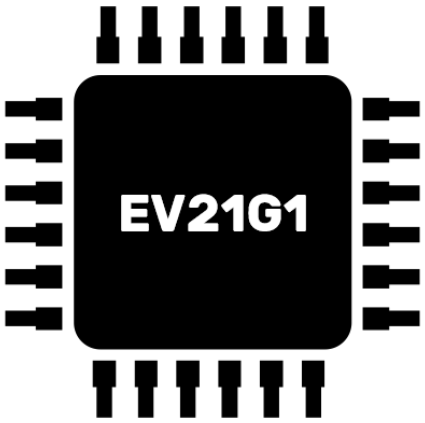


Arquitectura del procesador

EV21G1

v1.1.1

Santiago Arribere, Matías Francois, Joaquín Gaytan, Lucas Kammann, Pablo Scheinfeld

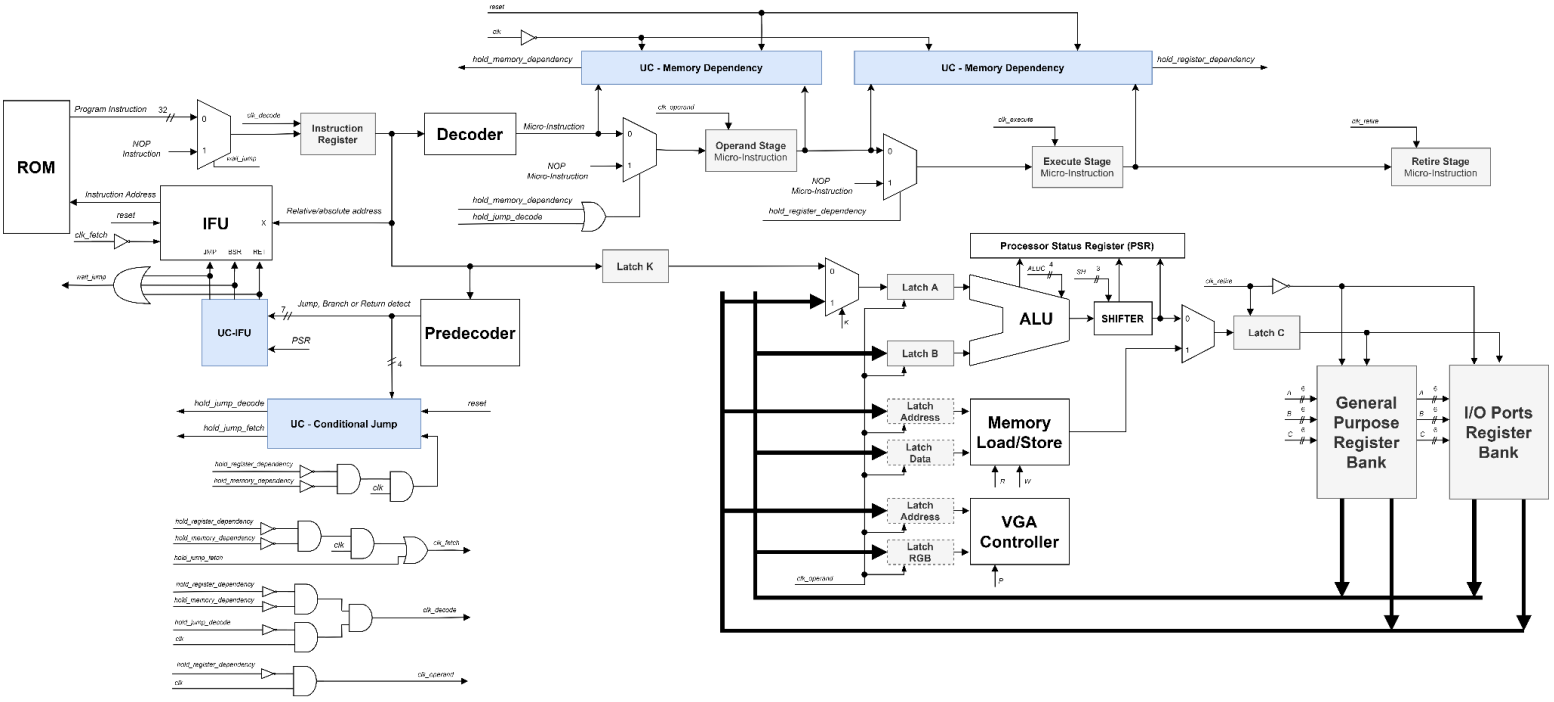
---

AÑO 2021

Índice

Arquitectura

El procesador EV21G1 posee una arquitectura de 32 bits, es un procesador programable de alta performance con una arquitectura de tres buses y una estructura de pipeline de 5 etapas. Se disponen de 28 registros de propósito general, 2 puertos de entrada y 2 puertos de salida y otros reservados.



Espacios de memoria

El procesador posee una arquitectura Harvard, según la cual existen dos memorias dedicadas para el programa (ROM) y los datos o variables (RAM). El procesador posee conexión a un bus de datos y direcciones de memoria, donde tanto las palabras como las direcciones poseen un ancho de 32 bits. La memoria se puede direccionar utilizando cualquiera de los registros de propósito general disponibles.

Dirección de inicio	**Dirección de fin**	**Región**
00000000	00001FFF	*RAM*
00002000	FFFFFFF	*RESERVED*

Para la memoria de programa, existe un bus de direcciones que posee un ancho de 12 bits a través del cual se comunica la posición del programa para buscar en memoria la próxima instrucción. Las palabras de instrucción ocupan un ancho de 32 bits en memoria.

Registros

Los registros son palabras de 32 bits, y se clasifican según si son de propósito general o si sirven a algún propósito específico dentro del procesador, de forma tal que los registros se encuentran categorizados funcionalmente. Este diseño presenta dos ventajas, la posibilidad de separar los bancos de

registros según el tipo de función que cumple, permitiendo que en un futuro se tenga en cuenta para una arquitectura superescalar si se buscará expandir la capacidad de procesamiento del procesador. Además, la identificación de grupos marcados de registros permite crear un espacio de registros dando lugar a futuras incorporaciones. Es decir, se reserva lugar específicamente contemplado para tipos de registros.

