Electronica digital 2

caso de estudio LM32

Ferney Alberto Beltrán Molina

Agosto 2019

Contacto

Nombre:	Ferney Alberto Beltrán Molina, Ing, MSc, PhD(c)
Email:	fabeltranm@unal.edu.co
oficina:	

Contenido

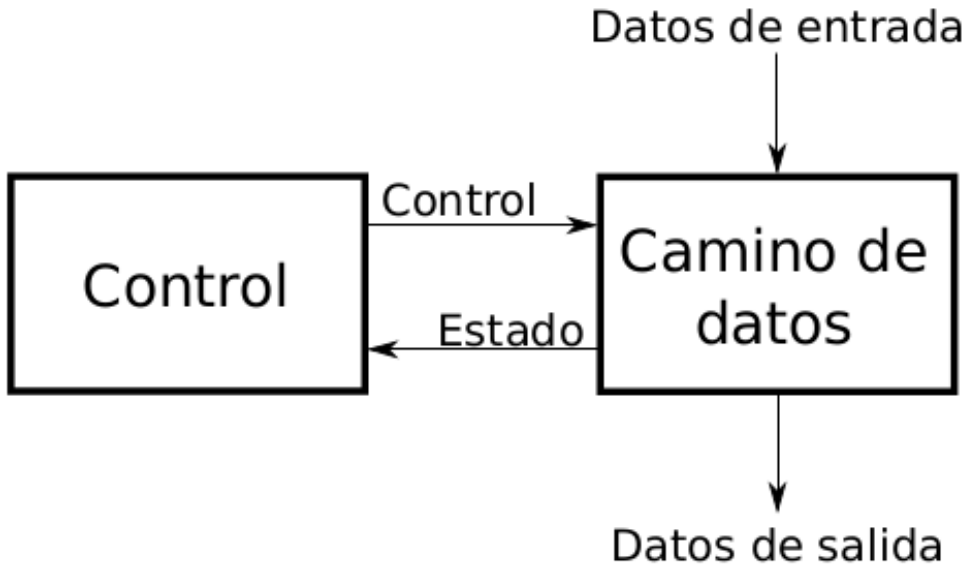
Índice

Antes..

En la clase anterior vimos:

- se presenta el datapath y la unidad de control Generica
- pipeline

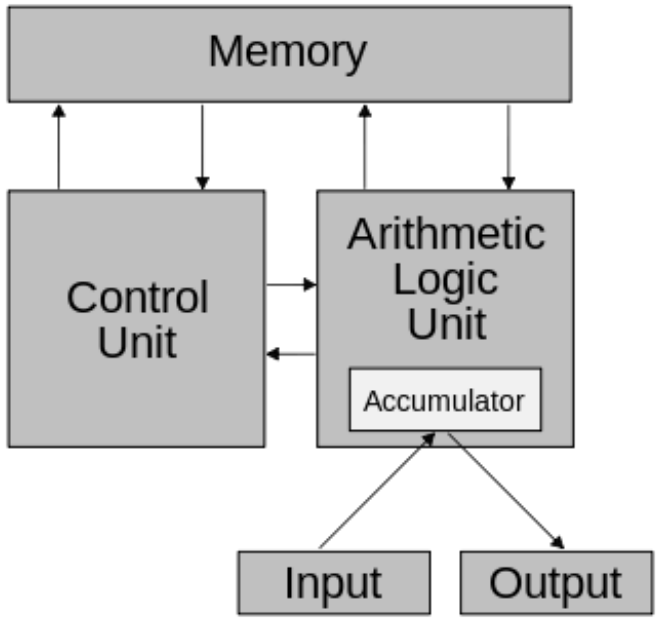
Recordando


Datos de entrada
Control
Control
Estado
Camino de datos
Datos de salida


CASIO

Índice

Hardware Software Interface



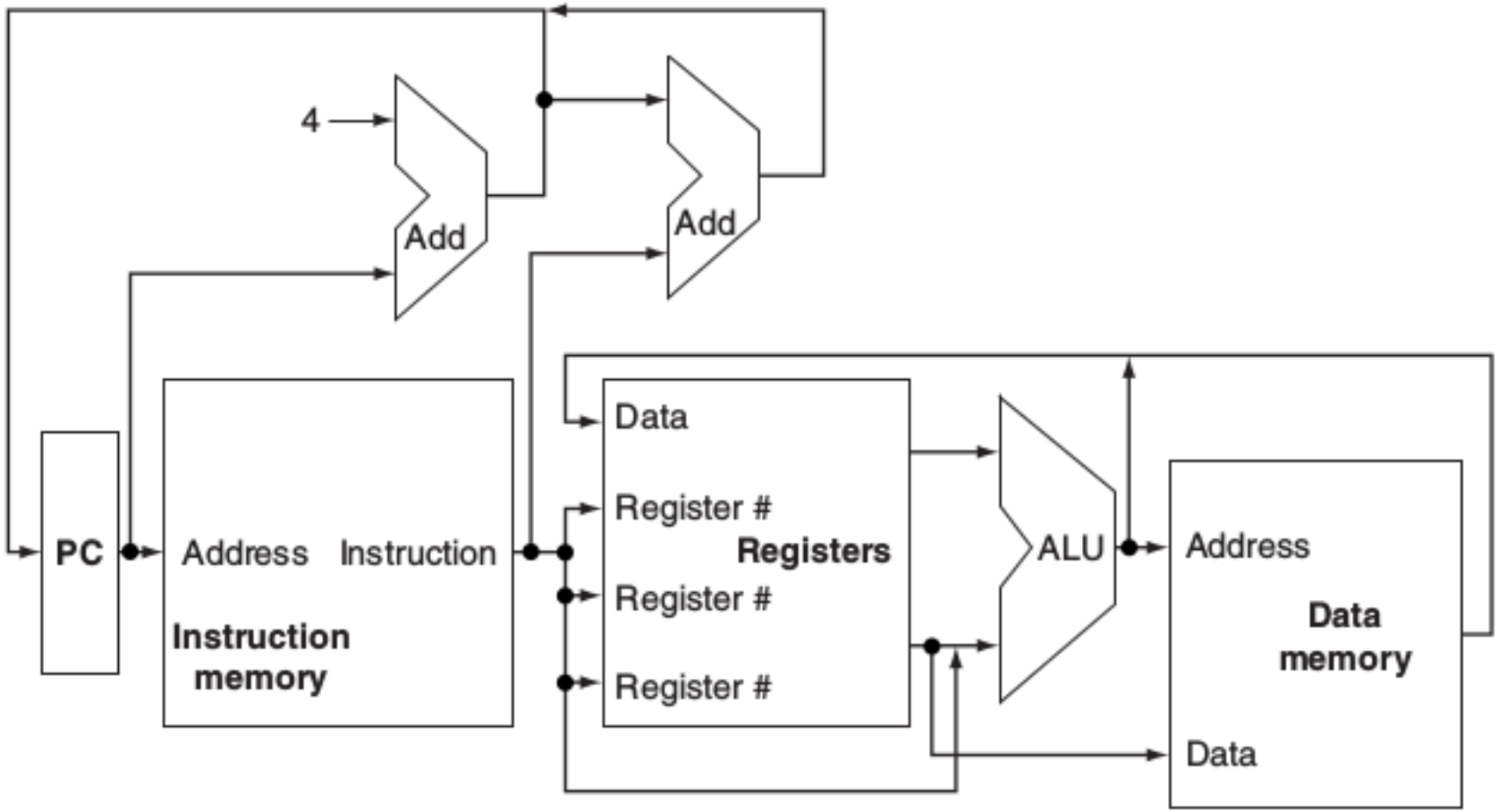
El procesador

Sigue las instrucciones del programa al pie de la letra. Suma y compara números, ordena activarse a los dispositivos de I/O, etc.

El procesador consta de dos componentes:

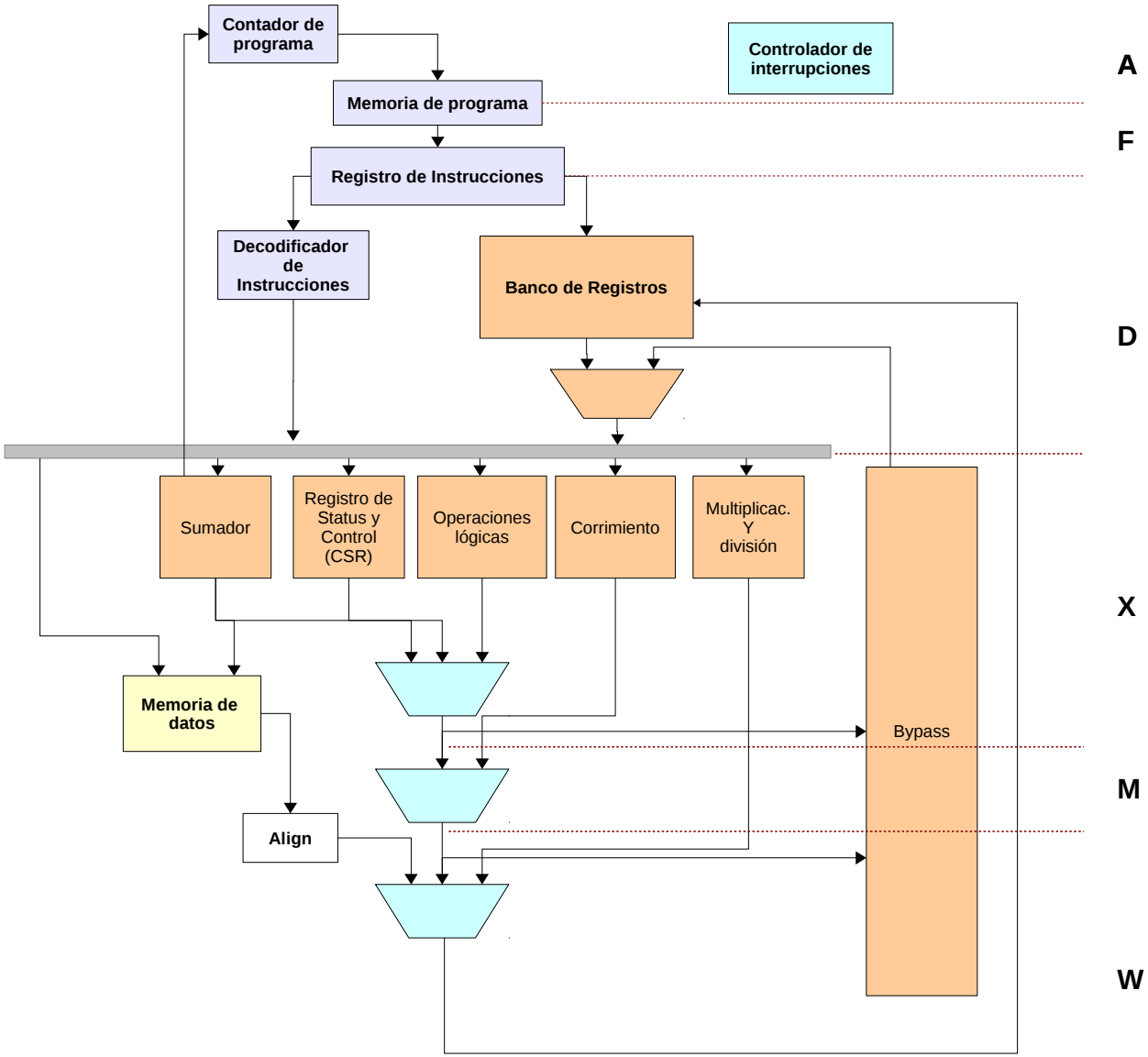
- El datapath. Ejecuta operaciones aritméticas y lógicas.
- El control. Ordena al datapath, memoria y dispositivos de I/O lo que hay que hacer de acuerdo al program

Procesador básico



Índice

soft-core LM32



Pipeline

A *Address*: Se calcula la dirección de la instrucción a ser ejecutada y es enviada al registro de instrucciones.

F *Fetch*: La instrucción se lee de la memoria.

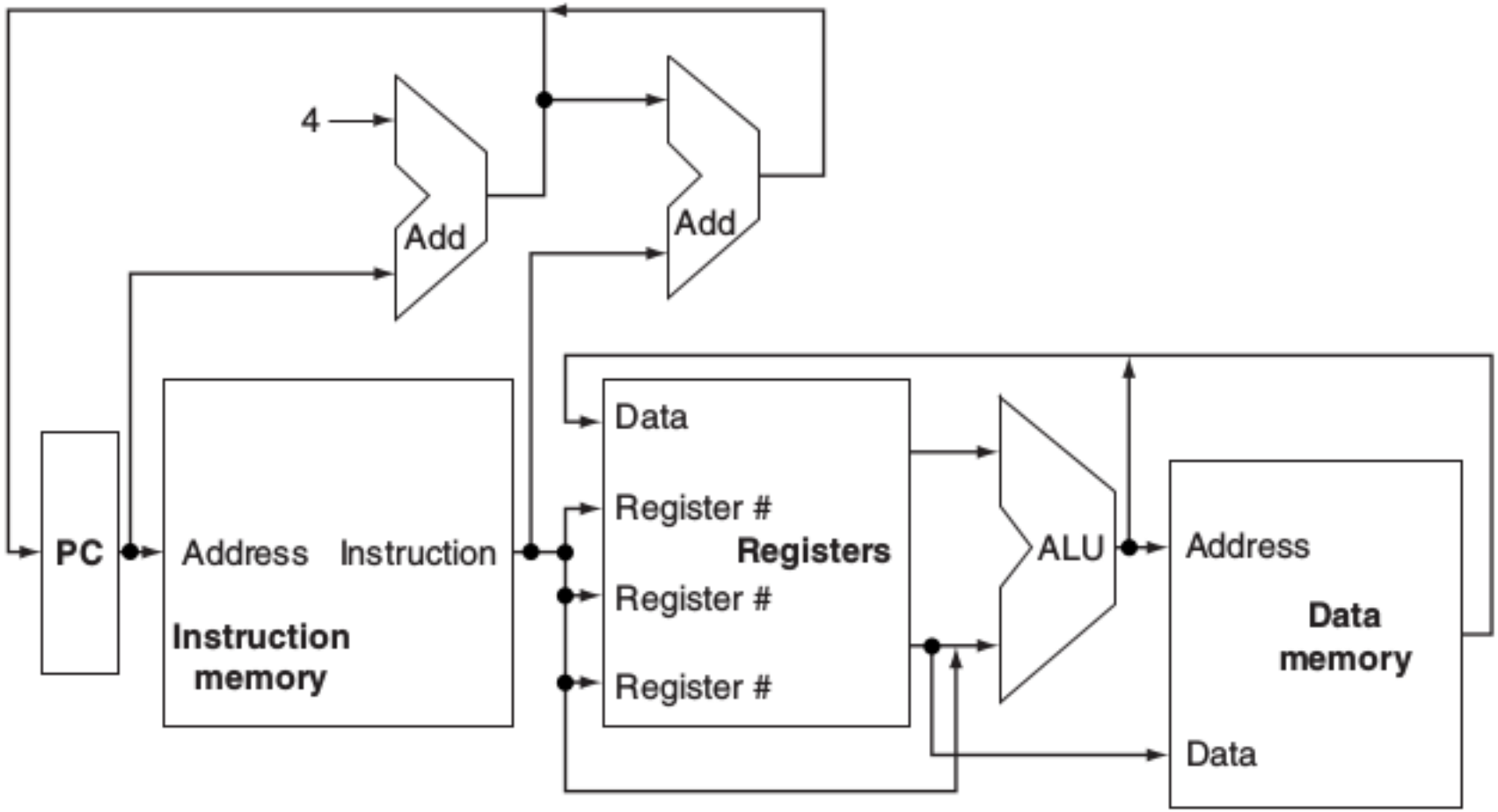
D *Decode*: Se decodifica la instrucción y se toman los operandos del banco de registros o tomados del bypass.

X *Execute*: Se realiza la operacion especificada por la instrucción. Para instrucciones simples (sumas y operaciones logicas), la ejecución finaliza en esta etapa, y el resultado se hace disponible para el bypass.

M *Memory*: Para instrucciones más complejas como acceso a memoria externa, multiplicación, corrimiento, división, es necesaria otra etapa.

D *Write back*: Los resultados producidos por la instrucción son escritas al banco de registros.

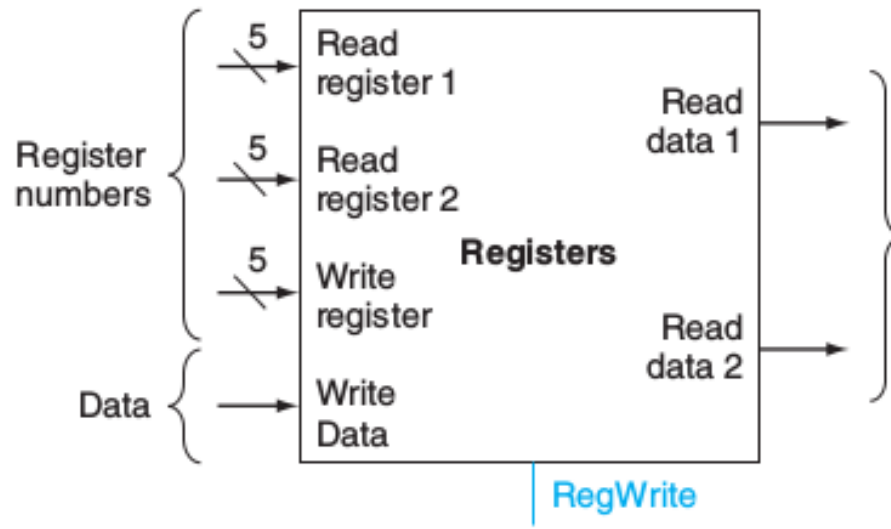
Banco de Registros



Banco de Registros (Introducción)

El banco de registros (register file) es un conjunto de registros para guardar y leer datos.

