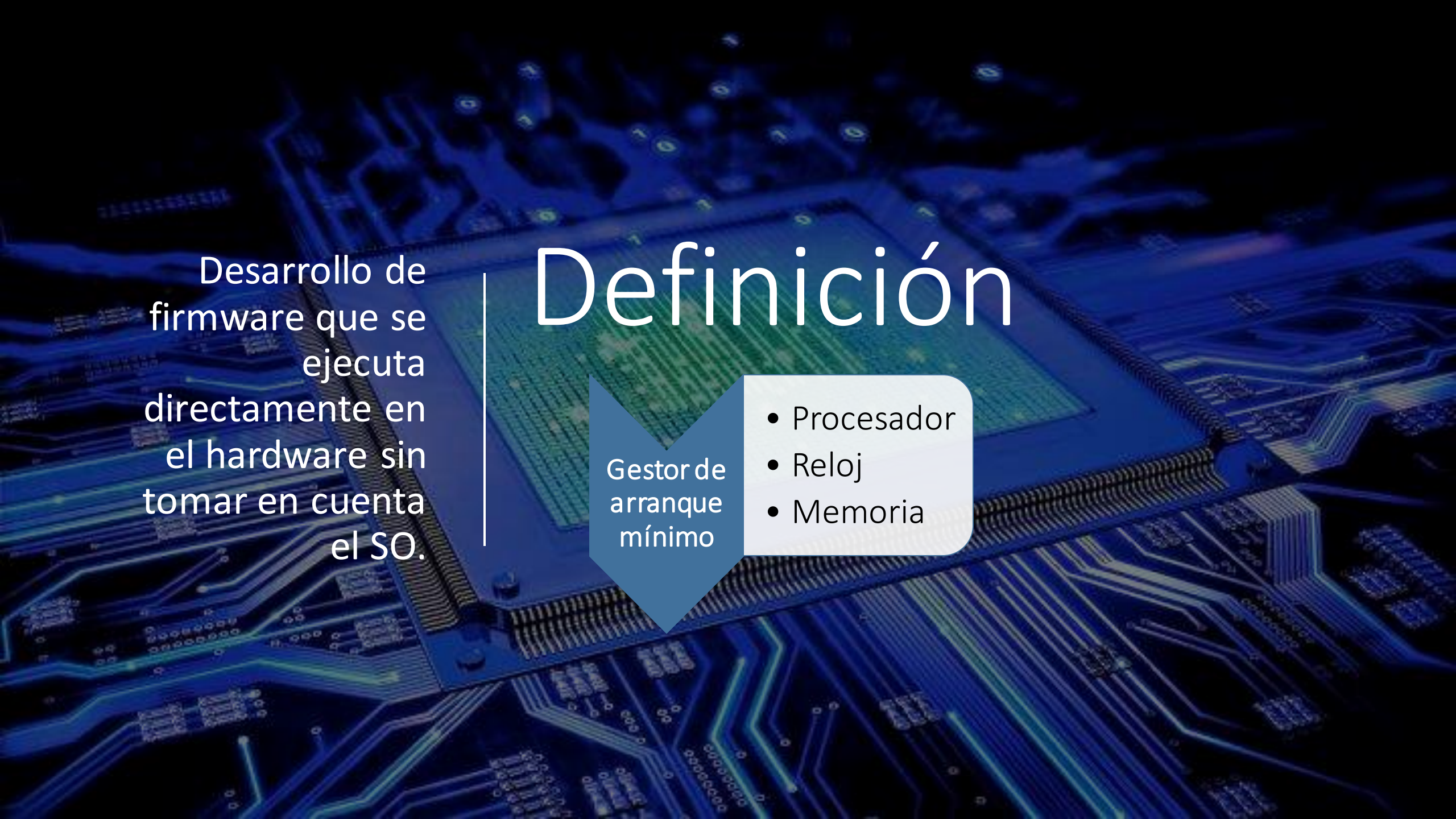
Programación Bare Metal

Arellano Yeo Nomar Alberto

Hernández García Luis Angel

Vázquez Sánchez Ilse Abril



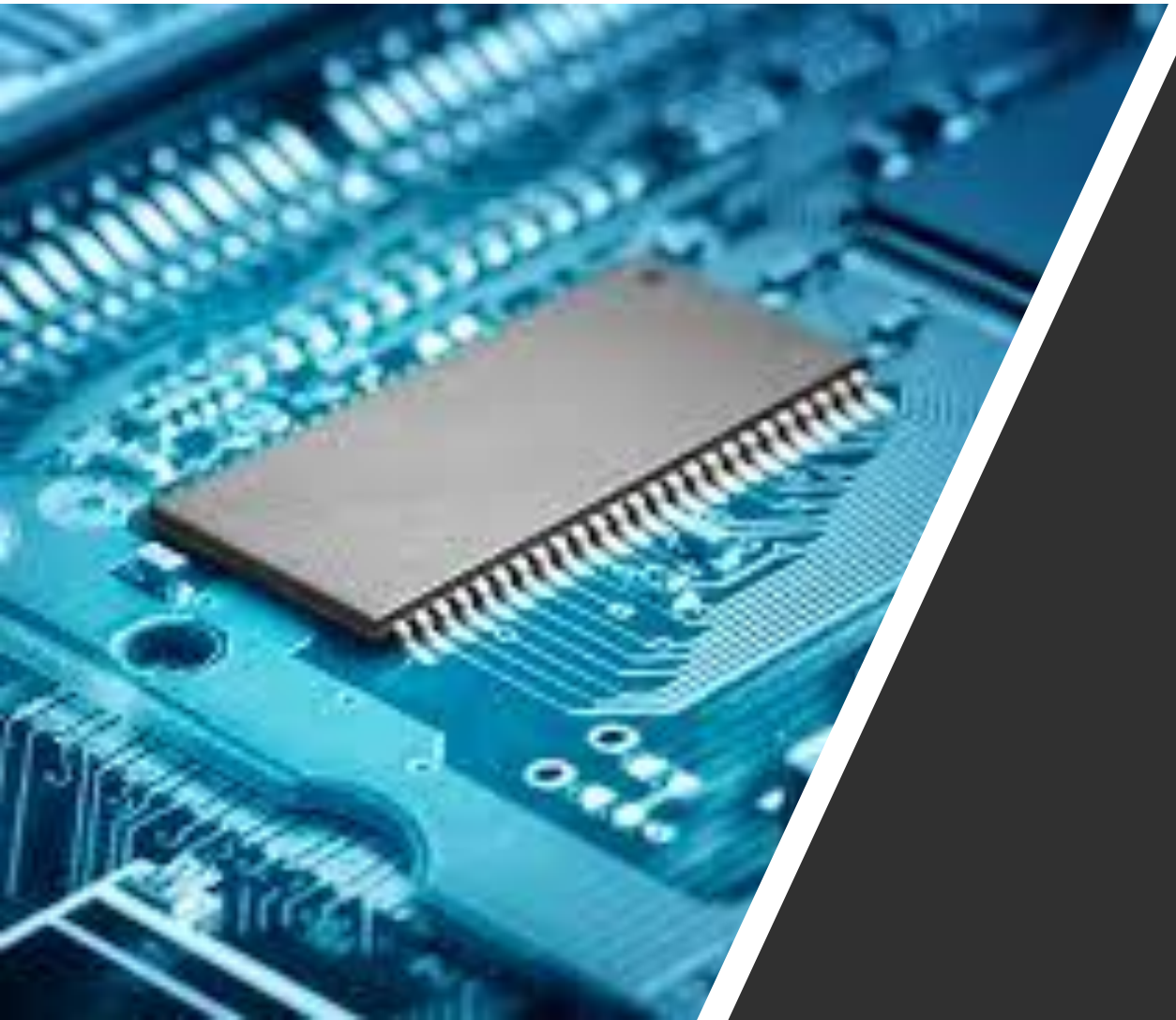
Definición

Desarrollo de firmware que se ejecuta directamente en el hardware sin tomar en cuenta el SO.

Gestor de arranque mínimo

- Procesador
- Reloj
- Memoria

Bare Metal en Sistemas Embebidos



Ciclos infinitos que llaman a subrutinas responsables de verificar entradas, realizar acciones específicas y escribir salidas.