N°	***Nombre***	***Tipo***
0	**R0**	*General*
1	**R1**	
2	**R2**	
3	**R3**	
4	**R4**	
5	**R5**	
6	**R6**	
7	**R7**	
8	**R8**	
9	**R9**	
10	**R10**	
11	**R11**	
12	**R12**	
13	**R13**	
14	**R14**	
15	**R15**	
16	**R16**	
17	**R17**	
18	**R18**	
19	**R19**	
20	**R20**	
21	**R21**	
22	**R22**	
23	**R23**	
24	**R24**	

25	**R25**	
26	**R26**	
27	**R27**	
28	**RESERVED**	
29	**RESERVED**	
30	**RESERVED**	
31	**RESERVED**	
32	**PI0**	*Puertos*
33	**PI1**	
34	**RESERVED**	
35	**RESERVED**	
36	**PO0**	
37	**PO1**	
38	**RESERVED**	
39	**RESERVED**	
40-62	**RESERVED**	
63	**NONE**	

PSR: Processor Status Register

El **PSR** o **Processor Status Register** es el registro que contiene el estado del procesador luego de la última operación ejecutada en la unidad aritmética lógica o en el módulo de desplazamiento, y según el cual se evalúa cómo deben resolverse las operaciones de salto condicional.

- El contenido del PSR sólo refleja el resultado de la operación inmediatamente anterior
- Para conocer cuándo y cómo pueden cambiar cada indicador del estado, ver las instrucciones disponibles en el procesador
- El indicador **N** especifica que el último resultado fue negativo
- El indicador **Z** especifica que el último resultado fue nulo
- El indicador **CY** especifica que en la última operación hubo carry
- El indicador **V** especifica que en la última operación hubo overflow

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	**Z**	**CY**	**V**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Set de instrucciones

Formato de instrucciones

En la siguiente tabla se muestran los formatos de instrucciones que se utilizan para codificar las instrucciones, se busca emplear un formato ortogonal para facilitar la decodificación. Es importante aclarar que algunas modificaciones se podrían hacer para aprovechar la característica de expanding opcodes, permitiendo que operaciones de menor cantidad de registros tengan más espacio de códigos de operación. Se descartó esta alternativa para mantener un formato más sencillo y simple, que cumple en abundancia con las necesidades del set de instrucciones y permite una decodificación simplificada.

Formato N° 1

El formato permite codificar instrucciones que trabajen sin operandos, pero con la opción de utilizar ninguno, uno, dos o tres registros. Estos registros se encuentran vinculados directamente con los buses de la arquitectura, y se puede seleccionar cuál de ellos está siendo utilizado.

Formato N° 2

El formato permite codificar instrucciones que trabajen con operandos, ya sea sin o con un registro. El registro puede ser vinculado a cualquiera de los dos buses disponibles.

Formato	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
***N° 1**: Sin operando*	0	A	B	C	OP												Rk (A)						Rj (B)								Ri (C)	
***N° 2**: Con operando*	1	0	B	C	OP												OPERAND (A)														Ri (B, C)	

Instrucciones

Nemónico	Instrucción	Significado	PSR				Formato	Opcode
			N	Z	CY	V		
NOP	No operation		x	x	-	-	N° 1	0 0 0 0 0 0 0 0 0 0
AND Ri,Rj,Rk	AND	$Ri = Rj\ AND\ Rk$	+	+	-	-	N° 1	0 0 0 0 0 0 1 1 0 1
OR Ri,Rj,Rk	OR	$Ri = Rj\ OR\ Rk$	+	+	-	-	N° 1	0 0 0 0 0 0 0 0 0 1
ORK Ri,K	OR with constant	$Ri = Ri\ OR\ K$	+	+	-	-	N° 2	0 1 0 0 0 0
ANK Ri,K	AND with constant	$Ri = Ri\ AND\ K$	+	+	-	-	N° 2	0 1 0 0 1 1
ADC Ri,Rj,Rk	ADD with carry	$Ri = Rj + Rk + CY$	+	+	+	+	N° 1	0 0 0 0 0 0 0 0 1 0
ADD Ri,Rj,Rk	ADD without carry	$Ri = Rj + Rk$	+	+	+	+	N° 1	0 0 0 0 0 0 0 0 1 1

MOV Ri,Rj	Move register to register	$Ri = Rj$	+	+	-	-	N° 1	0 0 0 0 0 0 0 1 0 0
CPL Ri,Rj	Complement register	$Ri = \bar{R} j$	+	+	-	-	N° 1	0 0 0 0 0 0 0 1 0 1
MMK Ri,K	Move constant to most significant word of register	$Ri = K << 16$	+	+	+	+	N° 2	0 1 0 0 0 1
MLK Ri,K	Move constant to least significant word of register	$Ri = K$	+	+	-	-	N° 2	0 1 0 0 1 0
STR Rk,Rj	Store data in memory	$M(Rj) = Rk$	x	x	-	-	N° 1	0 0 0 0 0 0 0 1 1 0
LDR Ri,Rj	Load data from memory	$Ri = M(Rj)$	x	x	-	-	N° 1	0 0 0 0 0 0 0 1 1 1
CLR CY	Clear the carry	$CY = 0$	x	x	0	x	N° 1	0 0 0 0 0 0 1 0 0 0
SET CY	Set the carry	$CY = 1$	x	x	1	x	N° 1	0 0 0 0 0 0 1 0 0 1
JMP X	Unconditional jump	$PC = X$	x	x	-	-	N° 2	0 0 0 1 0 0
JZE X	Jump if zero	$IF\ PSR\_Z = 1\ THEN\ PC = X$	x	x	-	-	N° 2	0 0 0 0 0 0
JNE X	Jump if negative	$IF\ PSR\_N = 1\ THEN\ PC = X$	x	x	-	-	N° 2	0 0 0 0 0 1
JOV X	Jump if overflow	$IF\ PSR\_O = 1\ THEN\ PC = X$	x	x	-	-	N° 2	0 0 0 0 1 0
JCY X	Jump if carry	$IF\ PSR\_CY = 1\ THEN\ PC = X$	x	x	-	-	N° 2	0 0 0 0 1 1
RET	Return from subroutine	$PC = Latest\ Stored\ PC\ \{+ 1\}$	x	x	-	-	N° 1	0 0 0 0 0 0 1 0 1 0
BSR S	Branch to subroutine	$Save\ PC;\ PC = PC + S$	x	x	-	-	N° 2	0 0 1 1 0 0
VGP Rk,Rj	Print the color Rj in the pixel Rk.	$P(Rk) = Rj$	x	x	-	-	N° 1	0 0 0 0 0 0 1 0 1 1

La convención usada para denotar el comportamiento de los indicadores del PSR en cada instrucción se muestra a continuación,

- **-** No hay cambios en el indicador
- **0** El indicador pasa a un estado lógico bajo
- **1** El indicador pasa a un estado lógico alto
- **+** El indicador cambia con la operación, según corresponda al resultado
- **x** El indicador puede tomar un valor desconocido sin significado o sentido

Microarquitectura

Formato de microinstrucciones

En la siguiente tabla se describe el formato de la palabra de control o microinstrucción empleada en el procesador para controlar las diferentes etapas de hardware de la microarquitectura.