1. Cada registro es un vector de flip-flops D.
2. Para leer un registro:
   - 2.1 Entrada: número de registro.
   - 2.2 Salida: dato contenido en el registro.
3. Para escribir un registro:
   - 3.1 Entrada: número de registro, dato y una señal de reloj para controlar la escritura.



Banco de Registros

El LM32 posee 32 registros de 32 bits

1. Registro *r0*
2. 8 registros (r1 a r7) son utilizados para paso de argumentos y retorno de resultados en llamados a funciones.
3. Los registros *r1 - r28* pueden ser utilizados como fuente o destino de cualquier instrucción.
4. Los registros *r26 (gp) r27 (fp)* y *r28 (sp)* son el puntero global, de frame y de pila respectivamente.
5. El registro *r29 (ra)*, la instrucción *call* para almacenar la dirección de retorno.
6. El registro *r30 (ea)*, almacenar el valor del *contador de programa* cuando se presenta una excepción.
7. El registro *r31 (ba)*, almacena el valor del contador de programa cuando se presenta una excepción tipo *breakpoint* o *watchpoint*.

Después del reset los 32 bits de los registros quedan indefinidos, por lo que la primera acción que debe ejecutar el programa de inicialización es asegurar un cero en el registro *r0*.

Registro de estado y control (CSR)

Nombre	**Index**	**Descripción**
PC		Contador de Programa
IE	0x00	(R/W)Interrupt enable
EID	—-	(R) Exception ID
IM	0x01	(R/W)Interrupt mask
IP	0x02	(R) Interrupt pending
ICC	0x03	(W) Instruction cache control
DCC	0x04	(W) Data cache control
CC	0x05	(R) Cycle counter
CFG	0x06	(R) Configuration
EBA	0x07	(R/W)Exception base address

Registro de estado y control (CSR)

Contador de Programa (PC)
Contiene la dirección de la instrucción que se ejecuta actualmente. Debido a que todas las instrucciones son de 32 bits, los dos bits menos significativos del PC siempre son zero. El valor de este registro después del reset es *h00000000*

IE Habilitación de interrupción
IE contiene la bandera IE, que determina si se habilitan o no las interrupciones. Si este flag se desactiva, no se presentan interrupciones a pesar de la activación individual realizada con IM. Existen dos bits *BIE* y *EIE* que se utilizan para almacenar el estado de IE cuando se presenta una excepción tipo breakpoint u otro tipo de excepción.

Registro de estado y control (CSR)

EID Exception ID

(3 bits) Indica la causa de la detención de la ejecución del programa.

- **0**: Reset; se presenta cuando se activa la señal de reset del procesador.
- **1**: Breakpoint; se presenta cuando se ejecuta la instrucción break o cuando se alcanza un punto de break hardware.
- **2**: Instruction Bus Error; se presenta cuando falla la captura en una instrucción, regularmente cuando la dirección no es válida.
- **3**: Watchpoint; se presenta cuando se activa un watchpoint.
- **4**: Data Bus Error; se presenta cuando falla el acceso a datos, típica mente porque la dirección solicitada es inválida o porque el tipo de acceso no es permitido.
- **5**: División por cero; Se presenta cuando se hace una división por cero.
- **6**: Interrupción; se presenta cuando un periférico solicita atención por parte del procesador. Para que esta excepción se presente se deben habilitar las interrupciones globales (IE) y la

IM Máscara de interrupción

La máscara de interrupción contiene un bit de habilitación para cada una de las 32 interrupciones, el bit 0 corresponde a la interrupción 0. Para que la interrupción se presente es necesario que el bit correspondiente a la interrupción y el flag IE sean igual a 1. Después del reset el valor de IM es *h00000000*

IP Interrupción pendiente

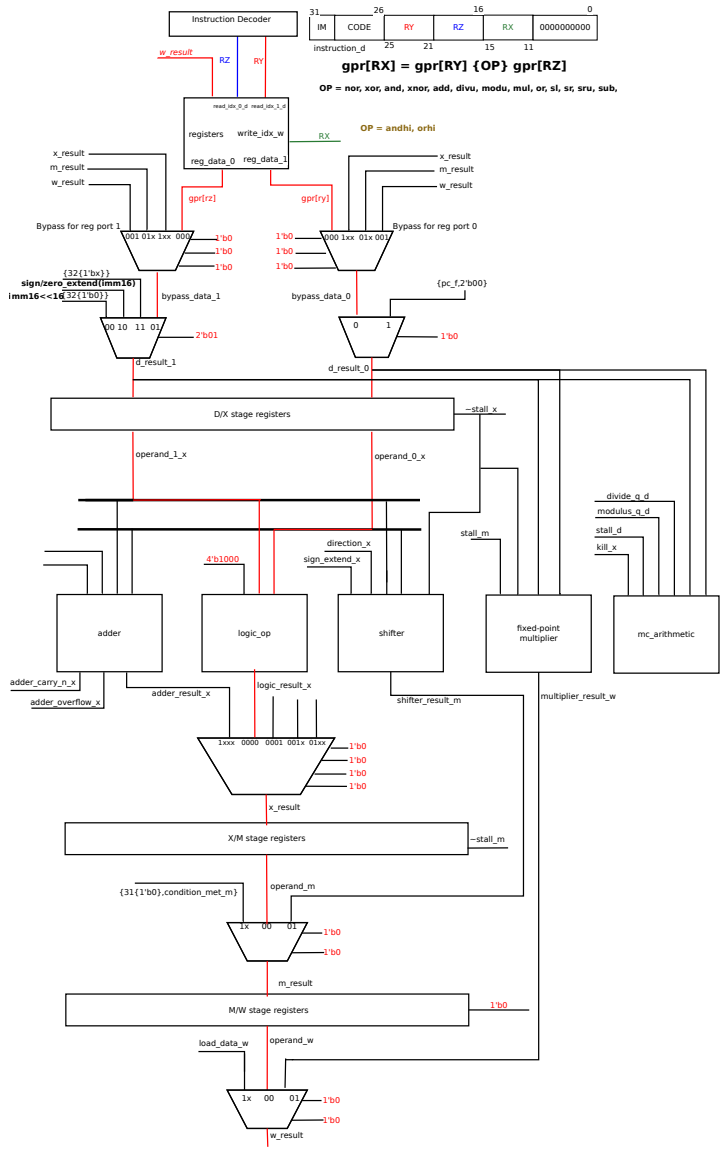
El registro IP contine un bit para cada una de las 32 interrupciones, este bit se activa cuando se presenta la interrupción asociada. Los bits del registro IP deben ser borrados escribiendo un 1 lógico.

Set de Instrucciones del procesador Mico32

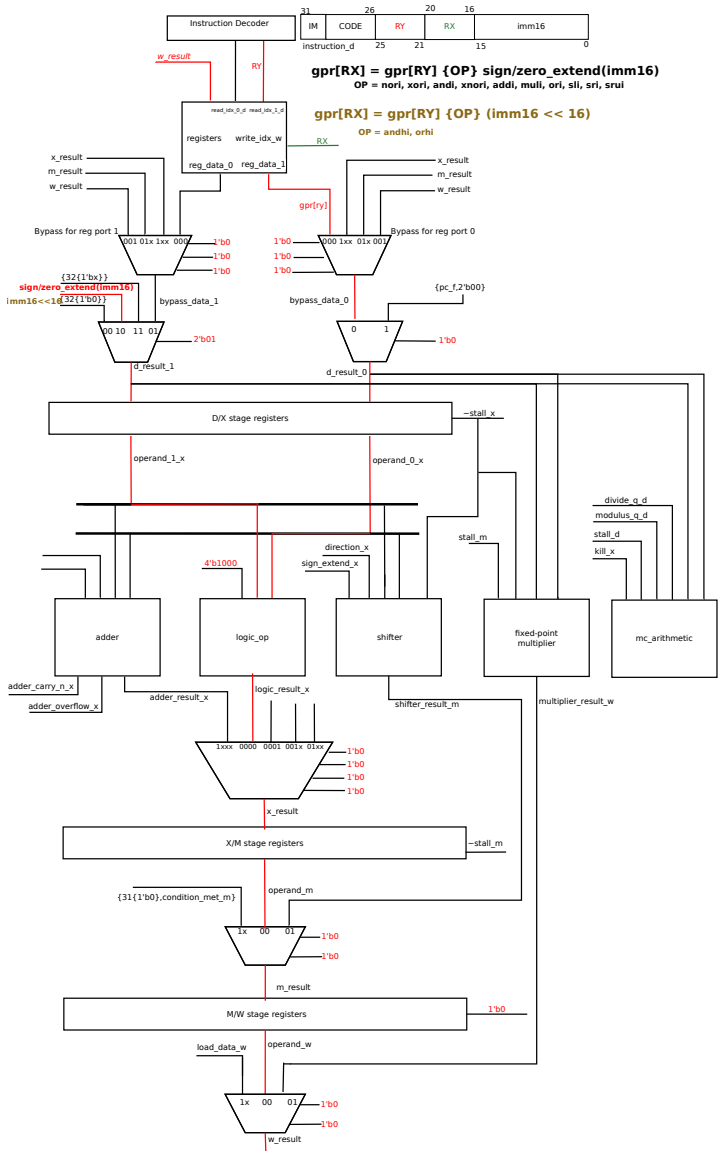
Tres (o cuatro)tipos de instrucciones:

- Instrucciones de referencia a memoria: load word y store word.
- Instrucciones aritmético-lógicas: *nor*, *xor*, *and*, *xnor*, *add*, *divu*, *modu*, *mul*, *or*, *sl*, *sr*, *sru*, *sub*
- Instrucciones de Salto: condicional y salto incondicional.
- interrupciones y excepciones

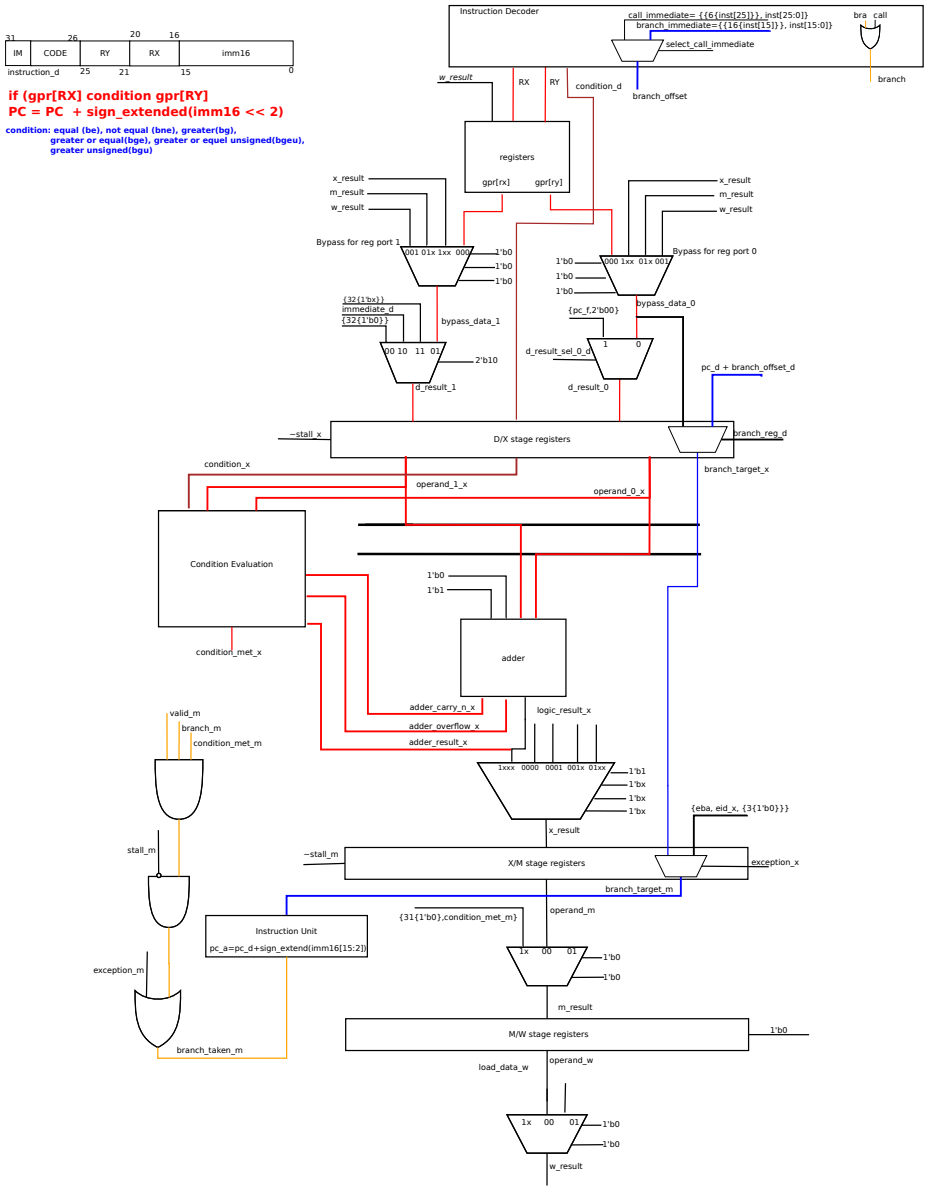
Camino de datos de las operaciones aritméticas y lógicas entre registros



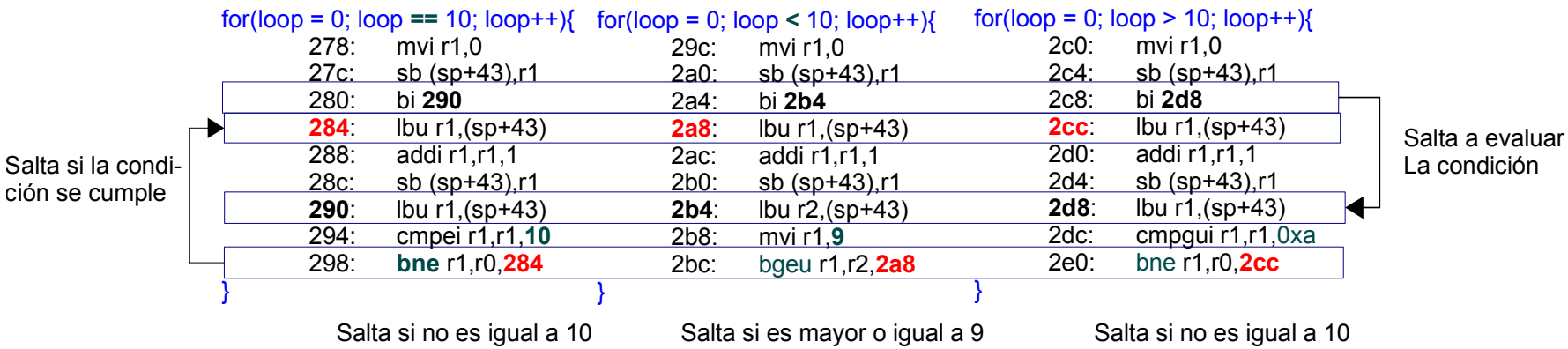
Camino de datos de las operaciones aritméticas y lógicas Inmediatas



