Electronica digital 2

caso de estudio SoC RISC-V

Ferney Alberto Beltrán Molina

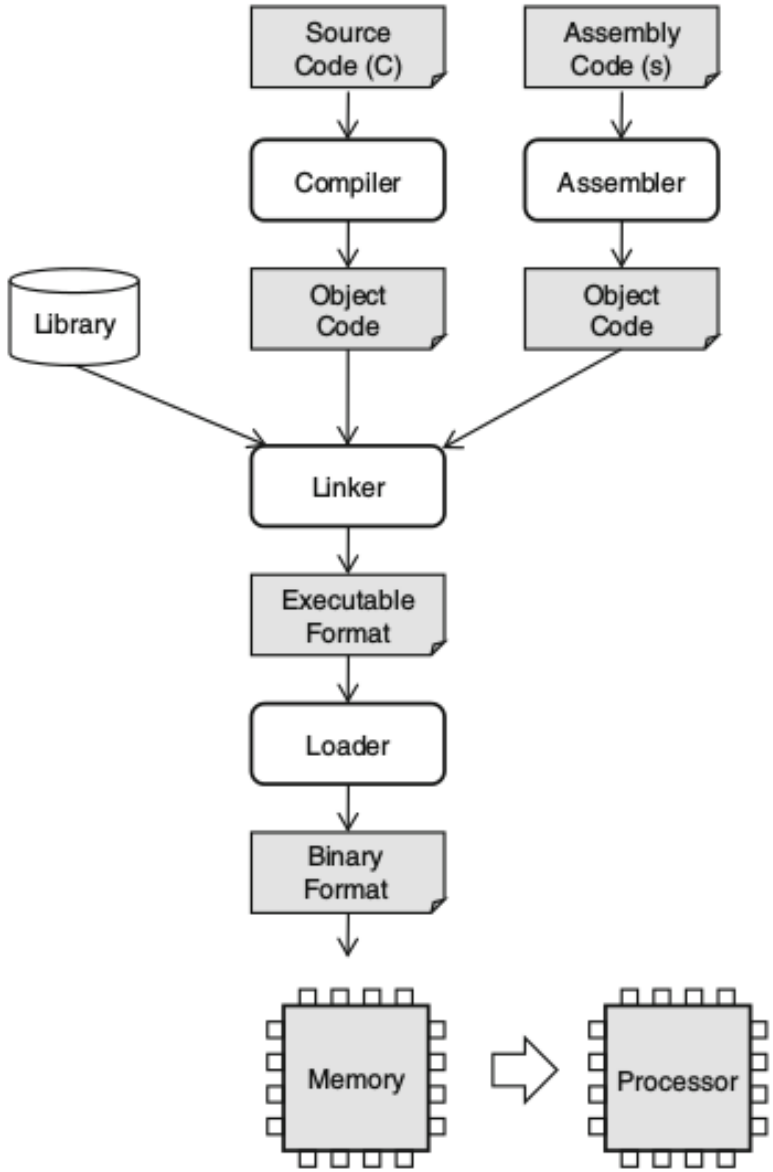
Agosto 2020

Contacto

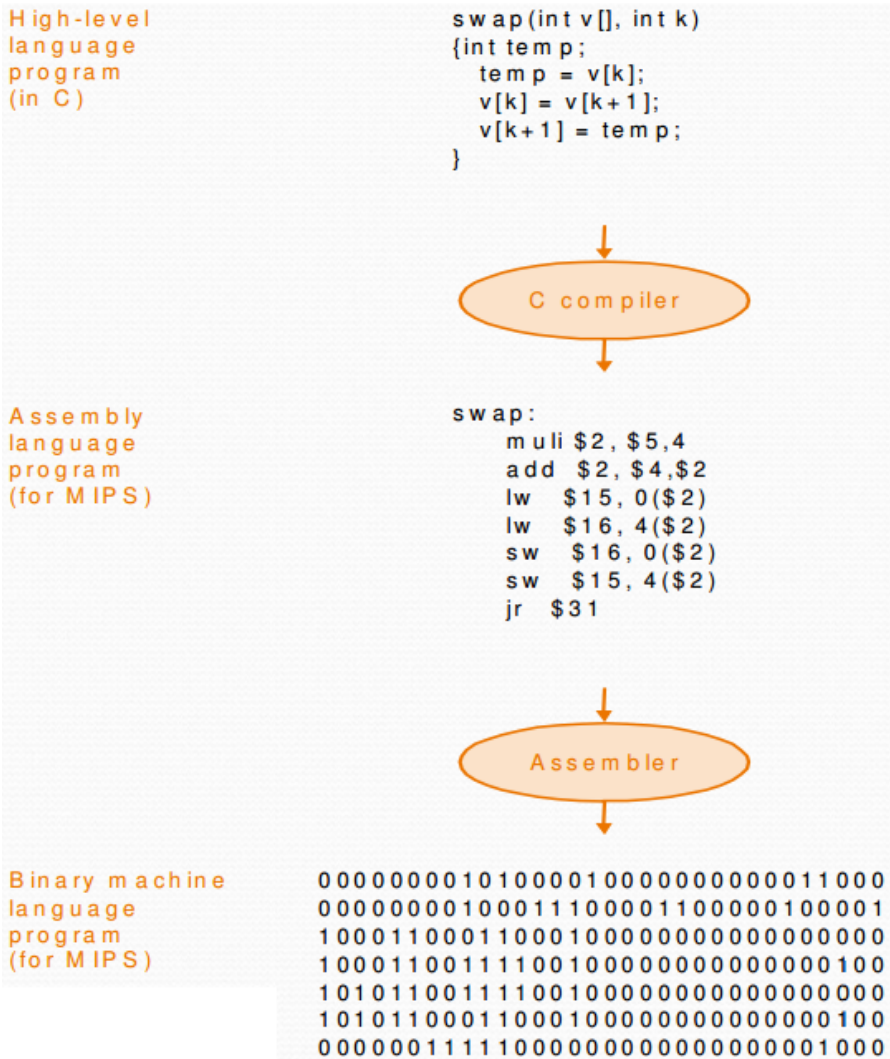
Nombre:	Ferney Alberto Beltrán Molina, Ing, MSc, PhD(c)
Email:	fabeltranm@unal.edu.co
oficina:	