Programación Bare Metal vs. Sistemas Operativos

Ventajas

- Hace más fácil escribir drivers de dispositivo para los periféricos
- El resultado es, a menudo, más rápido en velocidad de ejecución
- Permite reducir el consumo de energía y recursos del sistema
- Es más fácil cumplir objetivos de ejecución difíciles en tiempo real.

Desventajas

- Se deben escribir casi todos los dirvers de dispositivos para el sistema
- La concurrencia es más difícil de implementar
- La depuración puede ser más complicada de llevar a cabo

¿Cuándo hacer uso de programación Bare Metal?

Tiempos de arranque estrictos

Diseño de sistemas de baja gama

Mayor control del sistema

Eliminación de sobrecarga al SO

No se pueden pagar licencias

Conocimientos para desarrollo de Bare Metal

Lenguaje Ensamblador

Lenguaje C

Procesos de arranque

Llamadas al sistema

Plataformas de desarrollo

PROGRAMACIÓN BARE METAL EN RASPBERRY PI

ARCHIVOS NECESARIOS PARA PROGRAMAR

- **bootcode.bin**
- **start.elf**
- **kernel.img**

Compilar código
Bare Metal

Secuencia Normal

```
as -o ejemplo.o ejemplo.s
gcc -o ejemplo ejemplo.o
```

Secuencia para Bare Metal

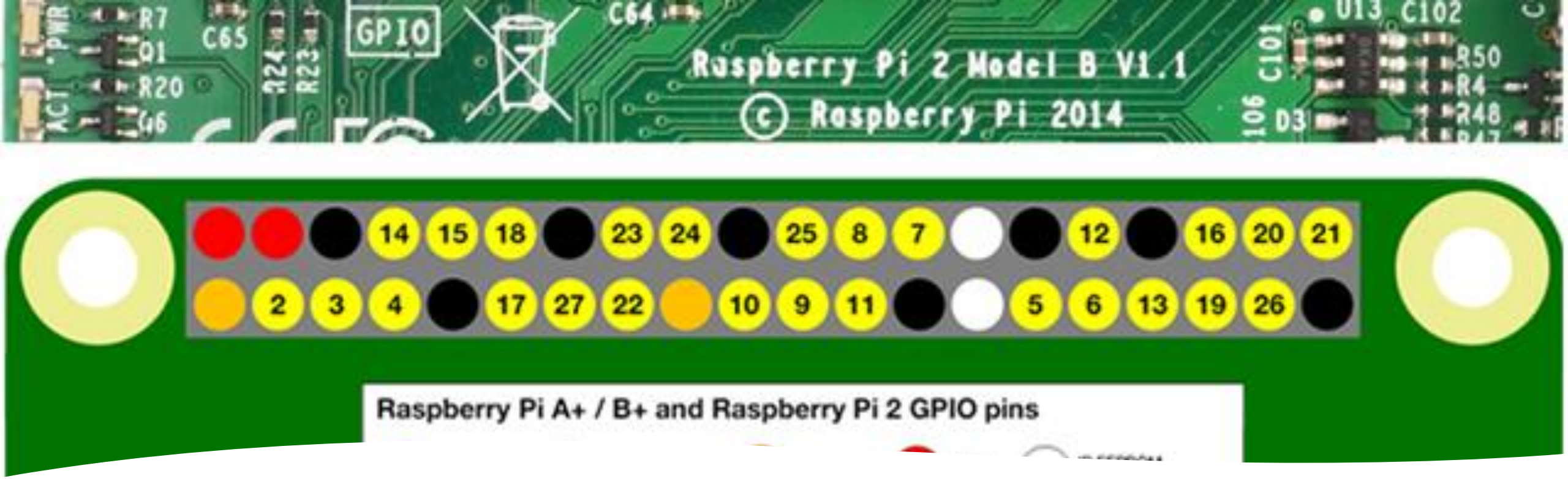
```
as -o ejemplo.o ejemplo.s
ld -e 0 -Ttext=0x8000 -o ejemplo.elf ejemplo.o
objcopy ejemplo.elf -O binary kernel.img
```

Procedimiento para creación de programas

1. Apagar la Rapsberry
2. Extraer la tarjeta SD
3. Introducir la tarjeta SD en el lector de la plataforma de desarrollo
4. Montar la unidad y sobrescribir el "kernel.img" nuevo
5. Desmontar y extraer la SD
6. Insertar de nuevo la SD en la Raspeberry

