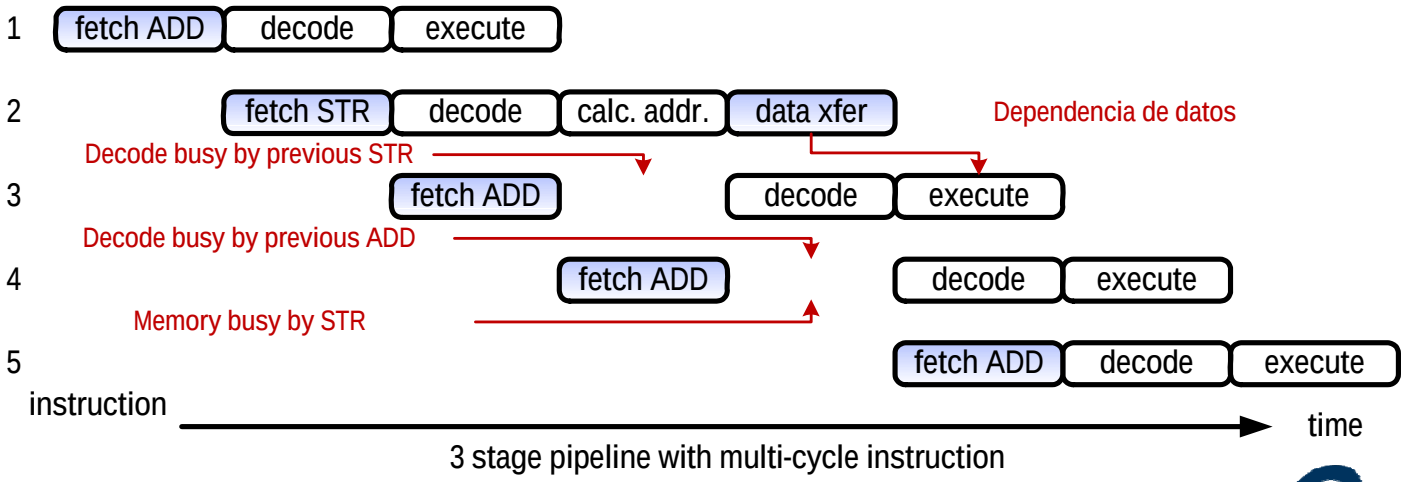
Sistemas electrónicos digitales

Tema 4:
ARM Organización e implementación



Índice del tema

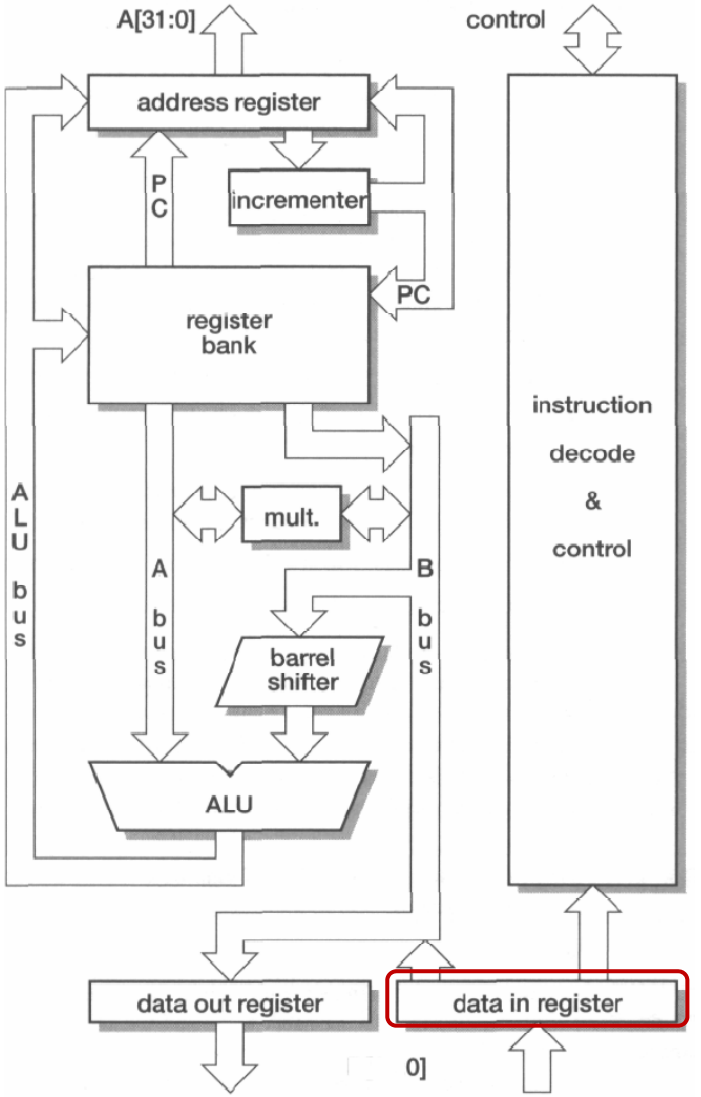
- Introducción
- Organización con 3 etapas de pipeline
- Organización con 5 etapas de pipeline
- Ejecución de instrucciones
- Implementación del ARM
- Interfaz de co-procesador
- Ejemplos

