

66.70 Estructura del Computador

Microarquitectura

```

    push ebp                                ;function prologue
    mov ebp, esp
    add esp, -12

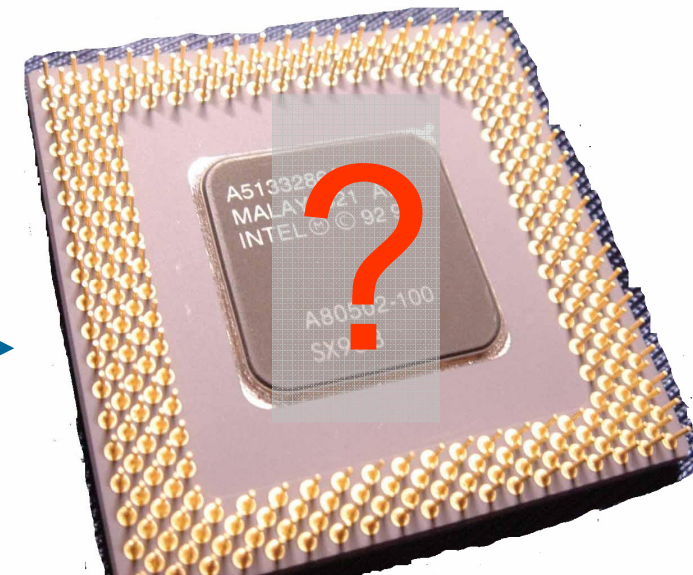
    ;if (x<2)
    cmp DWORD PTR [ebp + 8], 2             ;check if x<2
    jl  $IF_TRUE0
$IF_FALSE0:
    ;fibonacci(x-1)+fibonacci(x-2);
    mov eax, DWORD PTR [ebp + 8] ;calculate x-1
    sub eax, 1
    push eax                               ;push it on as an argument
    call _fib__fibonacci                 ;make the call to fib(x-1)
    mov DWORD PTR [ebp-8], eax           ;save the returned value in [ebp-8]

    mov eax, DWORD PTR [ebp-8] ;calculate x-2
    sub eax, 2
    push eax                               ;push it on as an argument
    call _fib__fibonacci                 ;make the call to fib(x-2)
    mov DWORD PTR [ebp-12], eax          ;save the returned value in [ebp-12]
    mov eax, DWORD PTR [ebp-12]
    add eax, DWORD PTR [ebp-8]           ;move fib(x-1)+fib(x-2) into eax
    mov DWORD PTR [ebp-4], eax          ;move the value into a temporary for returning
$IF_END0:
    mov eax, DWORD PTR [ebp-4]          ;move the temporary with the return
    ;value into the return register

$fibonacci_epi:
    mov esp, ebp;function epilogue
    pop ebp
    ret
$IF_TRUE0:

```

ISA



ISA

- *Set de instrucciones*
- *Hardware visible al **programador***

MICROARQUITECTURA

Implementación 1
en hardware

p.e.: mayor velocidad

Implementación 2
en hardware

p.e.: menor costo

...

Implementación N
en hardware

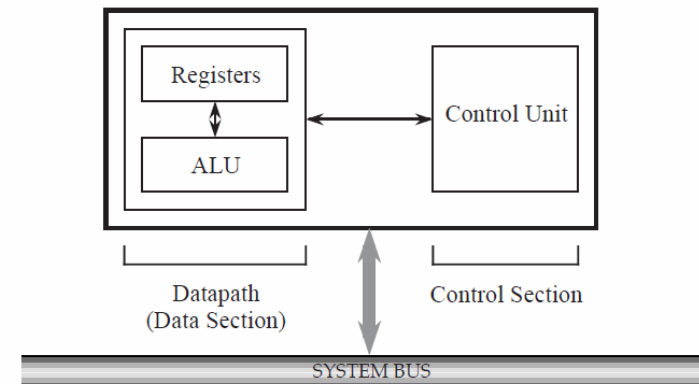
p.e.: menor consumo

← **Énfasis en**

Microarquitectura =

- ✓ Unidad de control
- ✓ Unidad aritmético lógica
- ✓ Registros visibles al programador
- ✓ Cualquier otro registro adicional necesario para el funcionamiento de la unidad de control

Objetivo: *ciclo de fetch*



Abordajes al diseño de la microarquitectura

Podemos diseñar la unidad de control según:

