CHAPTER 1

Unidad de Proceso Genérica y Unidad de Control Específica

Juan J. Navarro
Toni Juan

Unidad de proceso general

A la vista de los diseños de procesadores de propósito específico que hemos realizado podemos decir que una UP general debe tener un conjunto de registros y una o varias unidades funcionales (combinacionales que realizan cálculos variados) cuya interconexión se pueda modificar dinámicamente en cada ciclo. La Figura 1 muestra la UP genérica que utilizaremos. Está formada por un banco de 8 registros con dos buses de lectura y uno de escritura conectados a una ALU. Además se ha añadido una entrada y una salida de datos al/ desde el banco de registros. La estructura de este diseño puede considerarse una generalización de la UP específica para sumar 4 números que vimos en clase, cambiando el registro por un banco de registros y el sumador por una ALU.

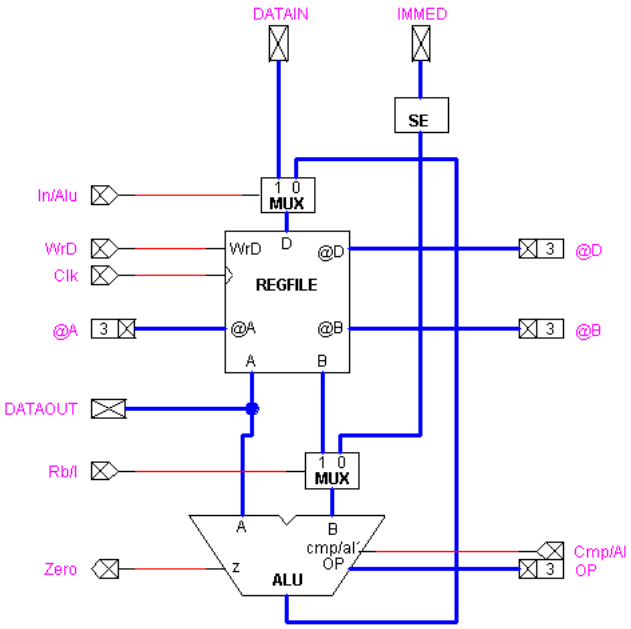


Figura 1. Unidad de Proceso genérica

En la Figura 2 se encuentra el detalle interno del banco de registros y en la Figura 3 la construcción de la ALU en dos grandes bloques: el CMP y la AL cuyos detalles de implementación se muestran en la Figura 4 y en la Figura 5 respectivamente. El bloque SE (Sign Extension) efectúa la extensión del bit de signo. Al bloque SE le entra un número codificado en complemento a 2 con 6 bits y obtiene a su salida ese mismo número codificado con 16 bits. El bloque z de la ALU detecta si los 16 bits de la salida de la ALU son todos ceros.

En la Figura 6 se muestra la tabla de codificación de las funciones de la ALU y los bits que forman la palabra de control de nuestra UP genérica.

En esta Unidad de Proceso genérica, en cada ciclo, se puede llevar a cabo una de las siguientes funciones:

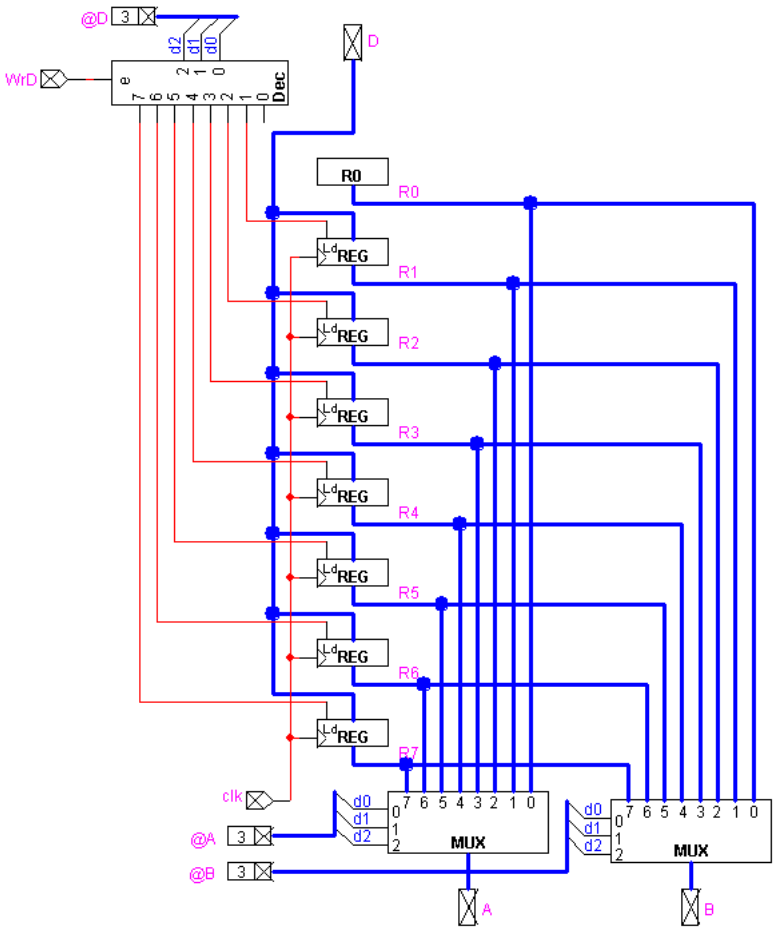


Figura 2. Detalle de la implementación interna del banco de registros

1. Leer dos registros, operarlos (con alguna de las 12 funciones de 2 operandos que puede realizar la ALU: 2 aritméticas, 2 lógicas y 4 de comparación de enteros y 4 de comparación de naturales- y dejar el resultado en otro registro al final del ciclo. Por ejemplo, escribir en el registro 6 la suma del contenido del registro 3 y del 5. Esta operación la denotaremos como:

   ADD R6, R3, R5

   (ADD es un mnemotécnico de la palabra inglesa *addition*, suma. Con Ri nos referimos al registro i del banco de registros).

   Otro ejemplo es: escribir en R3 un 1 si el contenido del registro 5 es mayor o igual que el contenido del registro R1 (interpretados los contenidos de los registros como números naturales codificados en binario con 16 bits) y un 0 en caso contrario. Esta operación la denotamos como:

   CMPGEU R3, R5, R1

   (CMPGEU es el mnemotécnico de *Comparison on Greather or Equal Unsigned*, comparación mayor o igual sin signo)

   En todos estos casos, el segundo operando puede ser, en vez del contenido de un registro, el número codificado en complemento a 2 con 6 bits que entra por IMMED. Estos 6 bits