Contenido

Introducción

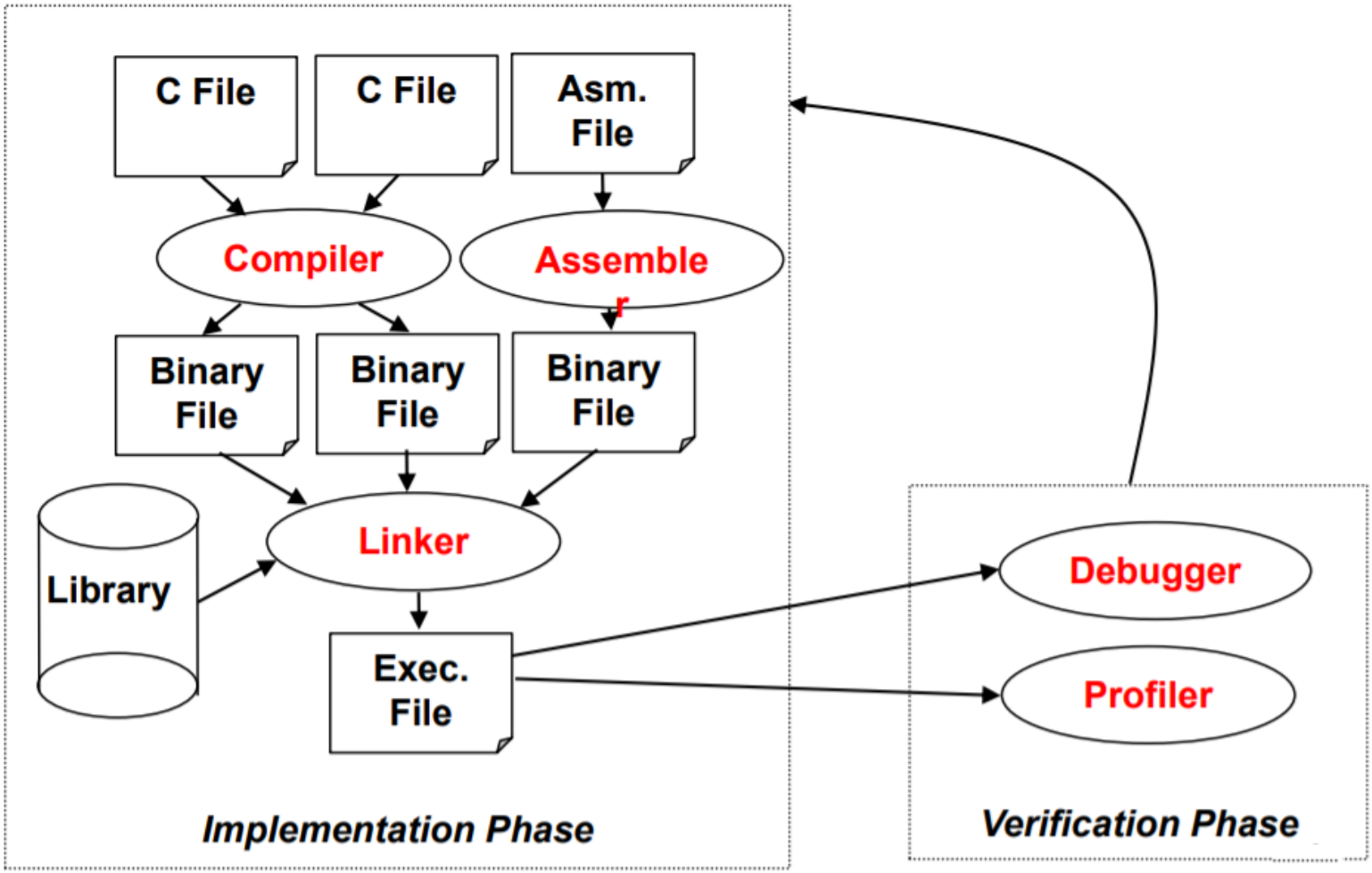


Introducción



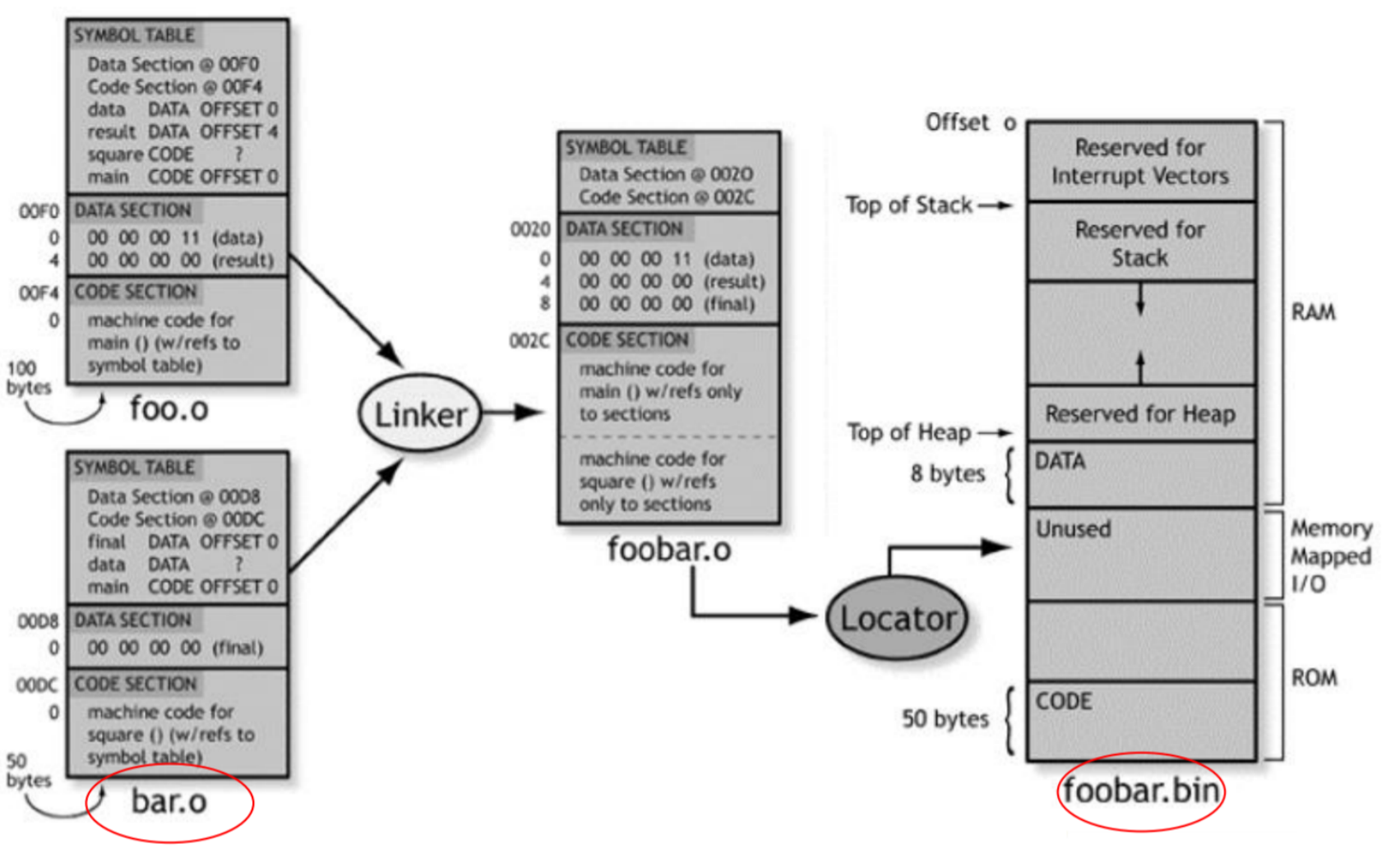
- lenguaje de programación de alto Nivel (C, C++, Java, etc.)
  - Sentencias
  - Variables
  - Operaciones
  - Funciones, procedimientos
- lenguaje Ensamblador
  - Instrucciones
  - Registro
  - Memorias
  - Funciones, procedimientos

Compilación Cruzada

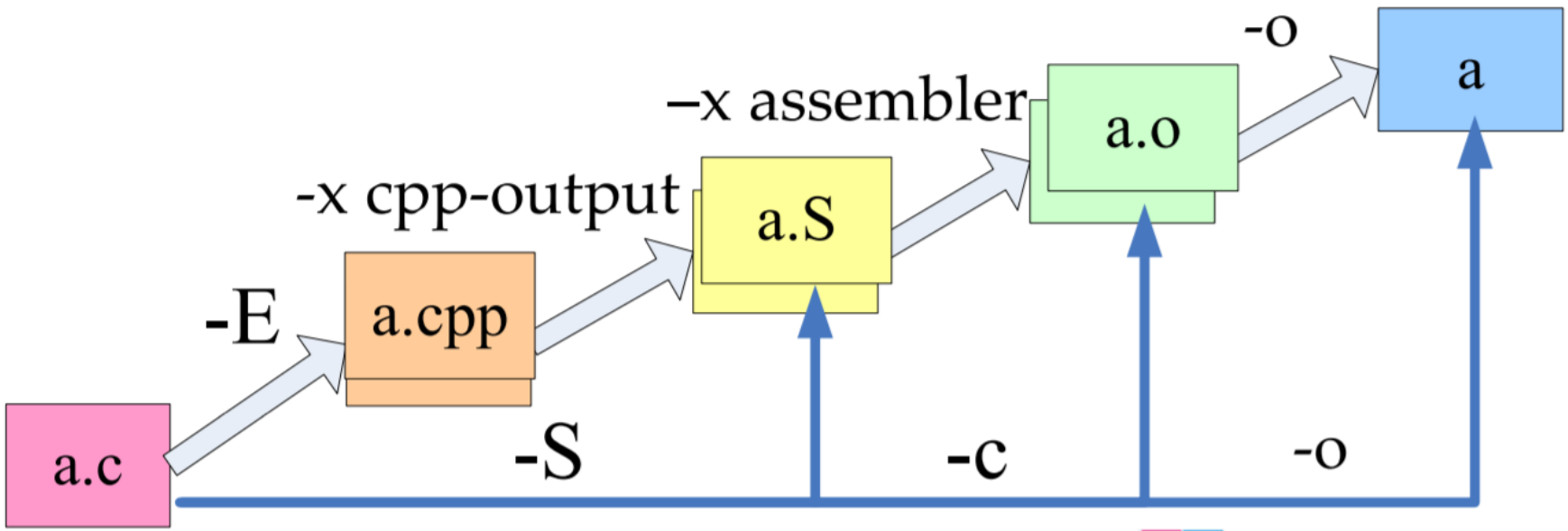


- correr en un procesador pero ejecutar en otro

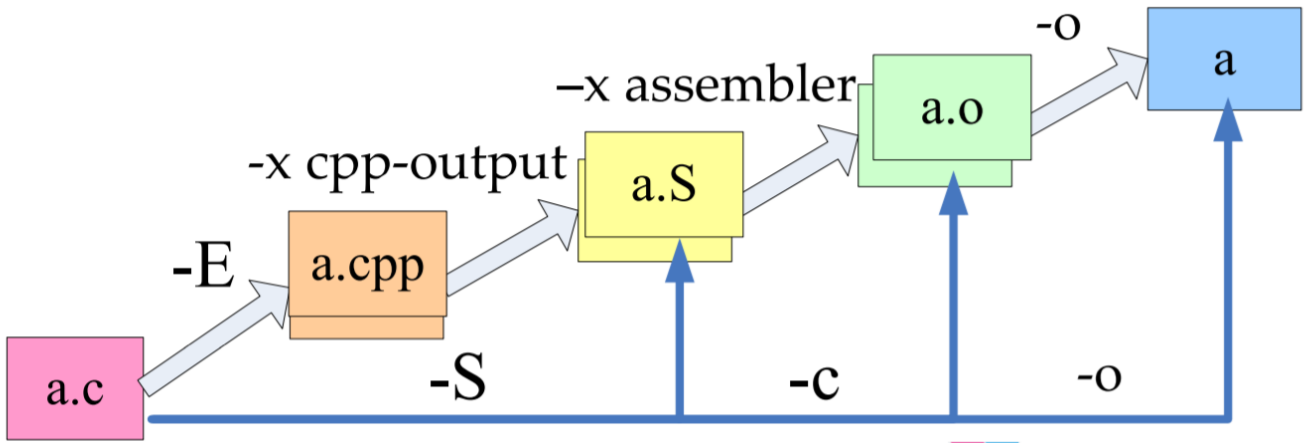
Compilación Cruzada



Procesos GCC



Proceso GCC



- gcc -o: Escriba la salida de compilación en un archivo de salida
- gcc -c: Compila archivos fuente sin vinculación.
- gcc -S: Crear código de ensamblador (*.S).
- gcc -wall: Habilita todos los mensajes de advertencia del compilador
- objdump: Muestra información de los archivos de objetos -d *.o

Proceso de optimización gcc -O

opción	nivel de optimización	Tiempo de ejecución	tamaño del código	uso de memoria	tiempo de compilación
-O0	optimización para el tiempo de compilación (predeterminado)	+	+	-	-
-O1 u -O	optimización para el tamaño del código y el tiempo de ejecución	-	-	+	+
-O2	optimización más para el tamaño del código y el tiempo de ejecución	-		+	++
-O3	optimización más para el tamaño del código y el tiempo de ejecución	---		+	+++
-Os	optimización para el tamaño del código		-		++
-Ofast	O3 con cálculos matemáticos rápidos y no precisos	---		+	+++

+ aumentar ++ aumentar más +++ aumentar aún más -reducir --reducir más --- reducir aún más