→ *Control microprogramado*

→ *Lógica cableada*

El trayecto de datos es esencialmente idéntico
en ambos casos

Implementación del Trayecto de Datos

Estructuras para realizar el movimiento de datos

Movimiento de datos
externo al CPU



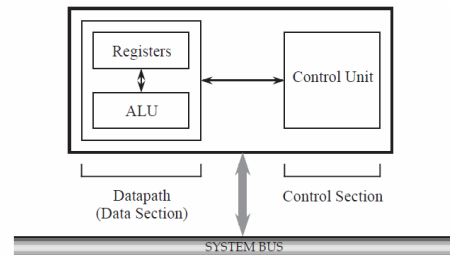
BUS DE SISTEMA

Movimiento de datos
dentro del CPU

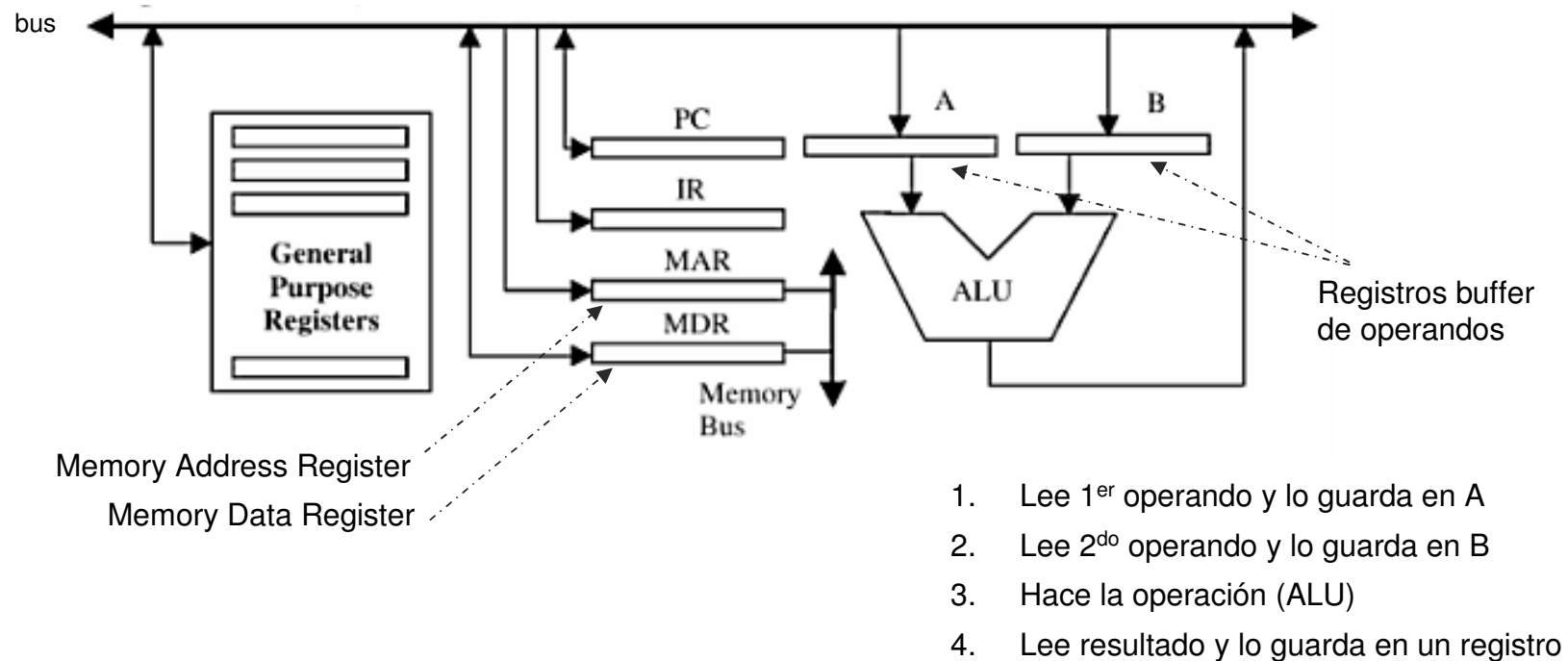


BUSES INTERNOS al CPU

- *Organizado en 1 bus*
- *Organizado en 2 buses*
- *Organizado en 3 buses*



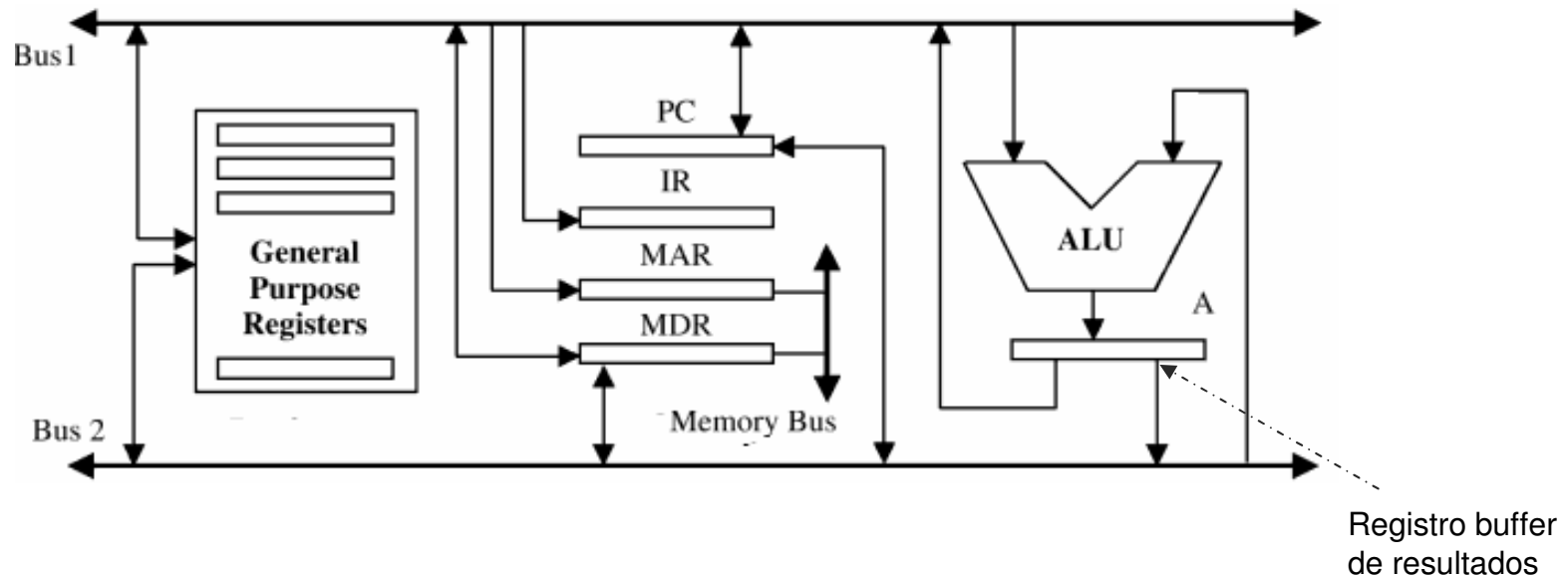
Trayecto de Datos Organizado en 1 bus



– Simple y de bajo costo

– Limitado flujo de datos por ciclo de reloj → Baja performance

Trayecto de Datos Organizado en 2 buses

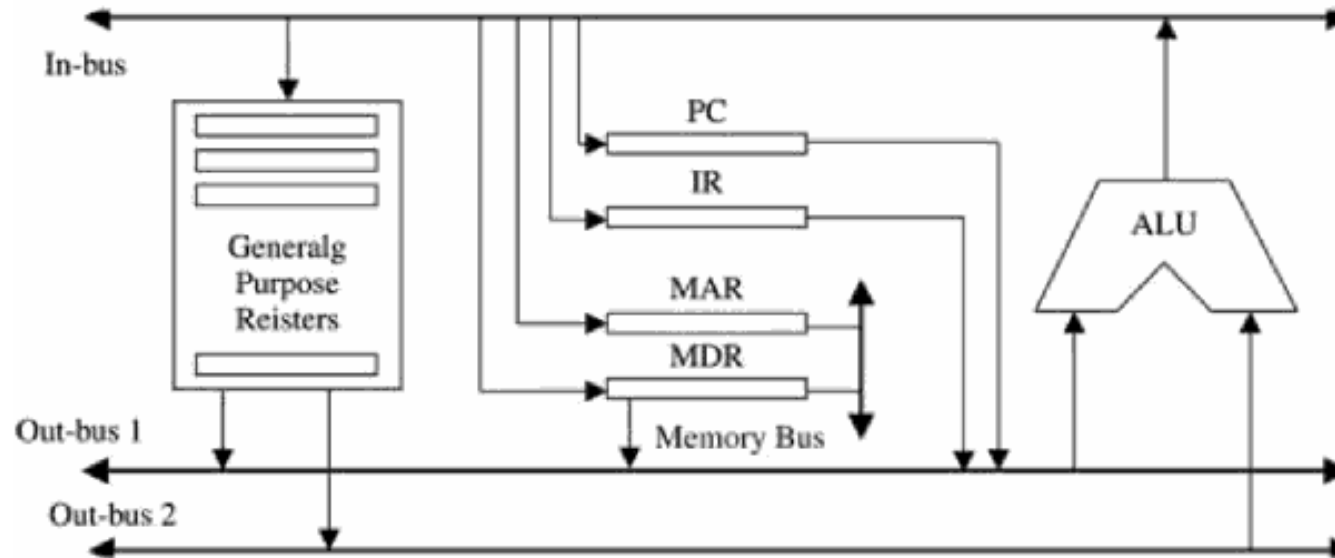


- *Obtiene los 2 operandos en el mismo ciclo de reloj*
- *Buffer para el resultado (bus1 y bus2 ocupados con operandos)*
- *Más rápido que organización en 1 bus*

Trayecto de Datos Organizado en 3 buses

