ITBA

Instituto Tecnológico de Buenos Aires

22.15 - Electrónica V

28 de Junio de 2022

Trabajo Práctico $N°2$ - EV22

Tobías Demeco
Alexander Moldovan
Gianfranco Muscariello
Matías Trípoli
Milagros Moutin

Profesores:

Ing. Andrés Carlos Rodriguez
Ing. Pablo Enrique Wundes

Índice

1. Introducción

En el presente informe se explica el detalle nuestra implementación procesador EV22. Si bien se tomó como punto de partida la arquitectura propuesta por la cátedra se le realizaron cambios que serán explicados en las secciones siguiente. Finalmente se comentaran aspectos funcionales y conclusiones a las que se llegó una vez implementado el procesador.

2. Set de instrucciones

Partiendo del set de instrucciones provisto por la cátedra, se diseñó otro que contiene las mismas instrucciones pero utilizando 20 bits por instrucción y de tipo expanding opcodes. En mismo se observa en la Tabla 1.

Código Binario EV22	Nemónico	Instrucción	Significado
0001 ssss ssss ssss ssss	BSR -S	Unconditional Branch (Relative) to Subroutine S	Save PC = PC +S
0010 yyyy yyyy yyyy yyyy	MOM Y, W	Move Working Register to Memory	M(Y) = W
0011 yyyy yyyy yyyy yyyy	MOM W,Y	Move Memory to Working Register	W = M(Y)
0100 kkkk kkkk kkkk kkkk	MOK W,K	Move Constant to Working Register	W = K
0101 kkkk kkkk kkkk kkkk	ANK W,K	ADD with Carry Constant with Working Register	W= W + K + CY
0110 kkkk kkkk kkkk kkkk	ORK W,K	OR Constant with Working Register	W= W OR K
0111 kkkk kkkk kkkk kkkk	ADK W,K	ADD with Carry Constant with Working Register	W= W + K + CY
1000 0000 0000 0000 0000	RET	Return From Subroutine	PC= Latest Stored PC + 1
1100 xxxx xxxx xxxx xxxx	JMP X	Unconditional Jump	PC = X
1101 xxxx xxxx xxxx xxxx	JZE X	Jump if Working Register is Zero	IF W = 0 THEN PC=X
1110 xxxx xxxx xxxx xxxx	JNE X	Jump if Working Register is Negative	F W_{15}=0 THEN PC=X
1111 xxxx xxxx xxxx xxxx	JCY X	Jump if Carry	IF CY THEN PC=X
0000 1000 iiiiii jjjjjj	MOV R_i, R_j	Move Register R_j to R_i R_i, R_j: 0 a 60	$R_i = R_j$
0000 1001 iiiiii jjjjjj	ADW R_i, R_j	Add with Carry R_j with Working Reg. to R_i	R_i = W + R_j + CY
0000 0000 0001 xx iiiiii	MOV R_i,W	Move Working Register to R_i	R_i = W
0000 0000 0100 xx jjjjjj	MOV W,R_j	Move R_j to Working Register	W= R_j
0000 0000 0101 xx jjjjjj	ANR W, R_j	AND R_j with Working Register	W= W AND R_J
0000 0000 0110 xx jjjjjj	ORR W,R_j	OR R_j with Working Register	W= W OR R_j
0000 0000 0111 xx jjjjjj	ADR W,R_j	ADD with Carry R_j with Working Register	W= W + R_j + CY
0000 0000 1000 xx jjjjjj	CPL R_j	Complement R_j Register	R_j =$\sim$ R_j
0000 0000 1010 xx jjjjjj	SHR R_j	Logical Shift Left for R_j	$R_j \ll 1$ (Logical)
0000 0000 1011 xx jjjjjj	SHAR R_j	Arithmetic Shift Right for R_j	$R_j \ll 1$ (Arithmetic)
0000 0000 1100 xx jjjjjj	INC R_j	Increment R_j Register	$R_j = R_j + 1$
0000 0000 1101 xx jjjjjj	DEC R_j	Decrement R_j Register	$R_j = R_j$ - 1
0000 0000 0000 0010 xxxx	CLR CY	Clear Carry	CY= 0
0000 0000 0000 0011 xxxx	SET CY	Set Carry	CY= 1
0000 0000 0000 1000 xxxx	CPL W	Complement Working Register	W =$\sim$ W
0000 0000 0000 1001 xxxx	SHL W	Logical Shift Left for W	$1 \gg W$ (Logical)
0000 0000 0000 1010 xxxx	SHR W	Logical Shift Right for W	$W \ll 1$ (Arithmetic)
0000 0000 0000 1011 xxxx	SHAR W	Arithmetic Shift Right for W	$W \ll 1$ (Arithmetic)
0000 0000 0000 1100 xxxx	INC W	Increment W	W = W + 1
0000 0000 0000 1101 xxxx	DEC W	Decrement W	W = W - 1
0000 0000 0000 1111 xxxx	NOP	No Action	