Rendimiento

CPU Time = Número de ciclos de reloj de la CPU * Tc

$$CPI = CLK / IC$$

$$IC \times CPI = CLK$$

$$T_{CPU} = CLK / f$$

$$T_{CPU} = CPI \times IC / f$$

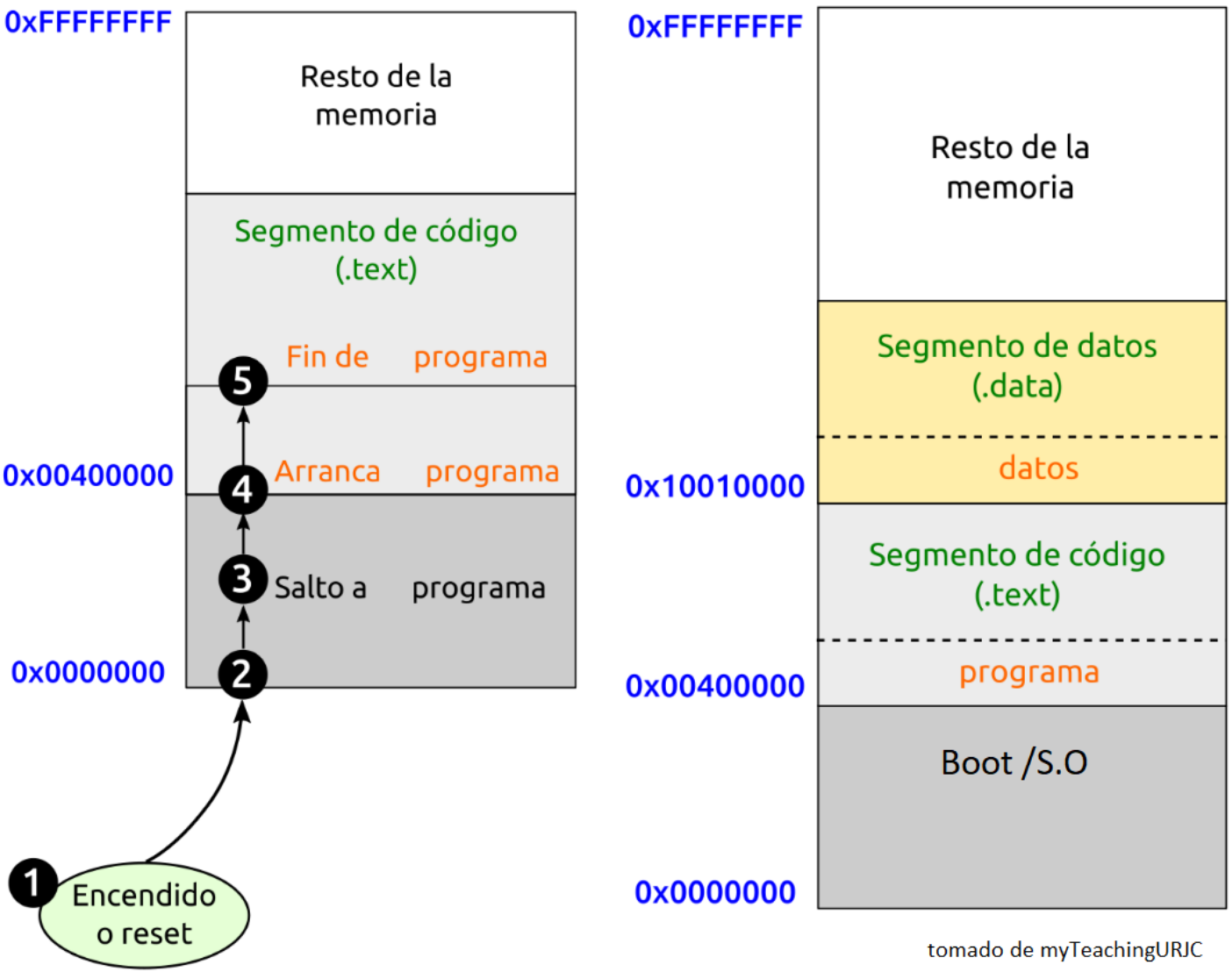
$$T_{CLK} = 1 / f$$

$$T_{CPU} = CPI \times IC \times T_{CLK}$$

Rendimiento

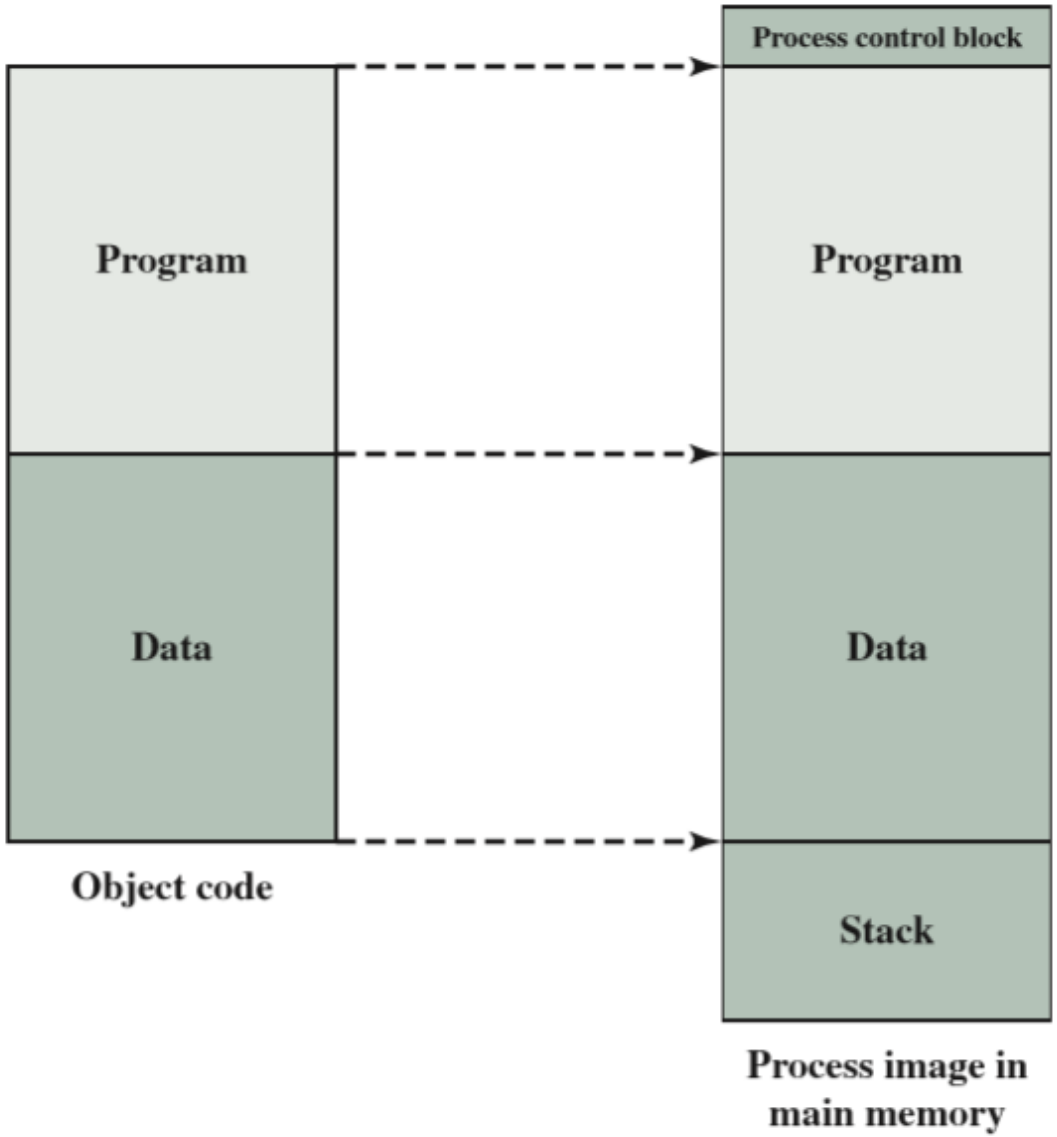
$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Proceso GCC

