METODOLOGÍA DE DISEÑO E IMPLEMENTACIÓN DE SISTEMAS EMBEBIDOS

—Utilizando Herramientas Abiertas—

CARLOS IVÁN CAMARGO BAREÑO

ÍNDICE GENERAL

CAPÍTULO 1

IMPLEMENTACIÓN DE TAREAS SOFTWARE UTILIZANDO PROCESADORES SOFT CORE

1.1. Introducción

En el capítulo anterior se estudió la forma de implementar tareas hardware utilizando máquinas de estado algorítmicas. La implementación de tareas hardware es un proceso un poco tedioso ya que involucra la realización de una máquina de estados por cada tarea; la implementación del camino de datos se simplifica de forma considerable ya que existe un conjunto de bloques constructores que pueden ser tomados de una librería creada por el diseñador. El uso de tareas hardware se debe realizar únicamente cuando las restricciones temporales del diseño lo requieran, ya que como veremos en este capítulo, la implementación de tareas software es más sencilla y rápida.

La estructura de una máquina de estados algorítmica permite entender de forma fácil la estructura de un procesador ya que tienen los mismos componentes principales (unidad de control y camino de datos), la diferencia entre ellos es la posibilidad de programación y la configuración fija del camino de datos del procesador.

En este capítulo se estudiará la arquitectura del procesador MICO32 creado por la empresa Lattice semiconductor y gracias a que fué publicado bajo la licencia GNU, es posible su estudio, uso y modificación. En la primera sección se hace la presentación de la arquitectura; a continuación se realiza el análisis de la forma en que el procesador implementa las diferentes instrucciones, iniciando con las operaciones aritméticas y lógicas siguiendo con las de control de flujo de programa (saltos, llamado a función); después se analizarán la comunicación con la memoria de datos; y finalmente el manejo de interrupciones.

En la segunda sección se abordará la arquitectura de un SoC (System on a Chip) basado en el procesador LM32, se analizará la forma de conexión entre los periféricos y la CPU utilizando el bus wishbone; se realizará una descripción detallada de la programación de esta arquitectura utilizando herramientas GNU.

1.2. Arquitectura del procesador LM32

La figura 1.1 muestra el diagrama de bloques del soft-core LM32, este procesador utiliza 32 bits y una arquitectura de 6 etapas del pipeline; las 6 etapas del pipeline son:

1. *Address*: Se calcula la dirección de la instrucción a ser ejecutada y es enviada al registro de instrucciones.
2. *Fetch*: La instrucción se lee de la memoria.
3. *Decode*: Se decodifica la instrucción y se toman los operandos del banco de registros o tomados del bypass.
4. *Execute*: Se realiza la operacion especificada por la instrucción. Para instrucciones simples como las lógicas o suma, la ejecución finaliza en esta etapa, y el resultado se hace disponible para el bypass.
5. *Memory*: Para instrucciones más complejas como acceso a memoria externa, multiplicación, corrimiento, división, es necesaria otra etapa.
6. *Write back*: Los resultados producidos por la instrucción son escritas al banco de registros.

Banco de Registros

El LM32 posee 32 registros de 32 bits; el registro *r0* siempre contiene el valor 0, esto es necesario para el correcto funcionamiento de los compiladores de C y ensamblador; los siguientes 8 registros (r1 a r7) son utilizados para paso de argumentos y retorno de resultados en llamados a funciones; si una función requiere más de 8 argumentos, se utiliza la pila (*stack*). Los registros *r1 - r28* pueden ser utilizados como fuente o destino de cualquier instrucción. El registro *r29 (ra)* es utilizado por la instrucción *call* para almacenar la dirección de retorno. El registro *r30 (ea)* es utilizado para almacenar el valor del *contador de programa* cuando se presenta una excepción. El registro *r31 (ba)* almacena el valor del contador de programa cuando se presenta una excepción tipo *breakpoint* o *watchpoint*. Los registros *r26 (gp) r27 (fp)* y *r28 (sp)* son el puntero global, de frame y de pila respectivamente. Después del reset el valor no se define el valor de los registros, por lo que la primera acción que debe ejecutar el programa de inicialización en colocar un cero en el registro *r0 (**xor** r0, r0, r0)*

Registro de estado y control

Table 4 shows all of the names of the control and status registers (CSR), whether the register can be read from or written to, and the index used when accessing the register. Some of the registers are optional, depending on the configuration of the processor. All signal levels are active high.

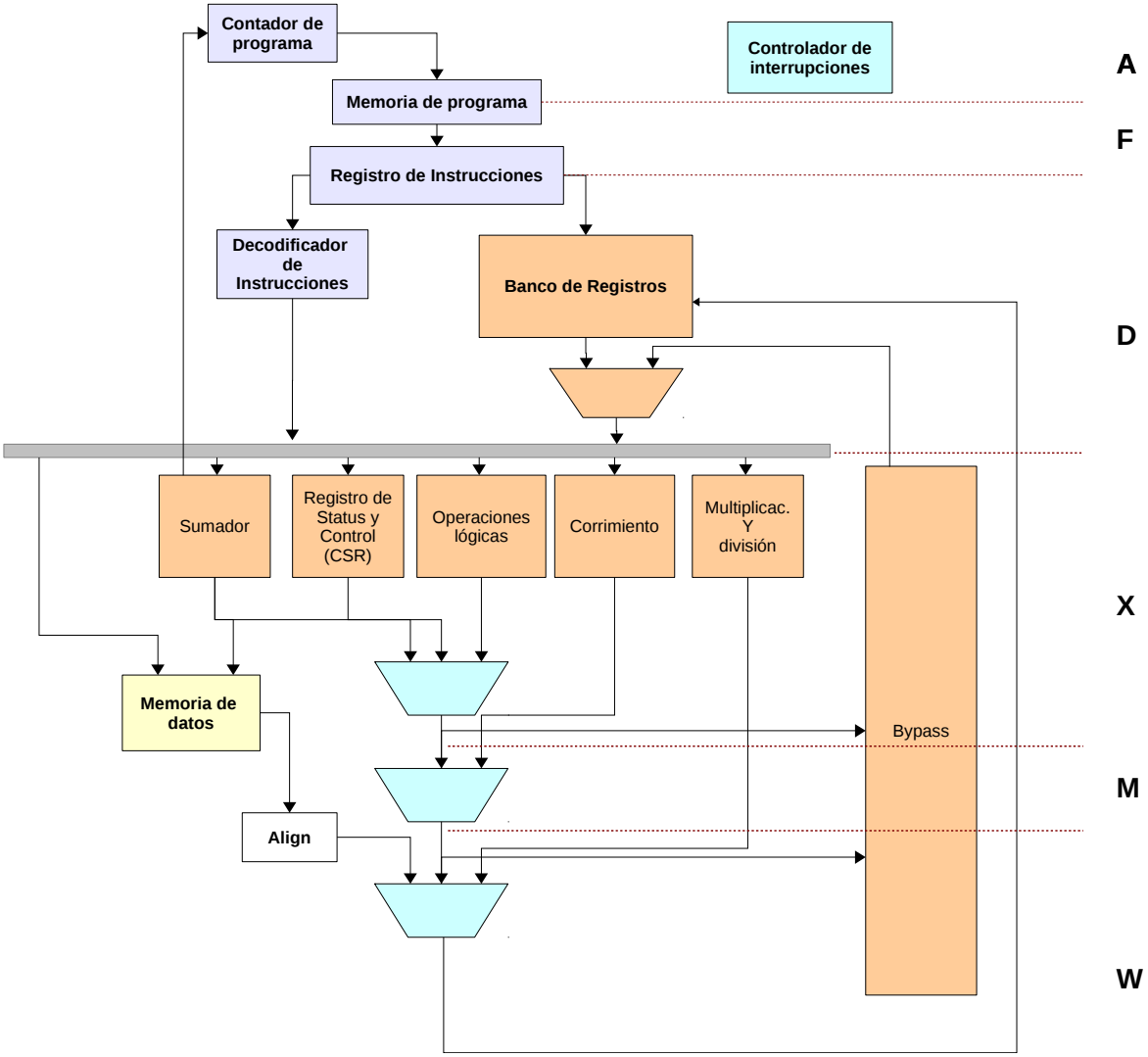


Figura 1.1: Diagrama de bloques del LM32

Cuadro 1.1: Registro de Estado y Control

Nombre	Index	Descripción
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

Contador de Programa (PC)

El contador de programa es un registro de 32 bits que contiene la dirección de la instricción 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*

EID Exception ID

El índice de la excepción es un número de 3 bits que indica la causa de la detención de la ejecución del programa. Las excepciones son eventos que ocurren al interior o al exterior del procesador y cambian el flujo normal de ejecución del programa. Los valores y eventos correspondientes son:

- **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 e una instrucción, típicamente 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, típicamente 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 dicisión 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 interrupción del periférico (IM).
- **7**: System Call; se presenta cuando se ejecuta la instricción *scall*.