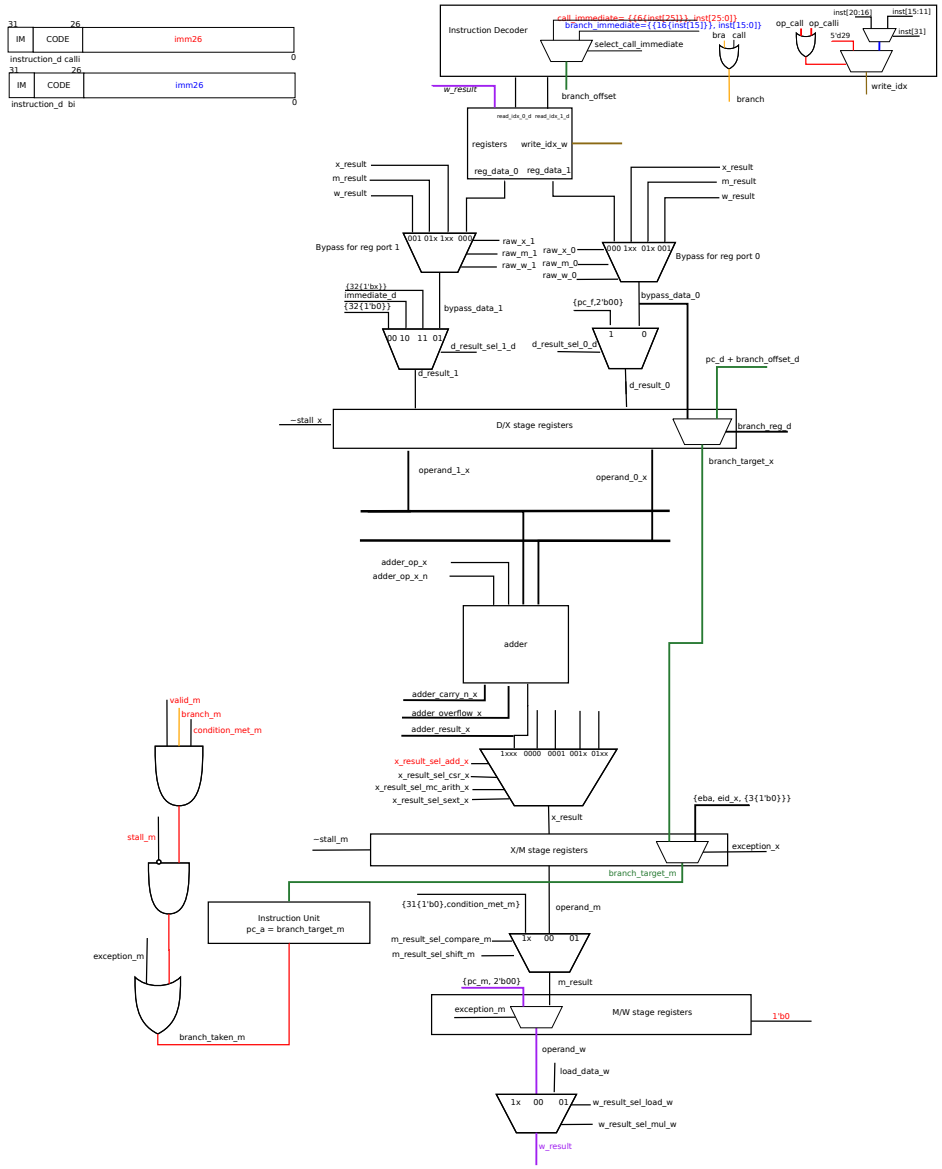
Saltos condicional



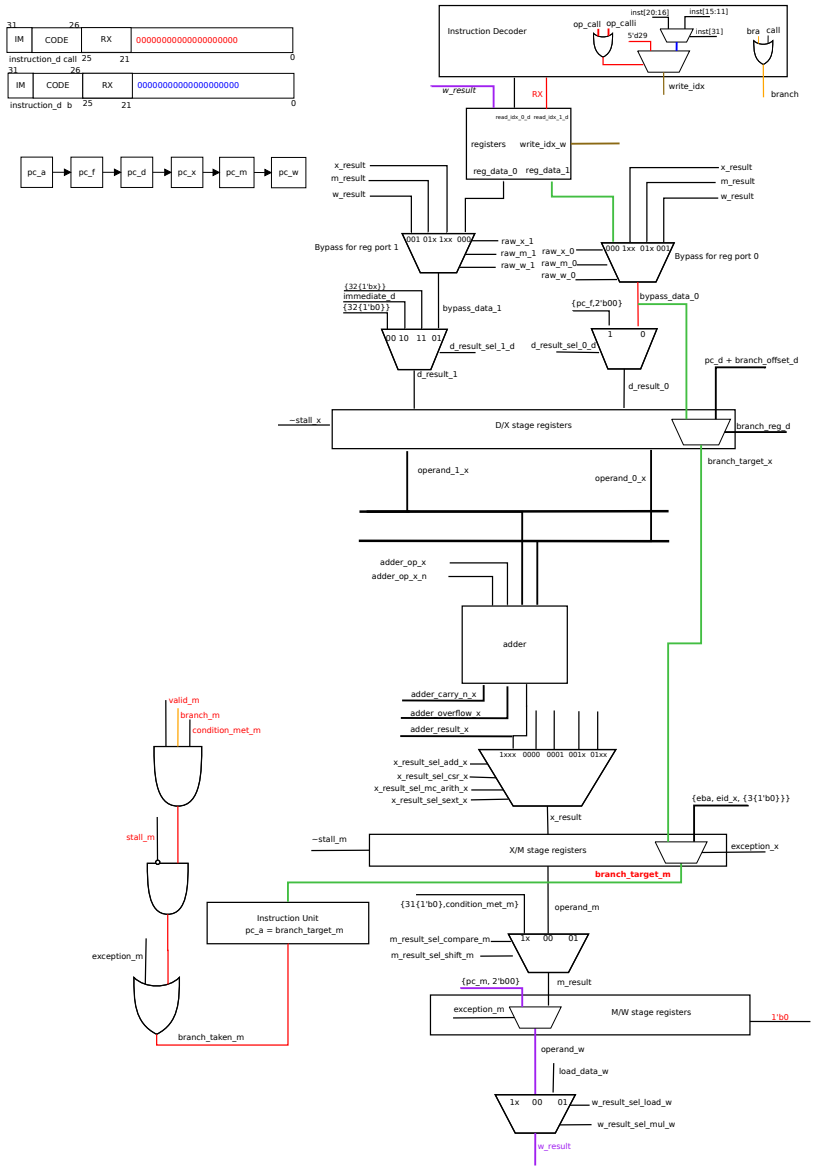
Saltos condicional ejemplo

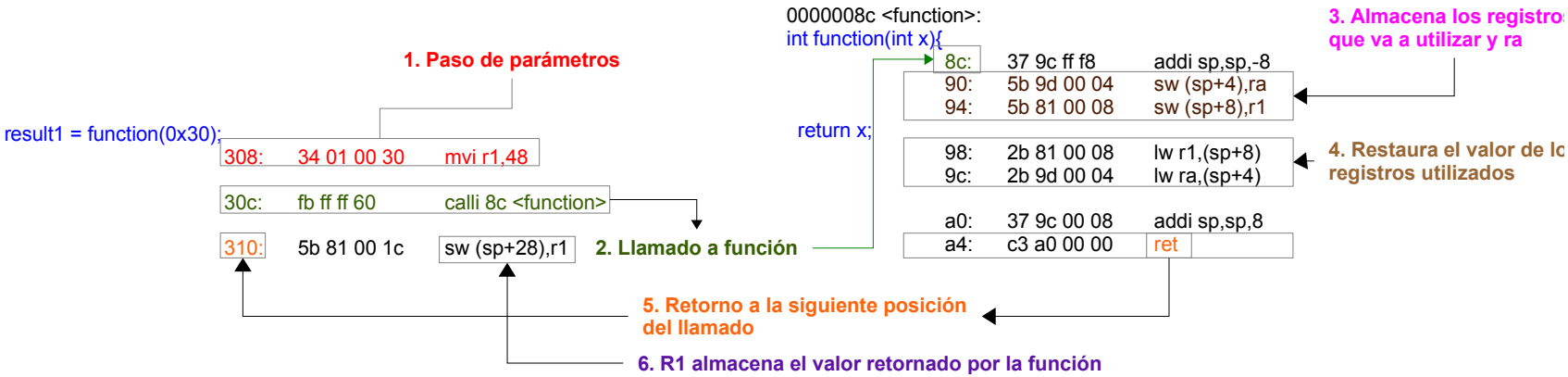


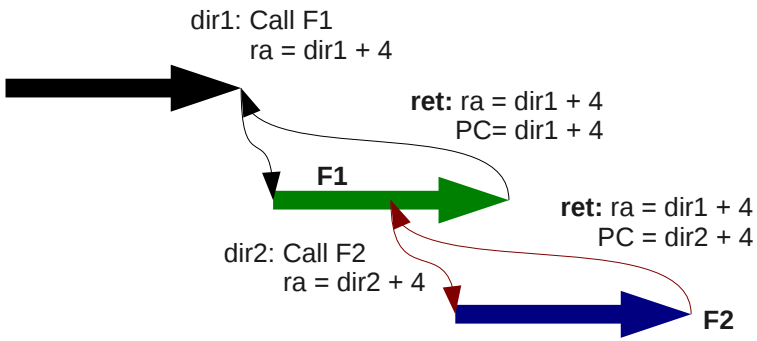
Camino de datos de los saltos y llamado a funciones inmediatos



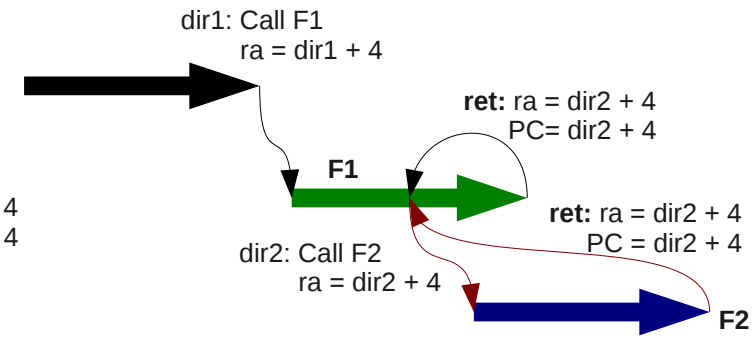
Camino de datos de los saltos y llamado a funciones







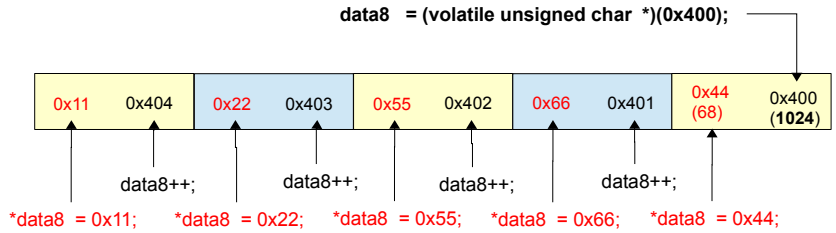
Almacenando ra



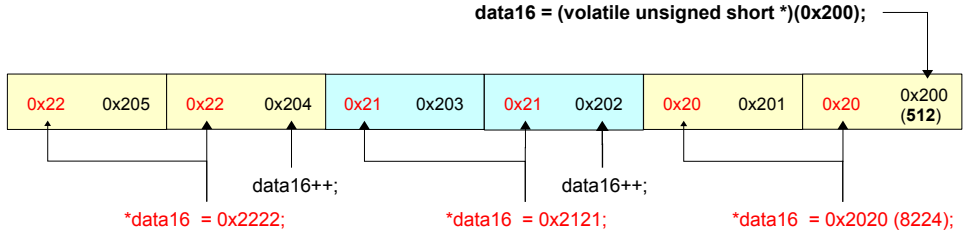
Sin almacenar ra

Data Types

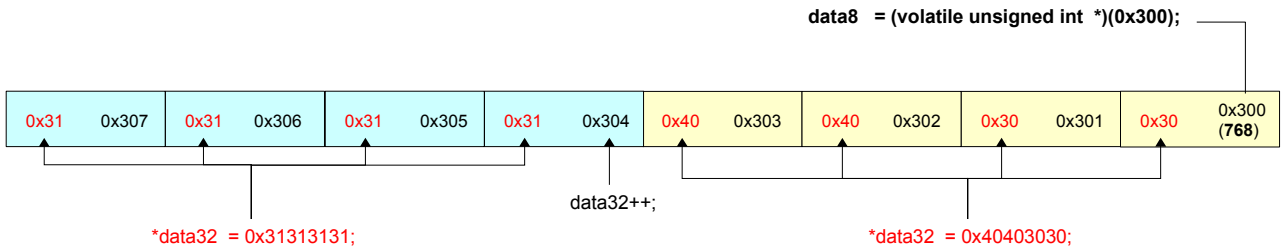
C data type	
Char	8-bit
Short	Signed 16-bit
Int	Signed 32-bit
Long	Signed 32-bit
Long long	Signed 64-bit




```
*data8 = 0x44;
8c:   34 02 00 44    mvi r2,68
90:   34 01 04 00    mvi r1,1024
94:   30 22 00 00    sb (r1+0),r2
```

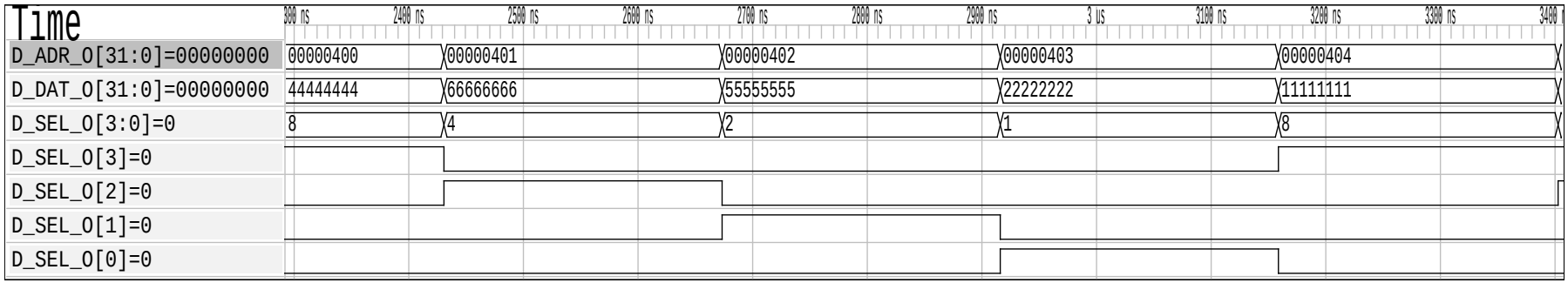


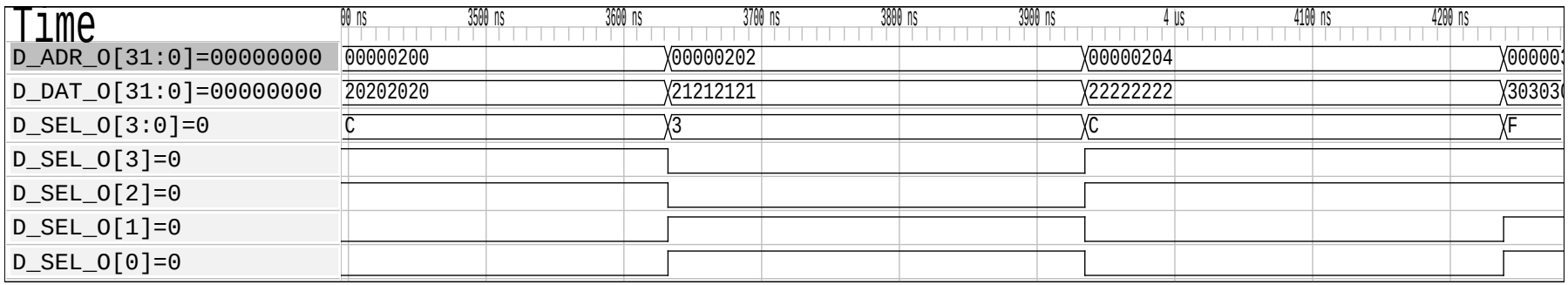

```
*data16 = 0x2020;
c8:   34 02 20 20    mvi r2,8224
cc:   34 01 02 00    mvi r1,512
d0:   0c 22 00 00    sh (r1+0),r2
```

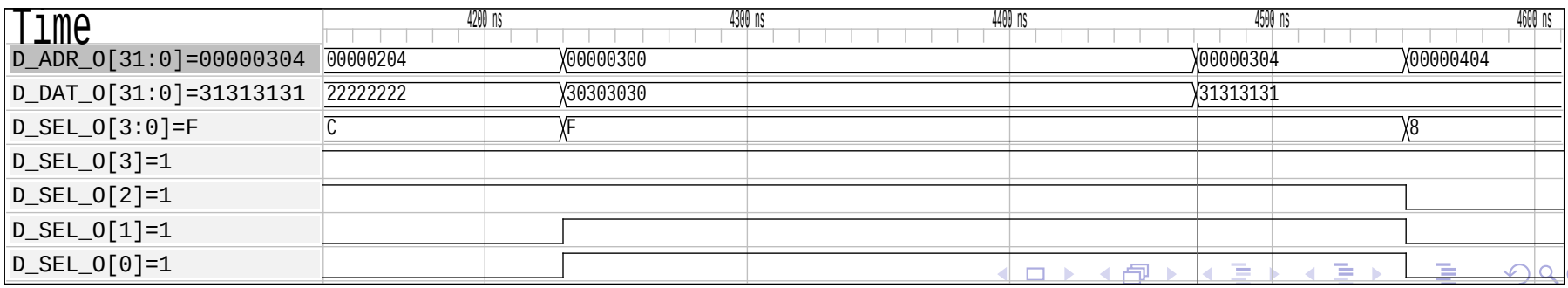



```
*data32 = 0x40403030;
e8:   78 02 40 40    mvhi r2,0x4040
f0:   38 42 30 30    ori r2,r2,0x3030
f8:   34 03 03 00    mvi r3,768
fc:   58 62 00 00    sw (r3+0),r2
```