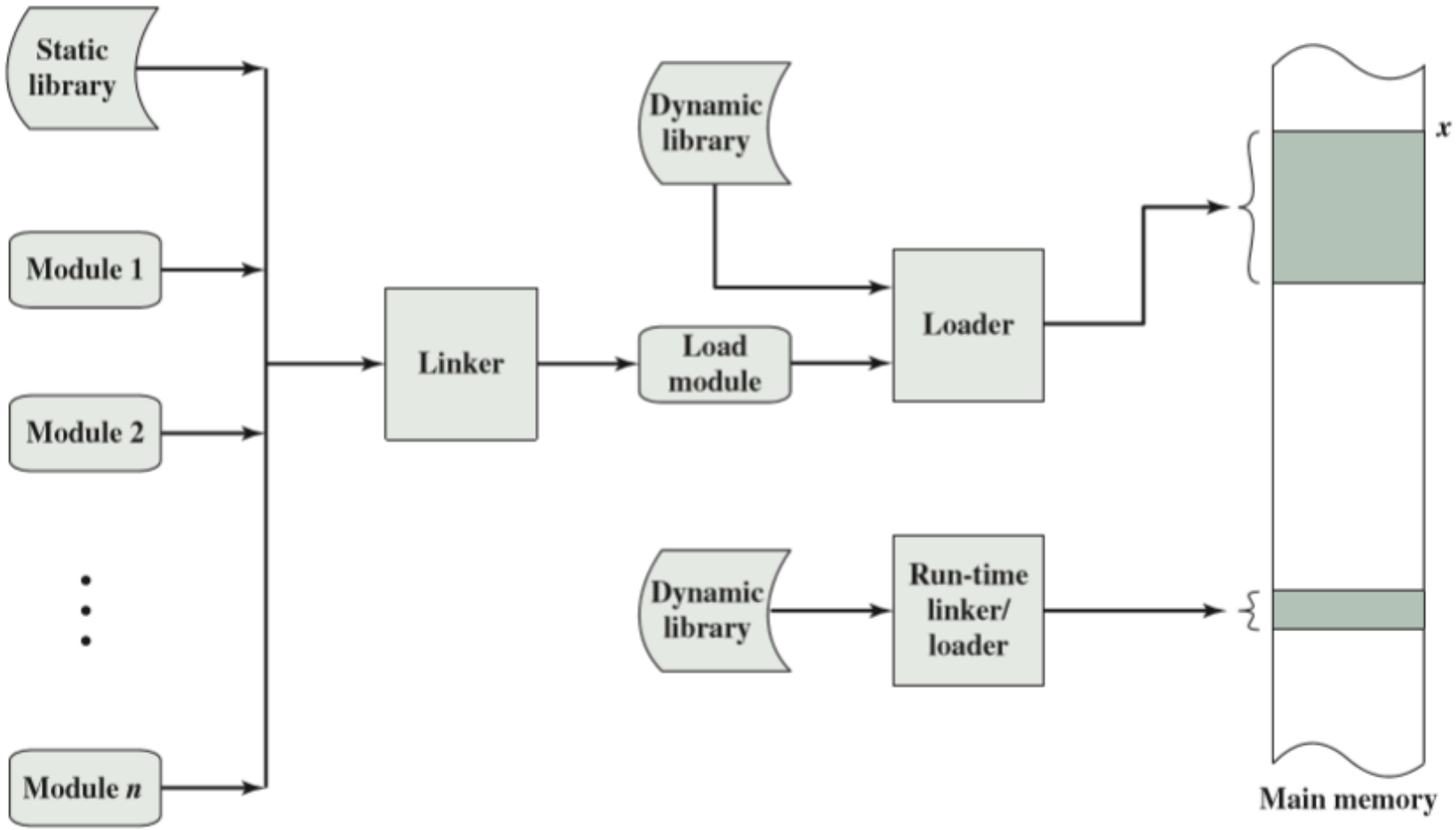


tomado de myTeachingURJC

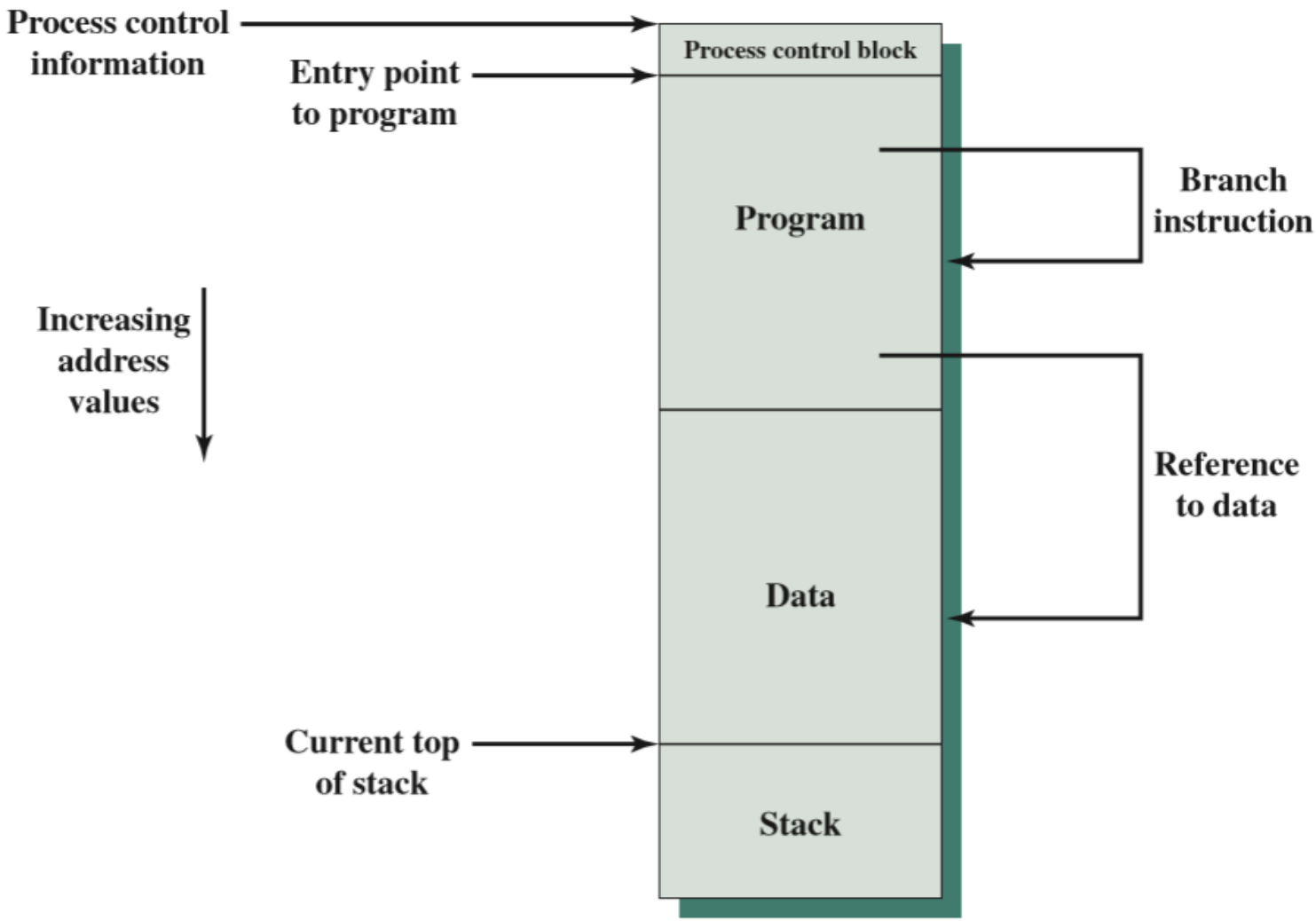
linking


Process control block
Program
Program
Data
Data
Stack
Object code
Process image in
main memory

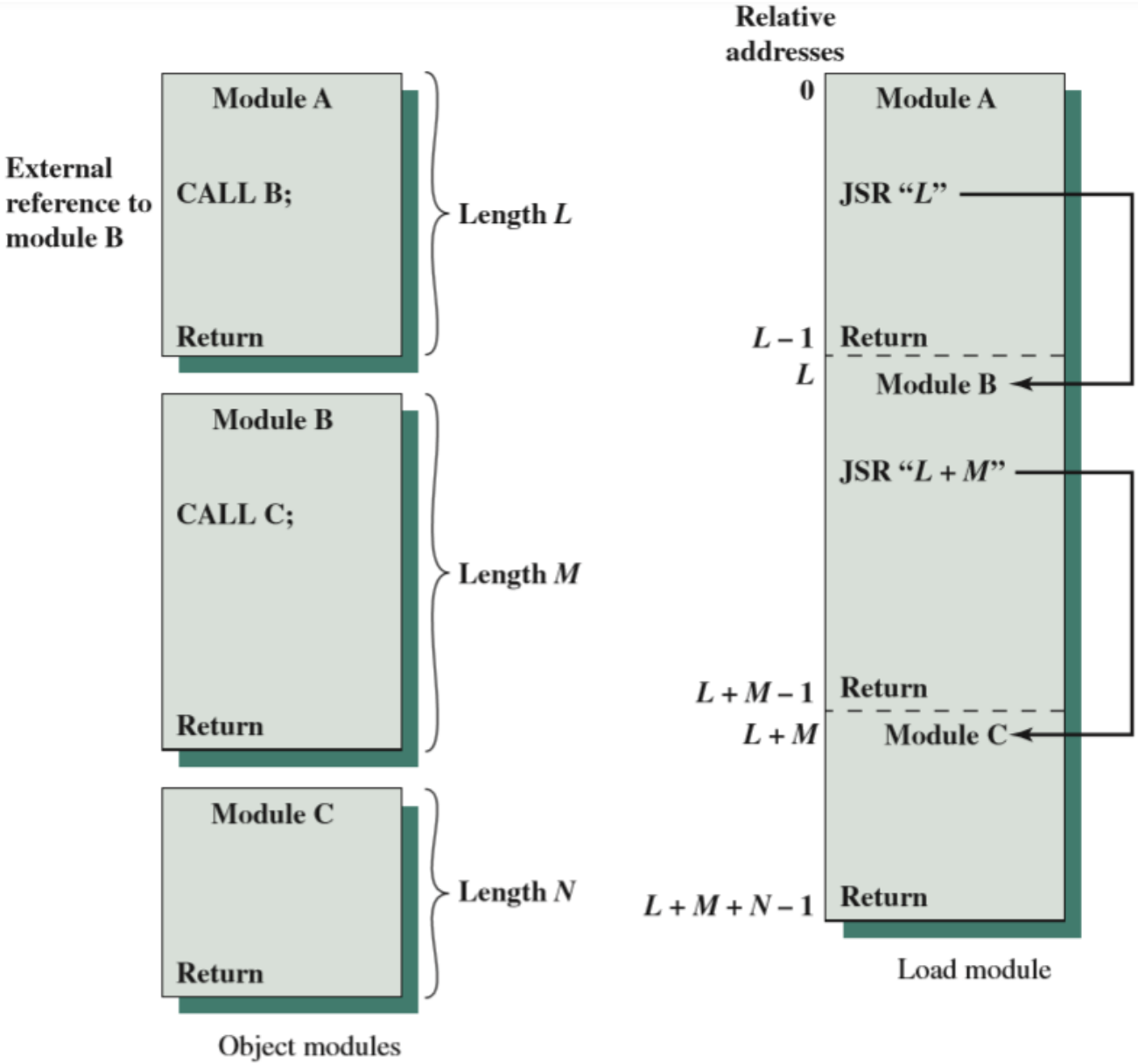
linking


Static library
Module 1
Module 2
Module n
Linker
Dynamic library
Load module
Loader
Dynamic library
Run-time linker/ loader
x
Main memory

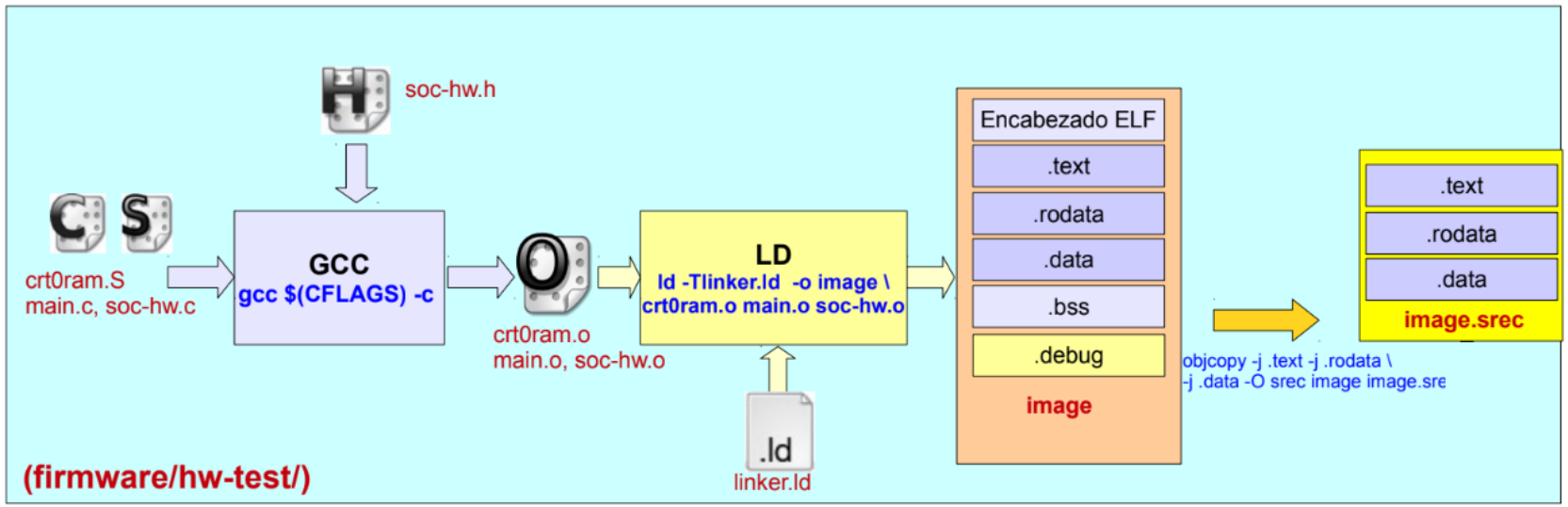
linking


Process control information
Entry point to program
Process control block
Program
Branch instruction
Increasing address values
Reference to data
Data
Current top of stack
Stack

linking


Module A
External reference to module B
CALL B;
Return
Length L
Module B
CALL C;
Return
Length M
Module C
Return
Length N
Object modules
Relative addresses
0
Module A
JSR "L"
L – 1
Return
L
Module B
JSR "L + M"
L + M – 1
Return
L + M
Module C
L + M + N – 1
Return
Load module