IE Habilitación de Interrupción

IE contiene el flag 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.

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 activa la interrupción asociada. Los bits del registro IP deben ser borrados escribiendo un 1.

- tareas software - Arquitectura del LM32 (componentes básicos de un procesador) - Compilación de progamas para el LM32, explicar un ejemplo sencillo puede ser el de tipos de datos comentandop todos los archivos, lm32, crt0.s, etc - Set de instrucciones, con ejemplos en donde sea necesario como en: - llamado a funciones: Ejemplo sencillo que muestre como se pasan parámetros a través de r0, r1, r2. - saltos: If, while, forma - interrupciones explicar como se debe modificar el crt0.s para incluir los vectores de excepción y como se atiende la interrupción. - acceso a memoria externa: Explicar como se mapean los registros de los periféricos a C, y tipos de datos. - Acceso a memoria externa: Bus wishbone: Topologías, señales del WB, arquitectura del conbus, explicar uart y timer. - Como se forma el SoC con el LM32. Diagrama de bloques del SoC, explicando donde quedan los diferentes periféricos.

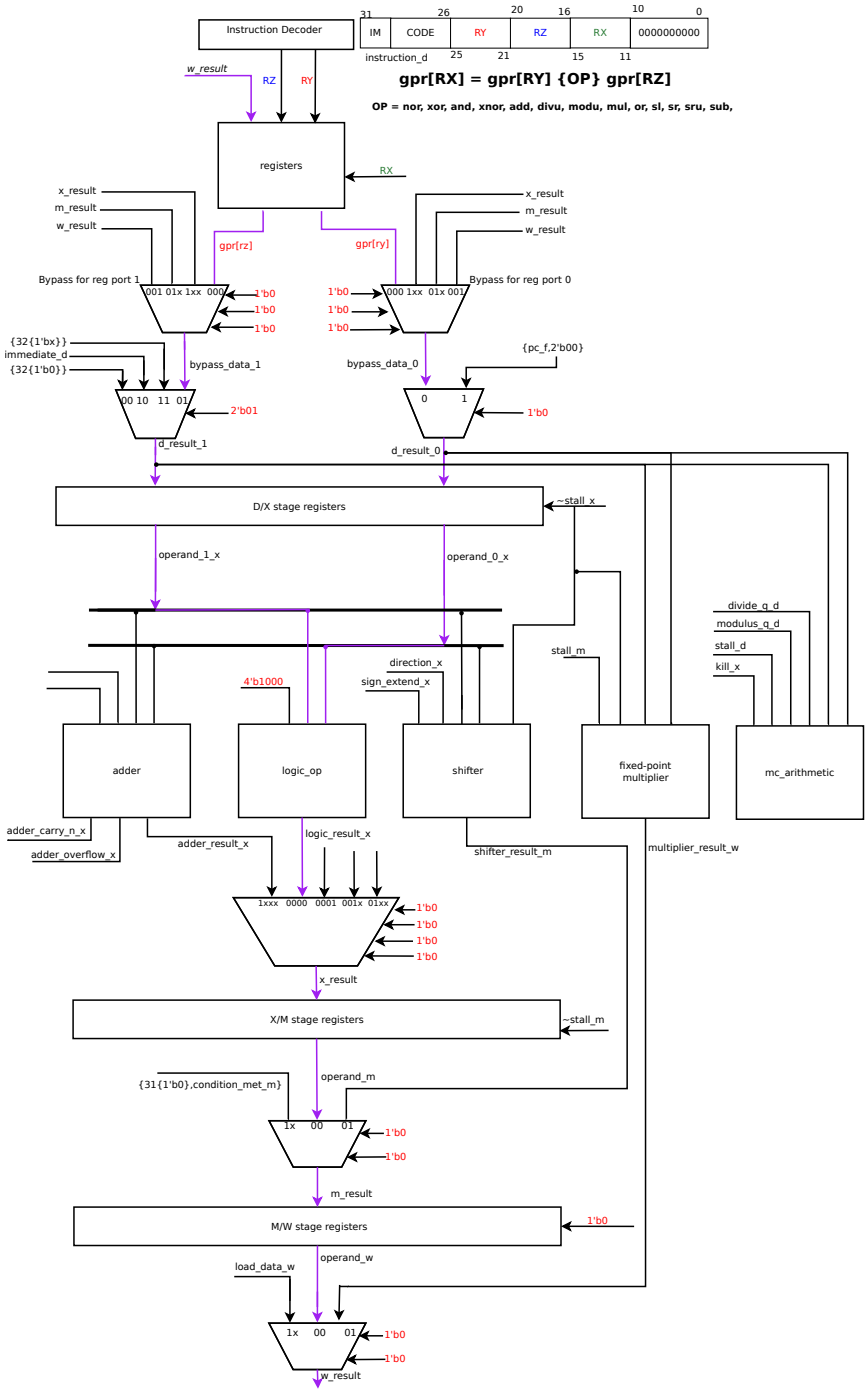


Figura 1.2: Camino de datos de las operaciones aritméticas y lógicas entre registros