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UC	UB	UA	ALUC				SH			K	R	W	A						B				C								X	P

UC Señal para indicar si la microinstrucción utiliza el bus C
UB Señal para indicar si la microinstrucción utiliza el bus B
UA Señal para indicar si la microinstrucción utiliza el bus A
ALUC Señal de control de la unidad lógica aritmética
SH Señal de control de la unidad de desplazamiento
K Indica que en el bus A se carga una constante
R Señal para comandar a la memoria una operación de lectura
W Señal para comandar a la memoria una operación de escritura
A Número de registro a cargar en el bus A
B Número de registro a cargar en el bus B
C Número de registro en el que se guarda el resultado de la ejecución
X Reservado
P Señal para comandar al controlador VGA una operación print

Microinstrucciones

Nemónico	UC	UB	UA	ALUC				SH			K	R	W	A	B	C	X	P
AND Ri,Rj,Rk	1	1	1	0	1	1	1	0	0	0	0	0	0	Rk	Rj	Ri	0	0
OR Ri,Rj,Rk	1	1	1	0	1	1	0	0	0	0	0	0	0	Rk	Rj	Ri	0	0
ORK Ri,K	1	1	0	0	1	1	0	0	0	0	1	0	0	0	Ri	Ri	0	0
ANK Ri,K	1	1	0	0	1	1	1	0	0	0	1	0	0	0	Ri	Ri	0	0
ADC Ri,Rj,Rk	1	1	1	0	1	0	1	0	0	0	0	0	0	Rk	Rj	Ri	0	0
ADD Ri,Rj,Rk	1	1	1	0	1	0	0	0	0	0	0	0	0	Rk	Rj	Ri	0	0
MOV Ri,Rj	1	1	0	0	0	0	1	0	0	0	0	0	0	0	Rj	Ri	0	0
CPL Ri,Rj	1	1	0	0	0	1	1	0	0	0	0	0	0	0	Rj	Ri	0	0
MMK Ri,K	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	Ri	0	0
MLK Ri,K	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	Ri	0	0
STR Ri,Rj	0	1	1	1	1	1	1	1	1	1	0	0	1	Rk	Rj	63	0	0

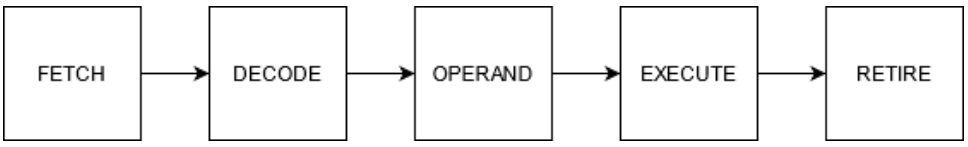
LDR Ri,Rj	1	1	0	1	1	1	1	1	1	1	0	1	0	0	Rj	Ri	0	0
CLR CY	0	0	0	1	0	1	1	1	1	1	0	0	0	0	0	63	0	0
SET CY	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	63	0	0
JMP X	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	63	0	0
JZE X	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	63	0	0
JNE X	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	63	0	0
JOV X	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	63	0	0
JCY X	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	63	0	0
RET	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	63	0	0
BSR S	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	63	0	0
VGP Rk, Rj	0	1	1	1	1	1	1	1	1	1	0	0	0	Rk	Rj	63	0	1
NOP	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	63	0	0

Pipeline de 5 etapas

El procesador implementa un pipeline de cinco etapas para mejorar la capacidad de procesamiento, mediante el paralelismo a nivel de instrucciones en etapas independientes entre sí que permiten avanzar de forma simultánea múltiples instrucciones.

Descripción general

En la siguiente figura se pueden observar las cinco etapas que componen al pipeline.



Fetch

Esta etapa se encarga de buscar en la memoria del programa la próxima instrucción, se encuentra asistida por una IFU o Instruction Fetch Unit que brinda funciones y/o capacidades para el manejo de saltos relativos, absolutos e incrementos en la posición del programa.

IFU: Instruction Fetch Unit

La unidad de fetch de instrucciones, posee el registro PC (Program Counter) el cual se actualiza con cada flanco descendente de clock, para buscar la siguiente instrucción a memoria en los flancos ascendentes del clock. El program counter posee 3 tipos de actualizaciones las cuales pueden ser: de forma absoluta, especial para las instrucciones de retornos de subrutina o de salto; o de forma relativa, incrementando

según una constante, para el caso de los saltos a subrutinas; o incrementando en 1 sola posición para el resto de las instrucciones.

Nombre	Tipo	Ancho	Significado
x	Entrada	16 bits	*Bus de entrada para carga de constantes*
bsr	Entrada	1 bit	*Señal de un salto relativo*
ret	Entrada	1 bit	*Señal de un salto de retorno*
jmp	Entrada	1 bit	*Señal de un salto absoluto*
clk	Entrada	1 bit	*Señal de clock para actualizar registros*
reset	Entrada	1 bit	*Señal de reset, para volver a 0 el PC*
next_address	Salida	16 bits	*Bus de salida*

La IFU cuenta con distintos bloques:

- Registro PC (Program Counter) y bloques lógicos: el registro y los bloques lógicos combinacionales que lo rodean para calcular la siguiente posición de memoria donde buscar la próxima instrucción. Cuenta con un sumador mediante el cual se incrementa el PC a la siguiente posición o se le suma un desplazamiento relativo para un salto a subrutina. Además, se puede cargar un valor de forma absoluta, para los retornos de subrutina o para los saltos absolutos.

- Stack Subroutine: Es una estructura de tipo LIFO, que utiliza la señal de return como pop, y branch como push, para guardar el valor siguiente al actual valor del Programa Counter. Internamente trabaja en el flanco opuesto al Program Counter para guardar su valor, y evitar inconvenientes en la propagación de los datos.

Decode

Esta etapa se encarga de decodificar la instrucción del programa acorde al set de instrucciones y generar las microinstrucciones que se ejecutan sobre la microarquitectura. Además, esta etapa cuenta con una unidad de pre decodificación para detección de instrucciones de salto condicional, incondicional y saltos o retornos por subrutina.

Operand

Esta etapa se encarga de preparar lo necesario para realizar la ejecución de la instrucción en la siguiente etapa, entre sus capacidades y tareas se encuentra,

- Buscar los operandos en los registros o constante
- Comunicar por el bus de datos y direcciones una operación de escritura o lectura a la memoria
- Comunicar una operación de impresión sobre píxeles al módulo VGA

Execute

Esta etapa se encarga de ejecutar las operaciones necesarias para resolver la instrucción, y se compone de diferentes unidades de ejecución para dar soporte a las diferentes operaciones de la arquitectura. Las unidades de ejecución actualmente disponibles son,

- ALU/Shifter (Unidad Lógica Aritmética)
- Load/Store (Memoria)
- Graphics (Controlador VGA)

ALU

La unidad lógica aritmética es un circuito combinacional que posee dos buses de entrada sobre los cuales realiza una operación acorde a la señal de control recibida, y el resultado se obtiene en su bus de salida. La unidad lógica aritmética trabaja con buses de ancho de 32 bits.