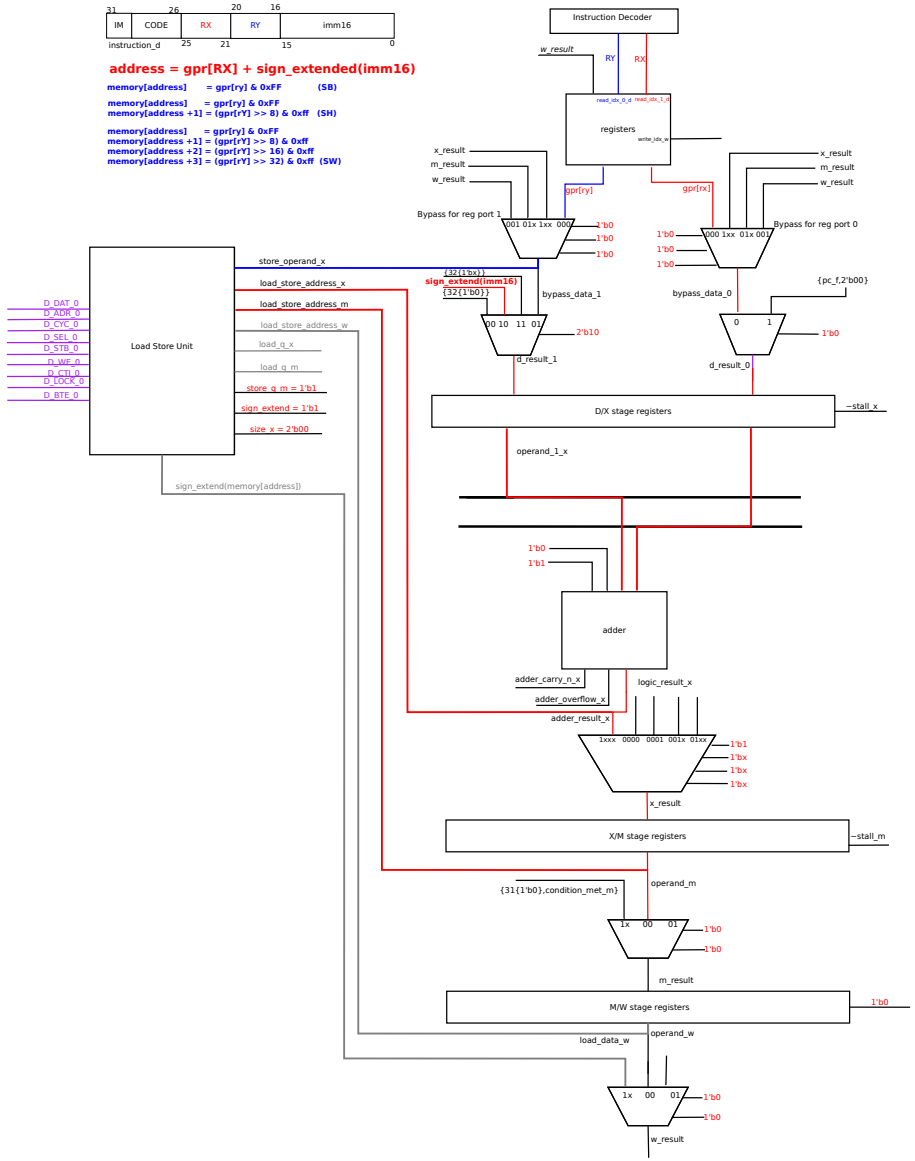
Accesos a Memoria



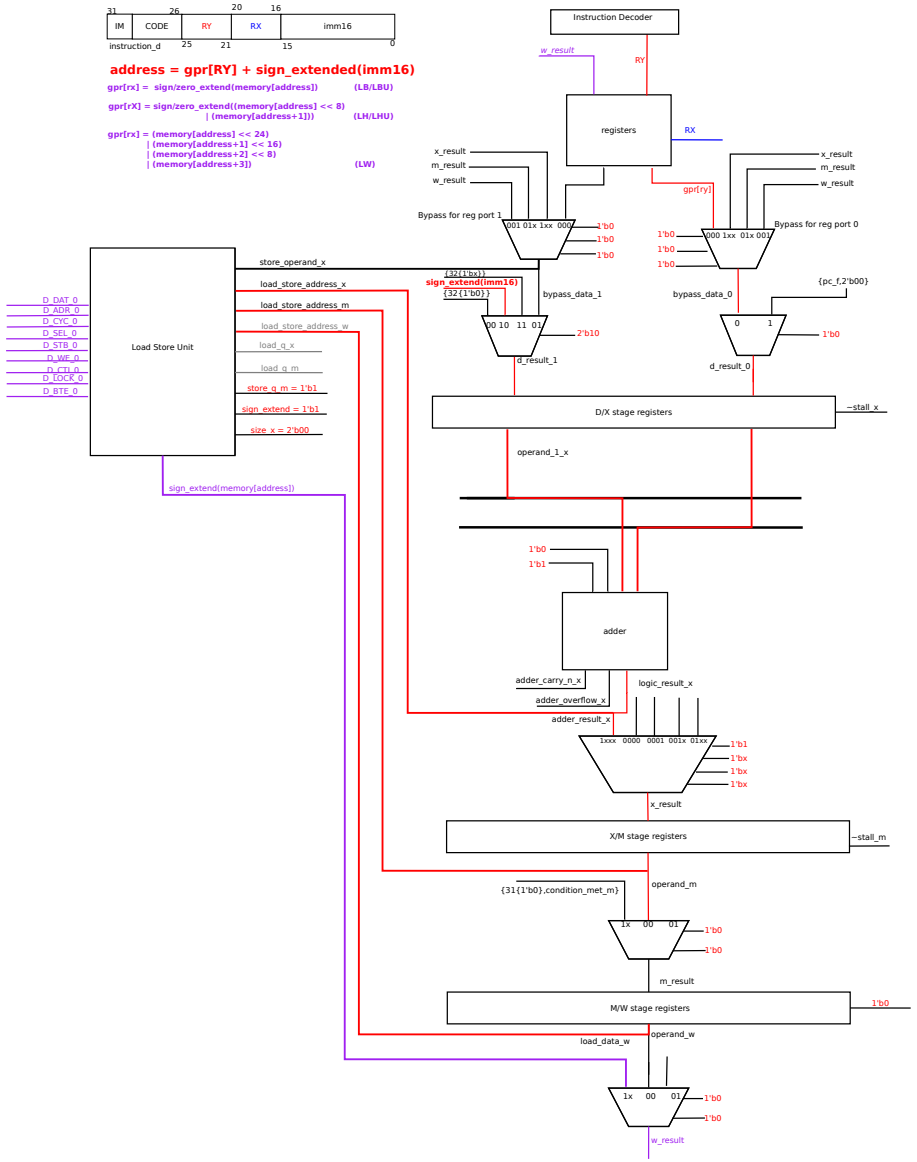




Escritura de memoria



Lectura


IM
CODE
RY
RX
imm16
instruction_d
address = gpr[RY] + sign_extended(imm16)
gpr[rx] = sign/zero_extend(memory[address]) (LB/LBU)
gpr[rX] = sign/zero_extend((memory[address] << 8) | (memory[address+1])) (LH/LHU)
gpr[rx] = (memory[address] << 24) | (memory[address+1] << 16) | (memory[address+2] << 8) | (memory[address+3]) (LW)
Instruction Decoder
registers
Load Store Unit
Bypass for reg port 1
Bypass for reg port 0
bypass_data_1
bypass_data_0
D/X stage registers
adder
X/M stage registers
M/W stage registers
operand_1_x
operand_m
operand_w
load_data_w
adder_carry_n_x
adder_overflow_x
adder_result_x
logic_result_x
x_result
m_result
w_result
d_result_1
d_result_0
store_operand_x
load_store_address_x
load_store_address_m
load_store_address_w
load_q_x
load_q_m
store_q_m = 1'b1
sign_extend = 1'b1
size_x = 2'b00
sign_extend(memory[address])
D_DAT_0
D_ADR_0
D_CYC_0
D_SEL_0
D_STB_0
D_WE_0
D_CTI_0
D_LOCK_0
D_BTE_0
~stall_x
~stall_m

Índice

Jerarquía de Memoria

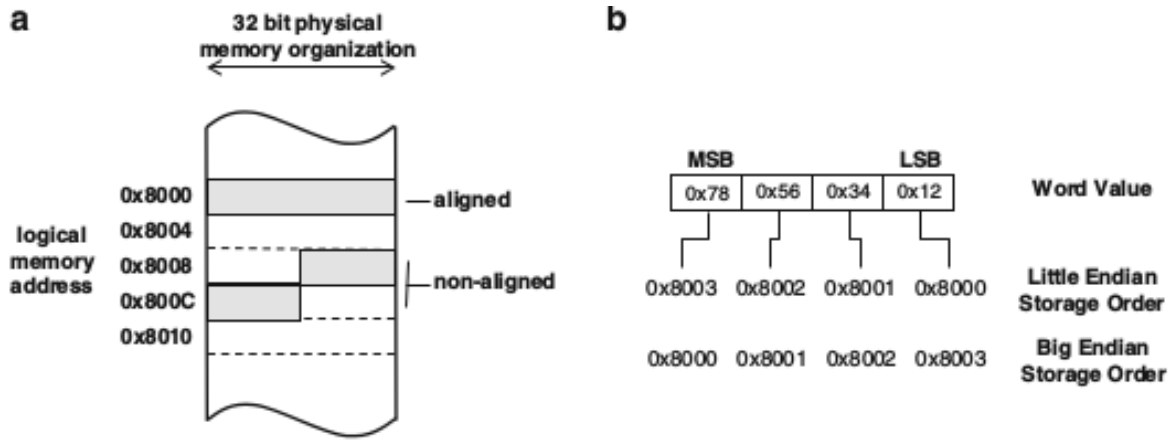
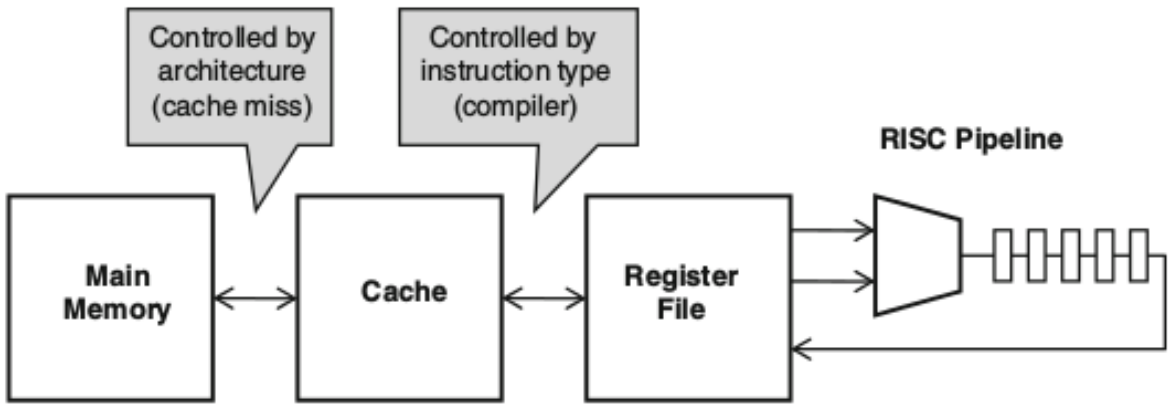
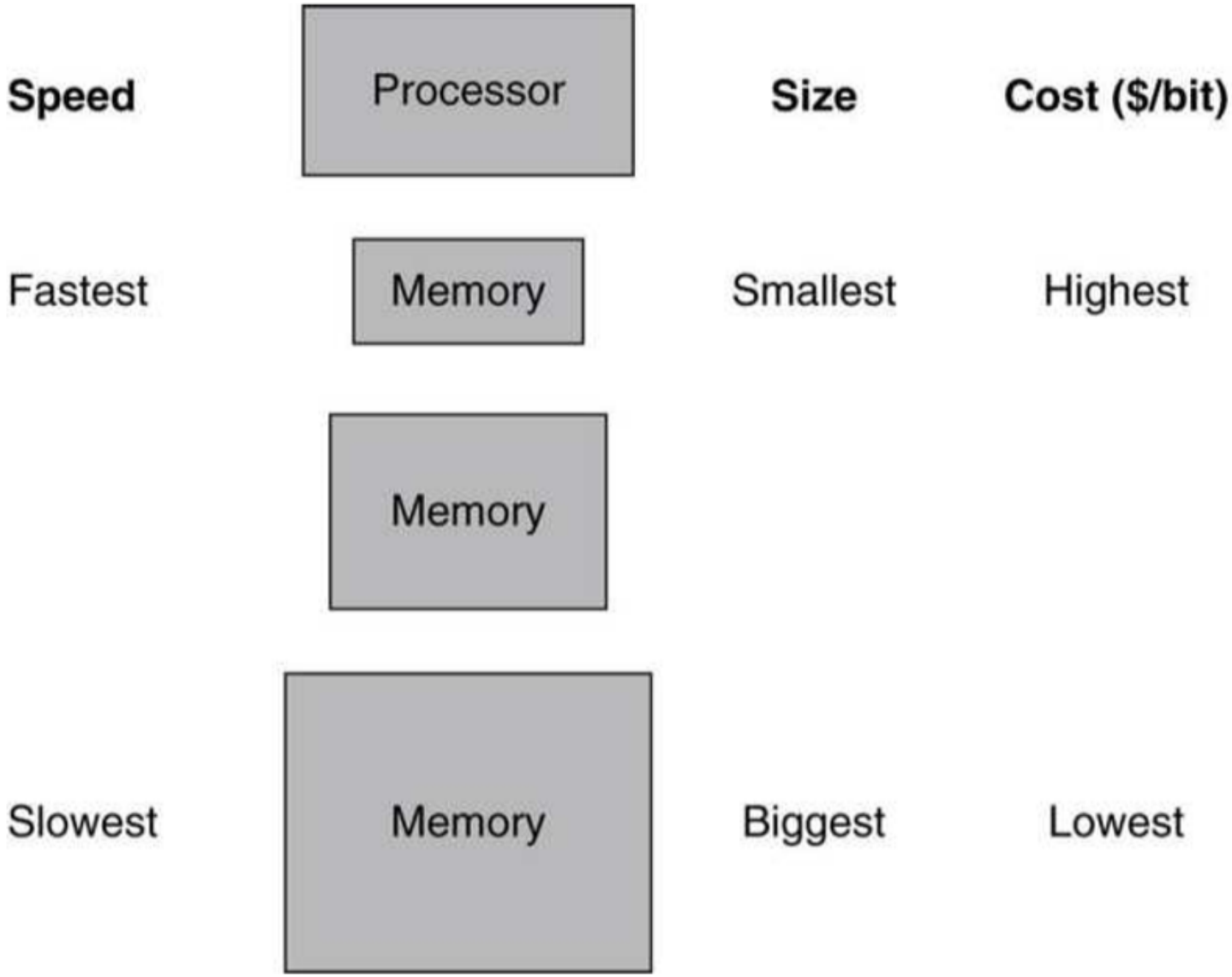


Fig. 7.7 (a) Alignment of data types. (b) Little-endian and Big-endian storage order