Out-bus: Dato sale de registro

In-bus: Dato entra a registro



- *Mueve 2 operandos y resultado en el mismo ciclo de reloj*
- *Buena performance*
- *Mayor complejidad de hardware*

Esta implementación

Además de su organización en 3 buses
Incluye la idea del **bus unidireccional**

La Unidad Aritmético-Lógica

Cuestiones relacionadas a su implementación

Performance

- *Velocidad*
- *Área ocupada en el integrado*
- *Disipación de potencia*

Ej.: Carry look-ahead:
- Alta velocidad
- Ocupa mucho espacio

Funcionalidad

- *Debe: mover datos y hacer operaciones aritméticas/lógicas*
- *Operaciones básicas de ALU para obtener el set de instrucciones dado*
(no necesariamente coinciden con las del ISA)

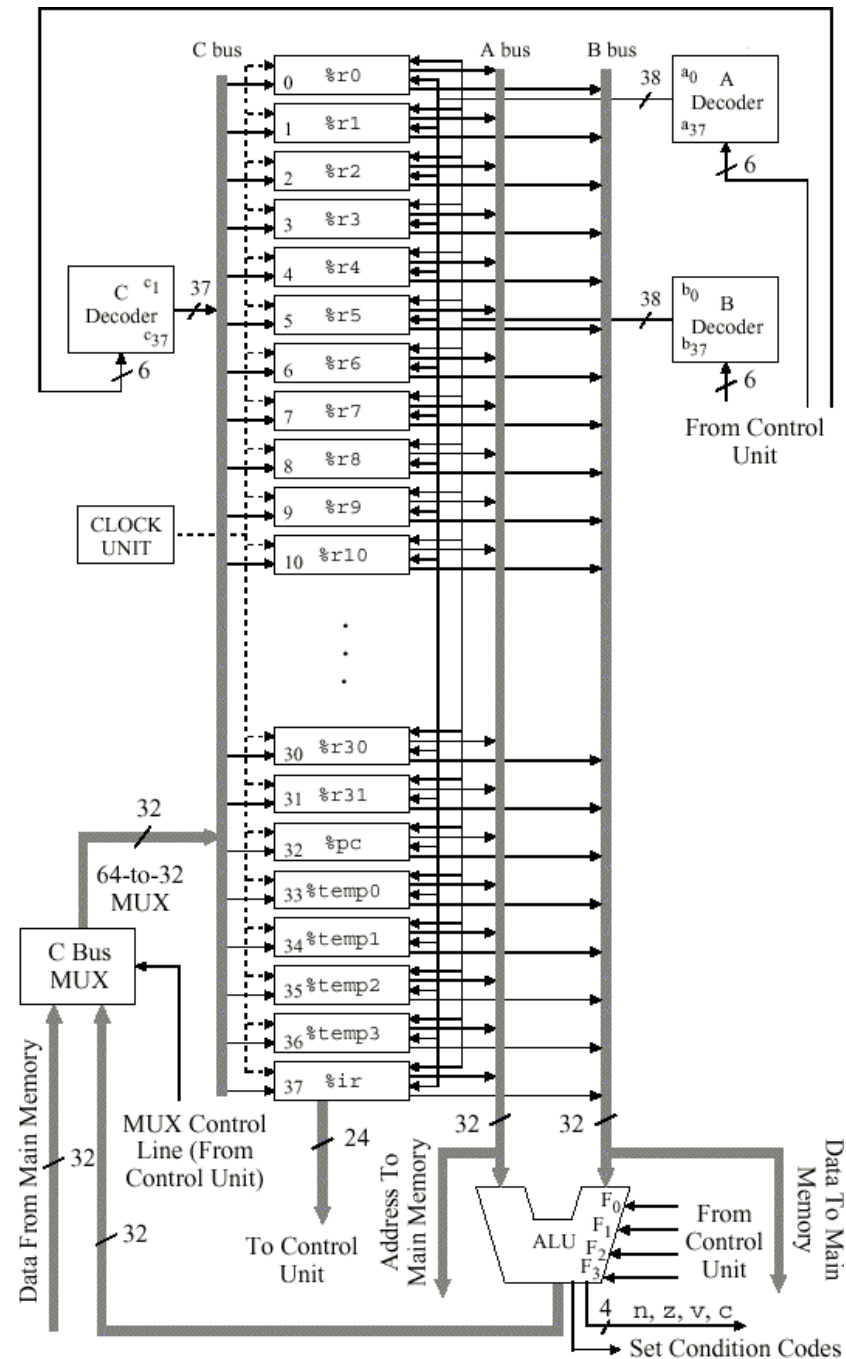
Implementación

- *Con 32 módulos de 1 bit puede obtenerse una ALU de 32 bits*
- *Puede basarse en el uso de una “Look-up table”*
- *Definir estructura de buses*