Tabla 1: Set de Instrucciones.

3. Esquema Modular

En la Figura se muestra el esquema con los distintos módulos del procesador. A continuación se explica la funcionalidad de cada uno de los bloques.

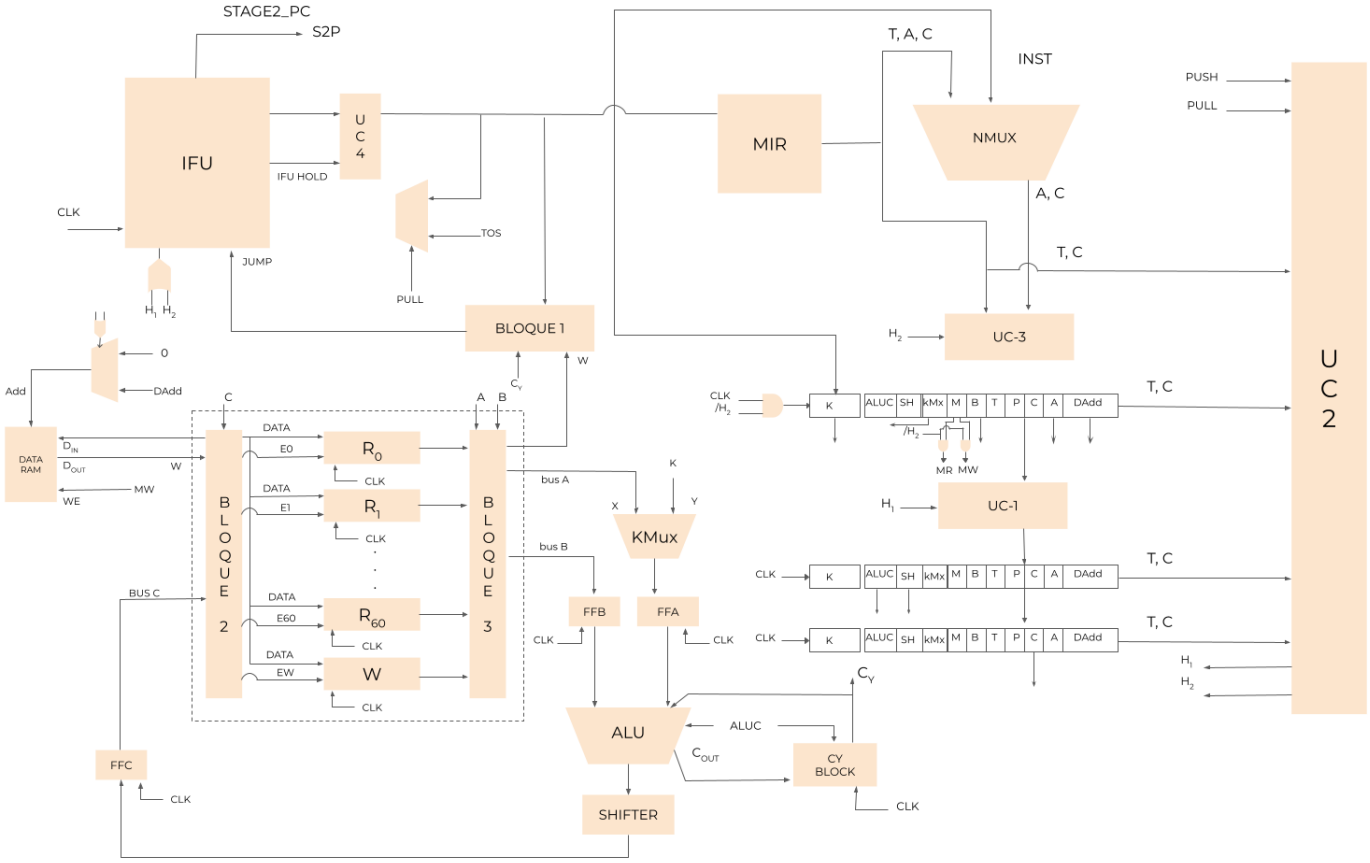

Figura 1: Diagrama en Bloques.

3.1. IFU

La IFU se encuentra en la etapa de Fetch del microprocesador. Su principal función es obtener de forma cómoda las instrucciones a realizar y ofrecer un manejo seguro de los saltos del programa. Cuenta con dos bloques fundamentales, el registro Program Counter y un incrementador, el cual fue creado de forma completamente combinacional. En la figura 2 se puede observar el diagrama en bloques de la IFU.

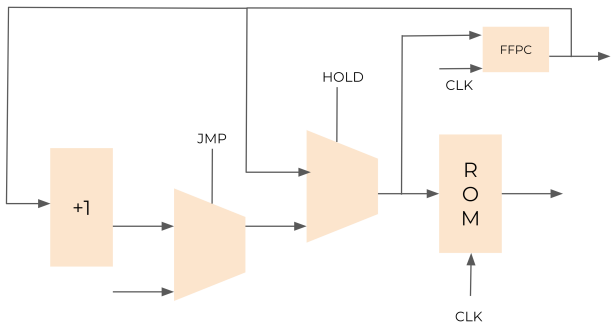

Figura 2: Diagrama de bloques de la IFU

Es posible dividir el bloque en tres casos particulares. El primero caso seria cuando el hold y el jmp se encuentren desactivados, es decir cunado no se requiere realizar ningún salto ni mantener el valor del PC. En este momento la IFU libera instrucciones de forma secuencial. El segundo caso es cuando se quiere realizar un salto, en esta instancia se activa jmp modificando la salida del mux y generando que la dirección que haya entrado a la IFu sea ubicada en el PC para recoger la instrucción consiguiente. Finalmente si el hold se encuentra activado se deberá mantener constante tanto el ultimo PC como su instrucción. En este caso la variable de jmp no tendrá ningún efecto en el resultado final.

En resumen se cambiara la salida de los muxes en función de la dirección de memoria que se quiera acceder.

3.2. MIR

Para la ROM de microinstrucción implementamos lógica combinacional y una decodificación que se podría definir como especulativa, ya que la decodificación de la instrucción se hace paralelamente en cuatro bloques distintos, cada uno decodificándola como si el opcode estuviese en el primer, segundo, tercer o cuarto nybble respectivamente, y luego mediante una lógica de selección utilizando MUXs y ANDs se elige cual de las cuatro microinstrucciones formadas es la correcta y es esta la que pasa a la siguiente etapa. Realizar la decodificación de esta manera fue posible gracias a la implementación de expanding opcodes, que nos otorgo cierta ortogonalidad entre microinstrucciones.