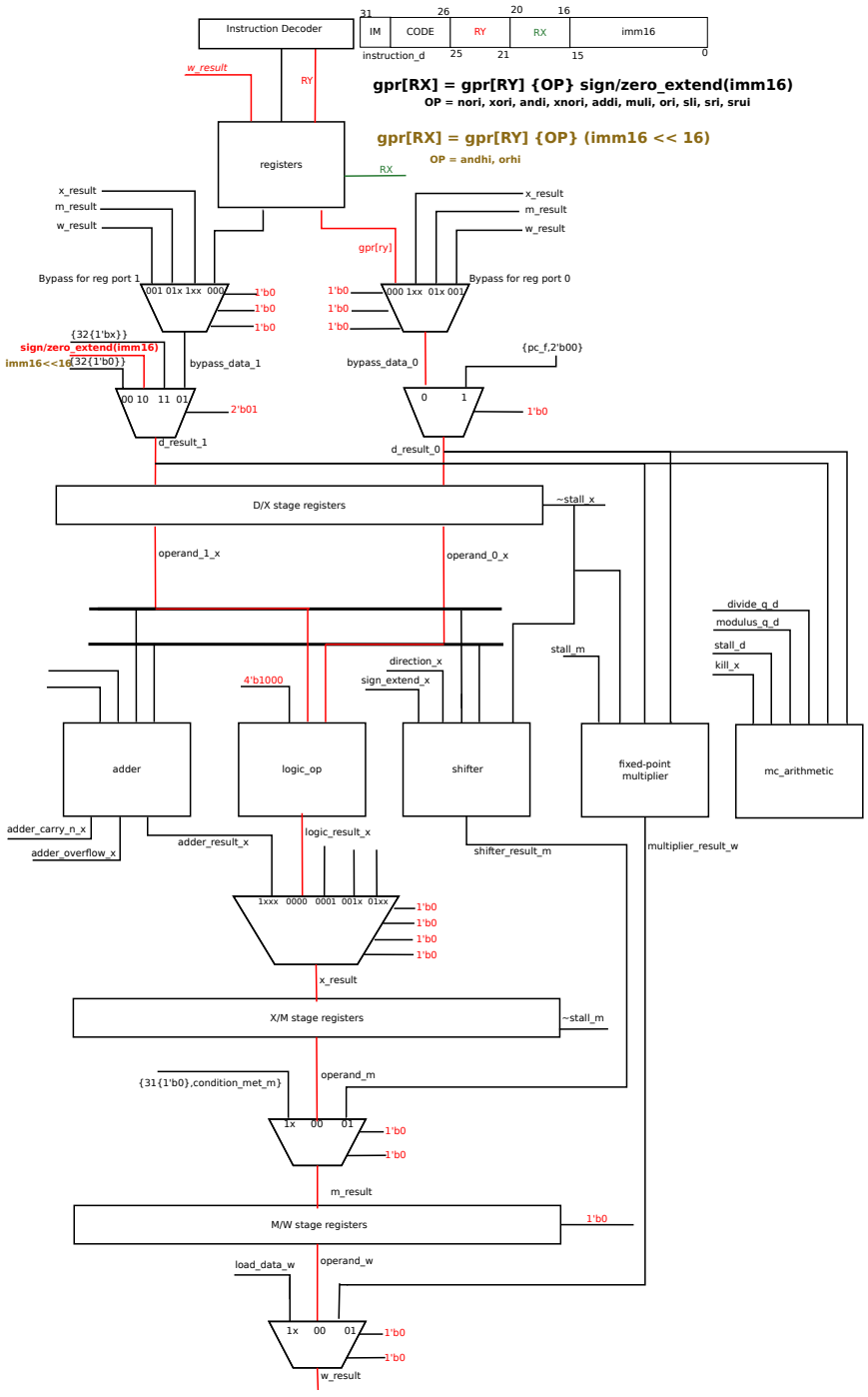
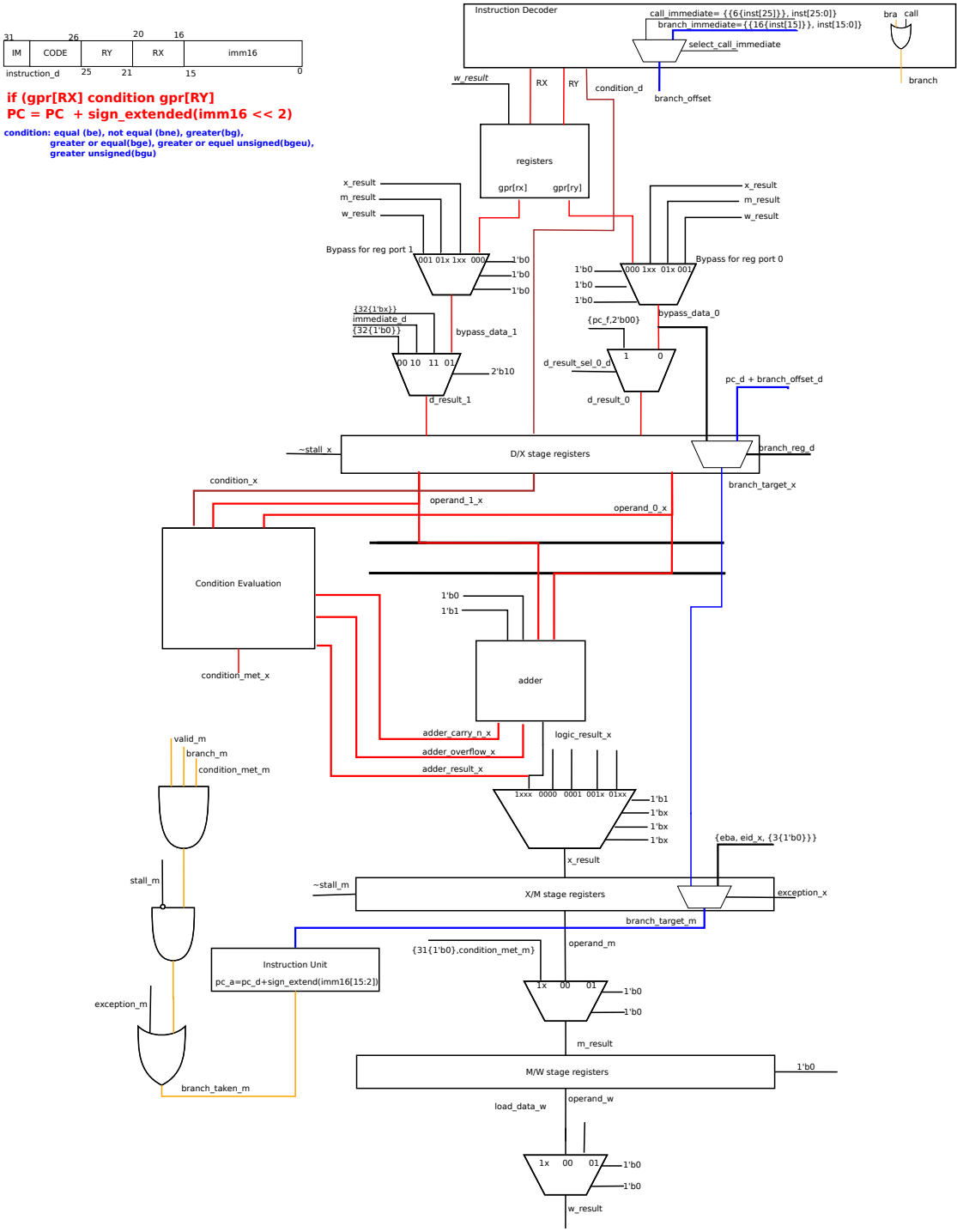


Figura 1.3: Camino de datos de las operaciones aritméticas y lógicas inmediatas


31
26
20
16
IM
CODE
RY
RX
imm16
instruction_d
25
21
15
0
if (gpr[RX] condition gpr[RY]
PC = PC + sign_extended(imm16 << 2)
condition: equal (be), not equal (bne), greater(bg),
greater or equal(bge), greater or equel unsigned(bgeu),
greater unsigned(bgu)
Instruction Decoder
call_immediate= {{6{inst[25]}}, inst[25:0]}
branch_immediate={{16{inst[15]}}, inst[15:0]}
select_call_immediate
bra
call
branch
w_result
RX
RY
condition_d
branch_offset
registers
gpr[rx]
gpr[ry]
x_result
m_result
w_result
Bypass for reg port 1
001 01x 1xx 000
1'b0
Bypass for reg port 0
000 1xx 01x 001
{32{1'bx}}
immediate_d
{32{1'b0}}
bypass_data_1
00 10 11 01
2'b10
d_result_1
{pc_f,2'b00}
bypass_data_0
d_result_sel_0_d
1
0
d_result_0
pc_d + branch_offset_d
~stall_x
D/X stage registers
branch_reg_d
branch_target_x
condition_x
operand_1_x
operand_0_x
Condition Evaluation
1'b1
adder
condition_met_x
adder_carry_n_x
adder_overflow_x
adder_result_x
logic_result_x
1xxx 0000 0001 001x 01xx
1'bx
valid_m
branch_m
condition_met_m
{eba, eid_x, {3{1'b0}}}
stall_m
~stall_m
X/M stage registers
exception_x
branch_target_m
operand_m
{31{1'b0},condition_met_m}
Instruction Unit
pc_a=pc_d+sign_extend(imm16[15:2])
1x 00 01
exception_m
m_result
M/W stage registers
branch_taken_m
load_data_w
operand_w
w_result