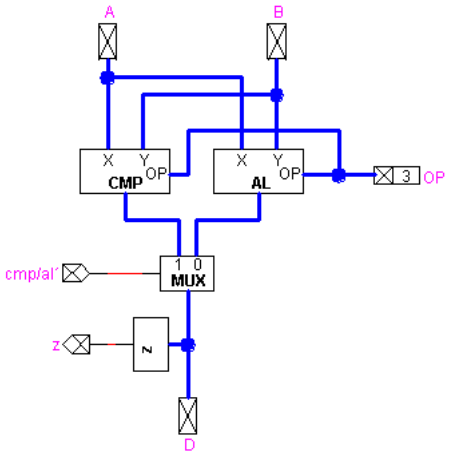


Figura 3. Diseño interno de la ALU

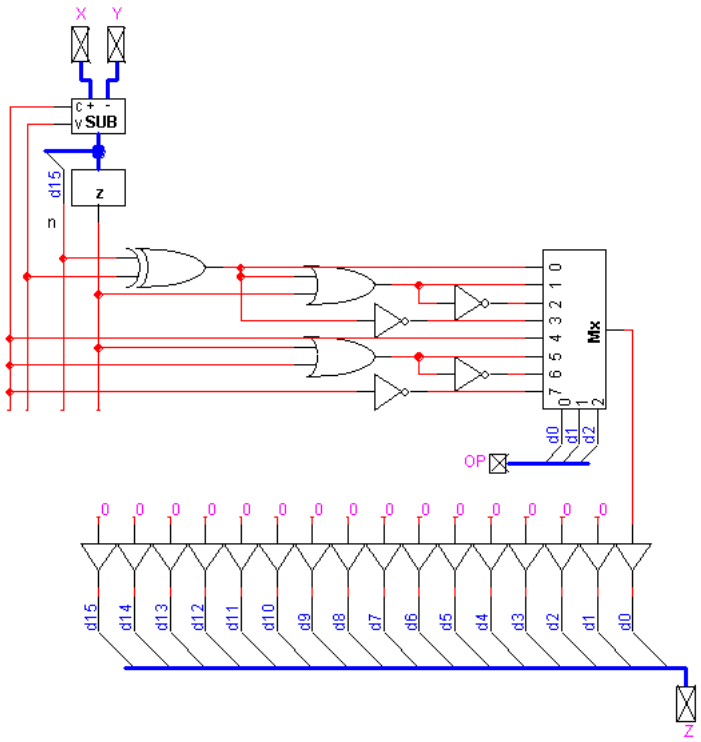


Figura 4. Detalle de la implementación interna del subsistema CMP

los proporcionará la Unidad de Control. Un ejemplo de esta operación a nivel de mnemotécnico es:

ADDI R7, R1, # -1

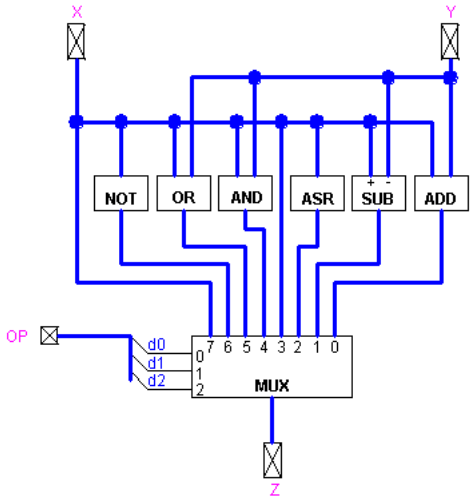


Figura 5. Detalle de la implementación interna del subsistema AL

Funcionalidad de la ALU

OP				
d_2	d_1	d_0	cmp/al = 0	cmp/al = 1
0	0	0	ADD (A, B)	CMPLT (A, B)
0	0	1	SUB (A, B)	CMPLE (A, B)
0	1	0	ASR (A)	CMPGT (A, B)
0	1	1	A	CMPGE (A, B)
1	0	0	AND (A, B)	CMPLTU (A, B)
1	0	1	OR (A, B)	CMPLEU (A, B)
1	1	0	NOT (A)	CMPGTU (A, B)
1	1	1	A	CMPGEU (A, B)

Palabra de Control

In/Alu	WrD	@D			@A			Cmp/Al	OP			Rb/I	@B			Immed					
		d_2	d_1	d_0	d_2	d_1	d_0		d_2	d_1	d_0		d_2	d_1	d_0	d_5	d_4	d_3	d_2	d_1	d_0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 6. Funcionalidad de la ALU y palabra de control de la Unidad de proceso genérica

2. Leer un registro por el bus A, operarlo -con alguna de las 3 funciones de 1 operando que puede realizar la ALU: ASR, NOT, A- y dejar el resultado en otro registro al final del ciclo. Por ejemplo, escribir en R4 el desplazamiento aritmético del contenido de R2. Esta operación la denotaremos como:

   ASR R4, R2

(ASR es un mnemotécnico de *arithmetic shift on rigth*, desplazamiento aritmético a la derecha.)