Por otro lado, posee salidas para indicar si en la operación se produjo carry o si en la operación se produjo overflow. La siguiente tabla de verdad describe el comportamiento de un full adder, un sumador binario con carry in y carry out, a modo de referencia para el diseño lógico interno de la ALU. De esto se deduce que,

$$Cout = \bar{R} \cdot (A + B) + B \cdot A$$

A	B	Carry In	Result	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ALUC	Operación
0000	$R = A$
0001	$R = B$
0010	$R = \bar{A}$
0011	$R = \bar{B}$
0100	$R = A + B$

0101	$R = A + B + Cin$
0110	$R = A\ OR\ B$
0111	$R = A\ \&\ B$
1000	$R = 0$
1001	$R = 1$
1010	$R = 0xFFFFFFFF$
1011	$Cout = 0$
1100	$Cout = 1$
1101	$Reserved$
1110	$Reserved$
1111	$Reserved$

Nombre	Tipo	Ancho	Descripción
ina	Entrada	32 bits	*Bus de entrada A*
inb	Entrada	32 bits	*Bus de entrada B*
aluc	Entrada	4 bits	*Señal de control para especificar operación*
cin	Entrada	1 bit	*Entrada de carry*
out	Salida	32 bits	*Bus de salida R*
cout	Salida	1 bit	*Indicador de si hubo carry*
v	Salida	1 bit	*Indicador de si hubo overflow*

Shifter

El shifter es un circuito lógico combinacional que permite aplicar operaciones de desplazamiento sobre su bus de entrada. Su resultado se obtiene sobre un bus de salida de 32 bits y posee una señal de control para determinar qué tipo de operación de desplazamiento aplicar.

SH	Operación
000	*No shift*
001	*Logical Shift Right (1 Bit)*
010	*Logical Shift Left (1 Bit)*
011	*Logical Shift Left (16 Bits)*

100	*Reserved*
101	*Arithmetic Shift Right (1 Bit)*
110	*Reserved*
111	*Reserved*

Nombre	Tipo	Ancho	Descripción
r	Entrada	32 bits	*Bus de entrada R*
sh	Entrada	3 bits	*Señal de control para especificar operación a realizar*
c	Salida	32 bits	*Bus de salida C*
cout	Salida	1 bit	*Indicador de si hubo carry*
v	Salida	1 bit	*Indicador de si hubo overflow*

Load/Store

Esta unidad de ejecución provee funciones de lectura y escritura sobre una memoria de datos RAM integrada junto con el procesador. La memoria se comunica con el procesador a través de los buses de dirección de memoria y datos, y las líneas de control para lectura o escritura. Por otro lado, se asume que la memoria opera a frecuencias inferiores que el procesador.

- El procesador se comunica con la memoria a través de los buses durante la etapa de operandos
- La memoria responde el resultado de la petición a través de los buses durante la etapa de ejecución

VGA: Video Graphics Array

El subsistema de video VGA del procesador es una unidad de ejecución que posee la arquitectura para generar las señales de video para controlar dispositivos compatibles con la interfaz VGA. El módulo cuenta con memoria interna de 64000 palabras de 3 bits para almacenar la información a transmitir a la pantalla. Esta información puede ser actualizada utilizando la instrucción **VGP Rk,Rj.** La capacidad del módulo VGA permite controlar una resolución de 320 píxeles de ancho y 200 píxeles de alto, con 8 posibles colores (3 bits RGB).

Nombre	Tipo	Ancho	Significado
pixel_address	Entrada	16 bits	*Dirección del pixel a escribir.* Se toma al píxel superior izquierdo de la pantalla como la dirección 0 y se avanza de izquierda a derecha y de arriba a abajo.
pixel_rgb	Entrada	3 bits	*Color RGB del pixel a escribir.* Cada bit representa un estado (encendido o apagado) de cada componente R-G-B.

print	Entrada	1 bit	*Señal de activación de la operación de print.*
cpu_clk	Entrada	1 bit	*Señal de clock para sincronizar la escritura a RAM.*
vga_clk	Entrada	1 bit	*Señal de clock para generar señales VGA.* Dado que el módulo funciona para una resolución de 320x200px, para obtener una frecuencia de 60Hz de refresco de la pantalla es necesario un clock de 5MHz.
vsync	Salida	1 bit	*Señal de sincronismo vertical VGA.*
hsync	Salida	1 bit	*Señal de sincronismo horizontal VGA.*
rgb	Salida	3 bits	*Color RGB del píxel actual de la pantalla.* Cada bit representa un estado (encendido o apagado) de cada componente R-G-B.

Retire

Esta etapa se encarga de guardar los resultados de la ejecución en los registros del procesador, para ello el espacio de registros tiene soporte en dos bancos de registros separados funcionalmente como se muestra a continuación,

- Banco de registros de propósito general
- Banco de registros de los puertos I/O (entradas y salidas del procesador)

Banco de registros generales

El banco de registros de propósito general contiene los registros del procesador cuyo uso no está vinculado a ninguna función específica, más que servir a las necesidades genéricas y variadas del programador para almacenar temporalmente información. El contenido de los registros se actualiza en los flancos descendentes del ciclo de retiro.
La siguiente tabla describe la interfaz del bloque funcional para el manejo del banco de registros.

Nombre	**Tipo**	**Ancho**	**Significado**
in	Entrada	32 bits	*Bus de entrada para cargar en registros*
in_sel	Entrada	6 bits	*Selección de registro para cargar la entrada*
outa_sel	Entrada	6 bits	*Selección de registro para cargar en el bus de salida A*
outb_sel	Entrada	6 bits	*Selección de registro para cargar en el bus de salida B*
clk	Entrada	1 bit	*Señal de clock para actualizar registros*
outa	Salida	32 bits	*Bus de salida A*
outb	Salida	32 bits	*Bus de salida B*

Banco de registros de puertos

El banco de registros de puertos contiene los registros del procesador que operan como puertos de entrada y/o salida del mismo. El contenido de los registros se actualiza en los flancos descendentes del ciclo de retiro.
La siguiente tabla describe la interfaz del bloque funcional para el manejo del banco de registros.

Nombre	Tipo	Ancho	Significado
in	Entrada	32 bits	*Bus de entrada para cargar en registros*
in_sel	Entrada	6 bits	*Selección de registro para cargar la entrada*
outa_sel	Entrada	6 bits	*Selección de registro para cargar en el bus de salida A*
outb_sel	Entrada	6 bits	*Selección de registro para cargar en el bus de salida B*
clk	Entrada	1 bit	*Señal de clock para actualizar registros*
outa	Salida	32 bits	*Bus de salida A*
outb	Salida	32 bits	*Bus de salida B*
input_port0	Entrada	32 bits	*Puerto de entrada 0*
input_port1	Entrada	32 bits	*Puerto de entrada 1*
output_port0	Salida	32 bits	*Puerto de salida 0*
output_port1	Salida	32 bits	*Puerto de salida 1*

Análisis de dependencias

Uno de los inconvenientes presentes en arquitecturas de CPU con pipeline, es que la paralelización de instrucciones puede provocar situaciones conflictivas cuando hacen uso de los mismos recursos del procesador. Por eso, es importante hacer un análisis para identificar cuáles recursos y en qué situaciones pueden ser utilizados de forma simultánea.