Introducción

- Modificación de la arquitectura del ARM
  - Avances en la tecnología CMOS. Reducción del tamaño en los procesos tecnológicos
  - 1990-1995: Cambio arquitectural en el diseño del ARM. Aparecen los cores ARM6 y ARM7 con diseño con tres etapas de pipeline
  - 1995 → Aumento de las prestaciones del ARM con la introducción de procesadores con 5 etapas de pipeline, ARM9
  - Avance hacia una arquitectura Harvard, con memoria de datos e instrucciones separadas, normalmente a modo de cachés, con memoria externa compartida, datos e instrucciones
  - Últimas versiones ofrecen pipeline de 8 etapas

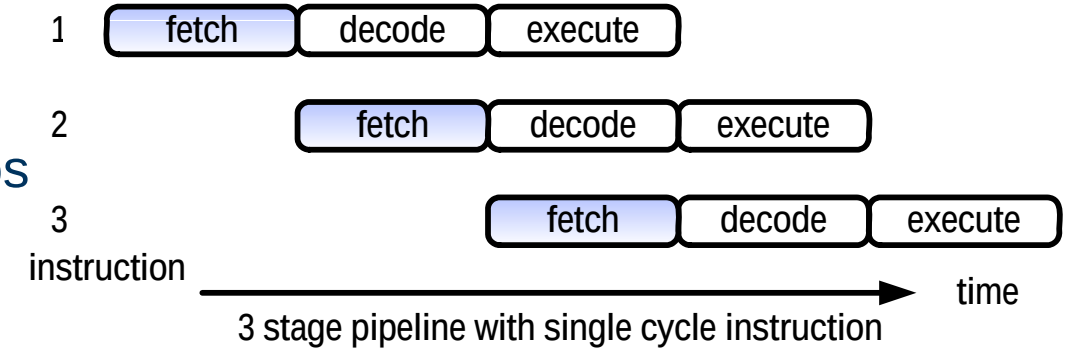
Organización con 3 etapas de pipeline: Arquitectura

- *Register bank*
  - Dos puertos de lectura + uno de escritura con acceso a todos los registros
  - Un puerto de lectura y otro de escritura con acceso a r15 (PC)
- *Barrel shifter*
  - Desplaza o rota el contenido de los registros
- *Address register and incrementer*
  - Permite mantener las direcciones de memoria así como facilitar su acceso secuencial
- *Data registers*
  - Transferencia con memoria. Buses de entrada y salida independiente