3. Escribir un registro, al final del ciclo, con la información presente durante este ciclo en la entrada DATAIN. Por ejemplo, cargar el registro 2 con el valor de la entrada de datos. Esto lo denotamos como:

   IN R2

   (IN es un mnemotécnico de *input*, entrada.)

4. Que el contenido de un registro del banco este presente en la salida DATAOUT durante este ciclo. Por ejemplo, sacar el contenido del registro 4. Esto lo denotamos como:

   OUT R4

   (OUT es un mnemotécnico de *output*, salida.)

Para que la UP haga una de estas funciones en un ciclo determinado, hay que proporcionar a la UP con la **palabra de control** adecuada durante ese ciclo. La Figura 7 muestra el valor de los bits de la palabra de control para cada uno de los casos anteriores. Por ejemplo, para hacer ASR R4, R2 se debe:

- indicarle al multiplexor que deje pasar lo que viene de la ALU, codificando un 0 en el campo In/Alu;
- poner la señal WrD a 1 para que al final del ciclo se escriba el contenido del bus D en el registro que indican los bits del campo @D;
- codificar un 4 en el campo @D para que se escriba el registro R4.
- leer por el puerto A del banco de registros el contenido de R2 (ya que la ALU puede desplazar a la izquierda lo que le entra por su entrada A), para ello se codifica un 2 en el campo @A y
- poner a 0 la señal Cmp/Al y poner 010 en el campo OP, bits $d_2d_1d_0$, para que la ALU desplace (aritméticamente: realimentando el bit de signo) la entrada A una posición a la derecha, (ver tabla de funcionalidades de la ALU en la Figura 6).

El campo Rb/I, @B e Immed, pueden codificarse de cualquier forma, pues es indiferente lo que entre por el bus B de la ALU, ya que la ALU no lo usa para nada.

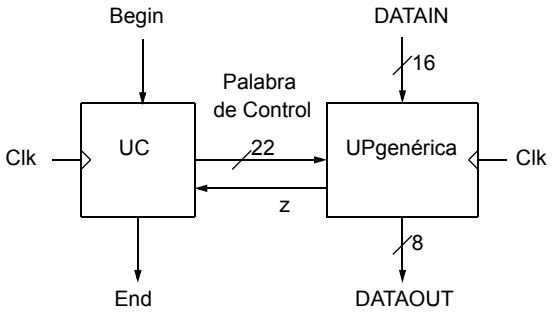
Mnemotécnico	**Palabra de Control**																					
	In/Alu	WrD	@D			@A			Cmp/Al	OP			Rb/I	@B			Immed					
			d_2	d_1	d_0	d_2	d_1	d_0		d_2	d_1	d_0		d_2	d_1	d_0	d_5	d_4	d_3	d_2	d_1	d_0
ADD R6, R3, R5	0	1	1	1	0	0	1	1	0	0	0	0	1	1	0	1	x	x	x	x	x	x
CMPGEU R3, R5, R1	0	1	0	1	1	1	0	1	1	1	1	1	1	0	0	1	x	x	x	x	x	x
ADDI R7, R1, # -1	0	1	1	1	1	0	0	1	0	0	0	0	0	x	x	x	1	1	1	1	1	1
ASR R4, R2	0	1	1	0	0	0	1	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x
IN R2	1	1	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
OUT R4	x	0	x	x	x	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 7. Cuatro ejemplos de operaciones que se pueden hacer en la UP en un ciclo y las palabras de control asociadas a cada operación