Procesos GCC



Índice

Concepto

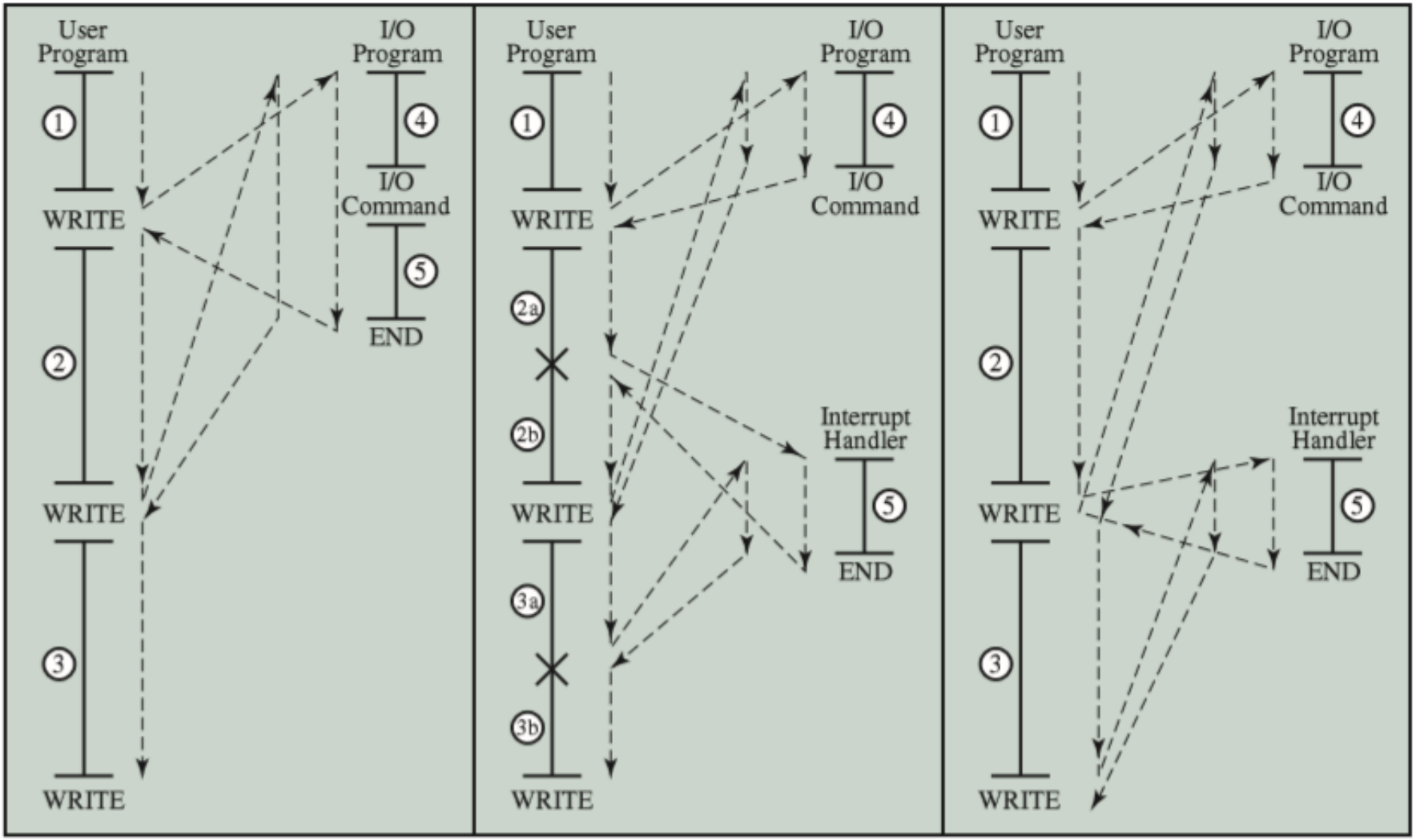
Eventos que cambian el flujo normal de la instrucciones. (no son call, jump o branch)

Excepciones Evento no programado que interrumpe la ejecución del programa; algunos procesadores lo usan Se utiliza para detectar instrucciones indefinidas.

Interrupción
En principio es una excepción que proviene del exterior del procesador. (Algunas arquitecturas usan el término interrupción para todas las excepciones).

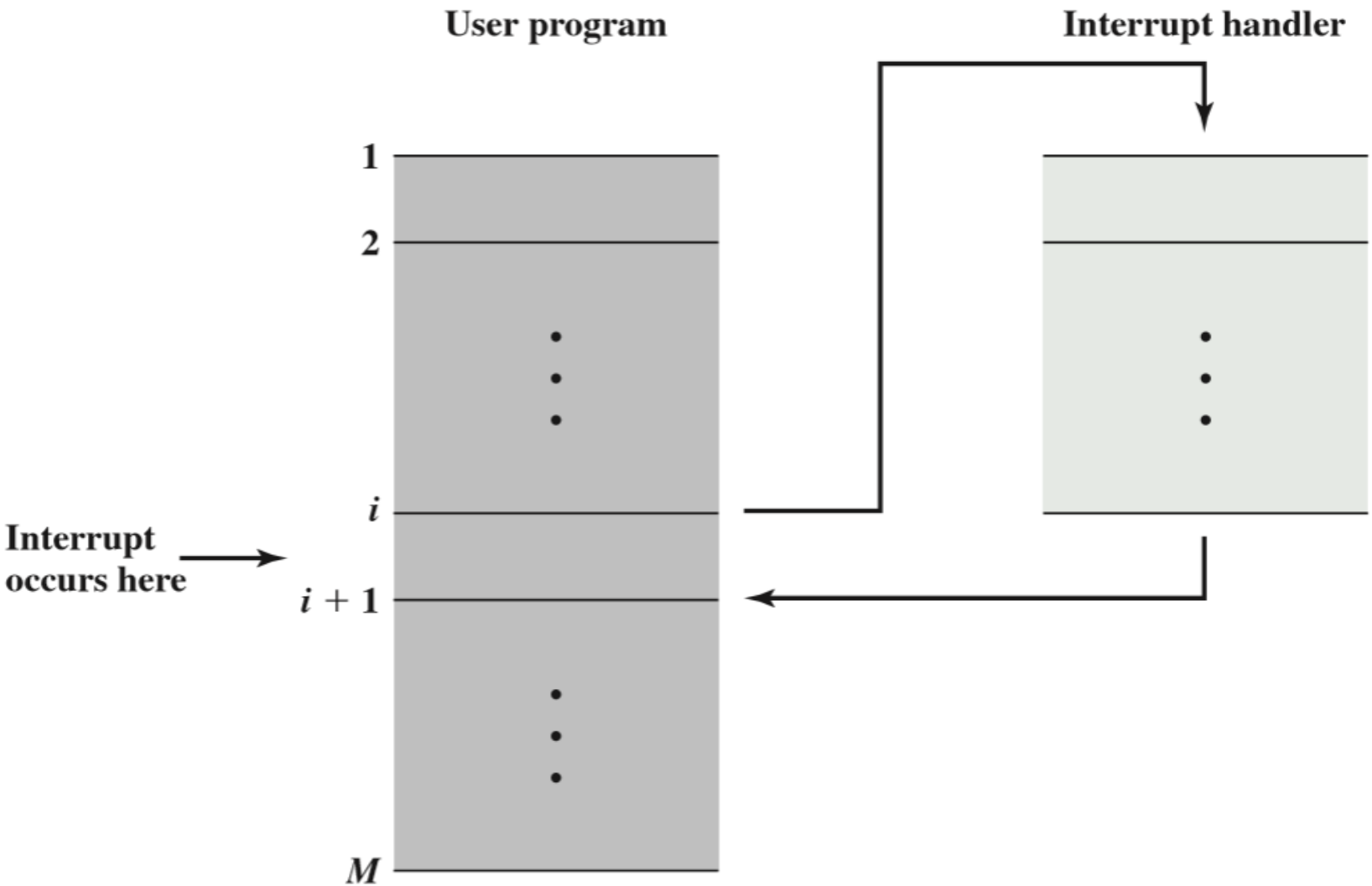
Las interrupciones se diseñan, principalmente, como una forma de mejorar la eficiencia del procesamiento