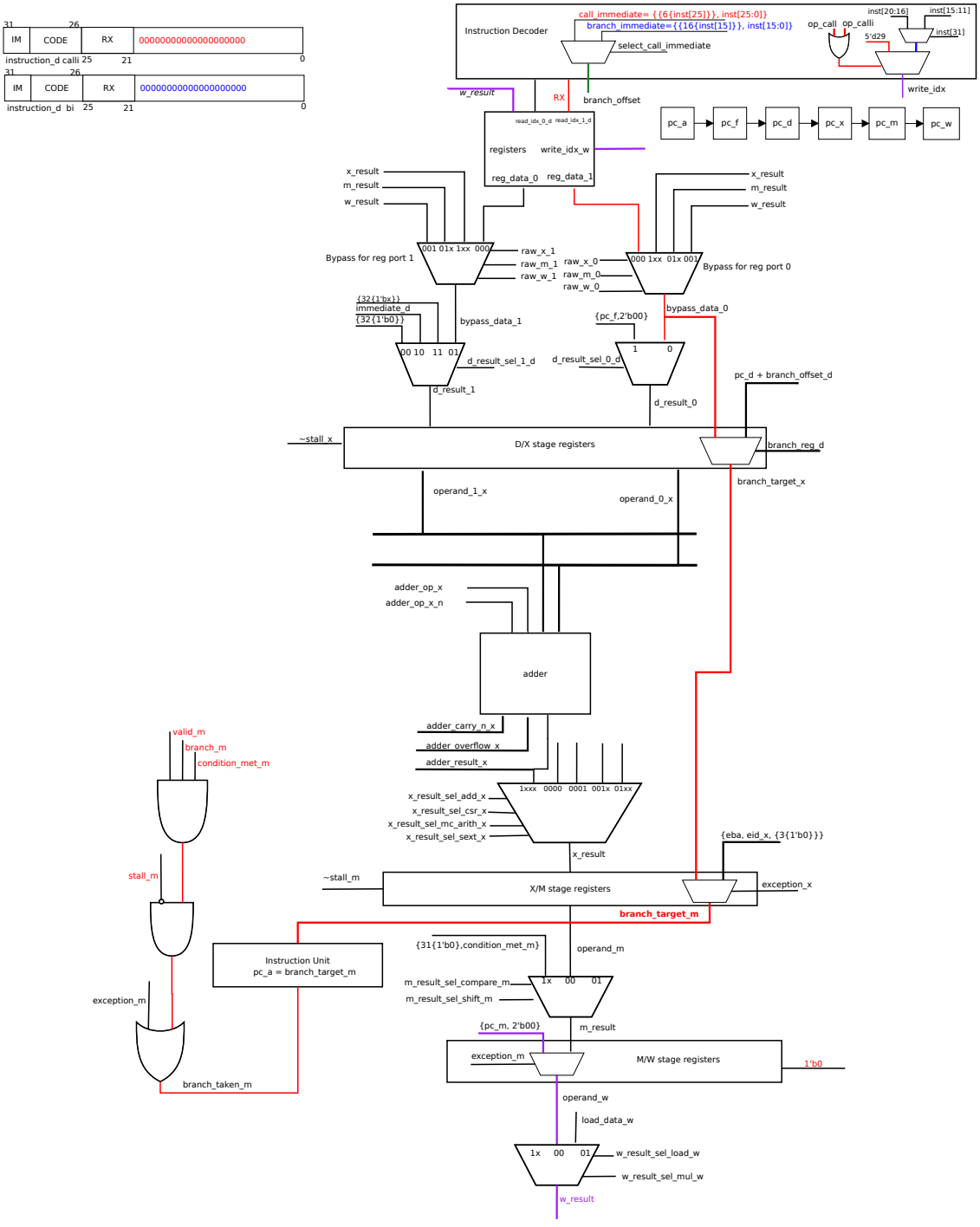


Figura 1.5: Camino de datos de los saltos inmediatos y llamado a funciones

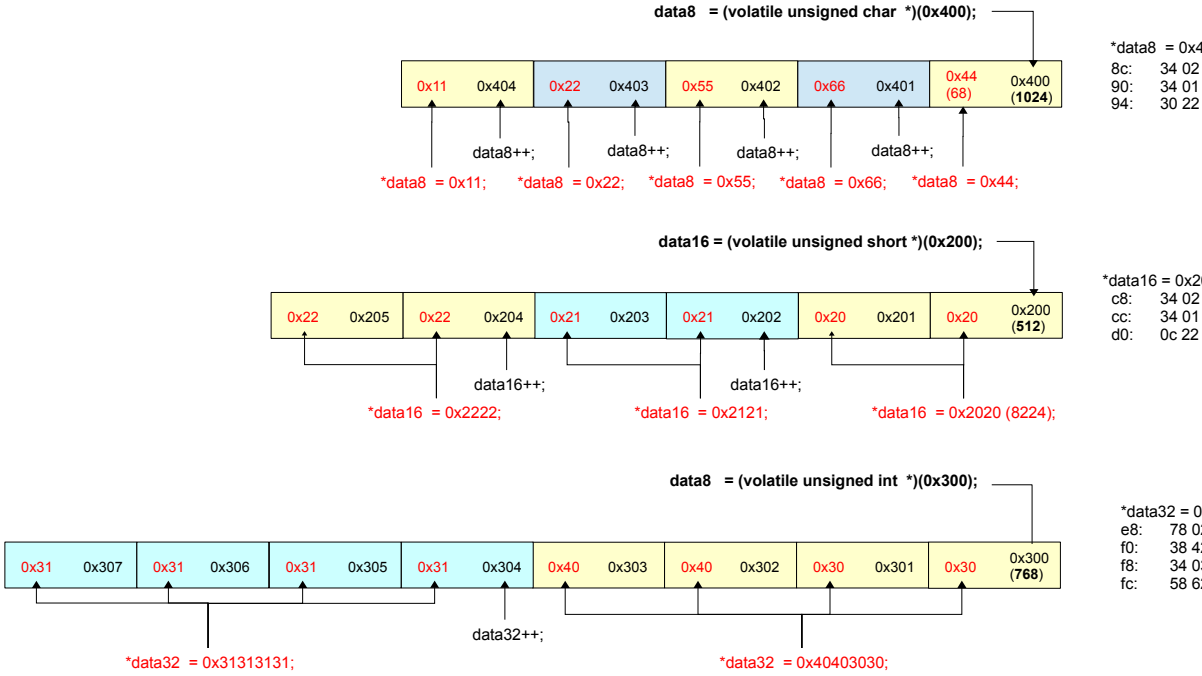


Figura 1.6: Tipos de datos soportados por el procesador Mico32

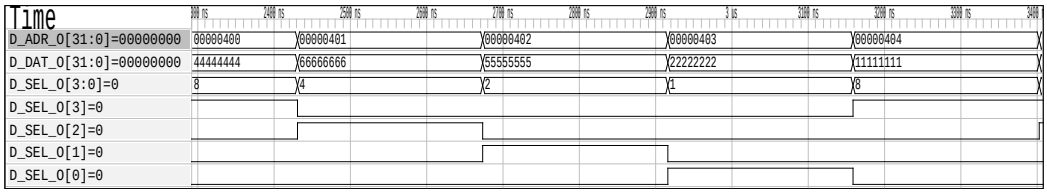


Figura 1.7: Acceso a un data tipo *char*

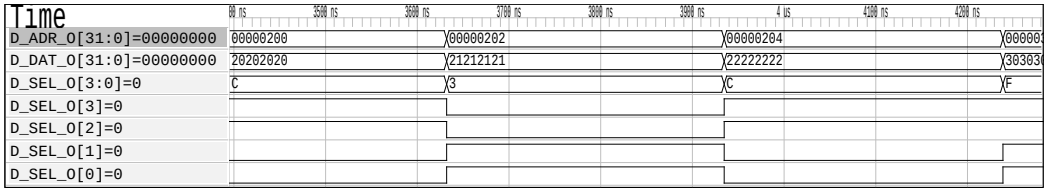


Figura 1.8: Acceso a un data tipo *short*

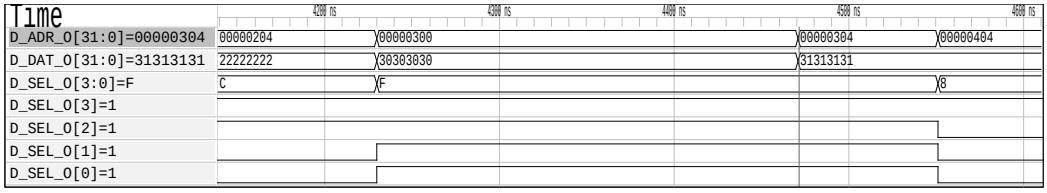


Figura 1.9: Acceso a un data tipo *int*

1.3. Set de Instrucciones del procesador Mico32

Instrucciones aritméticas

Entre registros

Inmediatas

Saltos

Condicionales

Llamado a función