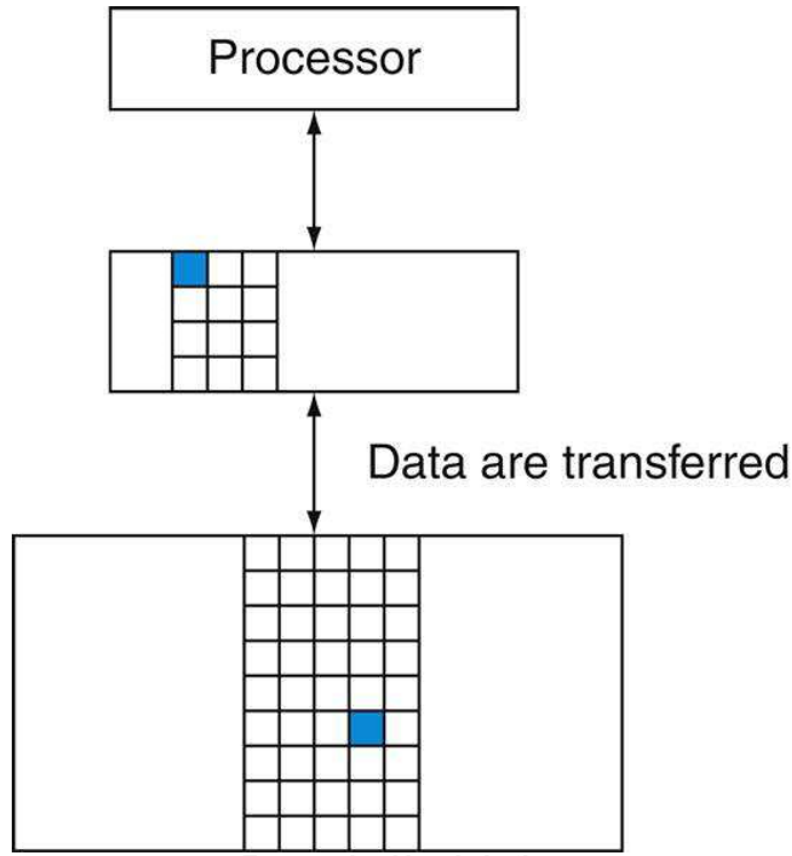


Jerarquía de Memoria

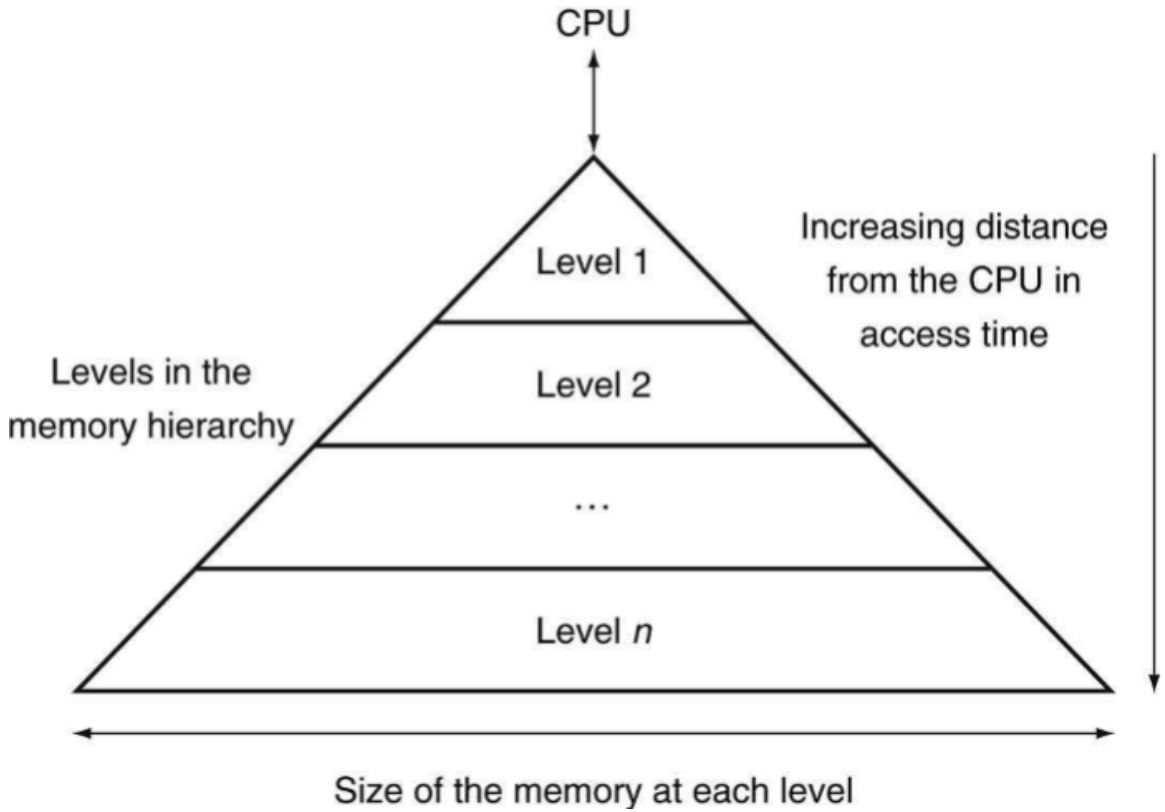


Jerarquía de Memoria

1. Blocks: Unidad mínima de información que puede estar presente o no en un caché.
2. Hit Rate: accesos a memoria que se encuentra en el nivel de la jerarquía de memoria.
3. Miss Rate: accesos de memoria que no se encuentra en la memoria

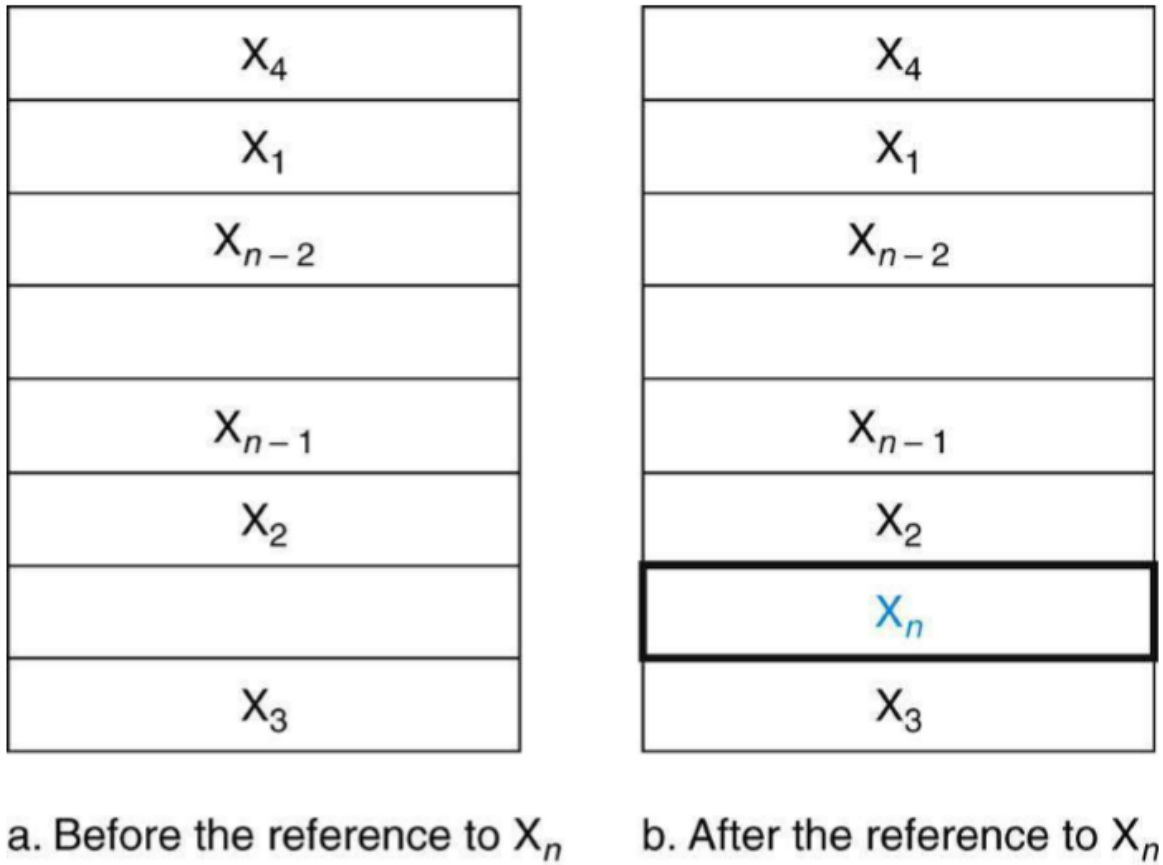


Jerarquía de Memoria



Hit time: tiempo requerido para acceder al nivel de memoria y tiempo necesario para determinar si es exitoso p no el acceso.
miss penalty: tiempo requerido para recuperar un bloque del nivel inferior + tiempo para acceder al bloque + tiempo transmisión de un nivel a otro + tiempo de insertar en el nivel que experimentó la falla.

Cache

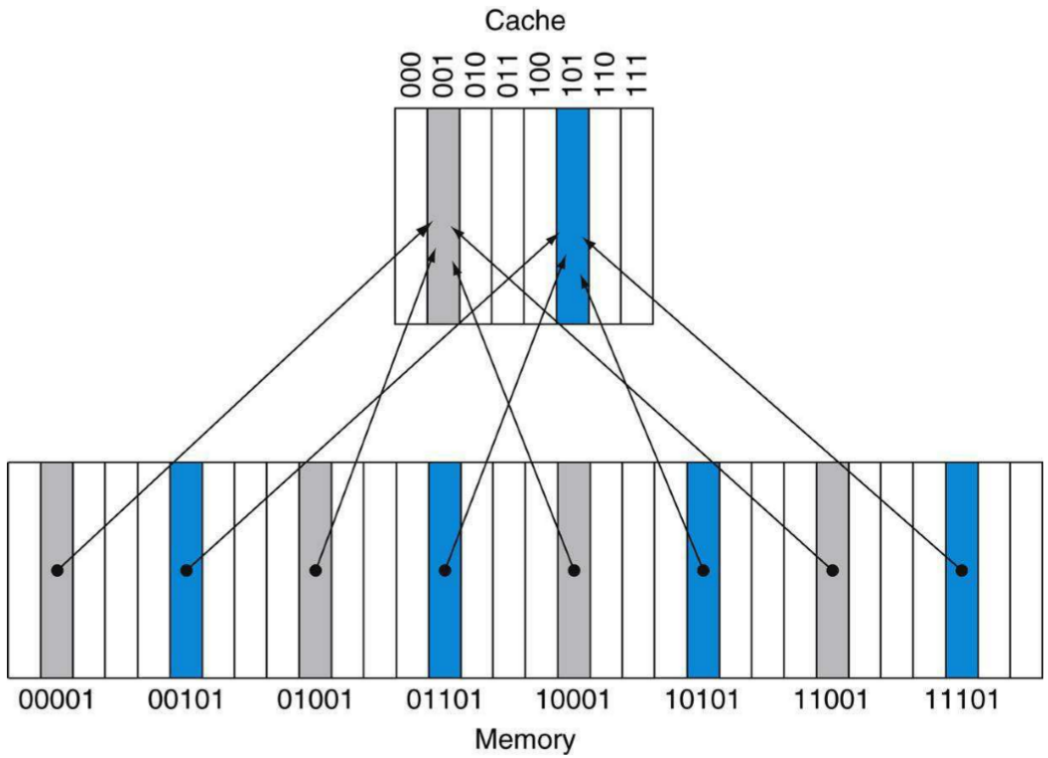


a. Before the reference to X_n b. After the reference to X_n

¿Cómo sabemos si un elemento de datos está en la memoria caché? ¿cómo lo encontramos?

direct-mapped cache

Cada ubicación de memoria se asigna exactamente a una ubicación de la memoria caché.

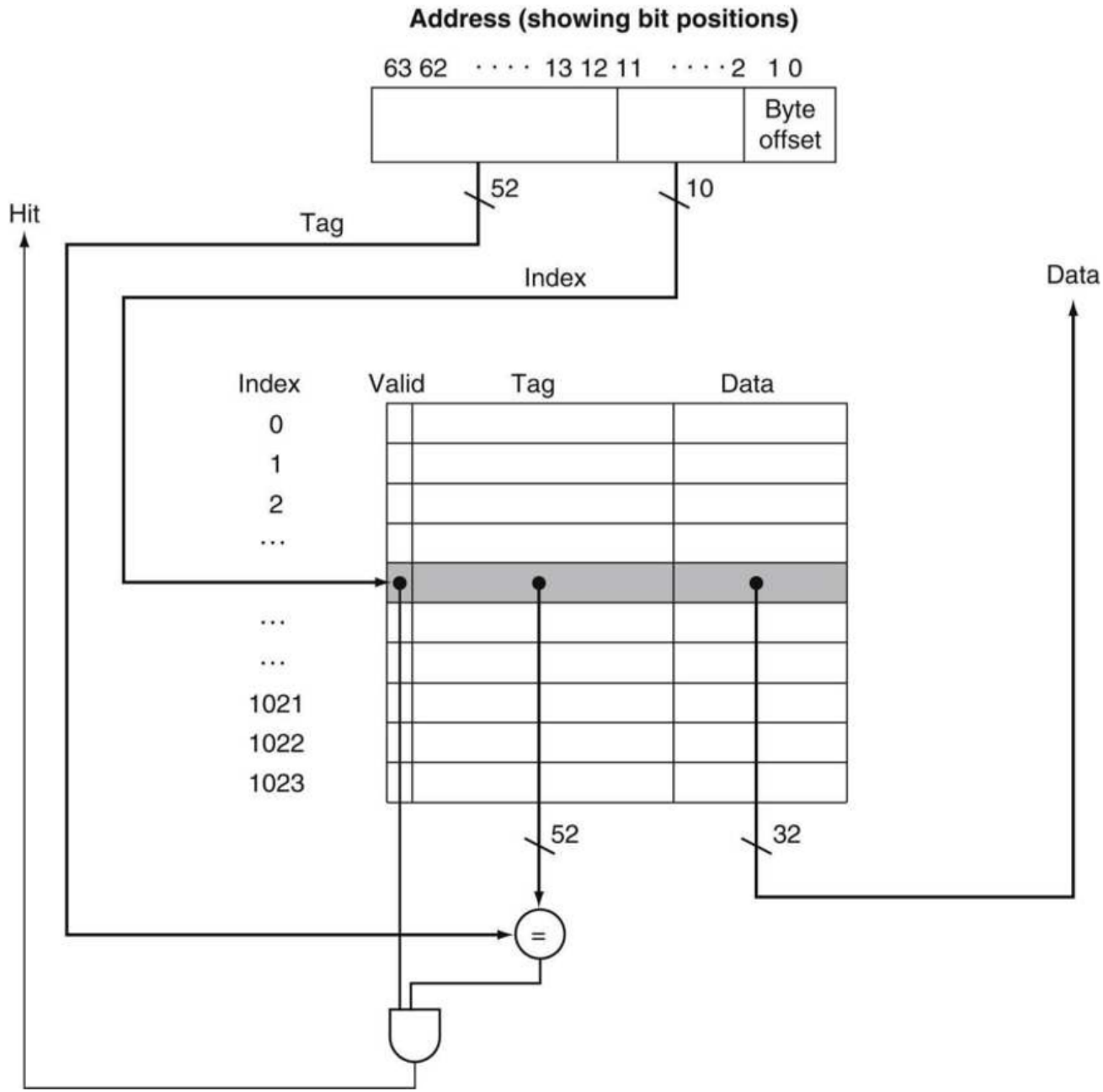


¿cómo sabemos si una palabra solicitada está en la memoria caché o no?

direct-mapped cache

1. **Tag**: utilizada para identificar si una jerarquía de memoria contiene el bloque asociado a la palabra solicitada en la dirección. ejm: 00011011
2. **valid bit**: indica que el bloque asociado en la jerarquía contiene datos válidos

Lectura Cache



Lectura Cache Ejercicio

Decimal address of reference	Binary address of reference	Assigned cache block (where found or placed)
22	10110_{two}	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	$(10000_{two} \bmod 8) = 000_{two}$

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Número de bit de Cache (direct-mapped cache)

El número total de bits necesarios para la memoria caché es función del tamaño de la caché y el tamaño de la dirección. La memoria caché incluye el almacenamiento de los datos y las etiquetas:

1. El tamaño de la caché es 2^n bloques. Se utilizan n bits para el índice.
2. El tamaño del bloque es de 2^m palabras (2^{m+2} bytes), Se utilizan m bits para la palabra dentro del bloque y 2 bits para la dirección de byte
3. El tamaño del tag es $TamañoDirección - (n + m + 2)$

El número total de bits en una memoria caché de asignación directa es:

$$2^n(2^m \text{ X } 32 + (64 - n - m - 2) + 1)$$

con $TamañoDirección = 64$

Ejercicio: Número de bit de Cache (direct-mapped cache)

¿Cuántos bits totales se requieren para la caché de asignación directa, si se requiere almacenar 16 KB de datos y cada bloques es de 4 palabras?. Suponga una dirección de 64 bits

Solución: Número de bit de Cache (direct-mapped cache)

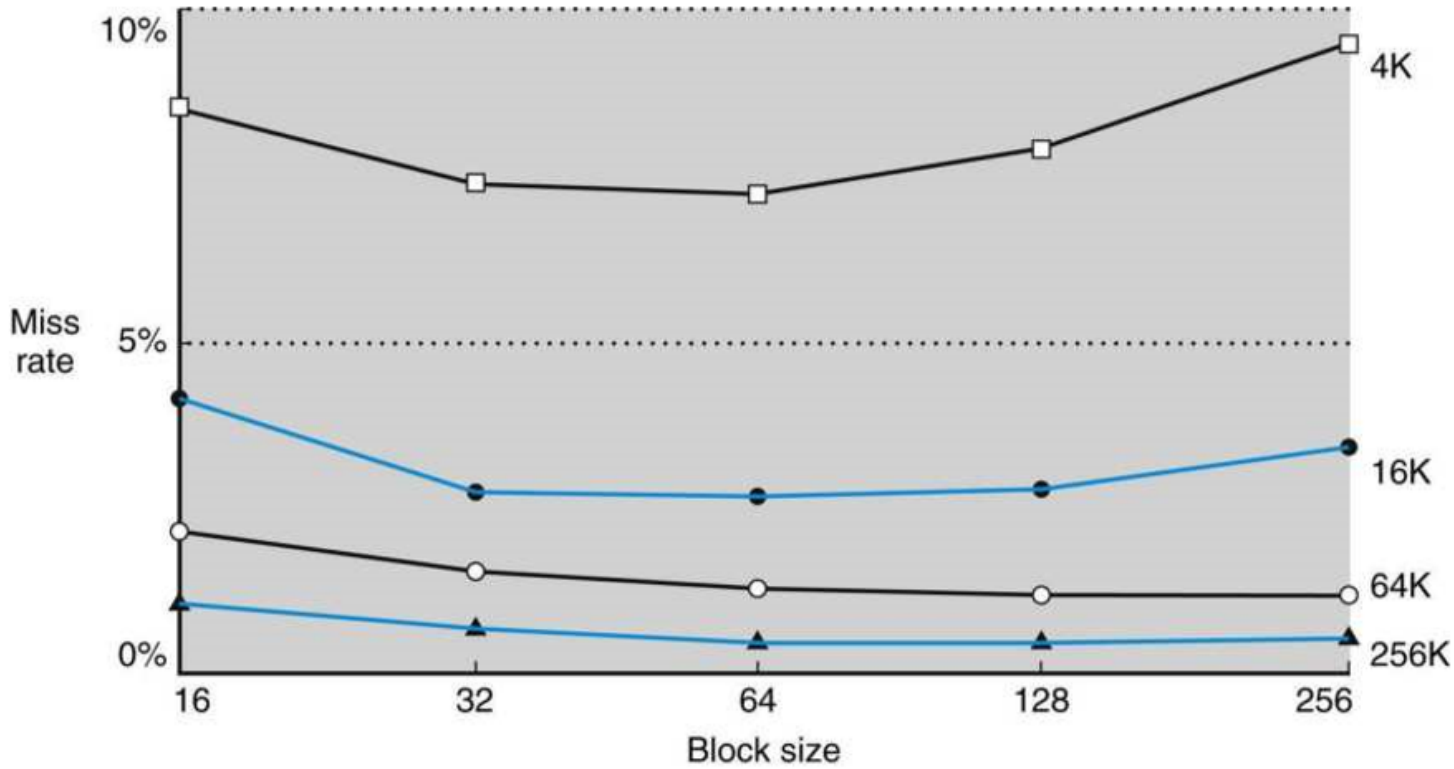
¿Cuántos bits totales se requieren para la caché de asignación directa, si se requiere almacenar 16 KB de datos y cada bloques es de 4 palabras?. Suponga una dirección de 64 bits

1. 16 kB = 4096 words (2^12).
2. 1 block = 4 words (2^2). $4X32 = 128bits$
3. total blocks = 1024 blocks (2^10).
4. tamaño de tag = $64 - 10 - 2 - 2 = 50bits$.