Algunas de las 4 funciones que puede realizar la UP genérica se pueden hacer en el mismo ciclo, con alguna restricción. Por ejemplo, en un ciclo se puede hacer IN R2 y OUT R2 codificando la palabra de control como: 11010100xxxxxxxxxxxxxx. Otro ejemplo es hacer en un solo ciclo SUB R1, R2, R3 y OUT R2. De hecho, en cada ciclo, la salida DATAOUT tiene el valor del registro que se lee por el bus A (y en todos los ciclos se lee algún registro, aunque pongamos el campo @B con xxx, ya que al implementarse cada x pasa a ser o un 0 o un 1). Un ejemplo de dos funciones que no pueden hacerse en el mismo ciclo es ADD R3, R5, R6 y IN R1, ya que sólo uno de los dos valores, o lo que sale de la ALU o lo que llega por DATAIN, puede atravesar el multiplexor para estar presente en el bus de entrada D del banco de registros.

Ejemplo de procesador de propósito específico usando la UP genérica

Vamos a implementar el procesador de propósito específico para sumar los cuatro números que llegan por la entrada DATAIN, diseñado en clase, pero ahora usando la UP general vista en la sección anterior. El esquema en dos grandes bloques se muestra en la Figura 8.



Así pues, como la UP ya esta diseñada nos centramos en la UC. La UC la especificaremos mediante un grafo de Moore y debe generar en cada ciclo:

- la palabra de control (de 22 bits, ver Figura 6) y las salidas de control (en este ejemplo solo el bit End) y
- el estado siguiente del grafo, en función del estado actual, la palabra de estado (bit z de la ALU) y de las entradas de control (bit Begin). El bit z no se usa en este ejemplo, pero se usará en el siguiente.

En la implementación de propósito específico vista en clase, el primer dato se guardaba en el registro (el único necesario), el segundo, que llega justo al ciclo siguiente de llegar el primero, se sumaba con el primero en el mismo ciclo que llegaba y al final del ciclo se almacenaba el resultado en el registro. Con el tercer y cuarto datos se procedía igual que con el segundo. Al siguiente ciclo después de llegar el dato cuarto, el resultado se encontraba en la salida. En total se requerían 5 ciclos.

Esto no se podrá hacer tan rápido con la actual UP, ya que (ver Figura 6):

- no se puede conectar directamente la entrada DATAIN con la ALU sin antes pasar por un registro del banco de registros (lo que lo retrasa al menos un ciclo),
- ni se puede guardar en el banco de registros lo que llega por la entrada DATAIN en un ciclo y lo que produce la ALU en ese mismo ciclo. Esto hace que ni siquiera se pueda estar entrando un dato a la vez que se están sumando dos datos que entraron al banco de registros en ciclos anteriores.

Por ello lo que haremos sera entrar primero los 4 datos, que llegan uno detrás de otro en ciclos consecutivos y después realizar las sumas. La Figura 9 muestra el grafo de estados de la UC encargada de generar la secuencia de señales de control adecuada para resolver este problema. Las distintas salidas de control se han denotado como Si para i=0,.., 7. Una tabla a la izquierda del grafo indica el valor de la palabra de control y de la salida de control End asociada a cada Si. La palabra de control se específica con mnemotécnicos, como se hizo en la sección anterior, ya que es una forma más clara que con ceros y unos. A modo de ejercicio, debéis *traducir* cada una de las palabras de control de mnemotécnicos a tira de 22 bits (ver solución en Figura 10).

	Palabra Control	End
S0	IN R1	0
S1	IN R2	0
S2	IN R3	0
S3	IN R4	0
S4	ADD R5, R1, R2	0
S5	ADD R5, R3, R5	0
S6	ADD R5, R4, R5	0
S7	OUT R5, IN R1	1