concepto

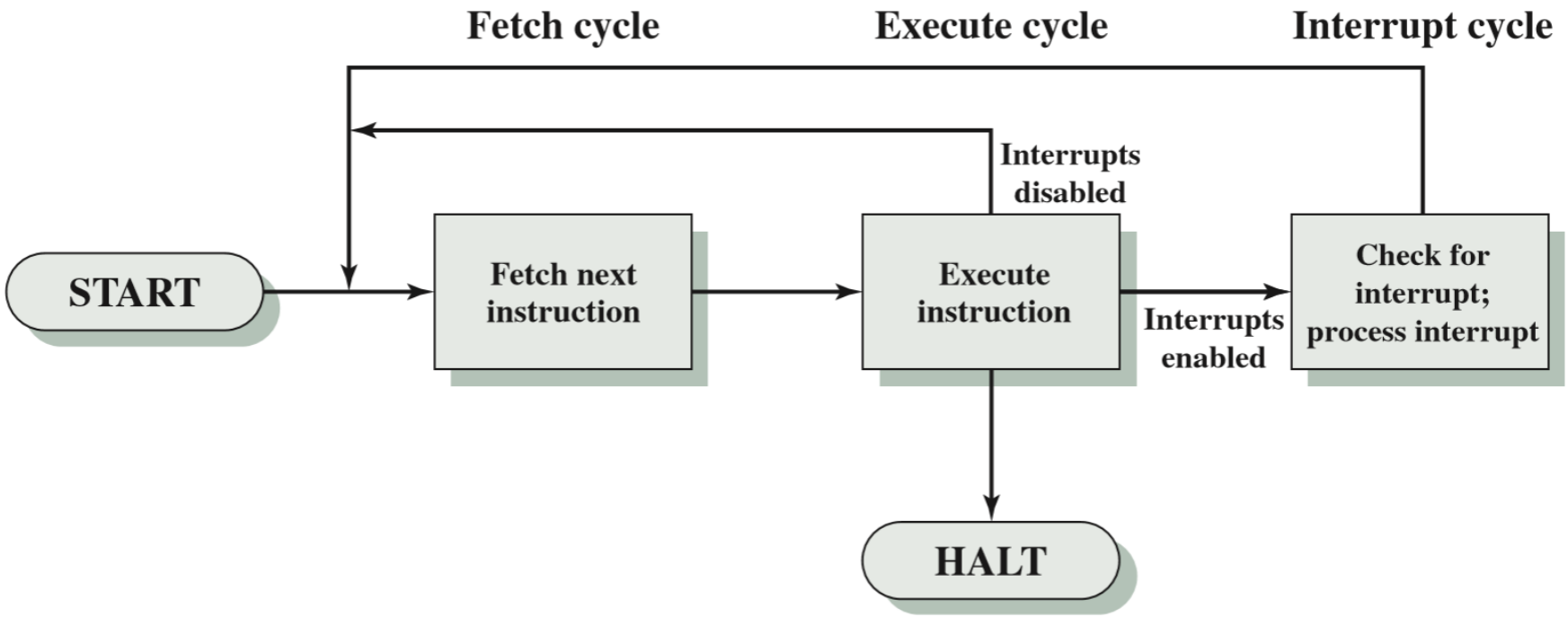


(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

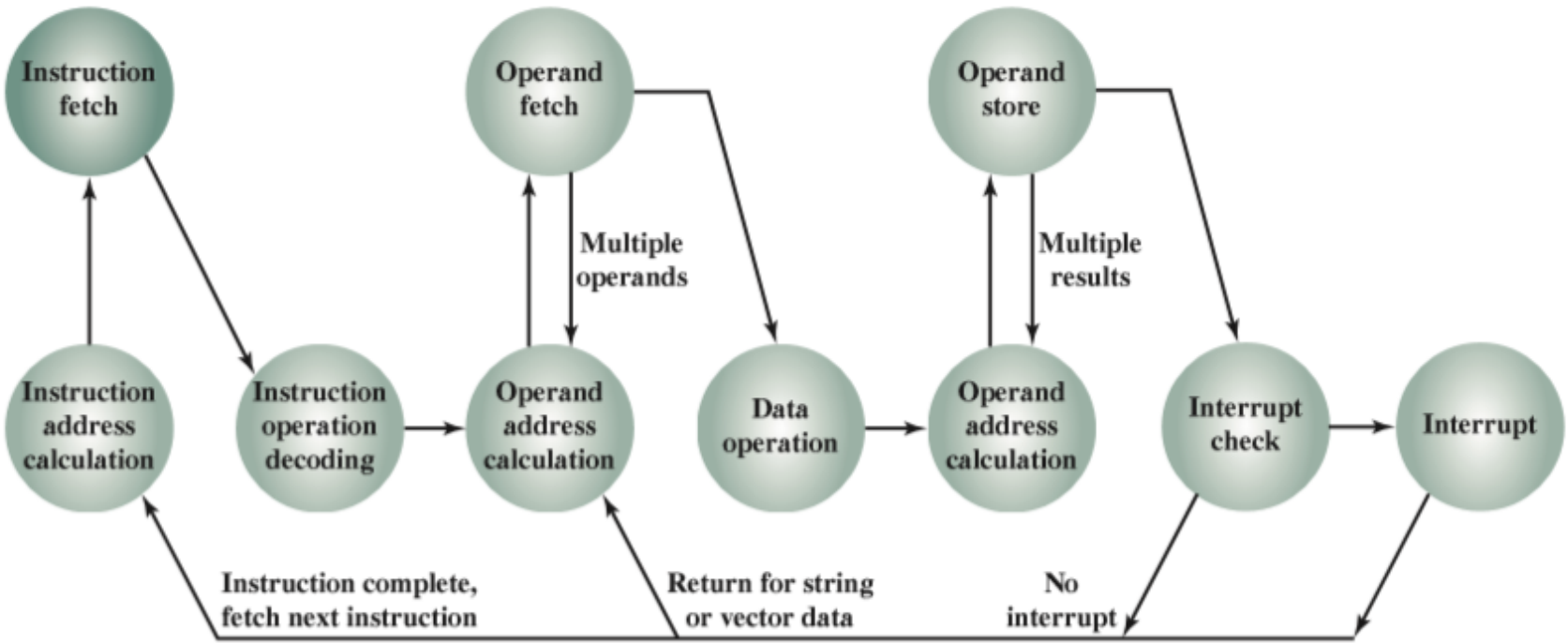
concepto


User program
Interrupt handler
1
2
i
i + 1
M
Interrupt
occurs here

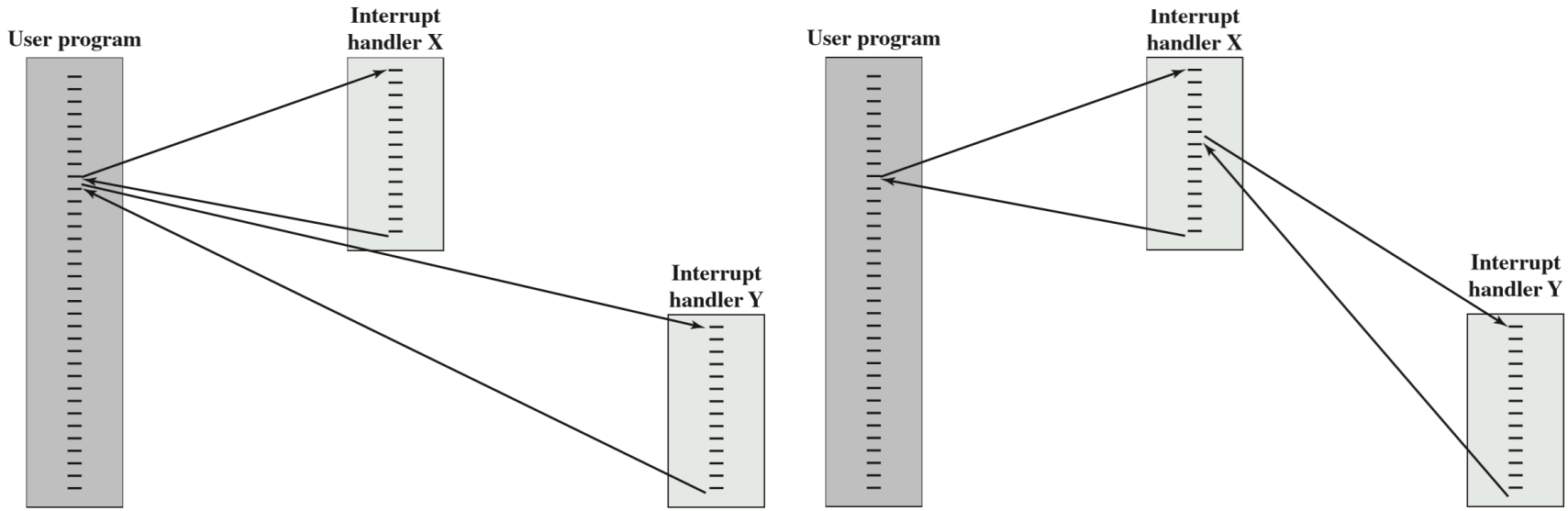
concepto


Fetch cycle
Execute cycle
Interrupt cycle
Interrupts disabled
START
Fetch next instruction
Execute instruction
Interrupts enabled
Check for interrupt; process interrupt
HALT