El tamaño total de bits para la cache de 16KB es:

$$2^{10}(128 + 50 + 1) = 2^{10}179 = 179Kbit = 22{,}375KB$$

Que pasa si se aumenta el tamaño de los blocks



A priori se evidencia disminución de miss pero ...
se aumenta la penalidad de fallo. Mayor latencia.

Recuerde: miss penalty es tiempo requerido para recuperar un bloque del nivel inferior y cargarlo en la memoria cache.

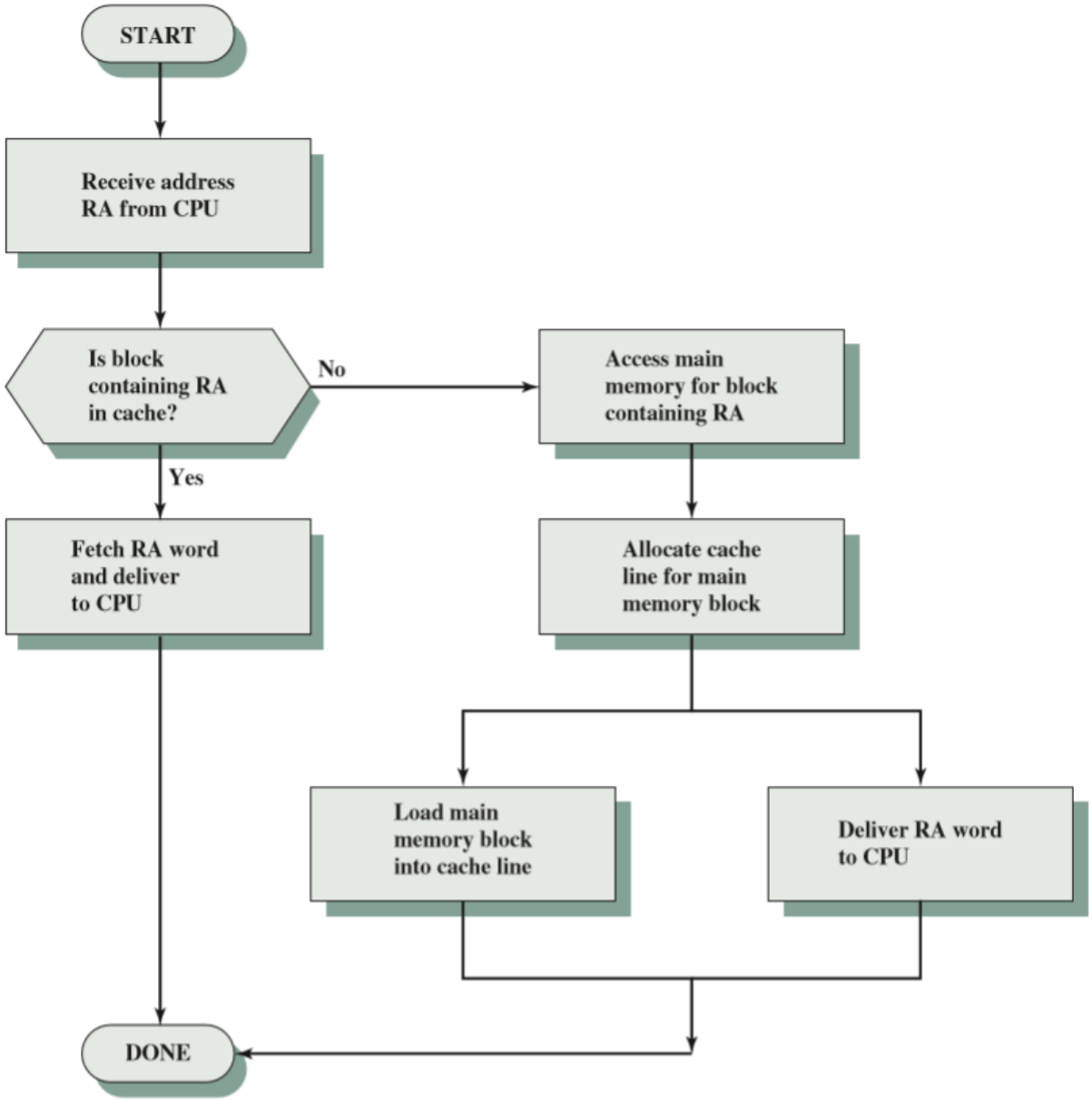
Ej: ¿Cómo mejorar la latencia? early restart, requested word first

Manejo de Cache Miss

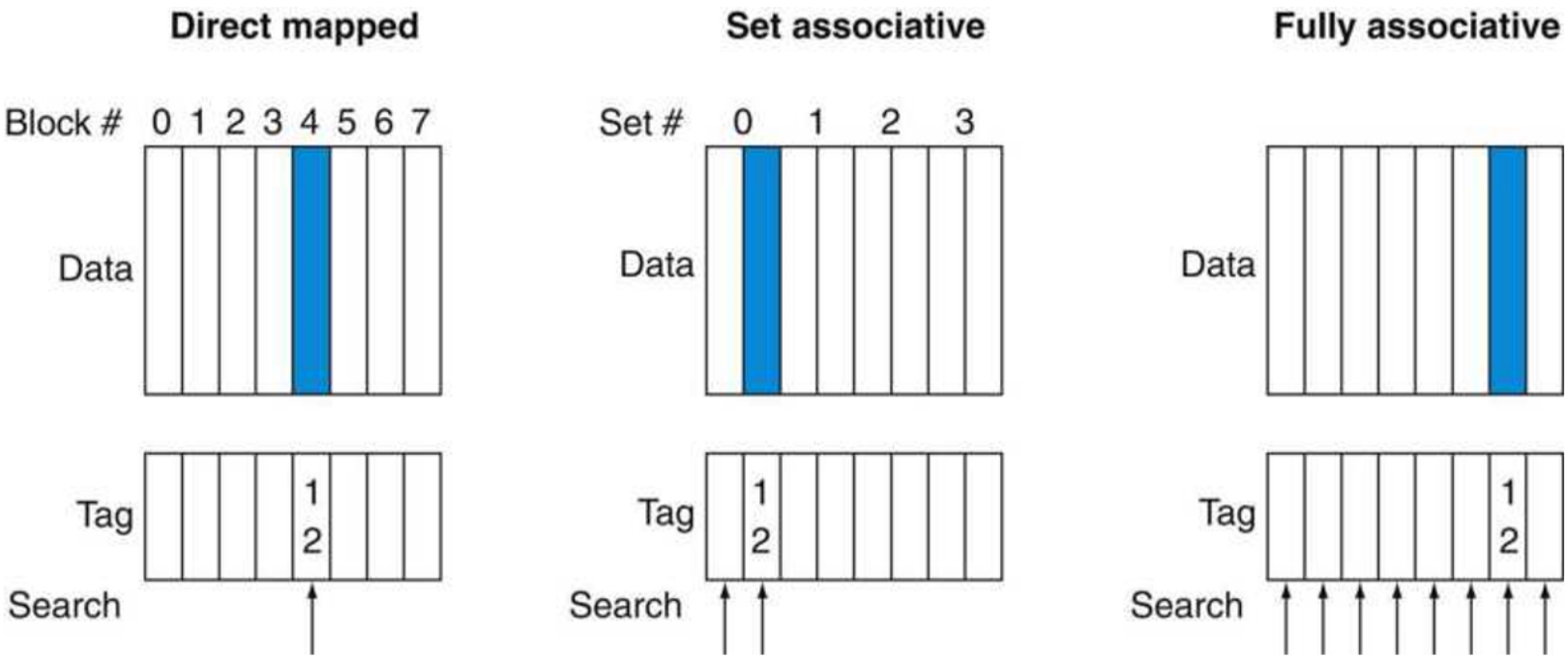
Básicamente se detiene el procesador (pipeline stall) hasta que la memoria responda con las instrucciones y datos.

1. Envíe el valor del PC a la memoria (anterior).
2. la UC solicita lectura de la instrucción a la memoria principal y espere a que la memoria complete su acceso.
3. se Escribe los datos en la caché, el tag (desde la ALU) y se activa el bit válido.
4. Se retorna la ejecución de la instrucción en el primer paso, lo que recuperará la instrucción, esta vez con caché Hit.