Una microarquitectura ARC

Trayecto de Datos

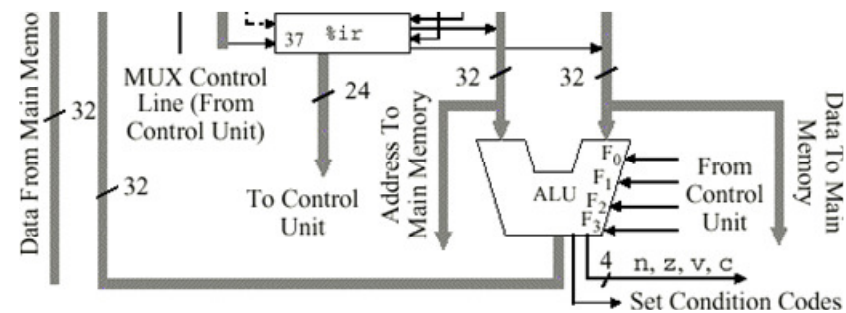
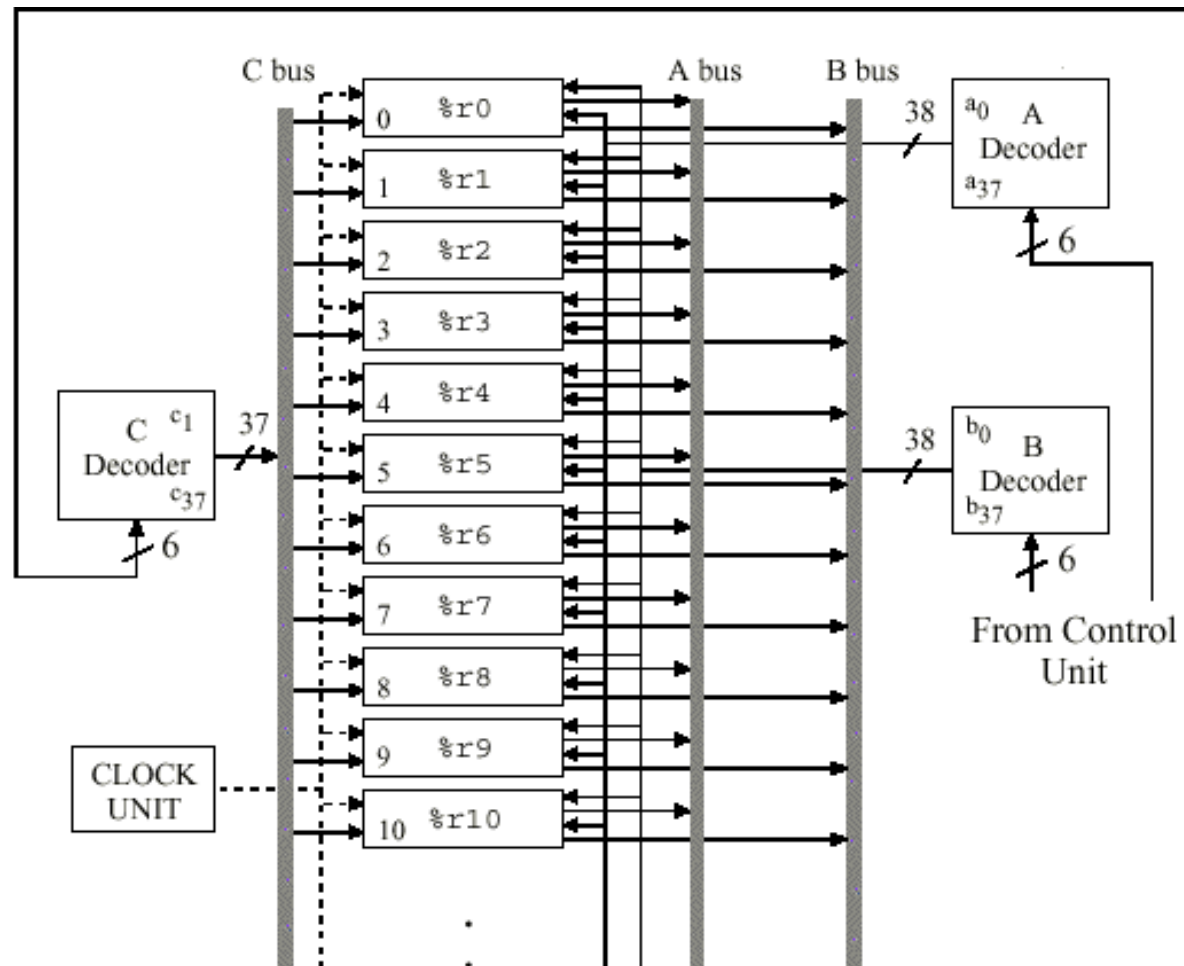
- ✓ Buses
- ✓ ALU
- ✓ Registros
- ✓ Decodificadores
- ✓ Multiplexores