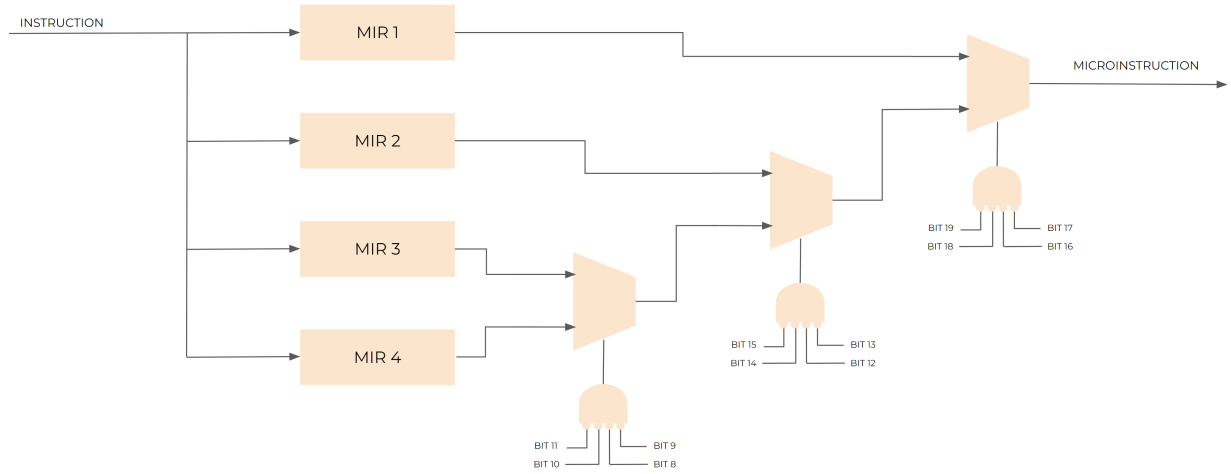

Figura 3: Diagrama en bloques del decoder .

Cabe mencionar que la lógica de cada bloque de decodificación usado se optimizó creando conexiones directas para parámetros que, o se mantenían iguales para todas las microinstrucciones que podían provenir de ese bloque, o no eran utilizados por las mismas.

3.3. StackBlock

Se utiliza un Stack para guardar las direcciones de retorno. Entonces se incorporó una unidad, cuyo diagrama en bloques se muestra en la Figura 4, que recibe el PC al activarse la señal *push*, y publica activa la dirección de retorno (PC+1) en la señal de salida (TOS). Para retirar el valor del stack se activa la señal de pull. En la Figura se muestra un diagrama en bloques funcional del StackBlock.

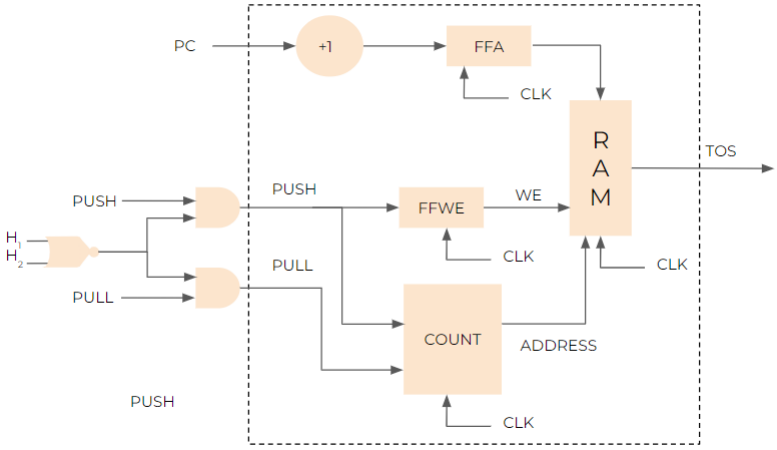

Figura 4: Stack Block.