concepto

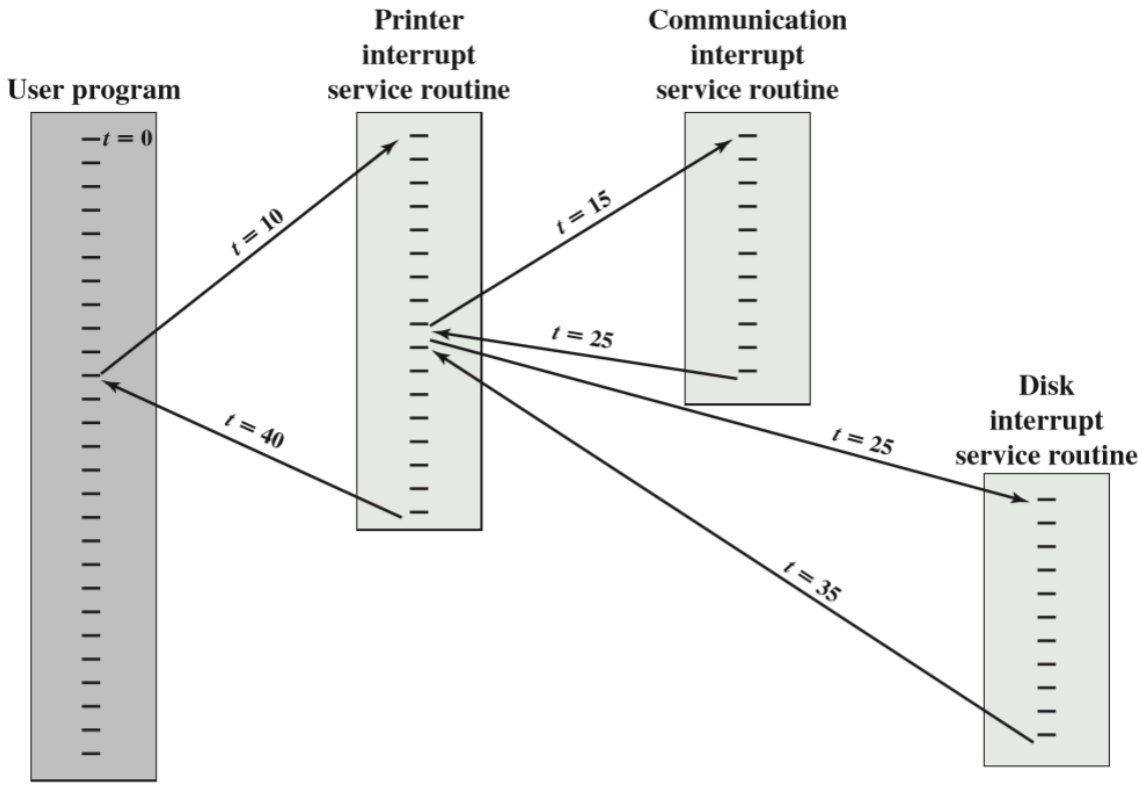

Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

Múltiples interrupciones



1. Opción 1: deshabilitar interrupciones
2. Opción 2: interrupciones por prioridad

Múltiples interrupciones



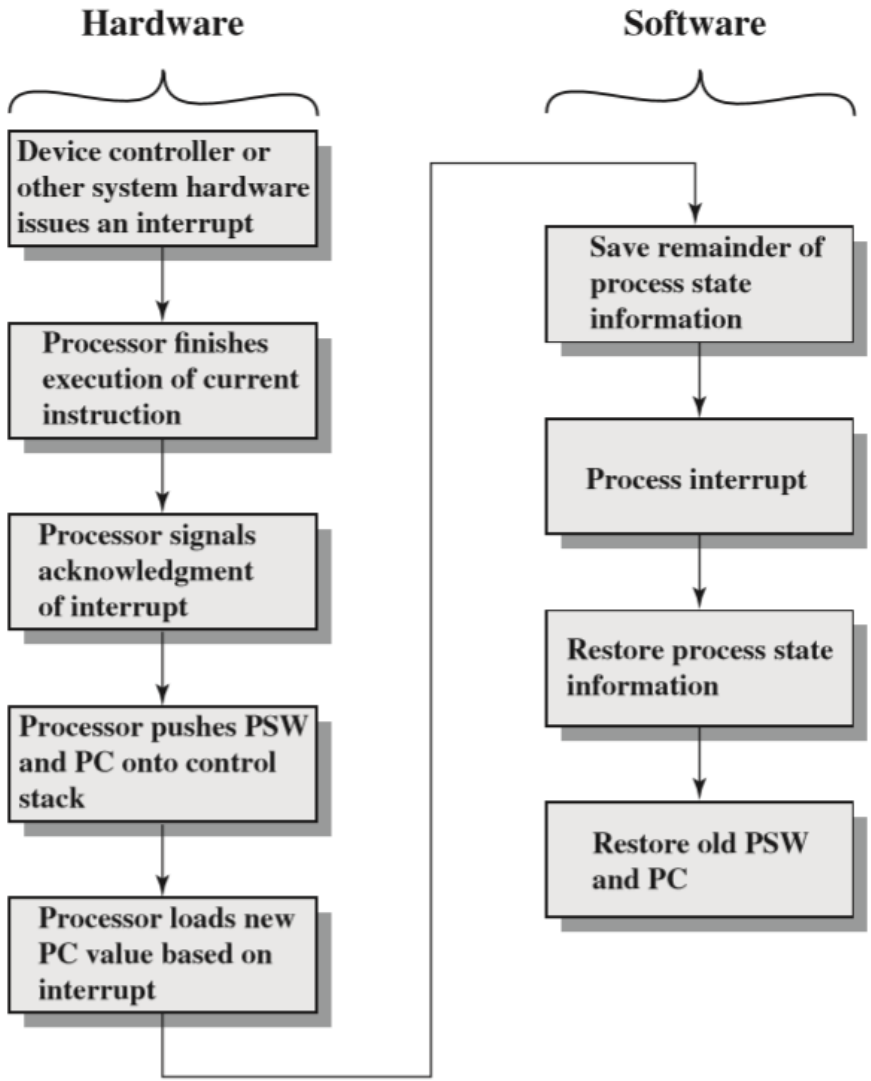
1. Prioridad 1: Comunicaciones
2. Prioridad 2: Disco
3. Prioridad 3: Impresora

Atención de interrupciones

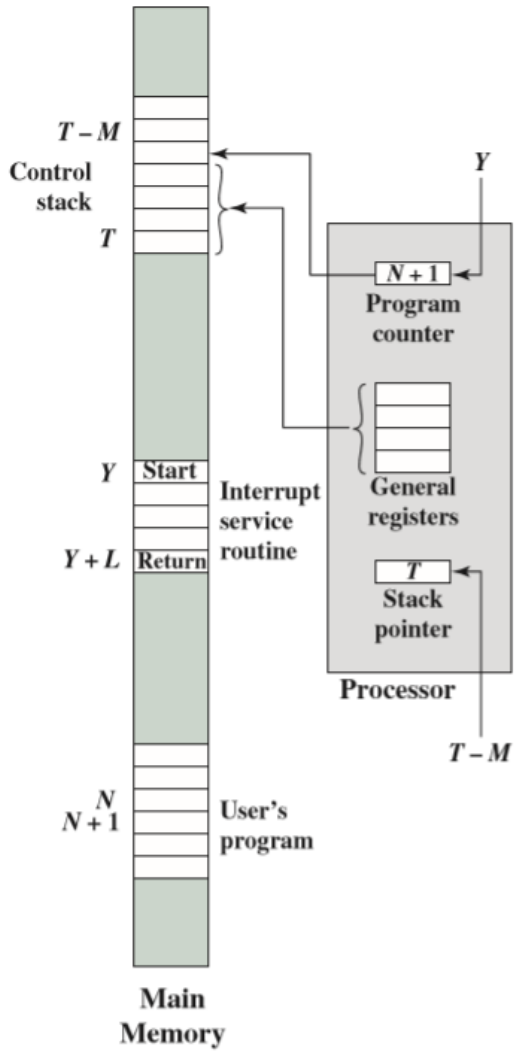
1. Vector de interrupciones: al tener interrupción se determina la dirección de programa a la que se transfiere el control. HW

2. Interrupción no vectorizada: El software conoce el motivo de la interrupción por la dirección en la que se inicia. Cuando la interrupción no está vectorizada, se puede usar un único punto de entrada para todas las interrupción