Una microarquitectura ARC

Trayecto de Datos

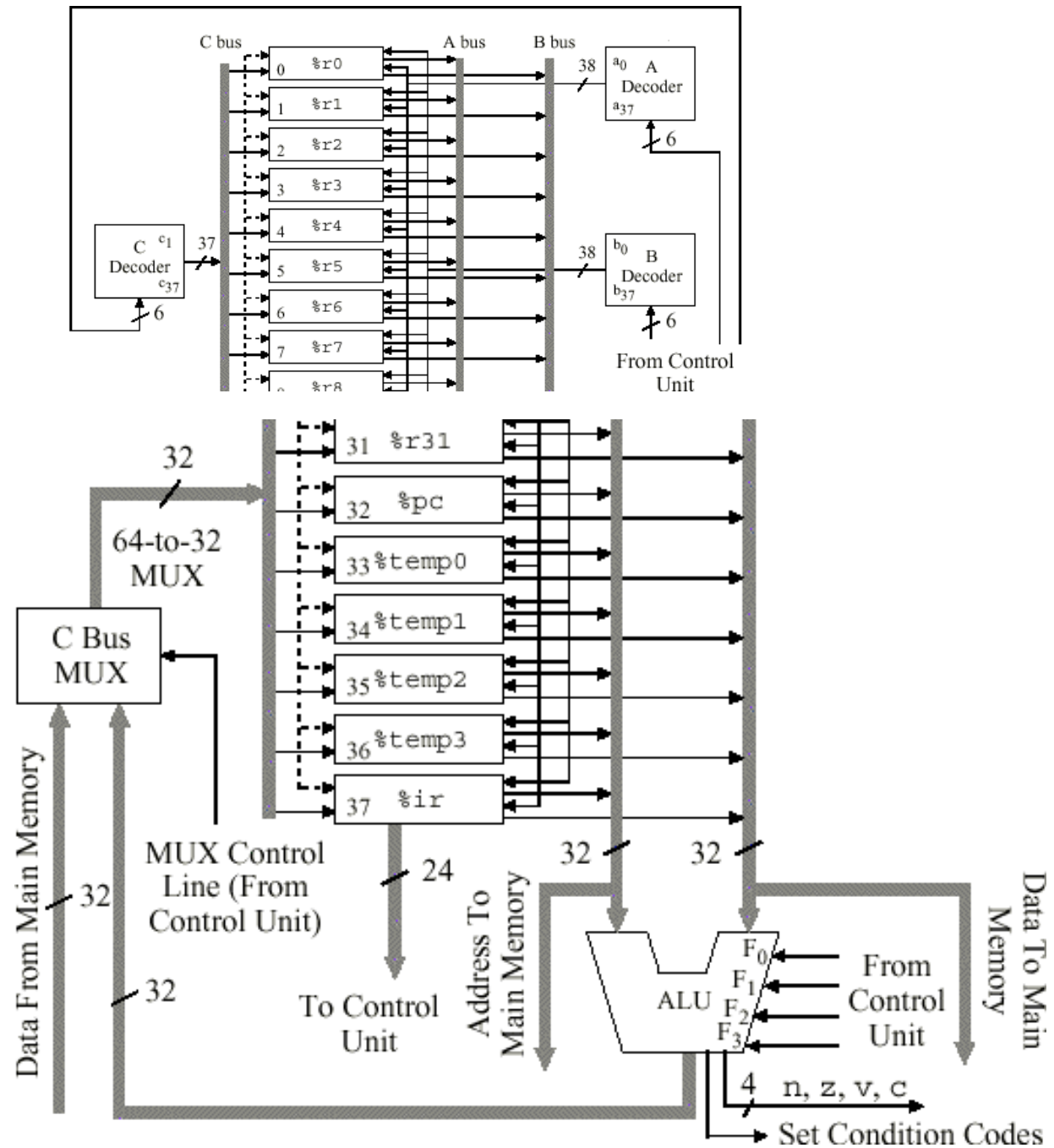
- ✓ Buses
- ✓ ALU
- ✓ Registros
- ✓ Decodificadores
- ✓ Multiplexores



Una microarquitectura ARC

Trayecto de Datos

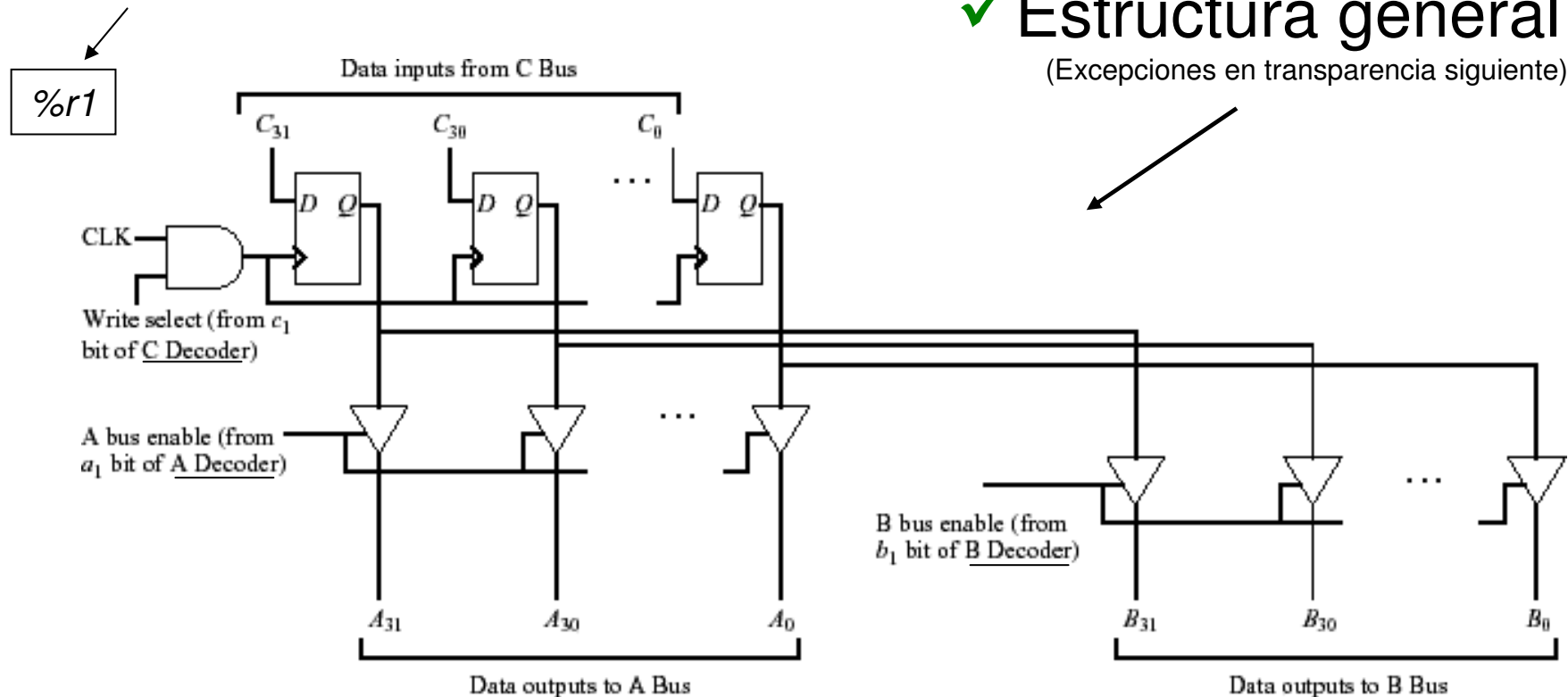
- ✓ Buses
- ✓ ALU
- ✓ Registros
- ✓ Decodificadores
- ✓ Multiplexores



ARC

Los registros

¿Porqué digo que es %r1 ?