Con esta implementación se tiene un delay de 2 tiempos de clock desde que se publica el PC en la linea de entrada y se activa la linea de push, hasta que el dato está visible en el TOS. De la misma forma, quitar el último valor almacenado en el stack tiene un delay de 2 tiempos de clock.

3.4. Bloques de Control - UC1, UC2, UC3, UC4

Los Bloques UC controlan el funcionamiento del pipeline, asegurándose que todas las instrucciones se ejecuten de forma correcta. La lógica principal se encuentra en el Bloque UC2, el cual recibe el estado de las instrucciones de las etapas de decodificación, operandos, ejecución y guardado. En función de sus entradas, genera las señales de Hold H1 y H2. La señal H1 detiene las instrucciones en la etapa de decodificación, mientras que la señal H2 detiene también a la instrucción de la etapa de operandos.

Los Bloques UC-2 y UC-3 sirven como puente entre las etapas, permitiendo el flujo de las microinstrucciones únicamente si la señal de Hold no está activa. De lo contrario, detienen a la instrucción que tienen detrás, transmitiendo a las próximas etapas la microinstrucción correspondiente a NOP. De esta forma, no se detienen las instrucciones más avanzadas del pipeline.

El Bloque UC4 tienen un funcionamiento similar: recibe la señal de Hold generada por la IFU, y si se encuentra activa, no permite el paso de la instrucción a la etapa de decodificación, transmitiendo en su lugar la instrucción NOP. De esta manera, la IFU puede activar la señal de Hold mientras se realiza la búsqueda de la próxima instrucción, en caso de que no esté inmediatamente disponible, sin afectar el funcionamiento del pipeline ni ejecutar instrucciones inválidas.

3.5. Register Bank

En la Figura 5 se observa el diagrama en bloques del Register Bank. Este contiene 62 registros de propósito general (numerados del 0 al 60) y un *Working Register*.

El Bloque 2 es una unidad funcional puramente combinacional que controla que información se publica en que registro. Cada registro recibe el clock y una señal de *write enable*, que activa el Bloque 2. La microinstrucción usa 6 bits para representar en cual de los 61 registros se va a publicar contenido. Estos bits los recibe el Bloque 2 a través de la linea de control C. Por lo tanto, no se guarda información en más de un registro por ciclo de clock. Los registros de propósito general reciben información proveniente del bus C mientras que el *working register* puede adicionalmente recibir información proveniente de memoria. En particular, si la linea de control C publica el valor $2'h2D$, en el *working register* se guarda contenido de memoria, mientras que si la linea de control C publica el valor $2'h2E$ en el *working register* se guarda contenido del bus C. Si no se desea que los registros guarden la información de los buses, se publica el valor $2'h2F$ en la linea de control C. Al recibir un flanco ascendente de clock, el registro cuyo *write enable* esté activo guarda el contenido correspondiente.

El Bloque 3, también puramente combinacional, recibe los valores de los registros y en base al valor de la lineas de control A y B publica o no los contenidos de los registros en los buses A y B respectivamente. Si no se deseada publicar contenido en un bus, la linea de control correspondiente deben publicar los valores $2'h2E$ y $2'h2F$.

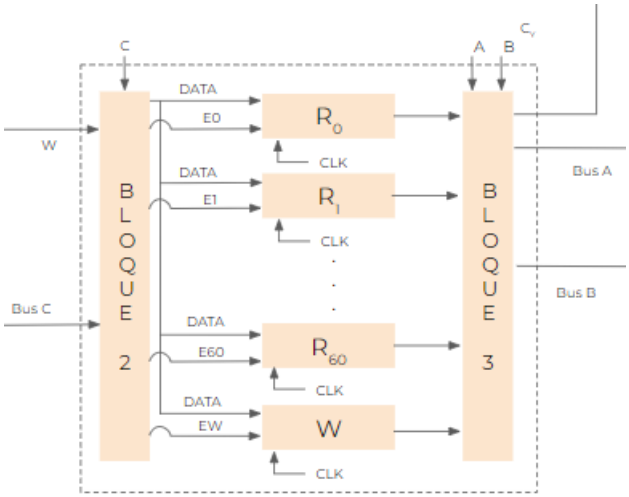

Figura 5: Register Bank.