Dependencia de registros

1. Una instrucción en la etapa de operandos necesita procurarse el contenido de un registro que va a ser modificado por otra instrucción en la etapa de ejecución.
2. Una instrucción en la etapa de operandos necesita procurarse el contenido de un registro que está siendo actualizado en la etapa de retiro.

Dependencia del PSR

1. El PSR sólo es accedido por aquellas instrucciones que son saltos condicionales. Los saltos condicionales se evalúan sobre la última operación realizada, por ello, cuando un salto condicional ingresa al procesador, la instrucción que le precede aún no fue resuelta y se desconoce la decisión del salto. Esto es un aspecto crítico dado que la arquitectura resuelve los saltos durante la etapa de decodificación.

Dependencia del bus de datos

1. La comunicación con los periféricos y la memoria a través del bus de datos se realiza con una topología master/slave, en la cual el procesador realiza una petición a través del bus de datos y de direcciones durante la etapa de operandos, y los dispositivos responden a través del bus de datos durante la etapa de ejecución. Una operación de escritura en etapa de operandos puede colisionar con una operación de lectura en etapa de ejecución, ya que tanto el procesador como el dispositivo buscan escribir sobre el bus.

Análisis de saltos

El procesador cuenta con instrucciones que son saltos condicionales, incondicionales, y saltos o retornos de subrutinas. Estos saltos pueden ser relativos a la posición actual del programa o pueden ser absolutos. La IFU es una unidad que se encarga de, por hardware, actualizar el estado actual del registro interno PC o Program Counter para buscar esa instrucción durante la etapa de fetch, para ello brinda funciones de incremento, saltos relativos, absolutos y saltos o retornos a subrutina. Por último, para minimizar el impacto de la presencia de saltos sin utilizar predicción para los casos condicionales, se busca resolver el salto cuando es detectado en la etapa de decodificación.

1. Los saltos se resuelven en la etapa de decodificación enviándole una señal correspondiente a la IFU, luego de lo cual existe un ciclo de máquina perdido para buscar la dirección correcta en memoria.
2. Los saltos condicionales no se pueden resolver hasta tanto no se haya ejecutado la operación que establece el resultado para la decisión.

Unidades de control

Dependencia de registros

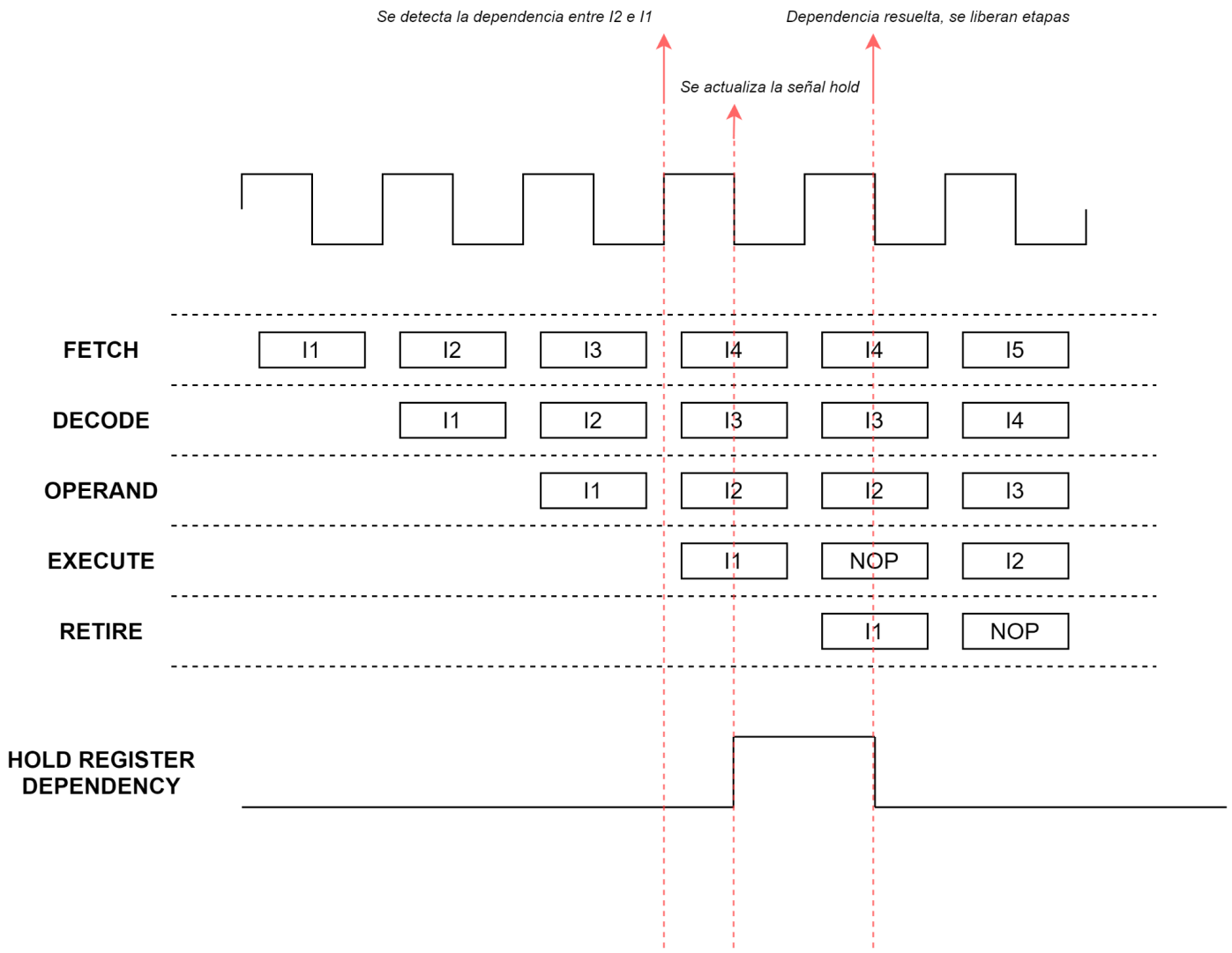
Cuando una instrucción en la etapa de operandos requiere de un registro que está siendo modificado en etapa de retiro, no es necesario tomar recaudos ya que se considera que el tiempo de demora en actualizar el registro no es un factor crítico o limitante en el tiempo del ciclo máquina. Entonces, la actualización llegará a propagarse hasta la etapa de operandos antes de la finalización del ciclo, ya que los tiempos que limitan la frecuencia son por ejecución y no por propagación en los registros.