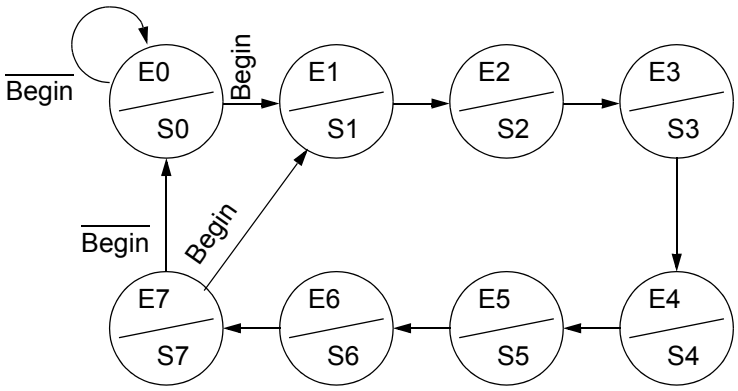


Figura 9. Grafo del sumador de 4 números para la UP genérica.

Como ya se anunció en un párrafo anterior, el tiempo de sumar los cuatro números usando la UP general es mayor que en el diseño totalmente específico del capitulo anterior. En concreto, 8 ciclos frente a 5. Esto ocurre en casi todos los ejemplos, la implementación más específica puede ser más rápida que una general. Por contra, esta última puede servir para resolver diferentes problemas con solo cambiar la UC.

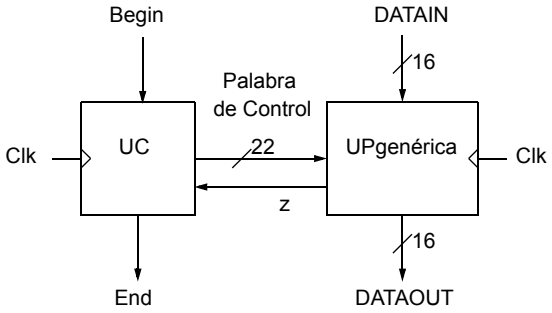
Para ser conscientes de que con la UP de la Figura 1 se pueden resolver diferentes problemas, en la sección siguiente diseñamos la UC para otro problema.

Mnemotécnico	Palabra de Control																						
	In/Alu	WrD	@D			@A			Cmp/Al	OP			Rb/I	@B			Immed						
			d_2	d_1	d_0	d_2	d_1	d_0		d_2	d_1	d_0		d_2	d_1	d_0	d_5	d_4	d_3	d_2	d_1	d_0
IN R1	1	1	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
IN R2	1	1	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
IN R3	1	1	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
IN R4	1	1	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
ADD R5, R1, R2	0	1	1	0	1	0	0	1	0	0	0	0	1	0	1	0	x	x	x	x	x	x	x
ADD R5, R3, R5	0	1	1	0	1	0	1	1	0	0	0	0	1	1	0	1	x	x	x	x	x	x	x
ADD R5, R4, R5	0	1	1	0	1	1	0	0	0	0	0	0	1	1	0	1	x	x	x	x	x	x	x
OUT R5, IN R1	1	1	0	0	1	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 10. Tabla de traducción de mnemotécnicos a tira de bits de la palabra de control.

Otro ejemplo de procesador de propósito específico usando la UP genérica

Diseñamos a continuación el grafo de la UC que junto a la UP genérica de la Figura 1 implemente el siguiente ejemplo: diseñar un procesador que obtenga la suma de *n* números naturales que llegan al procesador ciclo a ciclo por la entrada DATAIN. En el primer ciclo, cuando la entrada de control *Begin* vale 1, en la entrada de datos se encuentra el valor *n*. En los siguientes *n* ciclos están disponibles en esa misma entrada, los valores de los *n* números, a razón de uno por ciclo. La suma debe estar disponible en la salida DATAOUT cuando finalice el cálculo, lo que se indicara poniendo la salida de control *End* a 1 durante ese mismo ciclo. El esquema general diferenciando UC y UP se muestra en la Figura 11.



En la implementación totalmente específica que hicimos en clase teníamos 2 unidades aritméticas trabajando en paralelo. En un mismo ciclo el sumador sumaba el nuevo número que llegaba por la entrada DATAIN con el contenido del registro acumulador, mientras que el decrementador calculaba los datos que faltaban por llegar (y por sumar).