Opciones adicionales para montar programas

1. Cable JTAG con Software Opencode
2. Cable USB-Serie (UART)
3. Cable serie – serie (para dos Raspberry)



GPIO Raspberry

- **Se encuentran mapeados en memoria comenzando por la dirección 0x20200000**
- **Se puede acceder a ellos a través de los registros GPFSEL, GPSET y GPCLR**

Registros GPIO Raspberry

*** GPFSEL:**

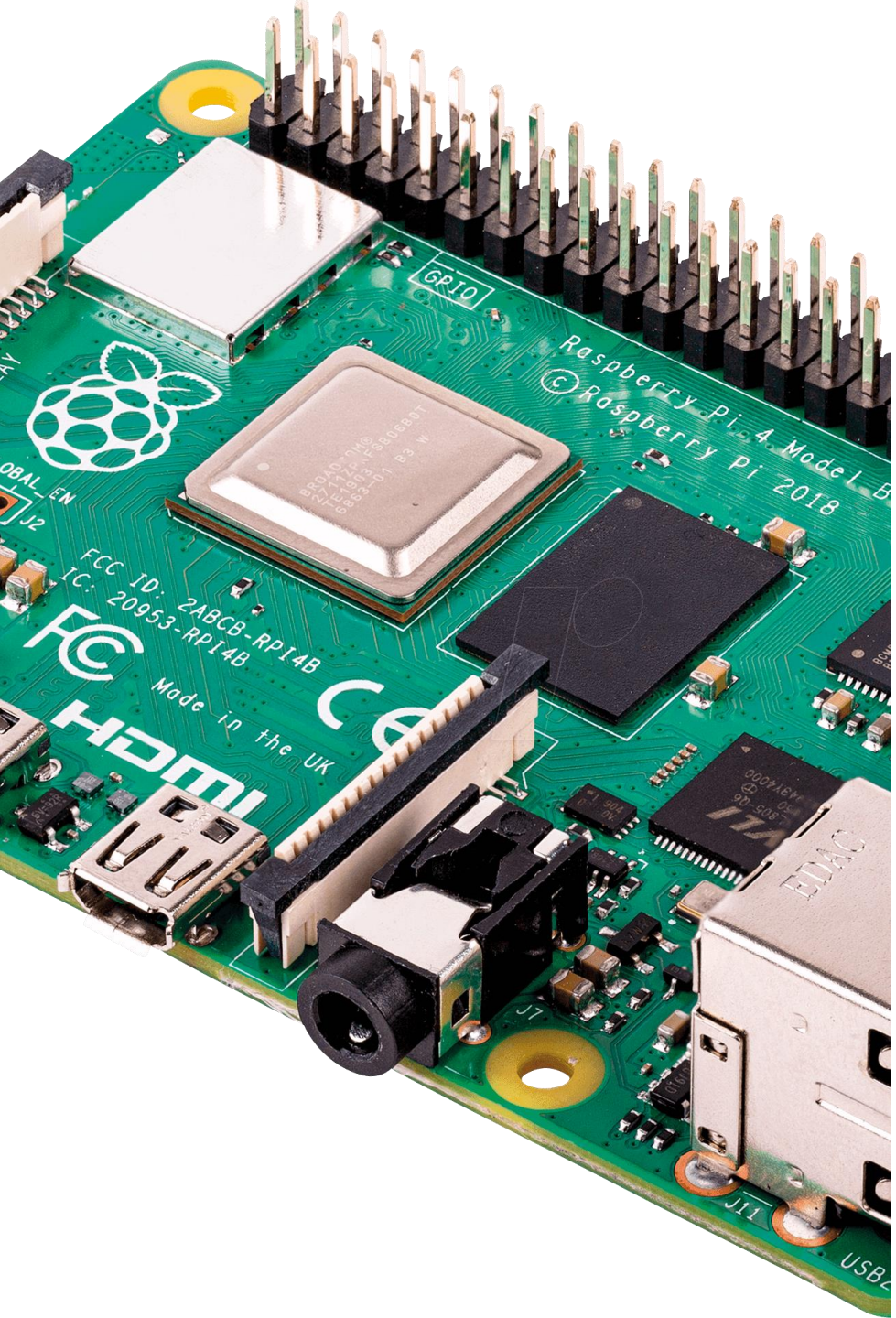
Las 54 señales GPIO se distribuyen en 6 grupos diferentes, 5 grupos con 10 señales y un último grupo con 4 señales. GPFSEL0, GPFSEL1, GFPSEL2, GPFSEL3, GPFSEL4 y GPFSEL5.