Comunicación con la memoria de datos

Tipos de datos

Escritura

Lectura

Interrupciones

Explicar el código de atención a la irq

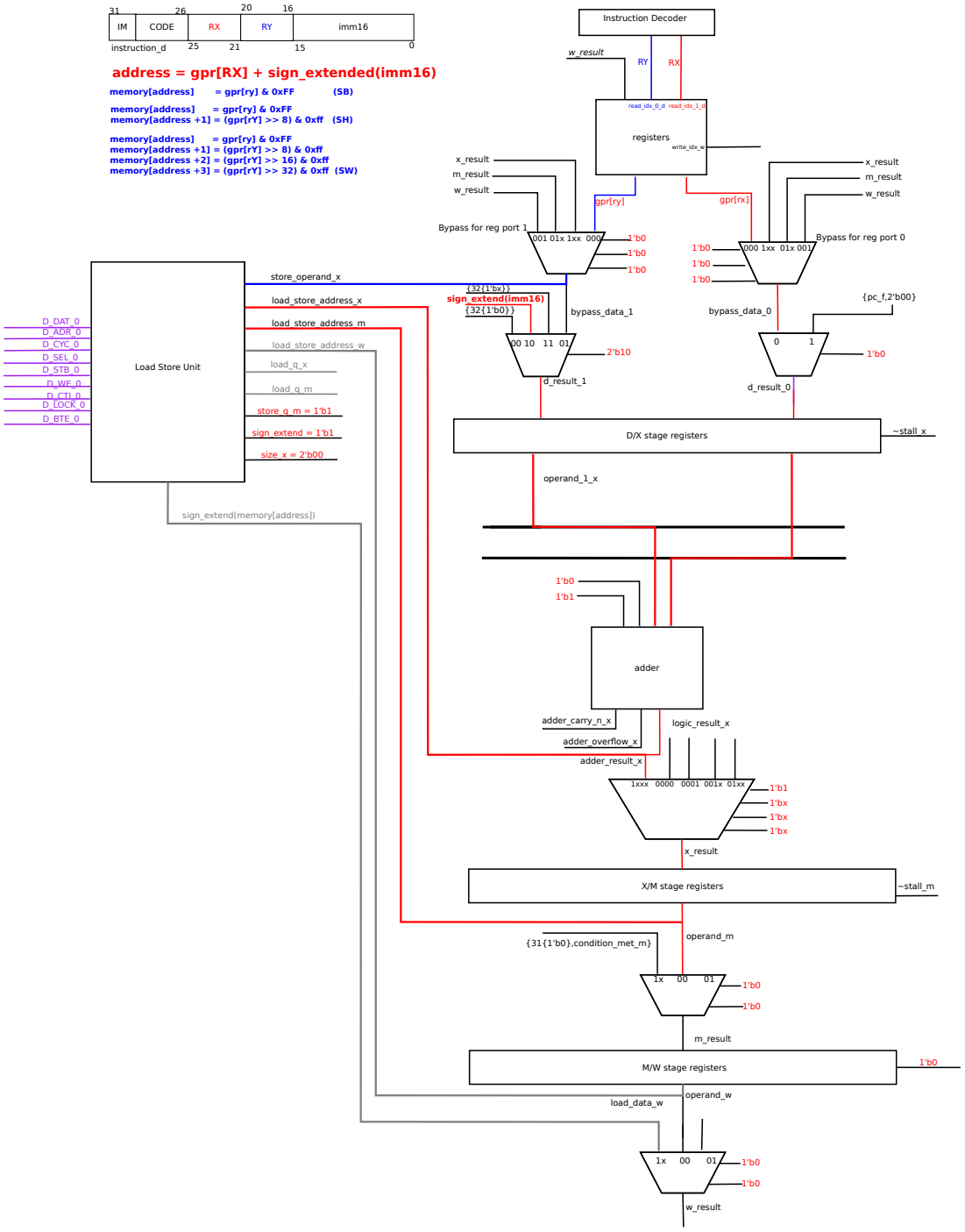


Figura 1.10: Camino de datos de las instrucciones de escritura a memoria

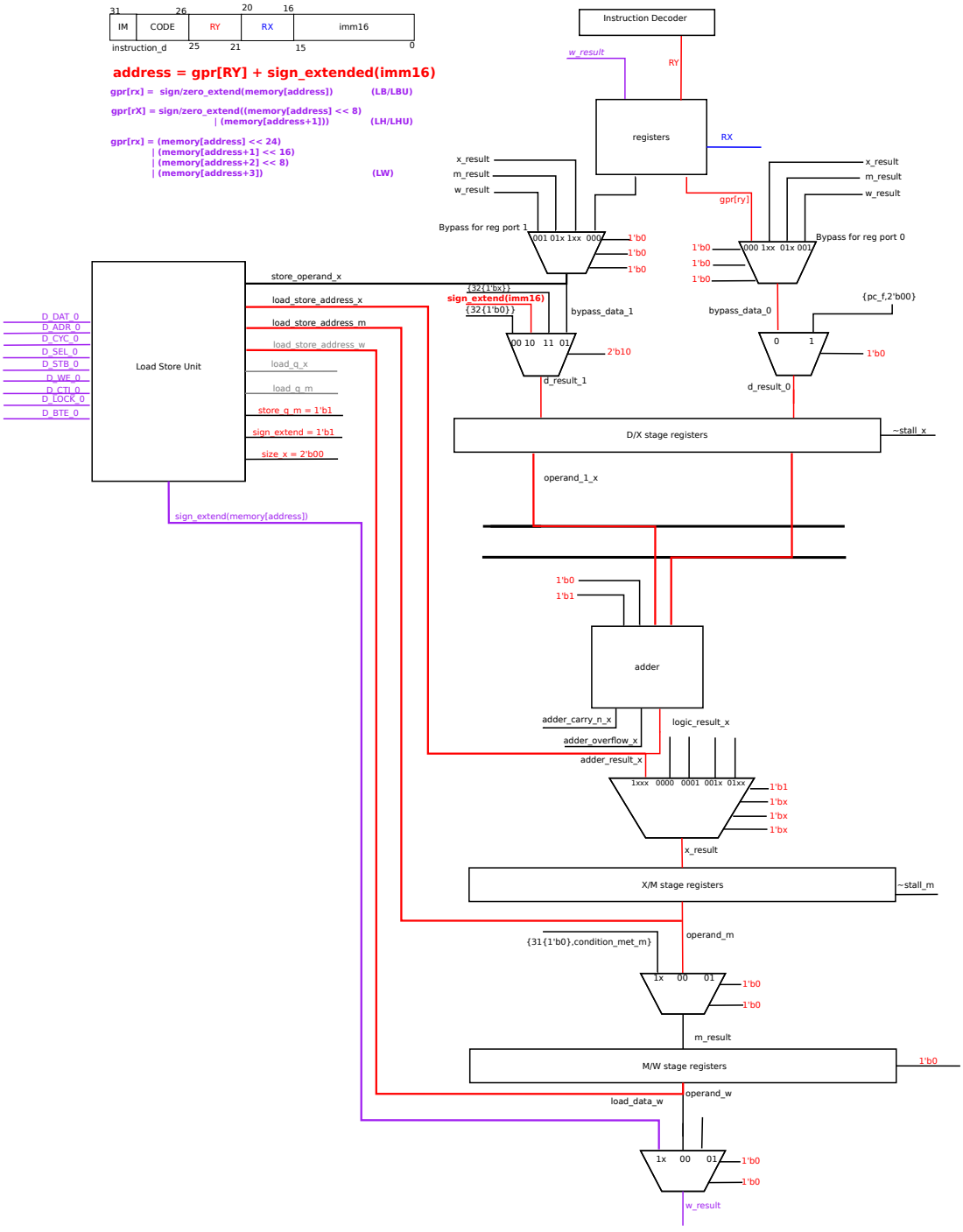


Figura 1.11: Camino de datos de las instrucciones de escritura a memoria

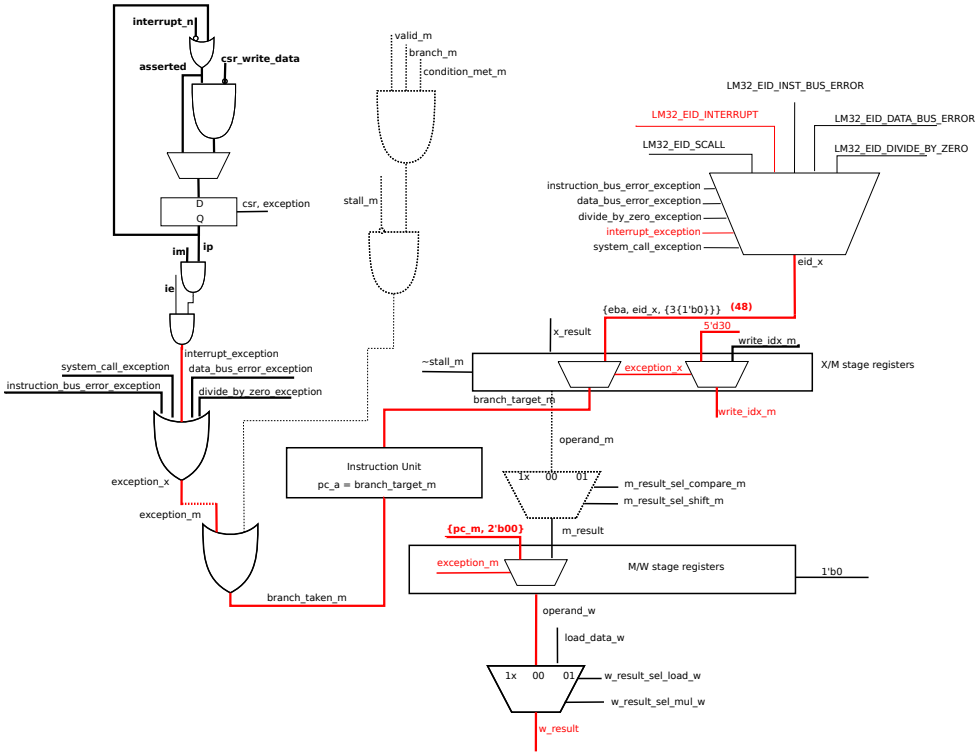


Figura 1.12: Camino de datos correspondiente a las generación de excepciones

1.4. Arquitectura del SoC LM32

Explicar el componente hardware y software,explicar la estructura sw y hw de los proyectos