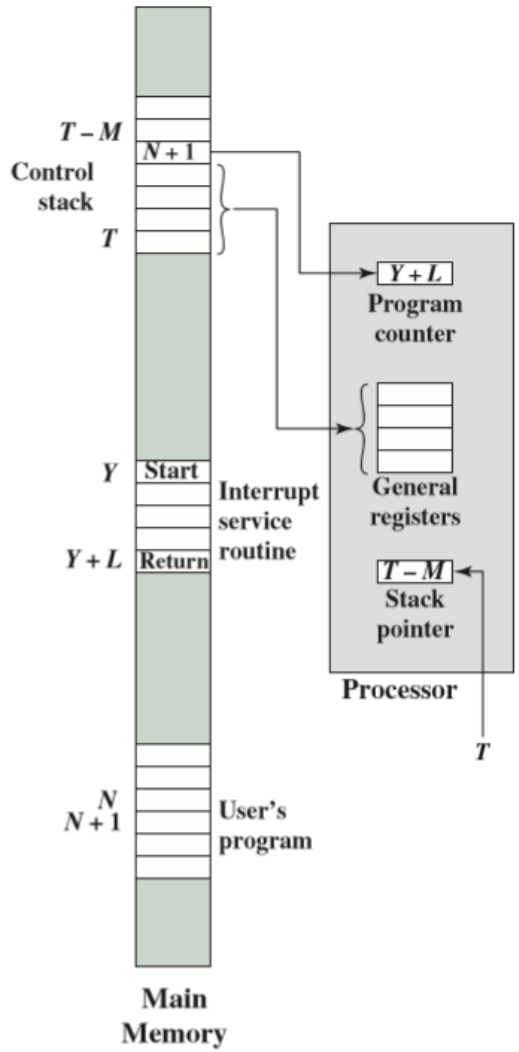
Manejo interrupciones



Manejo interrupciones

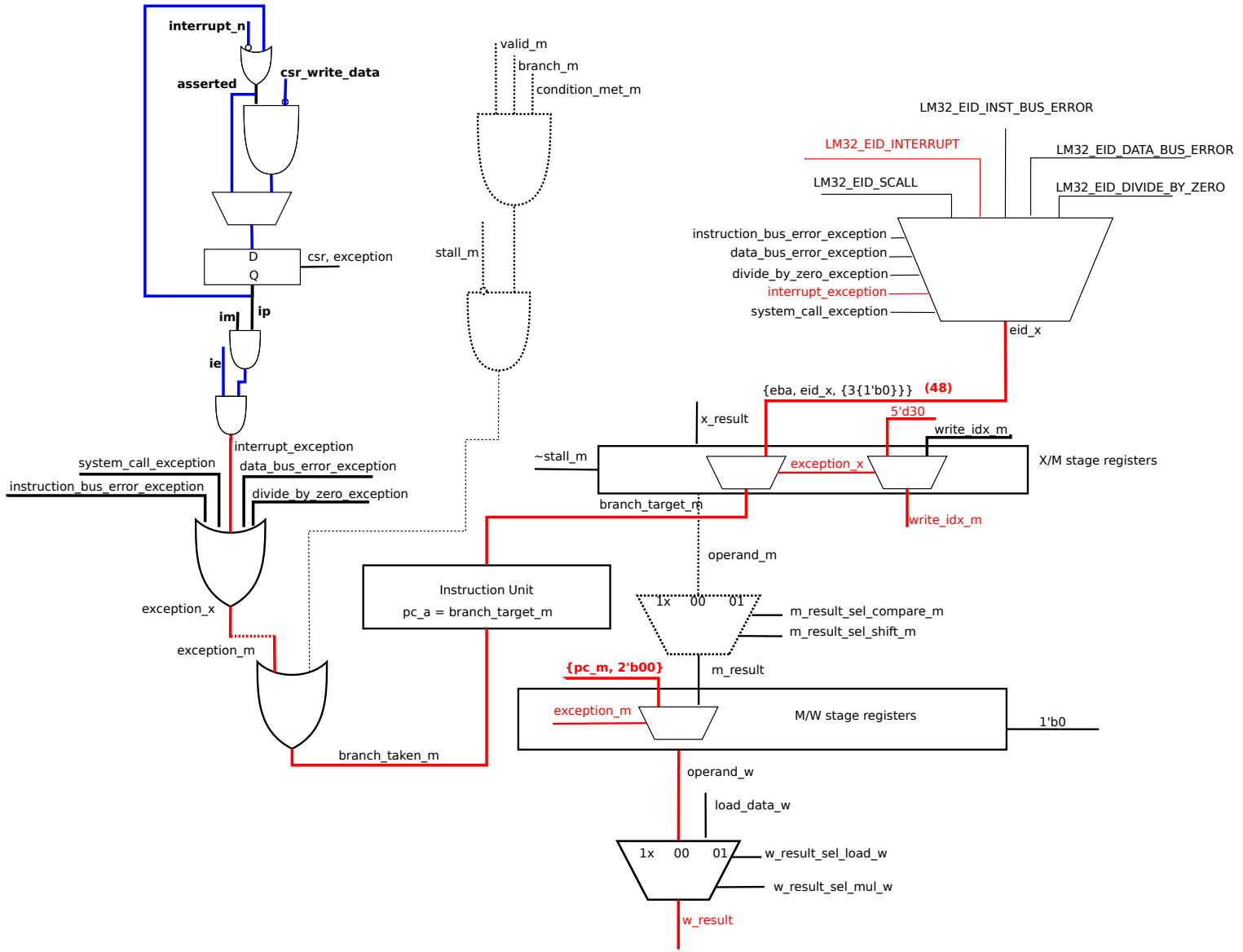


(a) Interrupt occurs after instruction at location N



(b) Return from interrupt

Interrupciones

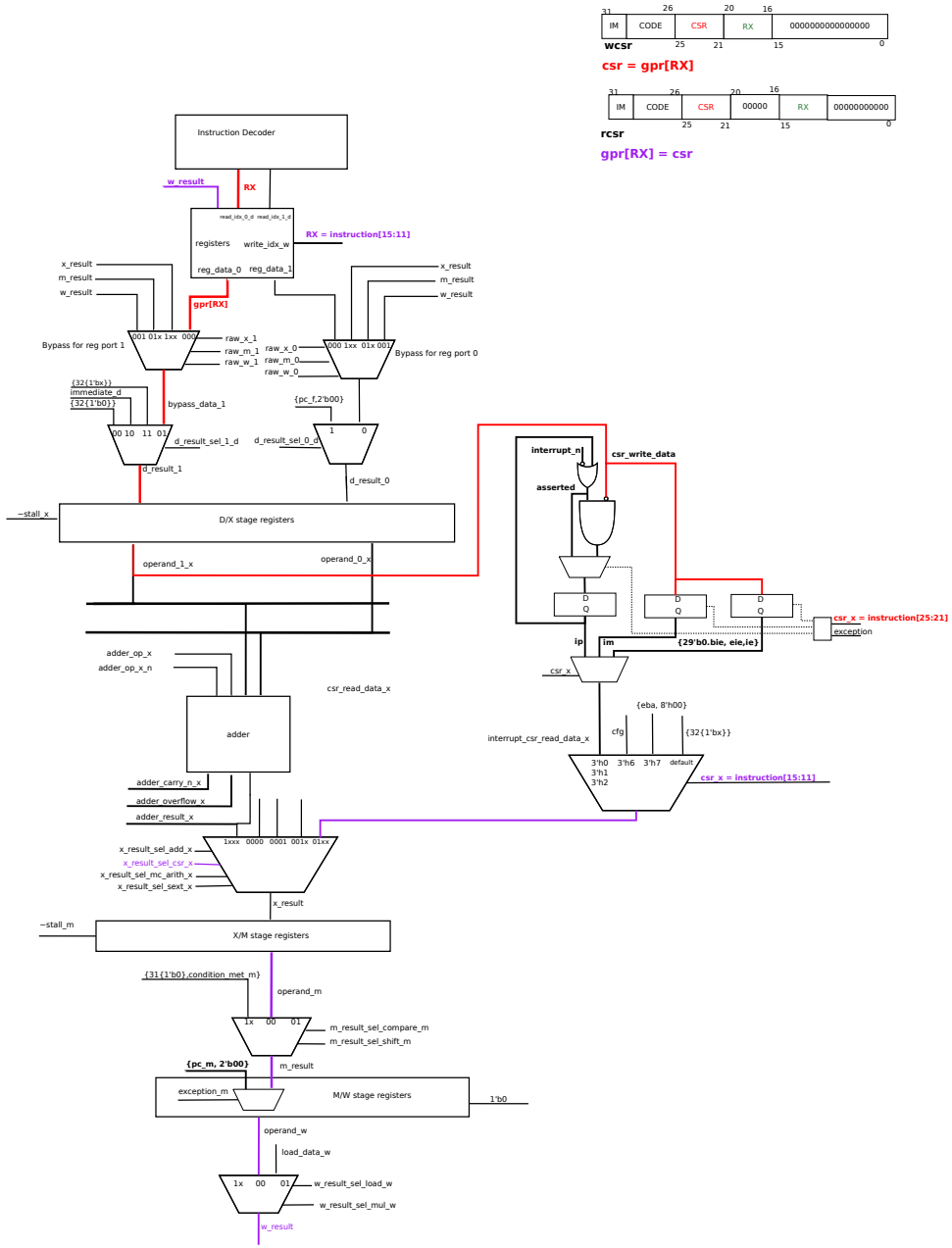


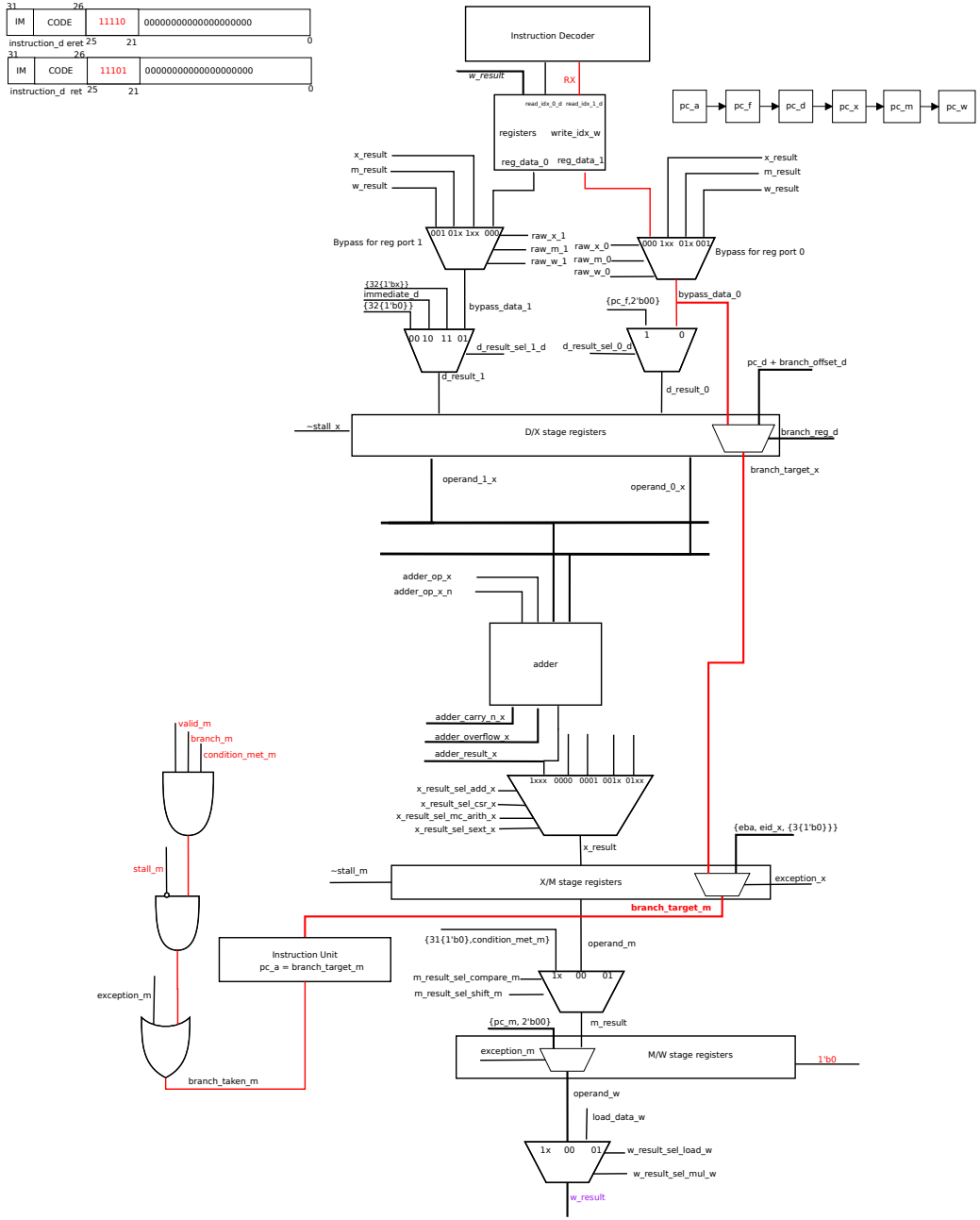
Rutina de atención a la interrupción

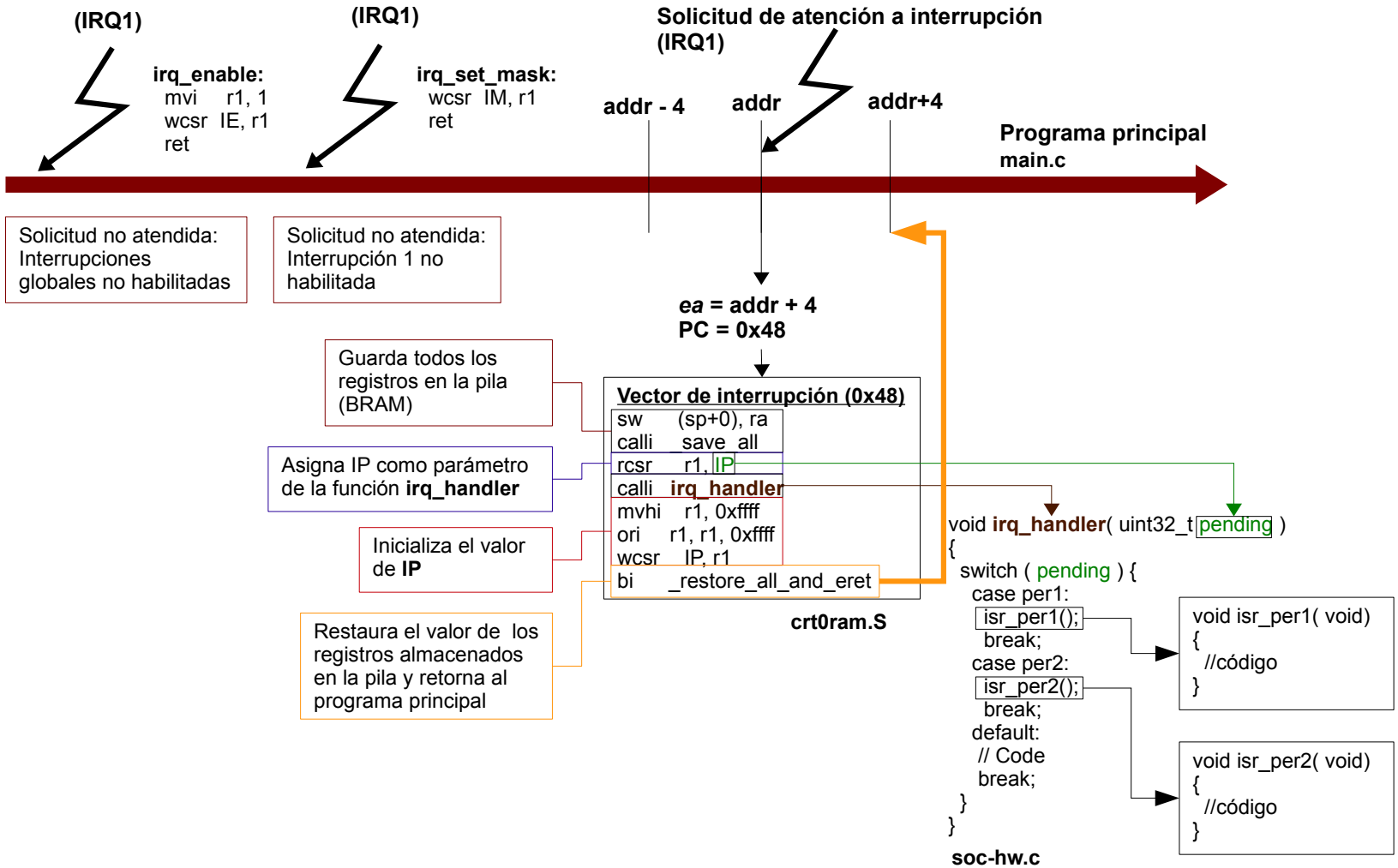
```
sw      (sp+0), ra
calli   _save_all
rcsr    r1, IP
calli   irq_handler
mvhi    r1, 0xffff
ori     r1, r1, 0xffff
wcsr    IP, r1
bi      _restore_all_and_eret
```

```
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
asserted
csr_write_data
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
read_idx_0_d read_idx_1_d
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
001 01x 1xx 000
Bypass for reg port 1
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
exception_m
branch_taken_m
operand_w
load_data_w
1x 00 01
w_result_sel_load_w
w_result_sel_mul_w
w_result


(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
}

PREGUNTAS