***GPSET y GPCLR:**

Las 54 señales GPIO se reparten en dos partes. GPSET0 y GPCLR0 almacenan las 32 primeras mientras que los registros GPSET1 y GPCLR1 las 22 restantes.

Tabla de Registros GPIO Raspberry

Dirección	Nombre	Descripción	Tipo
20200000	GPFSEL0	Selector de función 0	R/W
20200004	GPFSEL1	Selector de función 1	R/W
20200008	GPFSEL2	Selector de función 2	R/W
2020000C	GPFSEL3	Selector de función 3	R/W
20200010	GPFSEL4	Selector de función 4	R/W
20200014	GPFSEL5	Selector de función 5	R/W
2020001C	GPSET0	Pin a nivel alto 0	W
20200020	GPSET1	Pin a nivel alto 1	W
20200028	GPCLR0	Pin a nivel bajo 0	W
2020002C	GPCLR1	Pin a nivel bajo 1	W



Puertos y registros auxiliares

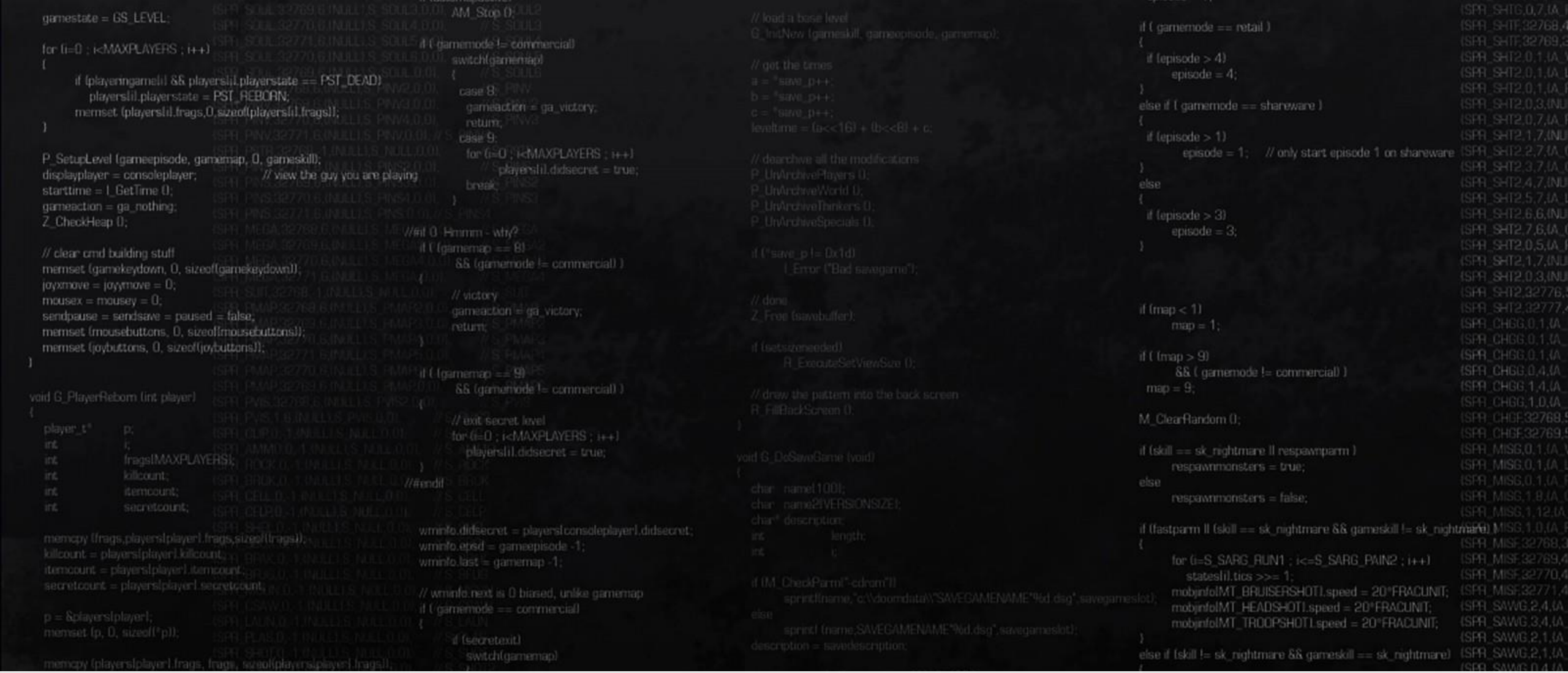
- **GPLVE:** Devuelve el valor del pin seleccionado. Se retorna 0 si hay 0[v] y se retorna 1 si hay 3.3[v].