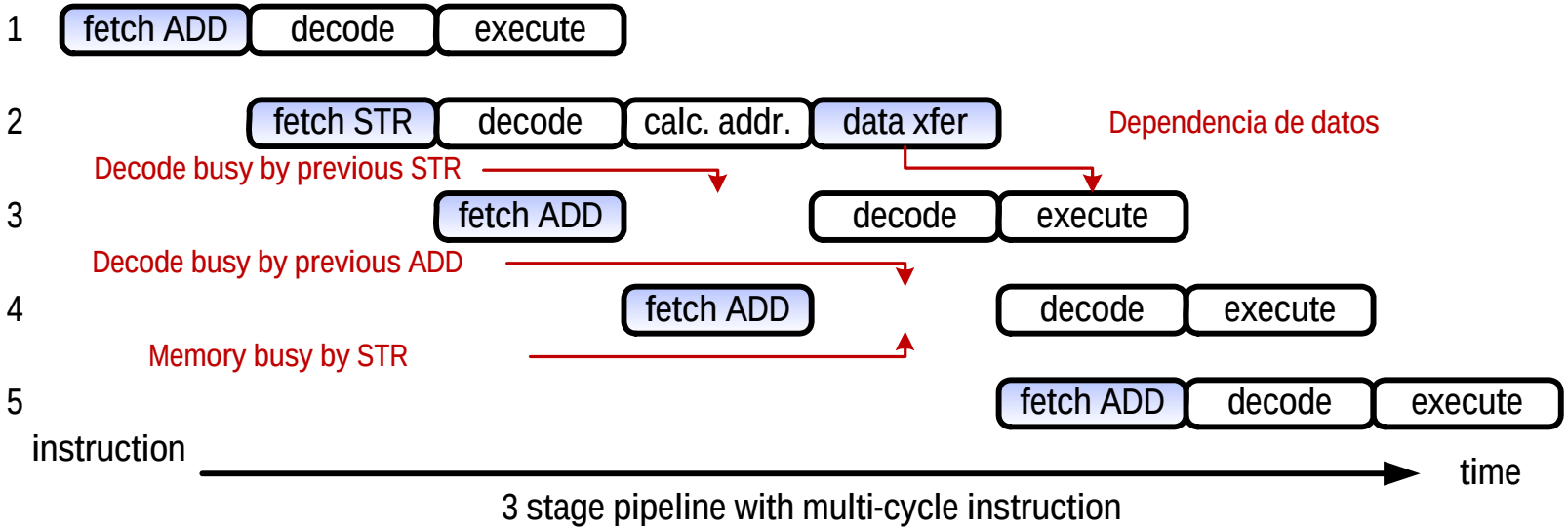
Organización con 3 etapas de pipeline: Etapas I

- 3 etapas de pipeline
  - *Fetch*: lectura de la instrucción y almacenamiento en pipeline de instrucciones
  - *Decode*: Se decodifica la instrucción y se preparan las señales para actuar sobre el *Datapath*. La instrucción no gobierna el *Datapath*
  - *Execute*: La instrucción toma el control del *Datapath*: banco de registro, desplazador, ALU, se genera el resultado y se almacena
  - 3 instrucciones en el pipeline
  - Latencia de 3 ciclos
  - 1 instrucción por ciclo



Organización con 3 etapas de pipeline: Etapas II

- 3 etapas de pipeline: ejecución de instrucciones con múltiples ciclos
  - Ejemplo: secuencia de ejecución sumas con una operación de almacenamiento intercalada tras la primera suma



