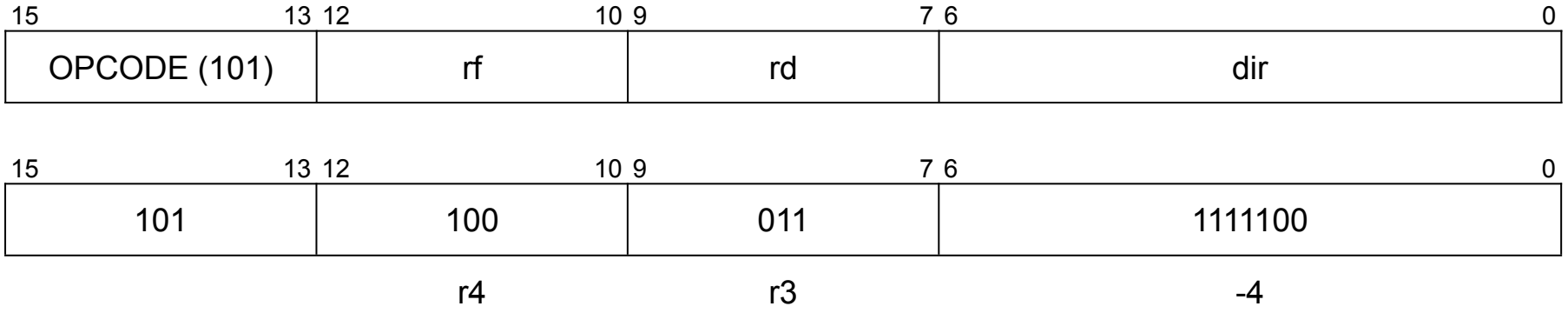
- Codificación de instrucción: `beq r1, r0, exit(+7)`
- Formato instrucción: **BEQ rf, rd, dir**
  - **rf → r1**
  - **rd → r0**
  - **dir → exit → +7**



- **Codificación: 1000010000000111 → 0x8407**

Procesador Simple de 16 bits: Programación - Codificación

- Codificación de instrucción: `blt r4, r3, loop(-4)`
- Formato instrucción: **BLT rf, rd, dir**
  - **rf → r4**
  - **rd → r3**
  - **dir → loop → -4**



- **Codificación: 1011000111111100 → 0xB1FC**

Procesador Simple de 16 bits: Programación - Codificación

0	addi r2, r0, 1	0010000100000001	0x2101
1	ldr r1, [r2]	0100100010000000	0x4880
2	addi r2, r2, 1	0010100100000001	0x2901
3	ldr r3, [r2]	0100100110000000	0x4980
4	loop: beq r1, r0, exit(+7)	1000010000000111	0x8407
5	ldr r4, [r2]	0100101000000000	0x4a00
6	addi r2, r2, 1	0010100100000001	0x2901
7	addi r1, r1, -1	0010010011111111	0x24ff
8	blt r4, r3, loop(-4)	1011000111111100	0xb1fc
9	addi r3, r4, 0	0011000110000000	0x3180
10	b loop(-6)	1101111111111010	0xdffa
11	exit: str r3, [r0]	0110000110000000	0x6180
12	end: b end(+0)	1100000000000000	0xc000