Por otro lado, cuando una instrucción en la etapa de operandos busca el contenido de un registro que va a ser modificado por una instrucción en etapa de ejecución, es necesario esperar. Para hacer esto, primero se busca detectar la situación comparando qué registros A y B se buscan en etapa de operandos, con el registro C de la microinstrucción en ejecución. Para facilitar la tarea de las unidades de control, la microinstrucción cuenta con bits para indicar que se hacen uso de esos buses y que la comparación debe hacerse.

En el ciclo máquina que se detecta la dependencia, debe garantizarse luego que el siguiente flanco ascendente no llegue a las etapas de fetch, decode y operand hasta que la operación llegue a retire. Además, para avanzar el pipeline, es necesario inyectar una microinstrucción nula en la etapa de ejecución. La señal que activa el mecanismo de dependencia se actualiza en los flancos descendentes.

En la siguiente figura, se muestra un diagrama temporal con la progresión del pipeline en una situación de dependencia de registros entre las etapas de operando y ejecución.

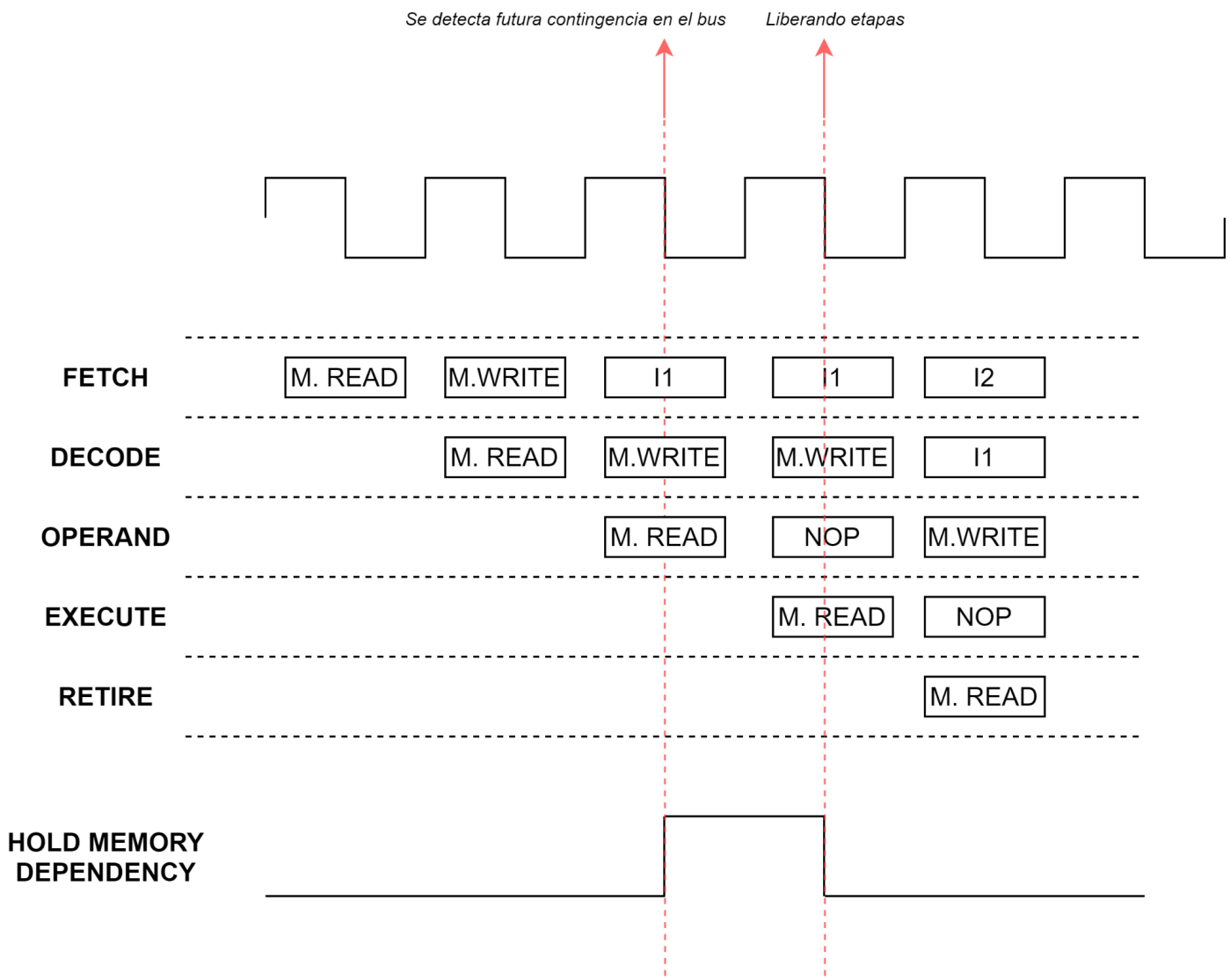
- La dependencia existe entre las instrucciones **I1** e **I2**



Dependencia del bus de datos

Una instrucción de escritura a memoria hace uso del bus de datos durante su estadío en la etapa de operandos, mientras que en una instrucción de lectura a memoria la misma responde a través del bus de datos en la etapa de ejecución. Para evitar la colisión, se debe detectar cuando la escritura se encuentra en etapa de decodificación y la lectura en etapa de operandos. La señal que activa el mecanismo de dependencia se actualiza en los flancos descendentes.

En la siguiente figura, se muestra un diagrama temporal de la progresión del pipeline en una situación de dependencias o contingencias del bus de datos.