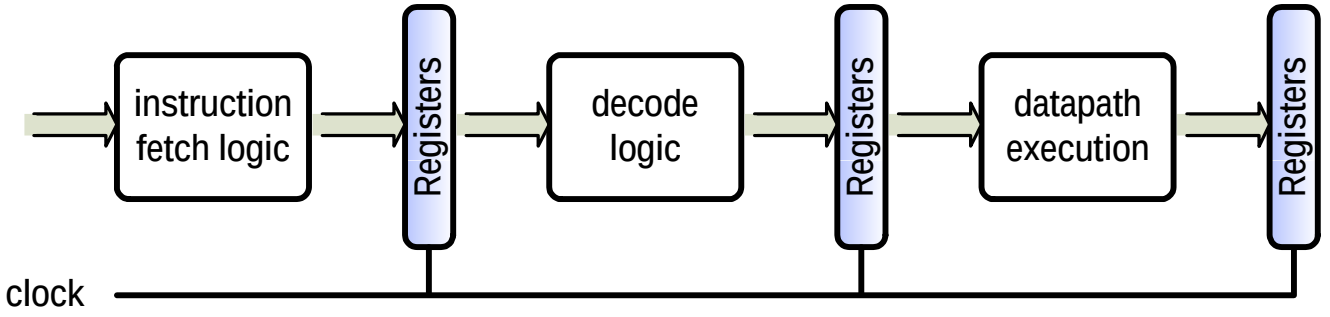
3 stage pipeline with multi-cycle instruction

  - Las etapas en sombreado realizan acceso a memoria
  - STR: decodificación en dos ciclos
    - Primer ciclo: decodifica la instrucción
    - Segundo ciclo: cálculo de la dirección efectiva

Organización con 3 etapas de pipeline: Consideraciones

Consideraciones generales

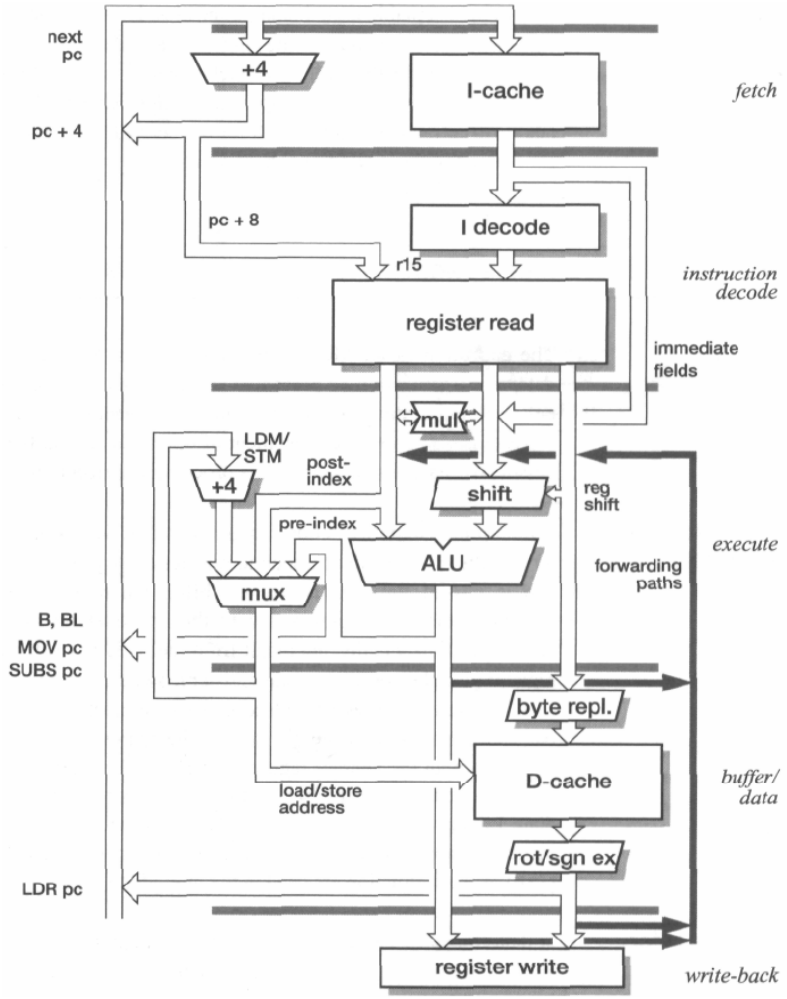
- Las instrucciones de salto provocan un *pipeline flushing*
- Durante el ciclo de decodificación, la instrucción no actúa sobre el pipeline sino sobre la lógica de control