- Flip-Flops D flanco descendente
- Escritura desde bus C habilitada por el correspondiente bit del decodificador C
- Lectura a buses A y B (a través de buffers Tri-state)

ARC

Registros que **no** responden a la estructura general

✓ Registro *%r0*

- No necesita flip-flops
- No tiene entrada de datos desde bus C
- No tiene entrada desde decodificador del bus C

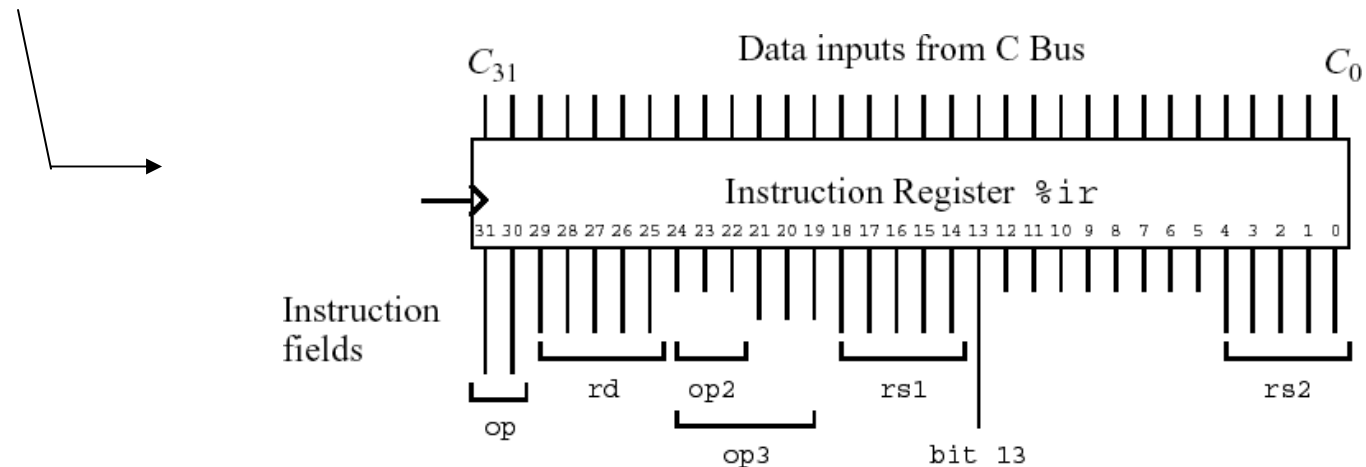
✓ Contador de programa *PC*

- Sólo almacena números múltiplos de 4

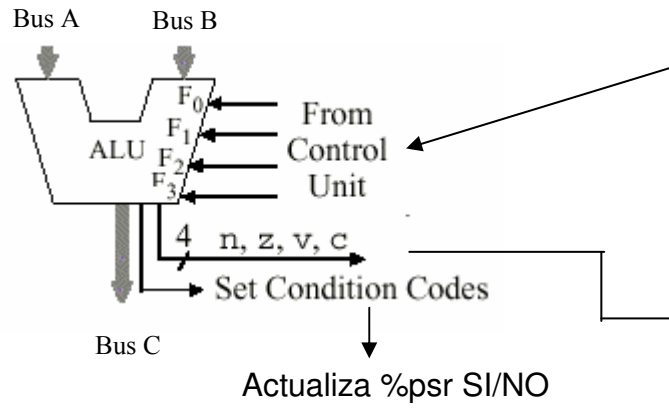
=> los 2 LSB cableados a cero

✓ Registro de instrucciones *IR*

- Tiene salidas específicas para campos del código de máquina



La Unidad Aritmético-Lógica



Operaciones a implementar en la ALU

F_3 F_2 F_1 F_0	Operation	Changes Condition Codes
0 0 0 0	ANDCC (A, B)	yes
0 0 0 1	ORCC (A, B)	yes
0 0 1 0	NORCC (A, B)	yes
0 0 1 1	ADDCC (A, B)	yes
0 1 0 0	SRL (A, B)	no
0 1 0 1	AND (A, B)	no
0 1 1 0	OR (A, B)	no
0 1 1 1	NOR (A, B)	no
1 0 0 0	ADD (A, B)	no
1 0 0 1	LSHIFT2 (A)	no
1 0 1 0	LSHIFT10 (A)	no
1 0 1 1	SIMM13 (A)	no
1 1 0 0	SEXT13 (A)	no
1 1 0 1	INC (A)	no
1 1 1 0	INCPC (A)	no
1 1 1 1	RSHIFT5 (A)	no

SRL: Shift right B pos.(0-31) el valor en A

LSHIFT2: (Bus A) shift left 2 pos.

LSHIFT10: (Bus A) shift left 10 pos.

SIMM13: LSB(13, Bus A) MSB = 0's

SEXT13: Simm13 en Bus A con Ext. Signo

INC: (BusA) + 1

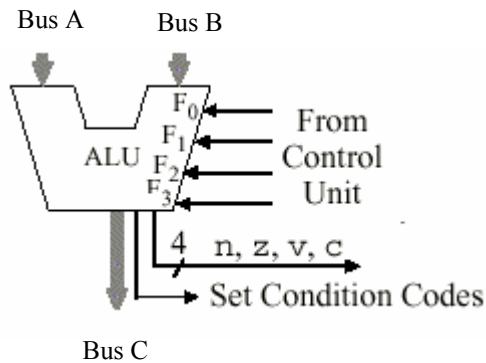
INCPC: (BusA) + 4

RSHIFT5: Shift right 5 pos. y ext. signo

ARC

ALU: *Funcionalidad*

Desde instrucciones básicas de ALU al set de instrucciones del procesador



(lenguaje ilustrativo)

Complemento a 2	NOR (%r2, %r2), %r1 INC %r1
Resta	NOR (%r2, %r2), %temp1 INC %temp1 ADDCC (%rtemp1, %r1), %r1
Shift Left 1 bit	ADD (%r2, %r2), %r2

F_3 F_2 F_1 F_0	Operation	Changes Condition Codes
0 0 0 0	ANDCC (A, B)	yes
0 0 0 1	ORCC (A, B)	yes
0 0 1 0	NORCC (A, B)	yes
0 0 1 1	ADDCC (A, B)	yes
0 1 0 0	SRL (A, B)	no
0 1 0 1	AND (A, B)	no
0 1 1 0	OR (A, B)	no
0 1 1 1	NOR (A, B)	no
1 0 0 0	ADD (A, B)	no
1 0 0 1	LSHIFT2 (A)	no
1 0 1 0	LSHIFT10 (A)	no
1 0 1 1	SIMM13 (A)	no
1 1 0 0	SEXT13 (A)	no
1 1 0 1	INC (A)	no
1 1 1 0	INCPC (A)	no
1 1 1 1	RSHIFT5 (A)	no


Implementando una ALU

Funciones que debe realizar:

- Operaciones aritméticas y lógicas
- Desplazamientos a derecha e izquierda

Soluciones alternativas:

- Para las operaciones aritmético-lógicas
 - Circuitos sumadores, restadores, etc. que vimos anteriormente
 - Usar una “look-up table”
- Para los desplazamientos
 - Registros de desplazamiento
 - Desplazador rápido o “Barrel-shifter”



¿Cuál es el problema de hacer un desplazamiento de n bits usando registros de desplazamiento?