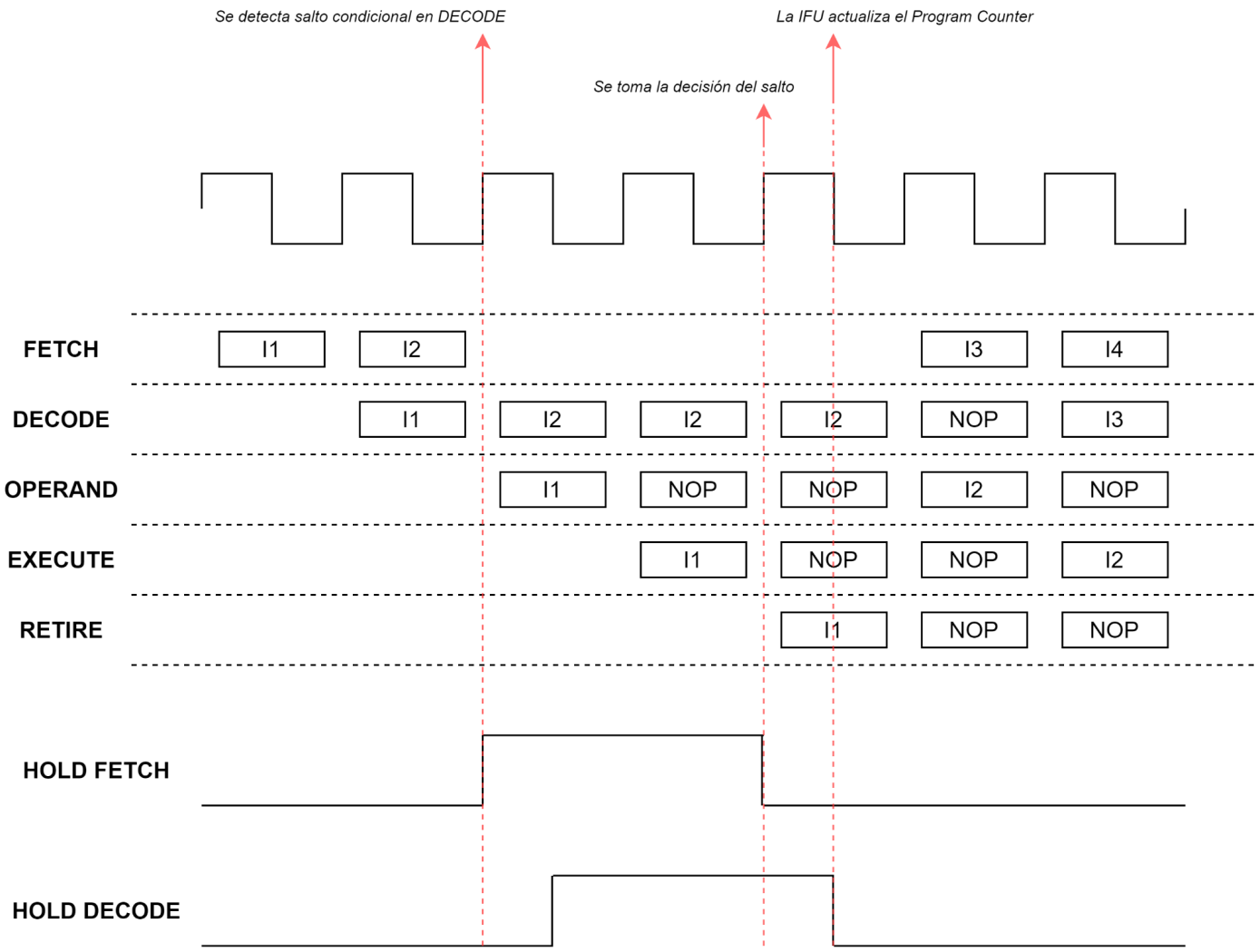
Manejo de saltos

Los saltos incondicionales absolutos o los saltos relativos a subrutina, se resuelven en la etapa de decodificación detectando su presencia y enviando una señal correspondiente a la IFU. Luego, existe un ciclo máquina durante el cual debe buscarse una instrucción nueva, para lo cual se inyecta una instrucción NOP en la etapa de decodificación.

Los saltos condicionales absolutos se resuelven en dos partes. En primer lugar, al momento de detectar que un salto condicional entra a la etapa de decodificación, se inician contadores que durante dos ciclos máquina activan un mecanismo que inyecta microinstrucciones NOP en la etapa de operandos y detiene la etapa de fetch. Luego de ese tiempo, la segunda parte para resolver un salto condicional es exactamente igual que para los incondicionales, ya que se conoce si debe tomarse o no el salto.

En la siguiente figura, se ilustra un diagrama de tiempos con la progresión del pipeline en un salto condicional, mostrando los eventos significativos y las señales de control correspondientes.

- La instrucción **I1** es una operación aritmética
- La instrucción **I2** es un salto condicional
- La instrucción **I3** es una instrucción arbitraria



Simulaciones

En esta sección se incluyen los resultados de la simulación de algunos programas de prueba, con el objetivo de mostrar puntos de interés sobre el comportamiento del procesador. Se desarrolló un ensamblador con el objetivo de facilitar la creación de las pruebas y el uso del procesador mismo, este permite obtener el código objeto en lenguaje máquina a partir del lenguaje ensamblador que emplea la arquitectura EV21G1.

Simulación #1

En esta simulación se busca realizar una operación entre dos registros, cargar el resultado en un tercer registro. La operación presentará carry, entonces se utiliza un salto condicional para generar un ciclo infinito. Esto permite visualizar,

- Operación entre registros
- Dependencia entre registros
- Salto condicional
- Múltiples dependencias en simultáneo

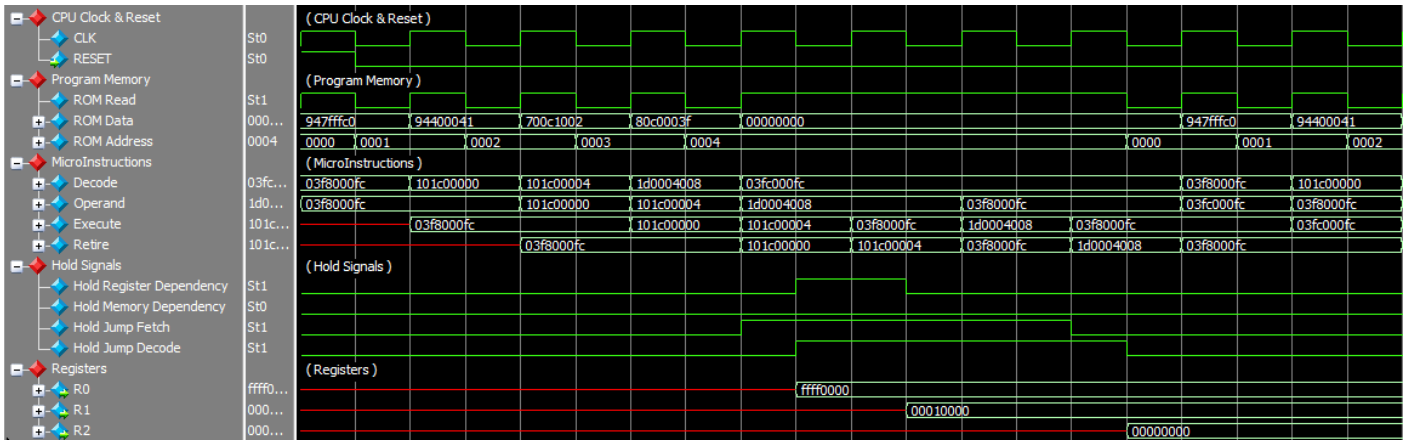
Código fuente

```
MMK 0,FFFF  // R0 = 0000FFFF << 16
MMK 1,0001  // R1 = 00000001 << 16
ADD 2,0,1   // R2 = R1 + R2
JCY 0       // If result is zero, jump to position 0
```

Código objeto

```
10010100011111111111111111000000
10010100010000000000000001000001
01110000000011000001000000000010
10000000110000000000000000111111
```

Resultado



Simulación #2

En esta simulación se busca guardar en una posición de memoria el resultado de una suma entre registros. Así, permite visualizar la operación entre dos registros, y la operación de escritura en memoria.