Resumen lectura Cache

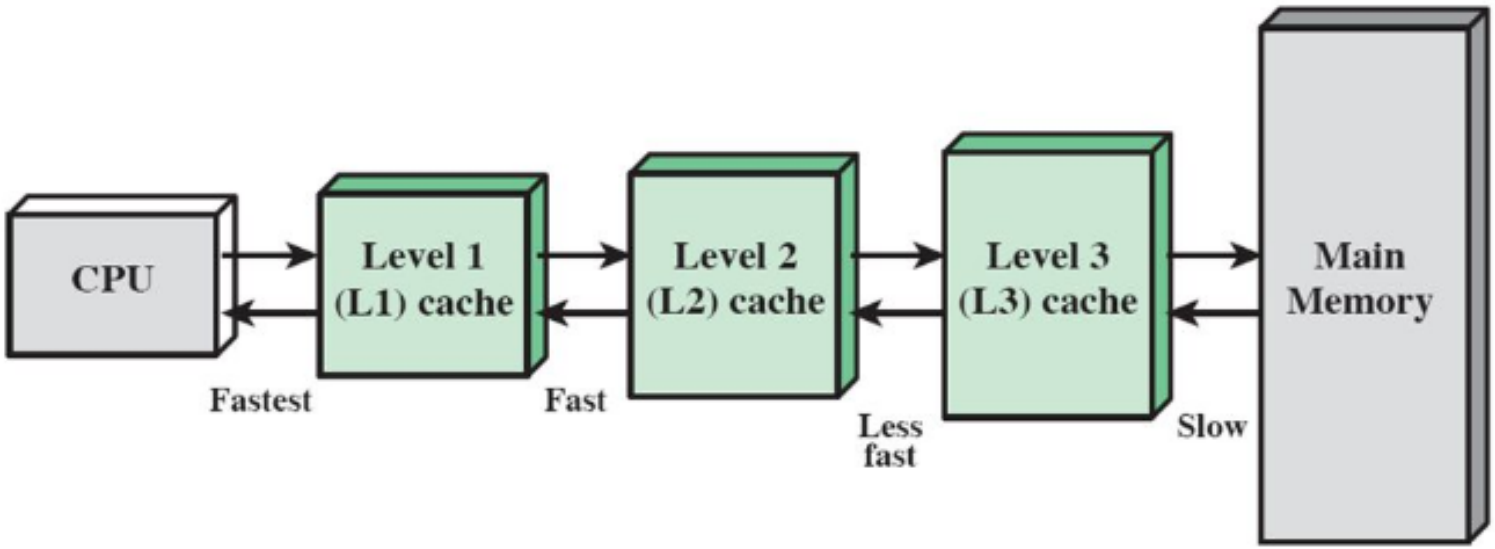


Técnicas de almacenamiento en Cache



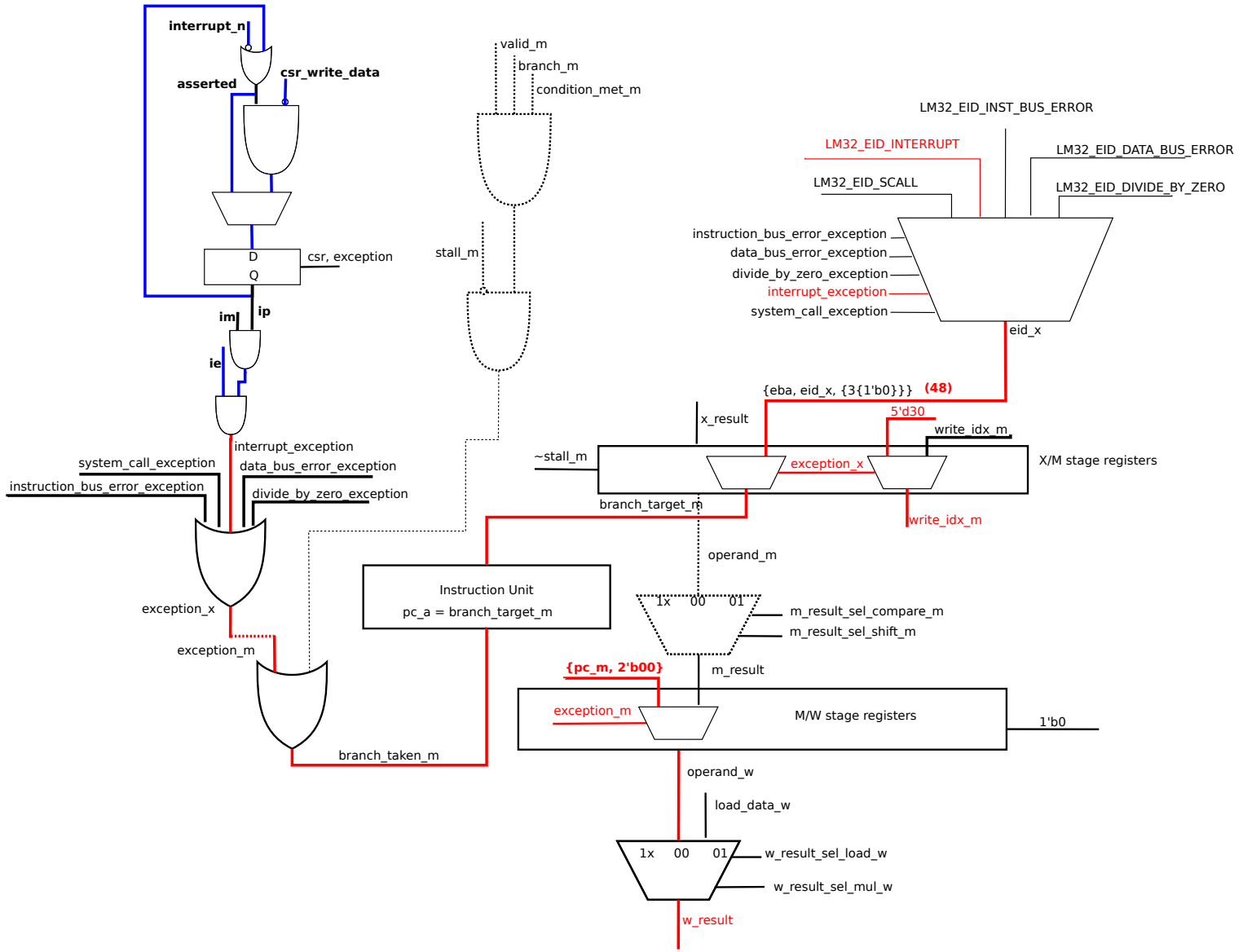
Ej: Buscar ventajas o desventajas

Niveles de Cache



Mejora la penalidad de fallos

Interrupciones

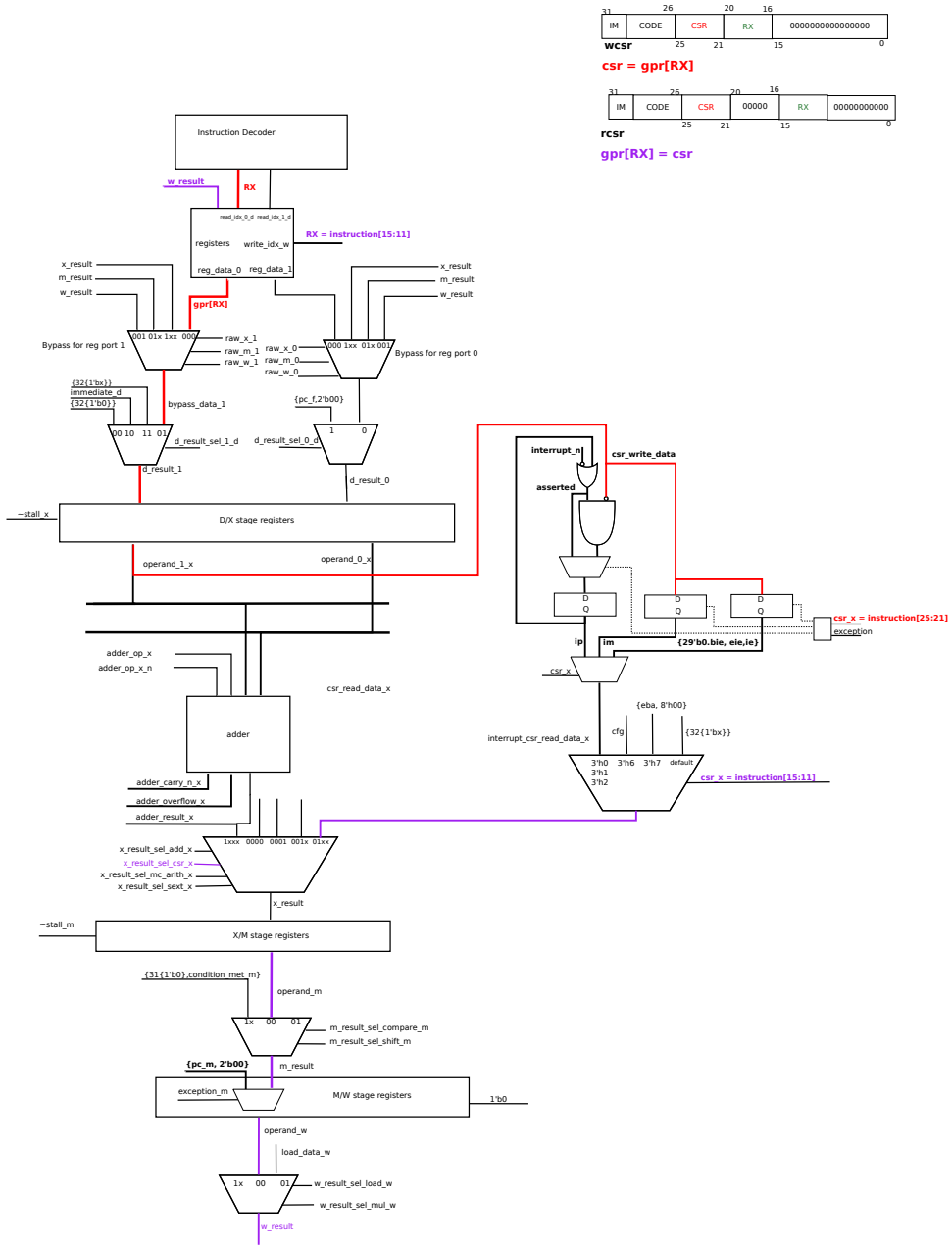


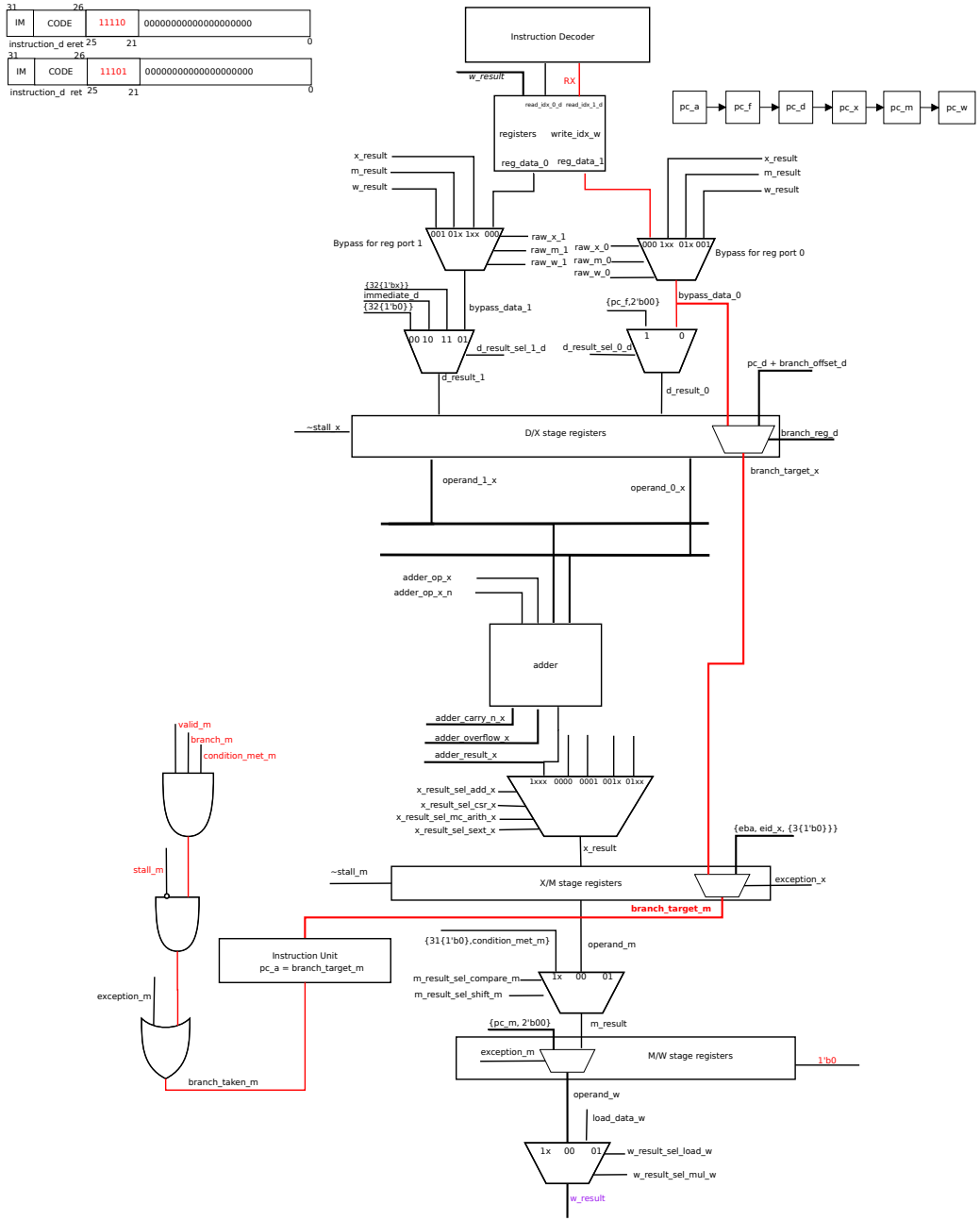
Rutina de atención a la interrupción

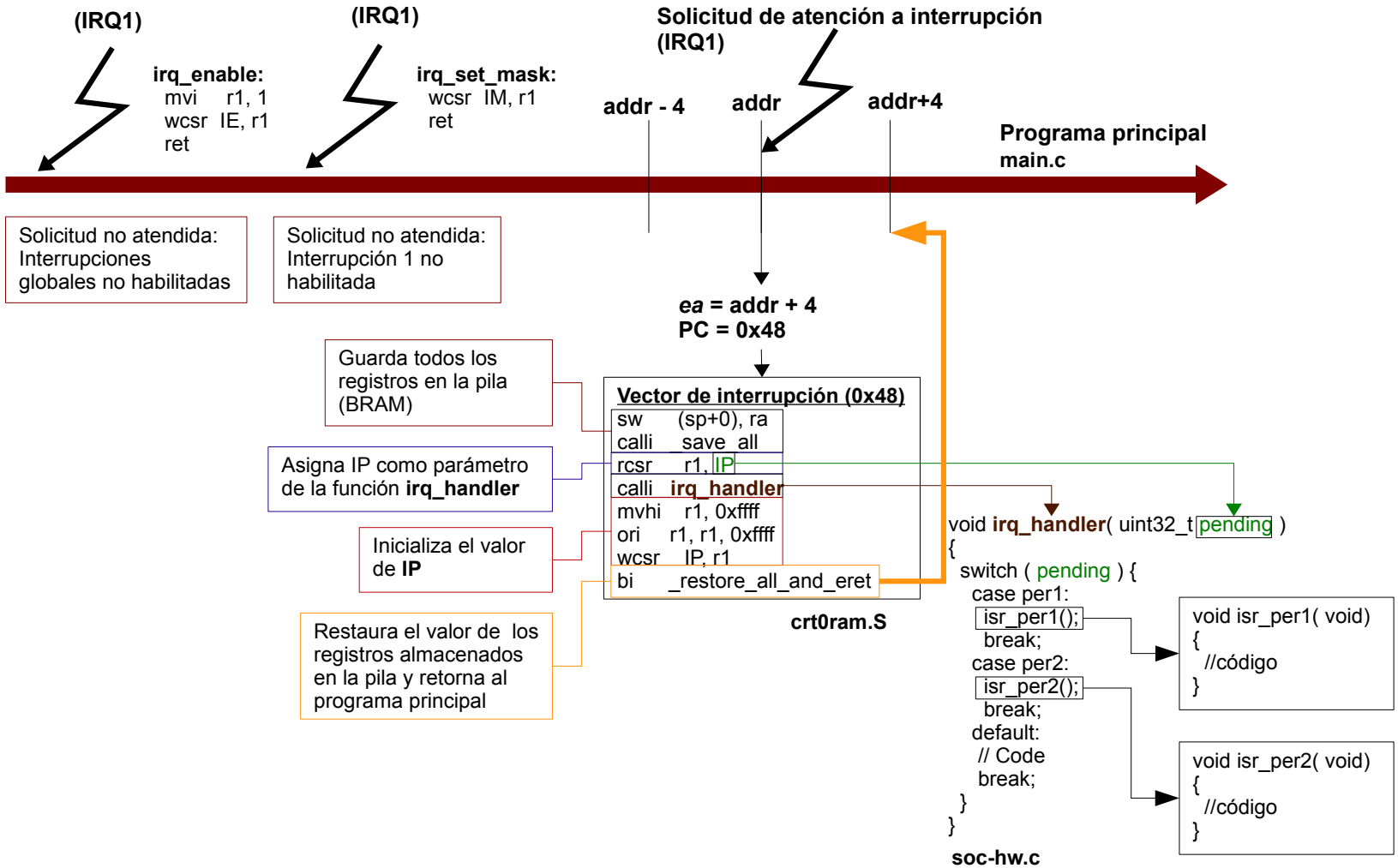
```
sw      (sp+0), ra
calli   _save_all
rcsr    r1, IP
calli   irq_handler
mvhi    r1, 0xffff
ori     r1, r1, 0xffff
wcsr    IP, r1
bi      _restore_all_and_eret
```

```
addi    sp, sp, -128
sw      (sp+4), r1
...
sw      (sp+108), r27
#endif
sw      (sp+120), ea
sw      (sp+124), ba
lw      r1, (sp+128)
sw      (sp+116), r1
mv      r1, sp
addi    r1, r1, 128
sw      (sp+112), r1
ret
```

```
lw      r1, (sp+4)
...
lw      r27, (sp+108)
lw      ra, (sp+116)
lw      ea, (sp+120)
lw      ba, (sp+124)
lw      sp, (sp+112)
eret
```

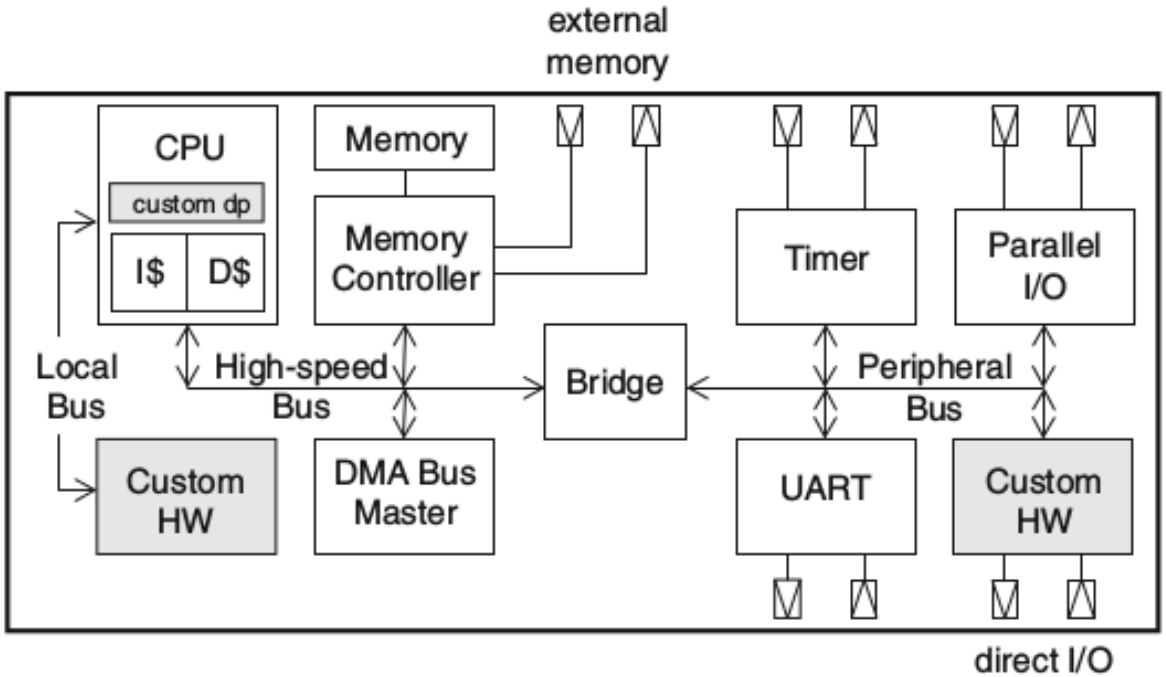

31 26 20 16
IM CODE CSR RX 0000000000000000
wcsr
25 21 15 0
csr = gpr[RX]
31 26 20 16
IM CODE CSR 00000 RX 00000000000
rcsr
25 21 15 0
gpr[RX] = csr
Instruction Decoder
w_result
RX
registers
write_idx_w
RX = instruction[15:11]
reg_data_0
reg_data_1
x_result
m_result
w_result
gpr[RX]
Bypass for reg port 1
Bypass for reg port 0
raw_x_1
raw_m_1
raw_w_1
raw_x_0
raw_m_0
raw_w_0
{32{1'bx}}
immediate_d
{32{1'b0}}
bypass_data_1
{pc_f,2'b00}
d_result_sel_1_d
d_result_sel_0_d
d_result_1
d_result_0
~stall_x
D/X stage registers
operand_1_x
operand_0_x
adder_op_x
adder_op_x_n
csr_read_data_x
adder
adder_carry_n_x
adder_overflow_x
adder_result_x
x_result_sel_add_x
x_result_sel_csr_x
x_result_sel_mc_arith_x
x_result_sel_sext_x
x_result
~stall_m
X/M stage registers
{31{1'b0},condition_met_m}
operand_m
m_result_sel_compare_m
m_result_sel_shift_m
{pc_m, 2'b00}
m_result
exception_m
M/W stage registers
1'b0
operand_w
load_data_w
w_result_sel_load_w
w_result_sel_mul_w
w_result
interrupt_n
csr_write_data
asserted
D
Q
csr_x = instruction[25:21]
exception
ip
im
{29'b0,bie, eie,ie}
csr_x
{eba, 8'h00}
interrupt_csr_read_data_x
cfg
{32{1'bx}}
default
csr_x = instruction[15:11]


31 26
IM
CODE
11110
00000000000000000000
instruction_d eret 25 21 0
31 26
IM
CODE
11101
00000000000000000000
instruction_d ret 25 21 0
Instruction Decoder
w_result
RX
registers
write_idx_w
reg_data_0
reg_data_1
pc_a
pc_f
pc_d
pc_x
pc_m
pc_w
x_result
m_result
w_result
Bypass for reg port 1
001 01x 1xx 000
raw_x_1
raw_m_1
raw_w_1
raw_x_0
raw_m_0
raw_w_0
000 1xx 01x 001
Bypass for reg port 0
{32{1'bx}}
immediate_d
{32{1'b0}}
bypass_data_1
00 10 11 01
d_result_sel_1_d
d_result_1
{pc_f,2'b00}
bypass_data_0
1 0
d_result_sel_0_d
d_result_0
pc_d + branch_offset_d
~stall_x
D/X stage registers
branch_reg_d
branch_target_x
operand_1_x
operand_0_x
adder_op_x
adder_op_x_n
adder
adder_carry_n_x
adder_overflow_x
adder_result_x
1xxx 0000 0001 001x 01xx
x_result_sel_add_x
x_result_sel_csr_x
x_result_sel_mc_arith_x
x_result_sel_sext_x
x_result
{eba, eid_x, {3{1'b0}}}
~stall_m
X/M stage registers
exception_x
branch_target_m
valid_m
branch_m
condition_met_m
stall_m
Instruction Unit
pc_a = branch_target_m
{31{1'b0},condition_met_m}
operand_m
m_result_sel_compare_m
m_result_sel_shift_m
1x 00 01
m_result
{pc_m, 2'b00}
exception_m
M/W stage registers
1'b0
operand_w
load_data_w
1x 00 01
w_result_sel_load_w
w_result_sel_mul_w
w_result
exception_m
branch_taken_m