ALU implementada con “look-up table” (LUT) y “barrel-shifter”

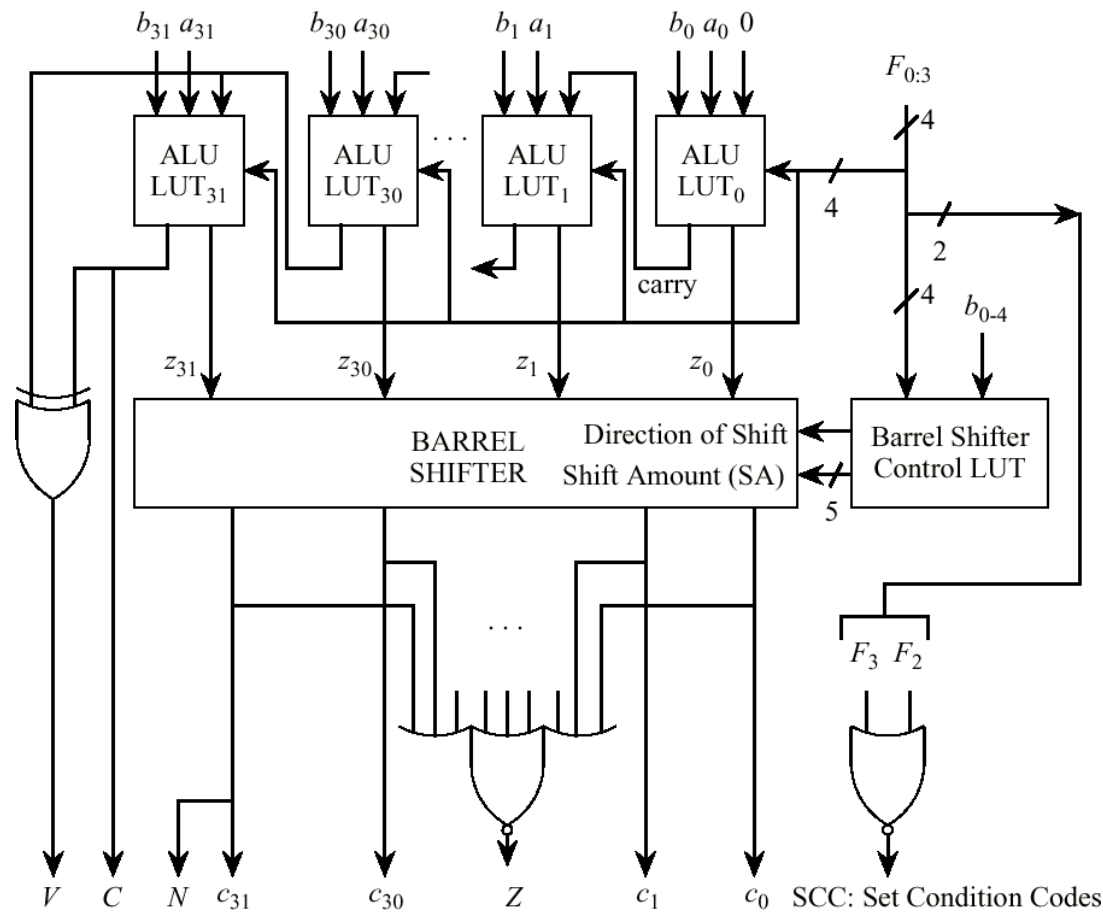
LUT

- operaciones **aritmético-lógicas**
 - 2 entradas de 32 bits (operandos)
 - entrada de control: 4 bits (elige operación)
 - salida de 32 bits (resultado)
 - Salida CC: n, z, v, c (código de condición)
 - Salida SCC (set CC): 1 bit (setea %psr: si/no)
 - Implementable con 32 módulos de 1 bit

Desplazador rápido (“Barrel Shifter”)

- **desplazamientos der/izq. de 0 a 31 bits en un solo ciclo de reloj**
 - entrada de operando: 32 bits
 - entrada n° bits despl.: 5 bits
 - entrada de dirección: 1 bit
 - salida 32 bits

ALU implementada con “look-up table” (LUT) y “barrel-shifter”



F_3	F_2	F_1	F_0	Operation
0	0	0	0	ANDCC (A, B)
0	0	0	1	ORCC (A, B)
0	0	1	0	NORCC (A, B)
0	0	1	1	ADDCC (A, B)
0	1	0	0	SRL (A, B)
0	1	0	1	AND (A, B)
0	1	1	0	OR (A, B)
0	1	1	1	NOR (A, B)
1	0	0	0	ADD (A, B)
1	0	0	1	LSHIFT2 (A)
1	0	1	0	LSHIFT10 (A)
1	0	1	1	SIMM13 (A)
1	1	0	0	SEXT13 (A)
1	1	0	1	INC (A)
1	1	1	0	INCPC (A)
1	1	1	1	RSHIFT5 (A)

ARC

ALU: *Implementación*

Tabla de verdad (parcial) de una LUT de 1 bit

		F_3	F_2	F_1	F_0	$Carry$ In	a_i	b_i	z_i	$Carry$ Out
ANDCC		0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	1	0	0
		0	0	0	0	0	1	0	0	0
		0	0	0	0	0	1	1	1	0
		0	0	0	0	1	0	0	0	0
		0	0	0	0	1	0	1	0	0
		0	0	0	0	1	1	0	0	0
		0	0	0	0	1	1	1	1	0
ORCC		0	0	0	1	0	0	0	0	0
		0	0	0	1	0	0	1	1	0
		0	0	0	1	0	1	0	1	0
		0	0	0	1	0	1	1	1	0
		0	0	0	1	1	0	0	0	0
		0	0	0	1	1	0	1	1	0
		0	0	0	1	1	1	1	1	0
						.				.
						.				.
						.				.