Bus wishbone

Señales principales

- *ack_o*: La activación de esta señal indica la terminación normal de un ciclo del bus.
- *addr_i*: Bus de direcciones.
- *cyc_i*: Esta señal se activa que un ciclo de bus válido se encuentra en progreso.
- *sel_i*: Estas señales indican cuando se coloca un dato válido en el bus *dat_i* durante un ciclo de escritura, y cuando deberían estar presentes en

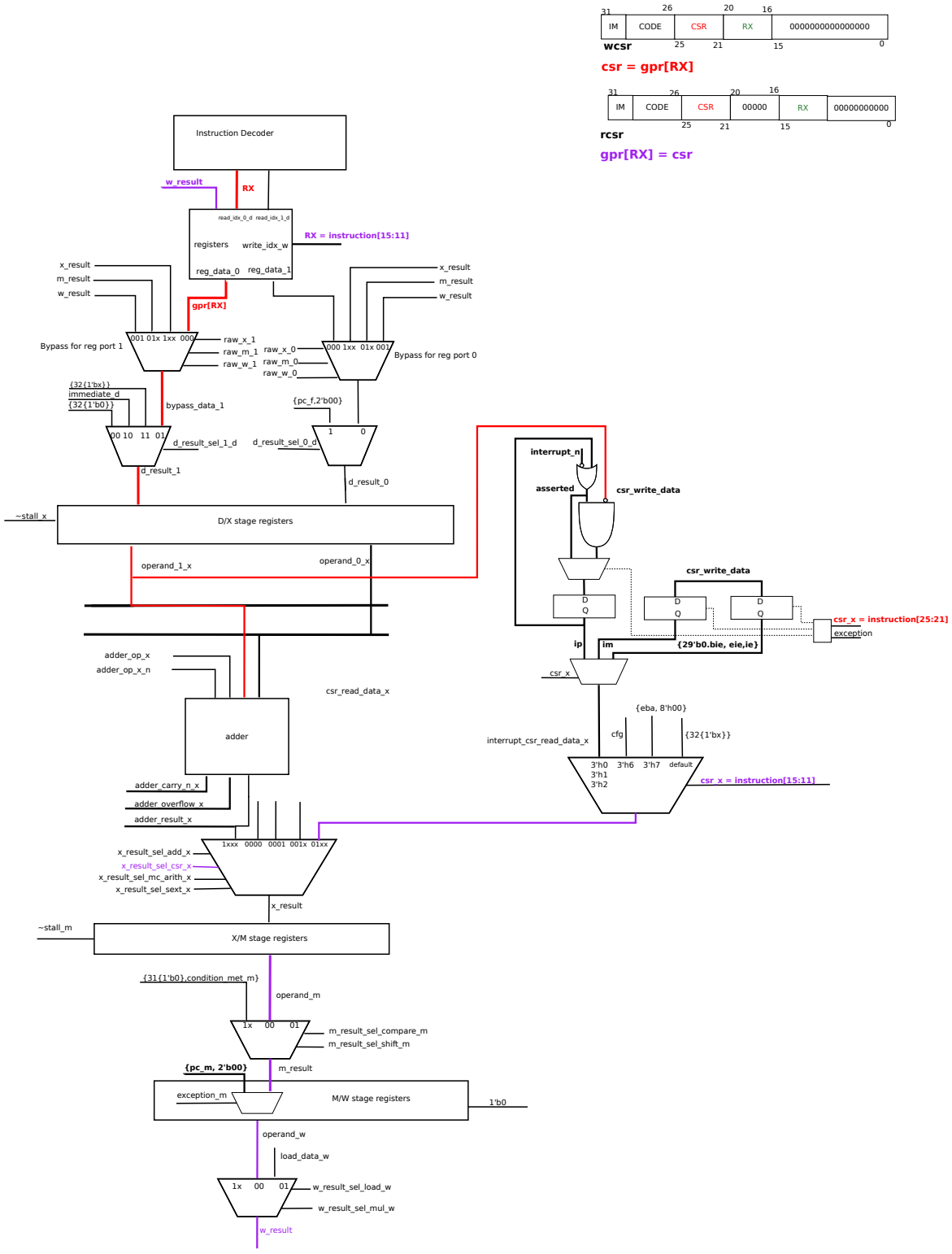


Figura 1.13: Camino de datos correspondiente al acceso de los registros asociados a las excepciones