Código fuente

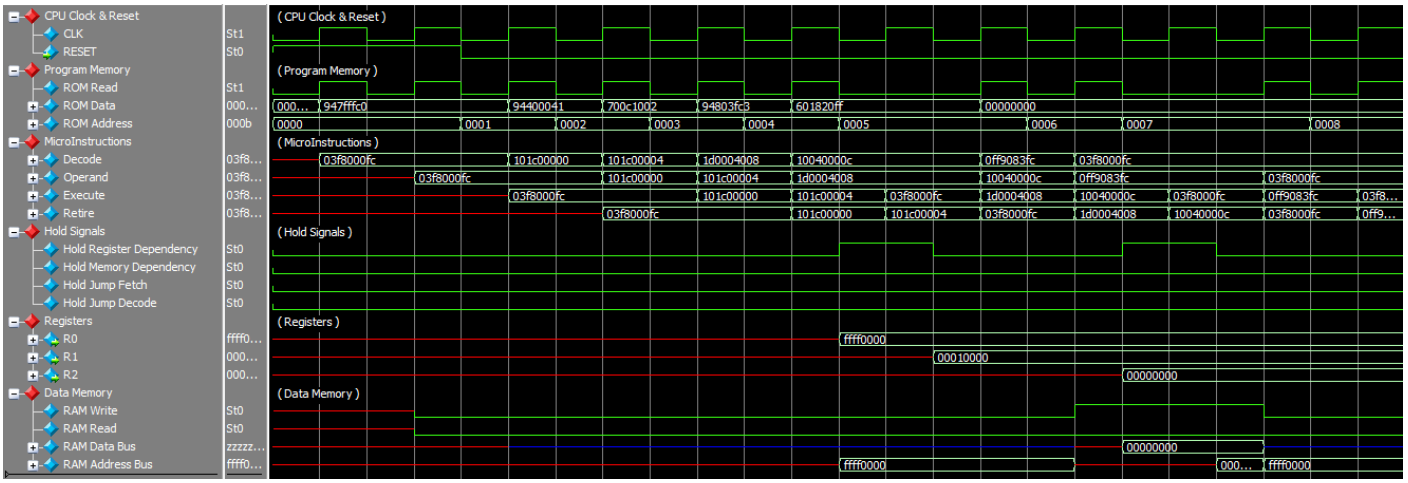
```
MMK 0,FFFF  // R0 = 0000FFFF << 16
MMK 1,0001  // R1 = 00000001 << 16
ADD 2,0,1   // R2 = R1 + R2
MLK 3,00FF  // R3 = 000000FF
STR 2,3     // M(R3) = R2
```

Código objeto

```
10010100011111111111111111000000
10010100010000000000000001000001
```

```
01110000000011000001000000000010
10010100100000000011111111000011
01100000000110000010000011111111
```

Resultado



Simulación #3

En esta simulación se reiteran las operaciones realizadas en la simulación anterior, pero se encadena una serie de operaciones a memoria con el objetivo de exponer una situación que active los mecanismos de dependencia por el bus de datos de memoria.

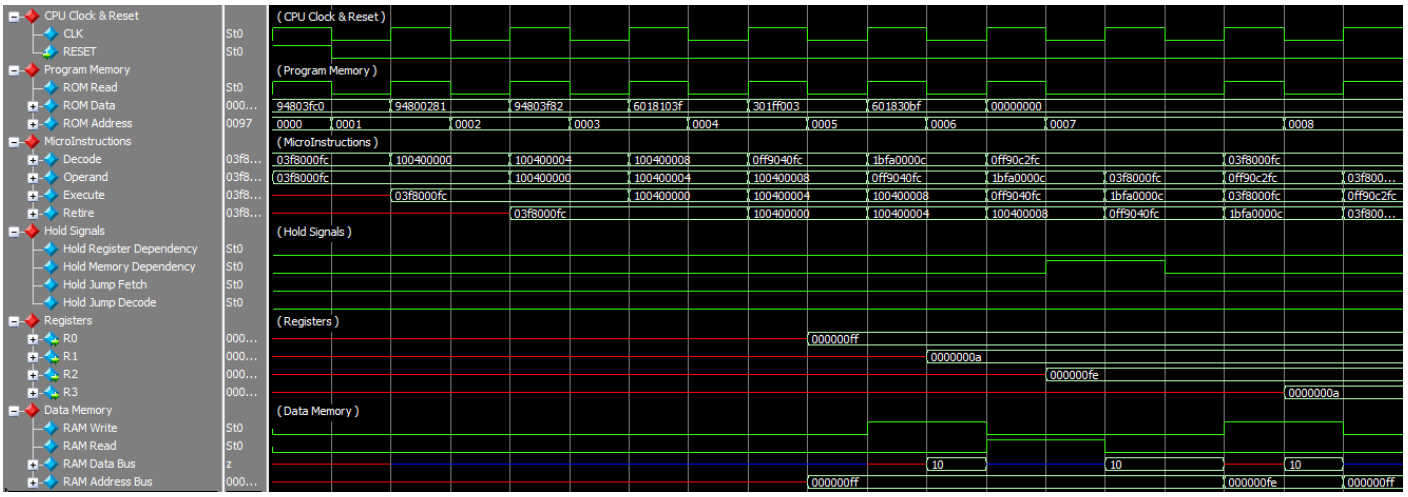
Código fuente

```
MLK 0,00FF  // R0 = 000000FF
MLK 1,000A  // R1 = 0000000A
MLK 2,00FE  // R2 = 000000FE
STR 1,0     // M(R0) = R1
LDR 3,0     // R3 = M(R0)
STR 3,2     // M(R2) = R3
```

Código objeto

```
10010100100000000011111111000000
10010100100000000000001010000001
10010100100000000011111110000010
01100000000110000001000000111111
00110000000111111111000000000011
01100000000110000011000010111111
```

Resultado


CPU Clock & Reset
CLK
RESET
Program Memory
ROM Read
ROM Data
ROM Address
MicroInstructions
Decode
Operand
Execute
Retire
Hold Signals
Hold Register Dependency
Hold Memory Dependency
Hold Jump Fetch
Hold Jump Decode
Registers
R0
R1
R2
R3
Data Memory
RAM Write
RAM Read
RAM Data Bus
RAM Address Bus
(CPU Clock & Reset)
(Program Memory)
(MicroInstructions)
(Hold Signals)
(Registers)
(Data Memory)