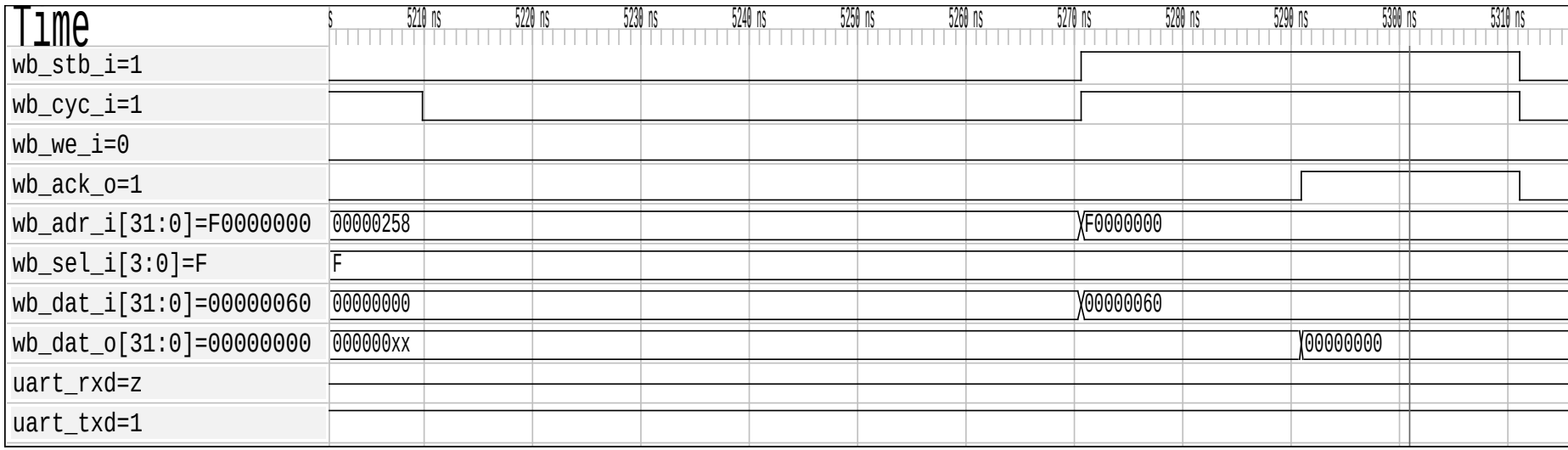

(IRQ1)
irq_enable:
mvi r1, 1
wcsr IE, r1
ret
(IRQ1)
irq_set_mask:
wcsr IM, r1
ret
Solicitud de atención a interrupción
(IRQ1)
addr - 4
addr
addr+4
Programa principal
main.c
Solicitud no atendida: Interrupciones globales no habilitadas
Solicitud no atendida: Interrupción 1 no habilitada
ea = addr + 4
PC = 0x48
Guarda todos los registros en la pila (BRAM)
Asigna IP como parámetro de la función irq_handler
Inicializa el valor de IP
Restaura el valor de los registros almacenados en la pila y retorna al programa principal
Vector de interrupción (0x48)
sw (sp+0), ra
calli _save_all
rcsr r1, IP
calli irq_handler
mvhi r1, 0xffff
ori r1, r1, 0xffff
wcsr IP, r1
bi _restore_all_and_eret
crt0ram.S
void irq_handler(uint32_t pending)
{
switch (pending) {
case per1:
isr_per1();
break;
case per2:
isr_per2();
break;
default:
// Code
break;
}
}
soc-hw.c
void isr_per1(void)
{
//código
}
void isr_per2(void)
{
//código
}

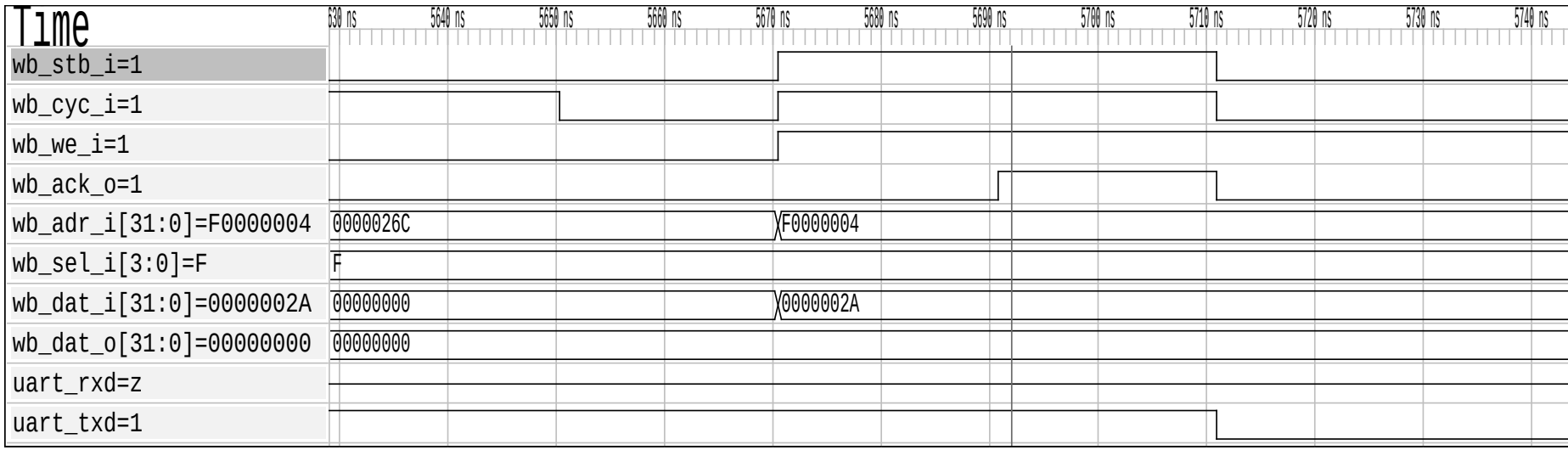
Índice

Estructura general de un SoC

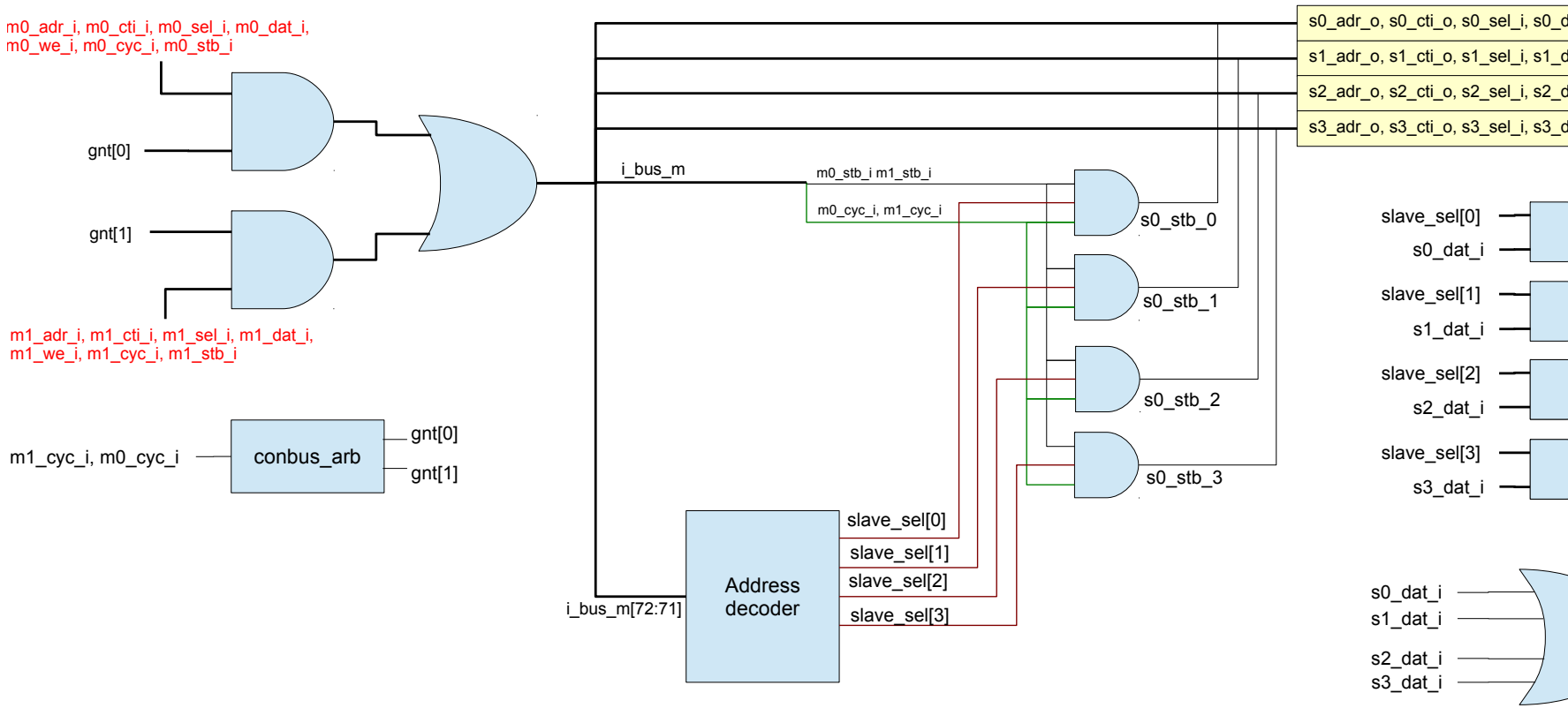


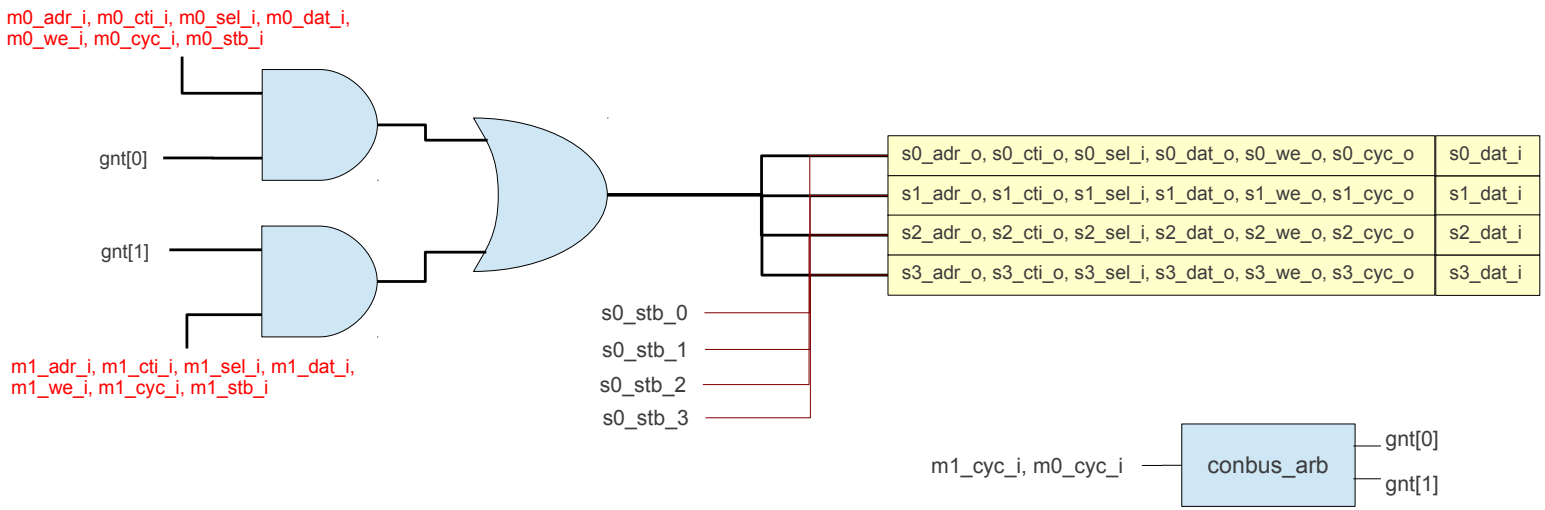
Índice

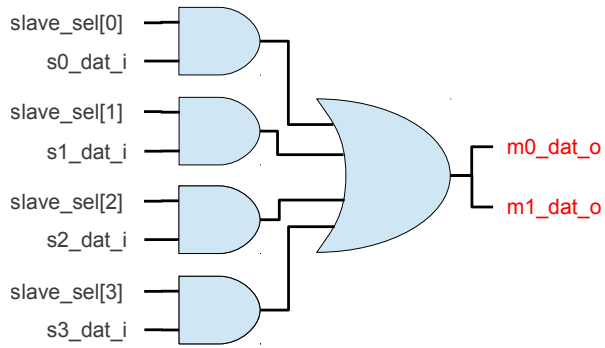

Time
5210 ns
5220 ns
5230 ns
5240 ns
5250 ns
5260 ns
5270 ns
5280 ns
5290 ns
5300 ns
5310 ns
wb_stb_i=1
wb_cyc_i=1
wb_we_i=0
wb_ack_o=1
wb_adr_i[31:0]=F0000000
00000258
F0000000
wb_sel_i[3:0]=F
F
wb_dat_i[31:0]=00000060
00000000
00000060
wb_dat_o[31:0]=00000000
000000xx
00000000
uart_rxd=z
uart_txd=1

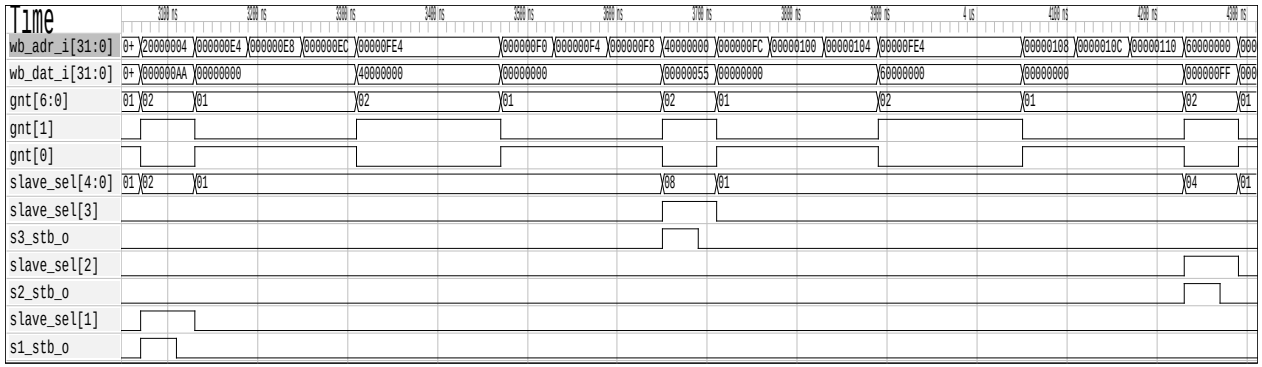

Time
630 ns
5640 ns
5650 ns
5660 ns
5670 ns
5680 ns
5690 ns
5700 ns
5710 ns
5720 ns
5730 ns
5740 ns
wb_stb_i=1
wb_cyc_i=1
wb_we_i=1
wb_ack_o=1
wb_adr_i[31:0]=F0000004
0000026C
F0000004
wb_sel_i[3:0]=F
F
wb_dat_i[31:0]=0000002A
00000000
0000002A
wb_dat_o[31:0]=00000000
00000000
uart_rxd=z
uart_txd=1

Arquitectura maestro/esclavo




m0_adr_i, m0_cti_i, m0_sel_i, m0_dat_i,
m0_we_i, m0_cyc_i, m0_stb_i
gnt[0]
gnt[1]
m1_adr_i, m1_cti_i, m1_sel_i, m1_dat_i,
m1_we_i, m1_cyc_i, m1_stb_i
s0_stb_0
s0_stb_1
s0_stb_2
s0_stb_3
s0_adr_o, s0_cti_o, s0_sel_i, s0_dat_o, s0_we_o, s0_cyc_o
s0_dat_i
s1_adr_o, s1_cti_o, s1_sel_i, s1_dat_o, s1_we_o, s1_cyc_o
s1_dat_i
s2_adr_o, s2_cti_o, s2_sel_i, s2_dat_o, s2_we_o, s2_cyc_o
s2_dat_i
s3_adr_o, s3_cti_o, s3_sel_i, s3_dat_o, s3_we_o, s3_cyc_o
s3_dat_i
m1_cyc_i, m0_cyc_i
conbus_arb
gnt[0]
gnt[1]


slave_sel[0]
s0_dat_i
slave_sel[1]
s1_dat_i
slave_sel[2]
s2_dat_i
slave_sel[3]
s3_dat_i
m0_dat_o
m1_dat_o


Time
wb_adr_i[31:0]
wb_dat_i[31:0]
gnt[6:0]
gnt[1]
gnt[0]
slave_sel[4:0]
slave_sel[3]
s3_stb_o
slave_sel[2]
s2_stb_o
slave_sel[1]
s1_stb_o

PREGUNTAS