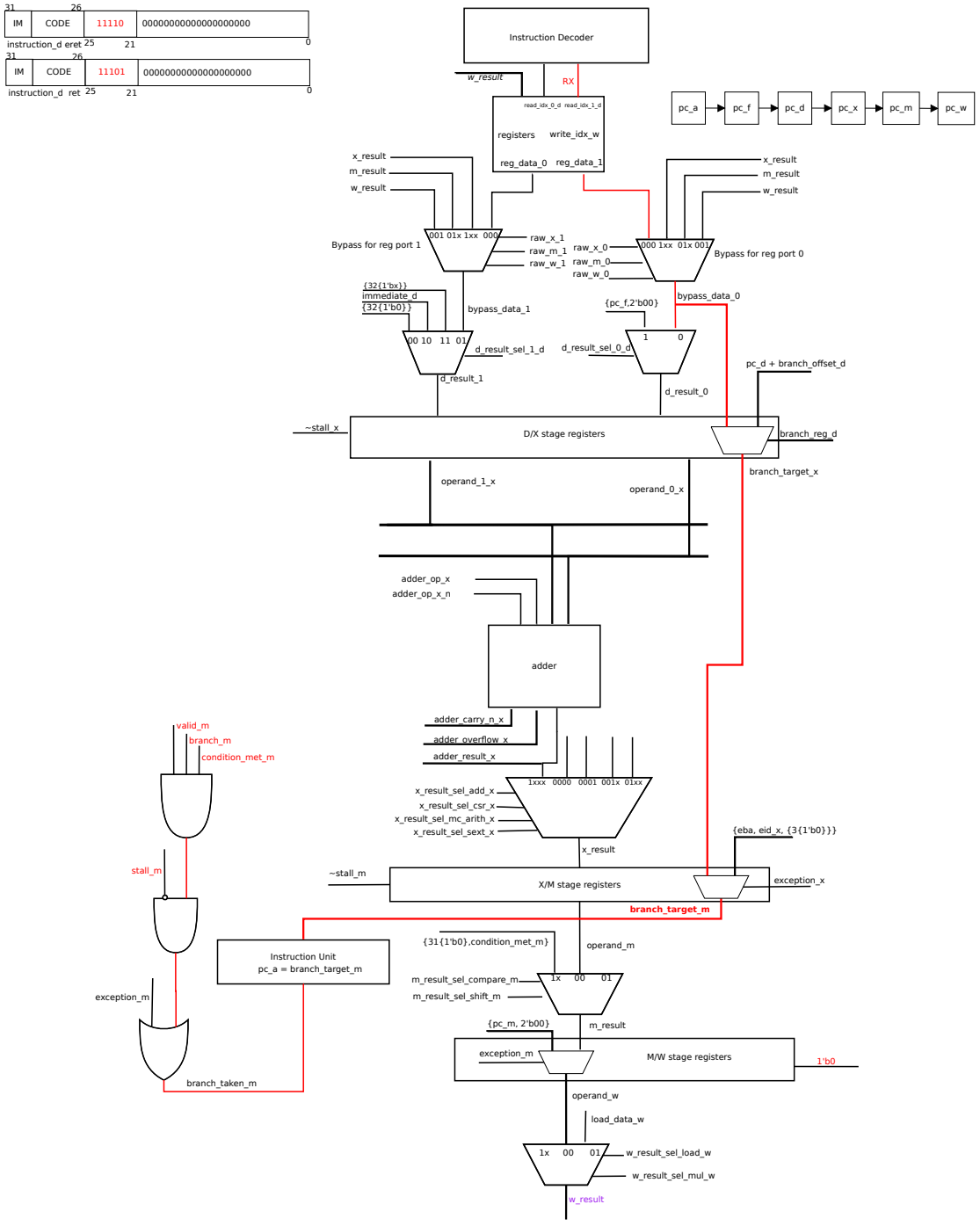


Figura 1.14: Camino de datos asociado al retorno de función y de excepción

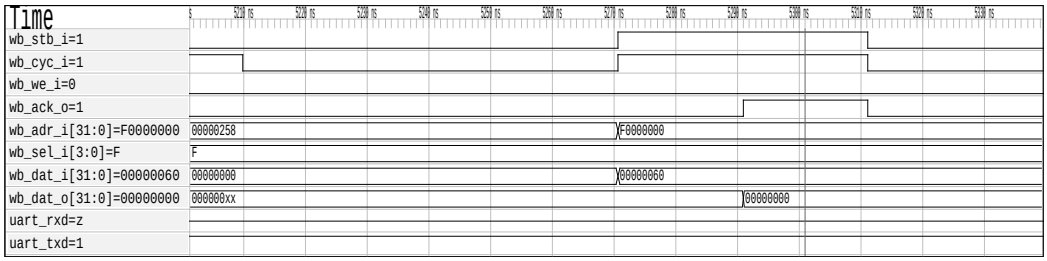


Figura 1.15: Ciclo de lectura del bus wishbone

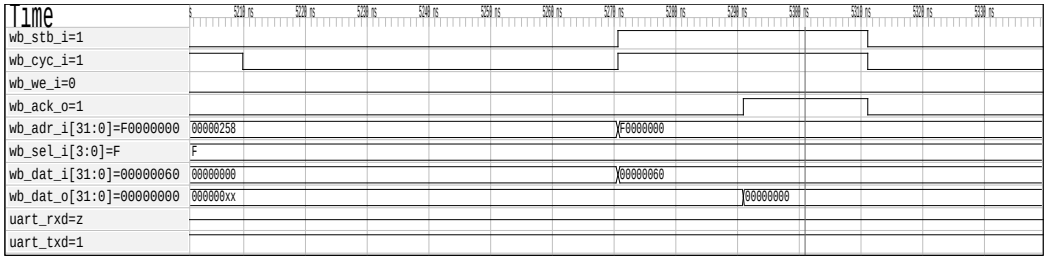


Figura 1.16: Ciclo de escritura del bus wishbone

el bus *dat_o* durante un ciclo de lectura. El número de señales depende de la granularidad del puerto. El LM32 maneja una granularidad de 8 bits sobre un bus de 32 bits, por lo tanto existen 4 señales para seleccionar el byte deseado (*sel_i(3:0)*).

- *stb_i*: Cuando se activa esta señal se indica al esclavo que ha sido seleccionado. Un esclavo wishbone debe responder a las otras señales únicamente cuando se activa esta señal. El esclavo debe activar la señal *ack_o* como respuesta a la activación de *stb_i*.

- *we_i*: Esta señal indica la dirección del flujo de datos, en un ciclo de lectura tiene un nivel lógico bajo y en escritura tiene un nivel lógico alto.

- *dat_i*: Bus de datos de entrada.

- *dat_o*: Bus de datos de salida.

Interface del bus wishbone

explicar como funciona el conmax, mostrar diagrama de flujo y simulación

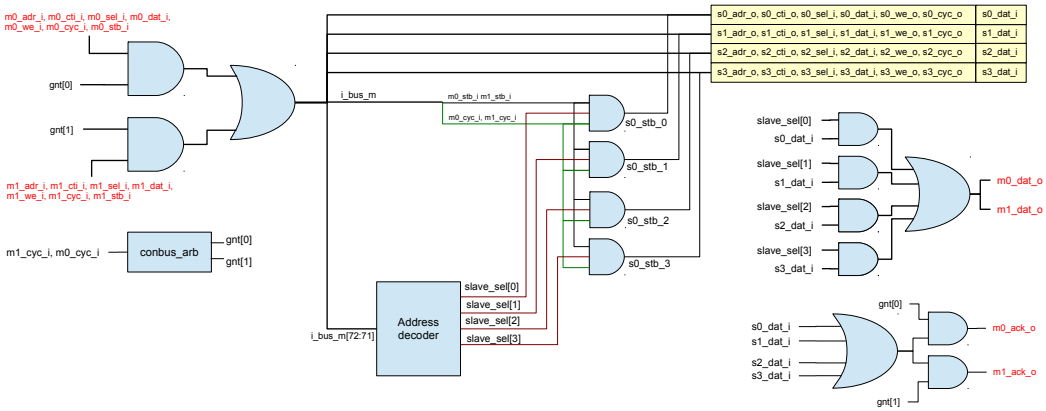


Figura 1.17: Circuito de interconexión del bus wishbone

Comunicación con periféricos

EXplicación de la UART y del TIMER globales interface en C.

Lectura

Escritura

Interfaz Software

Volatile keyword says the compiler that no optimiztion on the variable. The volatile keyword acts as a data type qualifier. The volatile qualifier alters the default behaviour of the variable and does not attempt to optimize the storage referenced by it. - Martin Leslie volatile means the storage is likely to change at anytime and be changed but something outside the control of the user program. This means that if you reference the variable, the program should always check the physical address (ie a mapped input fifo), and not use it in a cacheed way.