- **GPEDS:** Permite detectar qué pin ha provocado una interrupción en caso de usarlo como lectura.

- **GPREN:** Se selecciona cuáles son los pines que tengan permiso para generar interrupciones.

- **GPHEN:** Permite seleccionar cuáles son los pines que provocarán una interrupción al detectar 3.3[v].

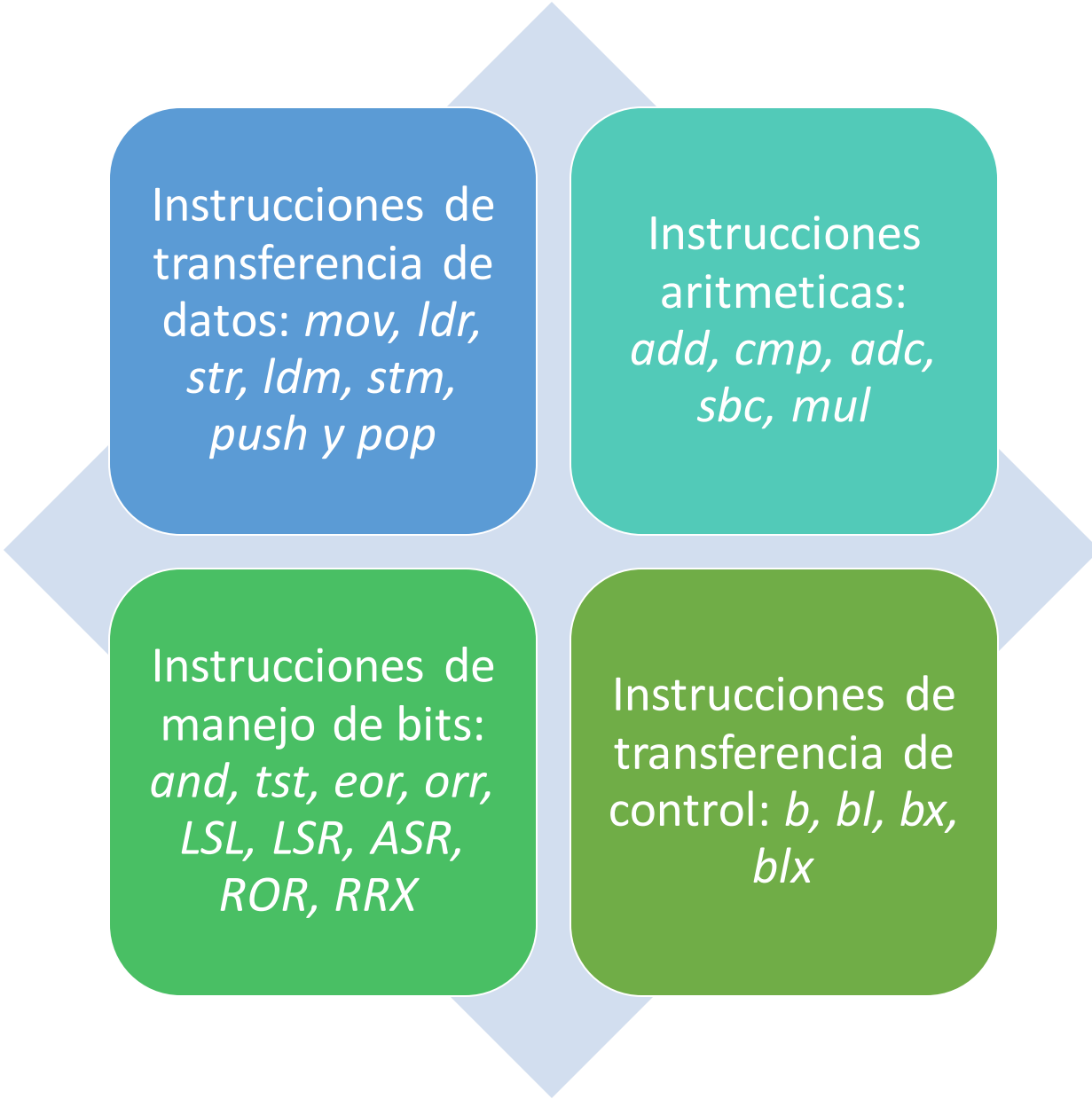
- **GPLEN:** Permite seleccionar cuáles son los pines que provocarán una interrupción al detectar 0[v].