Desplazador rápido (Barrel Shifter)

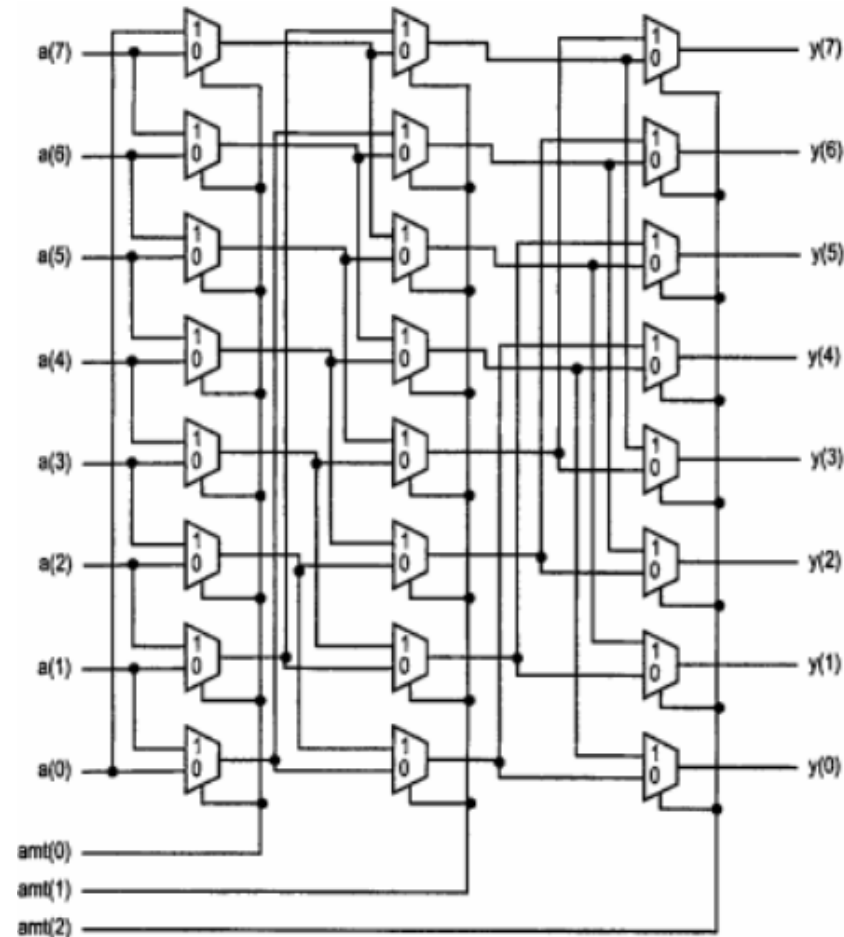
Funciones implementables

- Shift lógico de n bits der/izq
- Shift aritmético de n bits der/izq
- Shift circular (rotación) der/izq

Principio de funcionamiento

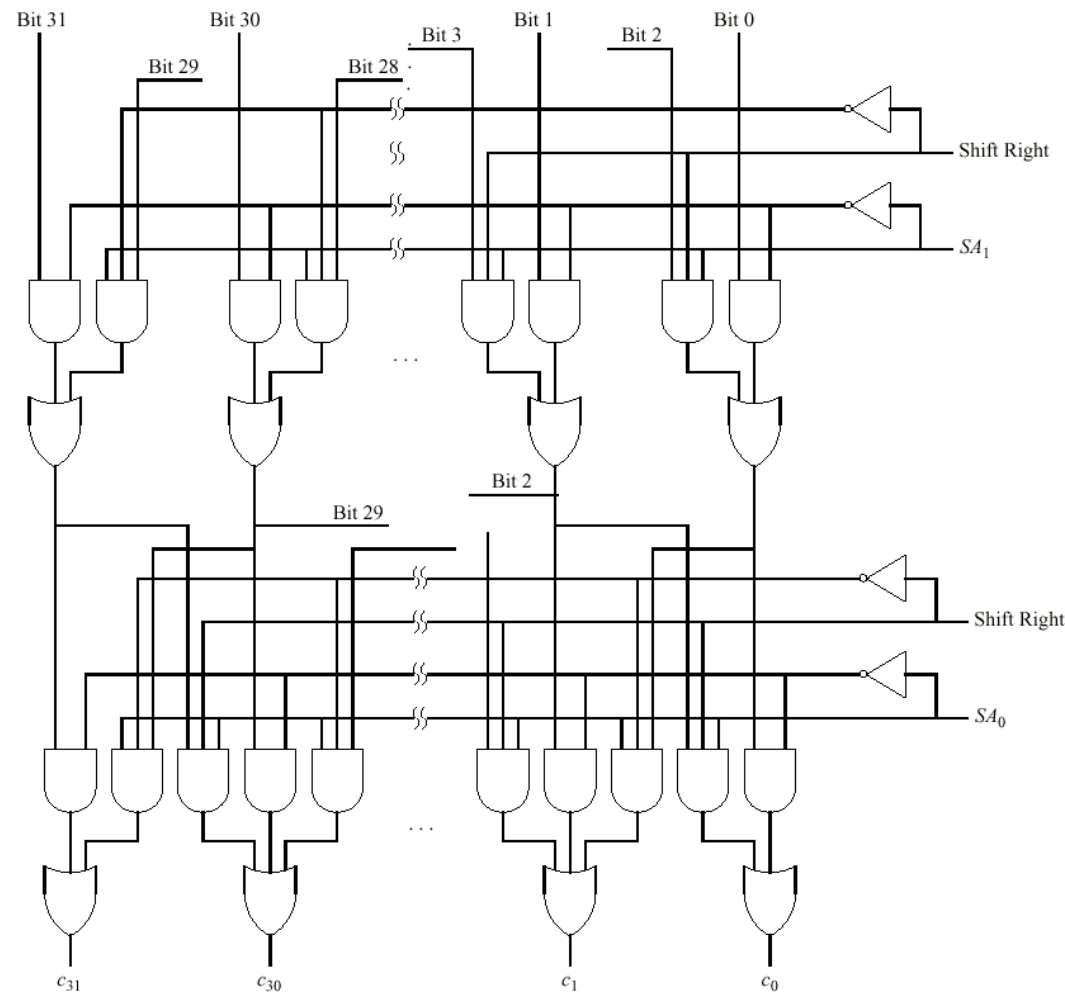
- Organización por niveles
- Cada nivel desplaza 2^n al anterior
- Bits de control definen rango del shift

Una implementación posible



Shift circular a derecha

Desplazador rápido (Barrel Shifter)



Implementación presentada en el libro de Murdocca-Heuring