3.6. Simulación

Para probar el funcionamiento del microprocesador, se realizaron códigos de prueba, los cuales consistieron en probar cada una de las instrucciones disponibles, el resultado de las mismas, y el comportamiento del pipeline cuando se activan las señales de Hold. El siguiente código es uno de ellos:

```
    ORG $0000
*               PC      Inst
    NOP         0000    000F0
    MOV w,#1    0001    40001
    MOV @0,w    0002    00100
    INC @0      0003    00C00
    MOV w,@0    0004    00400
    MOV @1,w    0005    00101
    INC @1      0006    00C01
    MOV w,@1    0007    00401

fin JMP fin     0008    C0008
```

Figura 6: Código de prueba

Los bits presentes en el campo T de la microinstrucción identifican si la instrucción correspondiente realiza una lectura o escritura de un registro R (identificado en el código fuente como 0 o 1) o el Working Register w. Como puede observarse en la Figura 6, cada instrucción de escritura es seguida por una de lectura del mismo tipo de registro. Debido a esto, el comportamiento esperado del pipeline es detenerse durante 2 ciclos de máquina entre cada instrucción, para asegurarse que los resultados se encuentren disponibles al momento de ejecutar la instrucción siguiente. Dicho comportamiento se observa en la Figura 7, el cual ilustra la simulación del pipeline durante la ejecución del código anterior. Las variables *aux_t* corresponden a los campos T de cada instrucción presente en el pipeline. Aquellas terminadas en "NOP" denotan a la microinstrucción transmitida por los bloques UC correspondientes; por ello, mientras la señal de Hold se encuentra activa, transmiten el valor $00, equivalente a la instrucción NOP.

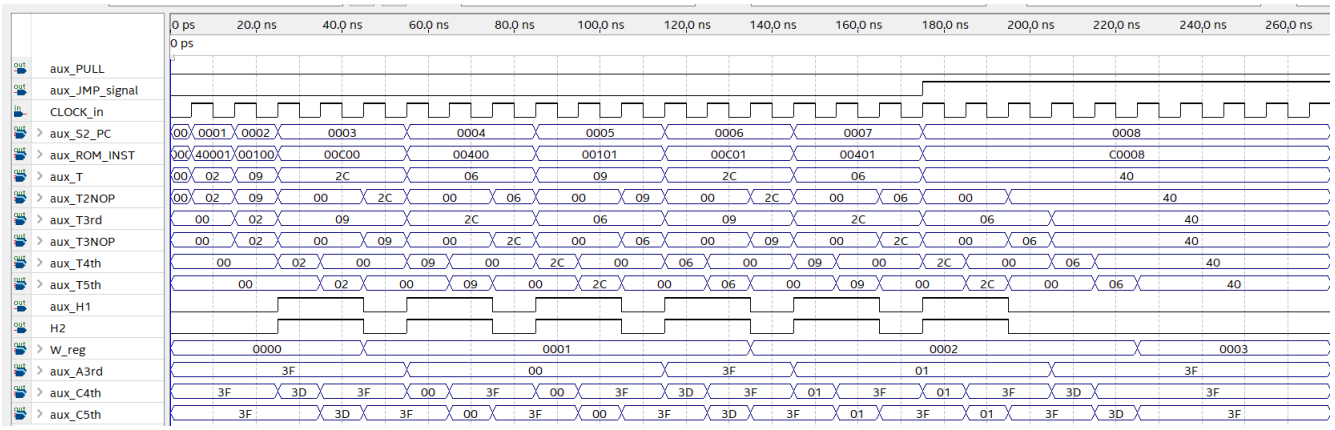

Figura 7: Simulación de un programa