Ejemplos para Bare Metal

Lenguaje ensamblador (ARMv6)

Instrucciones



Encender y apagar led con retardo

```
        .set    GPBASE,     0x20200000  /* dirección base de los puertos
                                        Selección del grupo 0 de GPIO (0-9)*/
        .set    GPFSEL0,    0x00        /* Se inicializan los pines como entradas */
        .set    GPSET0,     0x1c        /* inicializamos GPSET0 */
        .set    GPCLR0,     0x28        /* inicializamos GPCLR0 */

.text
        ldr     r0, =GPBASE /* carga dirección base en r0 */
        /* guia bits   xx99988877766655544433322211l000 */
        mov     r1, #0b00001000000000000000000000000000
        str     r1, [r0, #GPFSEL0]  @ GPIO 9 como salida
        /* guia bits                          9876543210 */
        mov     r1, # 0b00000000000000000000001000000000

bucle:  ldr     r2, =7000000        @ carga en r2
ret1:   subs    r2, #1              @ resta 1
        bne     ret1                @ regresa si Z = 0
        str     r1, [r0, #GPSET0]   @ enciende led GPIO9 = 1
        ldr     r2, =7000000
ret2:   subs    r2, #1
        bne     ret2
        str     r1, [r0, #GPCLR0]   @ apaga led GPIO9 = 1

        b       bucle
```

Ejercicio

Supongamos que una iteración del retardo (combinación de *subs* y *bne)* tarda 50 ciclos de reloj en ejecutarse. Tenemos una velocidad de 700MHz por lo que un ciclo dura aproximadamente 1.5ns así que cada iteración tarda 50 x $1.5E^{-9}$ s. Si para medio segundo necesitamos 7 millones de iteraciones, ¿Cuántas requeriremos para hacer un retardo de 2 segundos?

$$iteraciones = \frac{tiempo\ de\ consumo}{duración\ de\ 1\ iteración}$$

Solución

$$iteraciones = \frac{tiempo\ de\ consumo}{duración\ de\ 1\ iteración} = \frac{2s}{50 \times 1.5ns} \cong 27000000$$

Mismo caso en C

```c
/* Dirección base de perifericos GPIO */
#define GPIO_BASE       0x20200000UL

#define LED_GPFSEL      GPIO_GPFSEL0 // GPIO0-9
#define LED_GPFBIT      27
#define LED_GPSET       GPIO_GPSET0
#define LED_GPCLR       GPIO_GPCLR0
#define LED_GPIO_BIT    9

volatile unsigned int* gpio;
volatile unsigned int tim;

int main(void) {
    /* asignar direccion base GPIO */
    gpio = (unsigned int*)GPIO_BASE;

    /* Establece GPIO16 como salida */
    gpio[LED_GPFSEL] |= (1 << LED_GPFBIT);

    /* ciclo infinito */
    while(1)
    {
        for(tim = 0; tim < 500000; tim++)
            ;

        /* Apagar bit */
        gpio[LED_GPCLR] = (1 << LED_GPIO_BIT);

        for(tim = 0; tim < 500000; tim++)
            ;

        /* encender bit*/
        gpio[LED_GPSET] = (1 << LED_GPIO_BIT);
    }
}
```