Esto no va a poder hacerse a esta velocidad usando la UP general, ya que solo tiene una unidad funcional (la ALU) que deberá ser usada tanto para decrementar como para sumar, pero, eso si, no en el mismo ciclo las dos cosas, sino que en un ciclo se decrementará y al siguiente se sumará. Además, como se vio en la sección anterior, no se puede almacenar en el banco de registros en el mismo ciclo el resultado de la ALU y el dato de la entrada. Con todo ello, resulta imposible implementar el sistema con la UP general si los datos llegan a la velocidad de uno por ciclo. Necesitamos varios ciclos entre la llegada de un dato y la del siguiente, un ciclo para entrar el dato, otro para sumar y otro para decrementar y saber si faltan mas datos). Esto nos obliga a modificar el enunciado original en cuanto a la sincronizacion de la llegada de los datos, ahora los datos han de llegar según nos convenga en nuestro diseño.

Por otro lado, no almacenaremos primero los datos en el banco de registros y luego los sumaremos, como se hizo en la sección anterior, ya que esto supondría limitar *n* al tamaño del banco de registros y por consiguiente otro cambio del enunciado.

La Figura 12 muestra una especificación de la UC que junto a la UP genérica resuelve este problema.

	Mnemotécnico Palabra Control	End
S0	IN R1	0
S1	IN R2	0
S2	ADDI R1, R1, # -1	0
S3	IN R3	0
S4	ADD R2, R2, R3	0
S5	OUT R2, IN R1	1

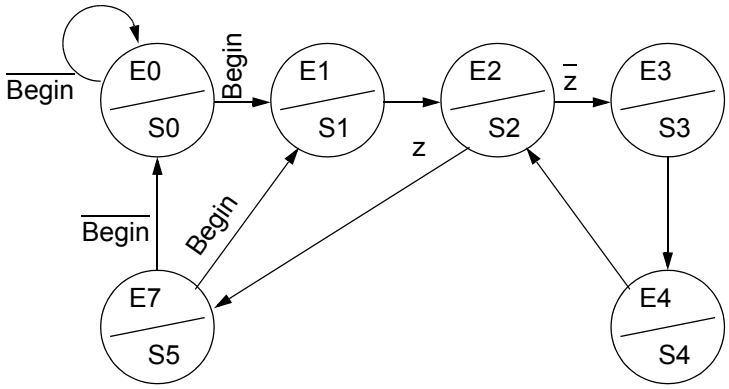


Figura 12. Grafo del sumador de *n* números para la UP genérica.

A la vista del grafo que hemos obtenido, la llegada síncrona de datos para que el sistema funcione correctamente debe formularse como sigue.

La señal Begin se podrá a 1 durante un ciclo indicando al sistema que debe comenzar un cálculo y que los datos son validos en la entrada DATAIN con la siguiente secuenciación:

- Durante el mismo ciclo que Begin vale 1 en DATAIN se encuentra el valor *n*. Supongamos que este es el ciclo *c*.
- Al ciclo siguiente, durante el ciclo *c*+1, en DATAIN se encuentra el primer dato a sumar.

- En el ciclo *c*+3 llega el tercer dato (si *n*>1), y a partir de este momento cada 3 ciclos llega un nuevo dato por DATAIN (ciclos *c*+6, *c*+9, ...).

Ejercicio:

Efectuar las modificaciones oportunas en el grafo de la Figura 12 para que el sistema de la sección anterior funcione correctamente con la siguiente sincronización de los datos que llegan por DATAIN.

- Durante el mismo ciclo que Begin vale 1 en DATAIN se encuentra el valor *n*. Supongamos que este es el ciclo *c*.
- A partir del ciclo *c*, cada 3 ciclos llega un dato a sumar. Esto es, el primer dato a sumar llega en el ciclo *c*+3, el segundo en el *c*+6, el tercero en el *c*+9,...