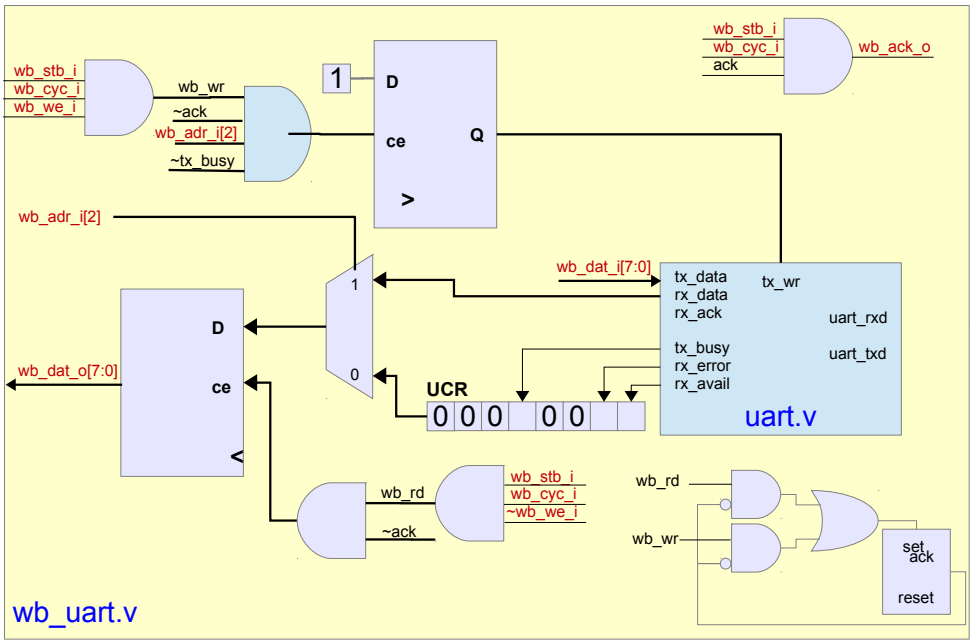


Figura 1.18: Ejemplo de periférico wishbone: UART

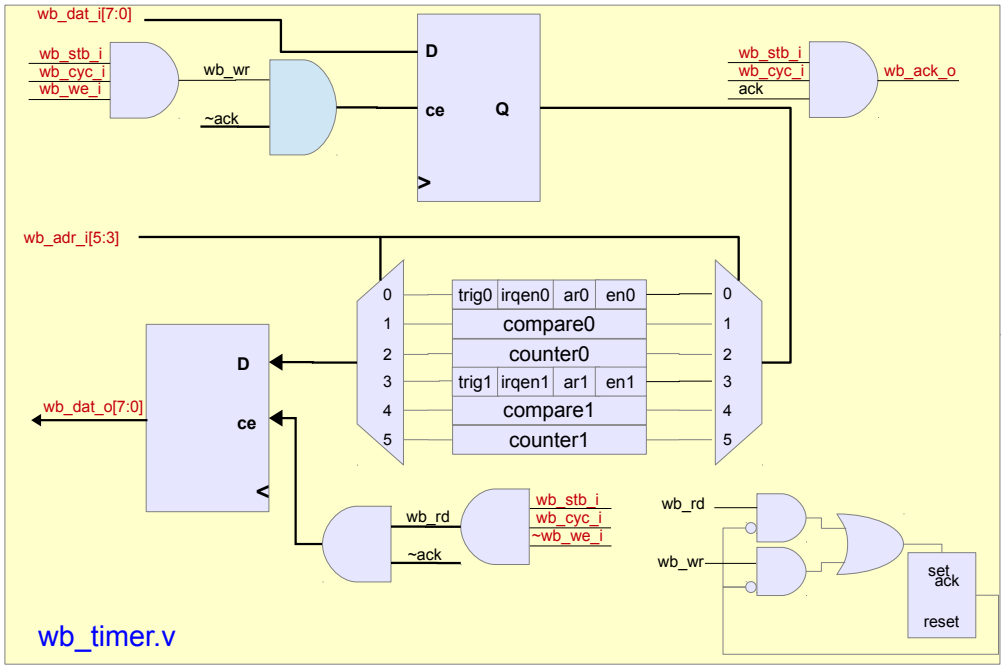


Figura 1.19: Ejemplo de periférico wishbone: TIMER

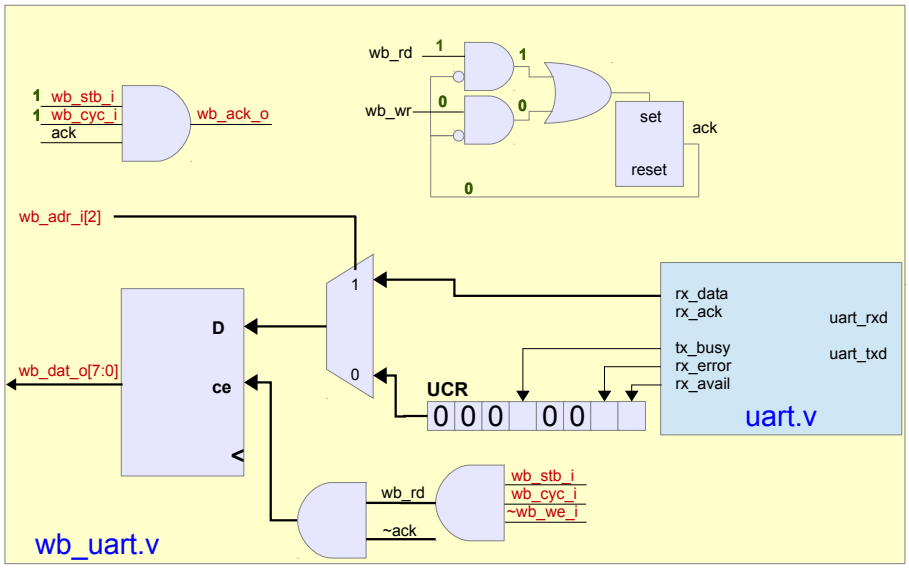


Figura 1.20: Circuito equivalente de lectura de la UART

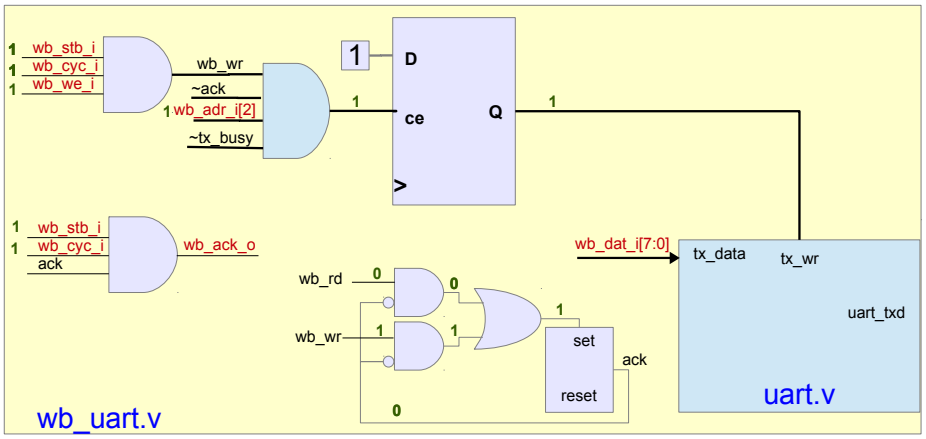


Figura 1.21: Circuito equivalente de escritura de la UART

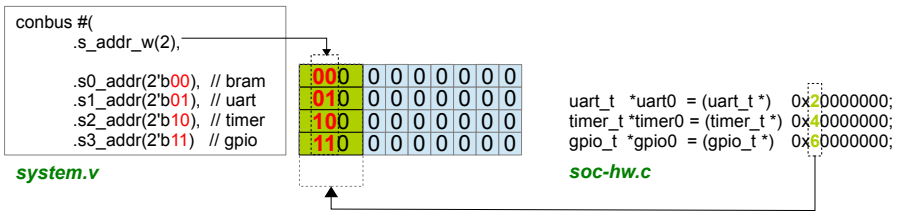


Figura 1.22: Asignación de la dirección de memoria a los periféricos

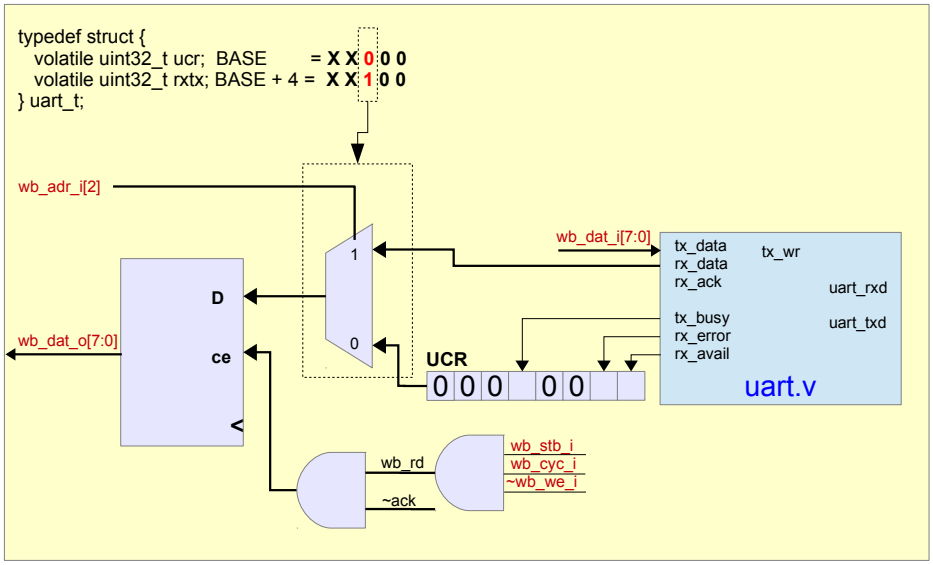


Figura 1.23: Definición de la dirección de los registros internos de la UART

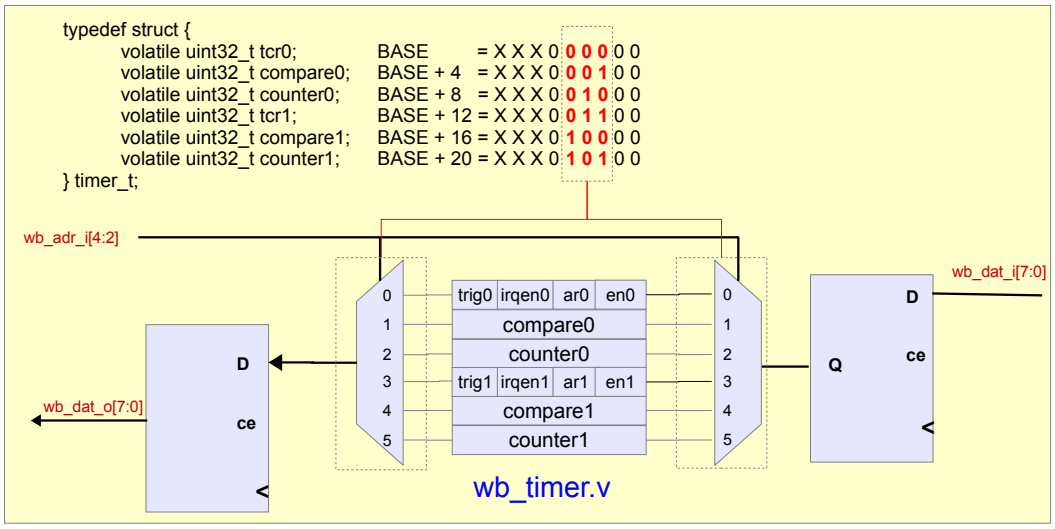


Figura 1.24: Definición de la dirección de los registros internos del TIMER

Bibliografía