- Durante el primer ciclo de *Datapath* cada instrucción realiza el *fetch* de la siguiente instrucción
  - En caso de usar el PC como registro de propósito general, durante la etapa de ejecución, éste apunta a la segunda instrucción posterior, PC + 8

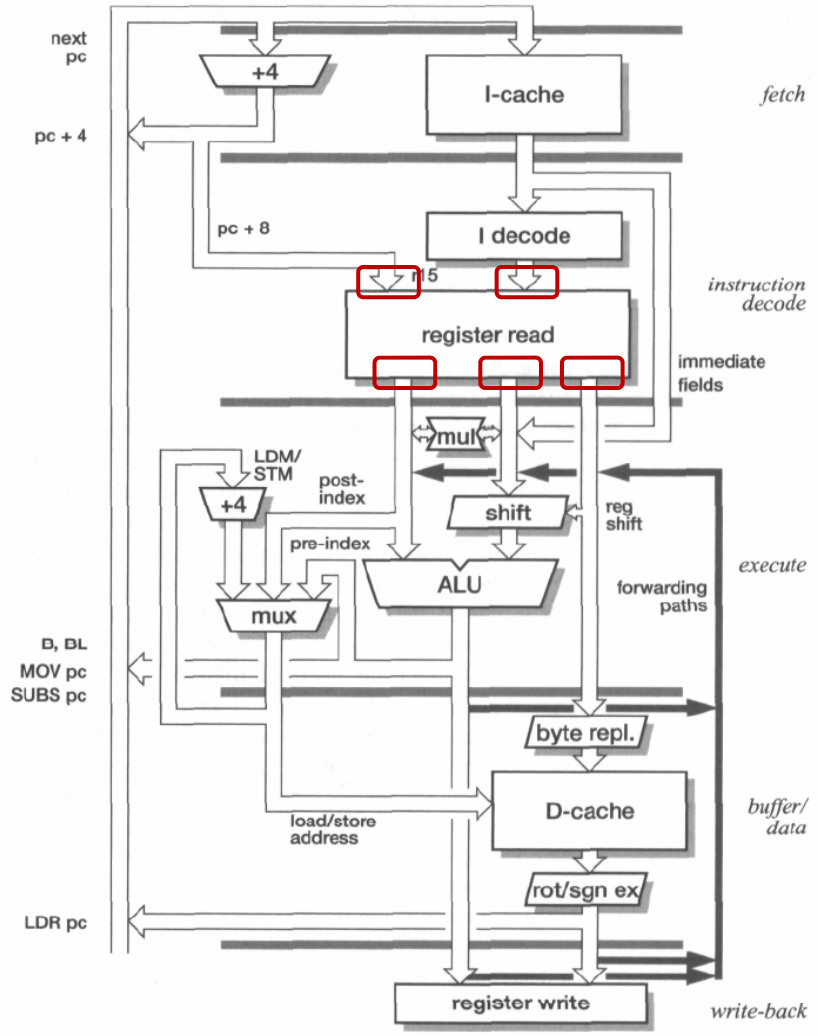
Organización con 5 etapas de pipeline: Consideraciones

- Tiempo de ejecución de un programa
  - $T_{prog} = (N_{ist} \times CPI)/F_{clk}$
  - Mejora de prestaciones
    - Aumento de la frecuencia
    - Reducir el CPI
- *Memory bottleneck*
  - Arquitectura Von Neumann
    - No permite la transferencia de más de una palabra por ciclo de reloj
  - Arquitectura Harvard
    - Permite el acceso a instrucción y dato en un ciclo ciclo de reloj
- Solución adoptada
  - 5-stage pipeline
  - Arquitectura Harvard



Organización con 5 etapas de pipeline: Etapas

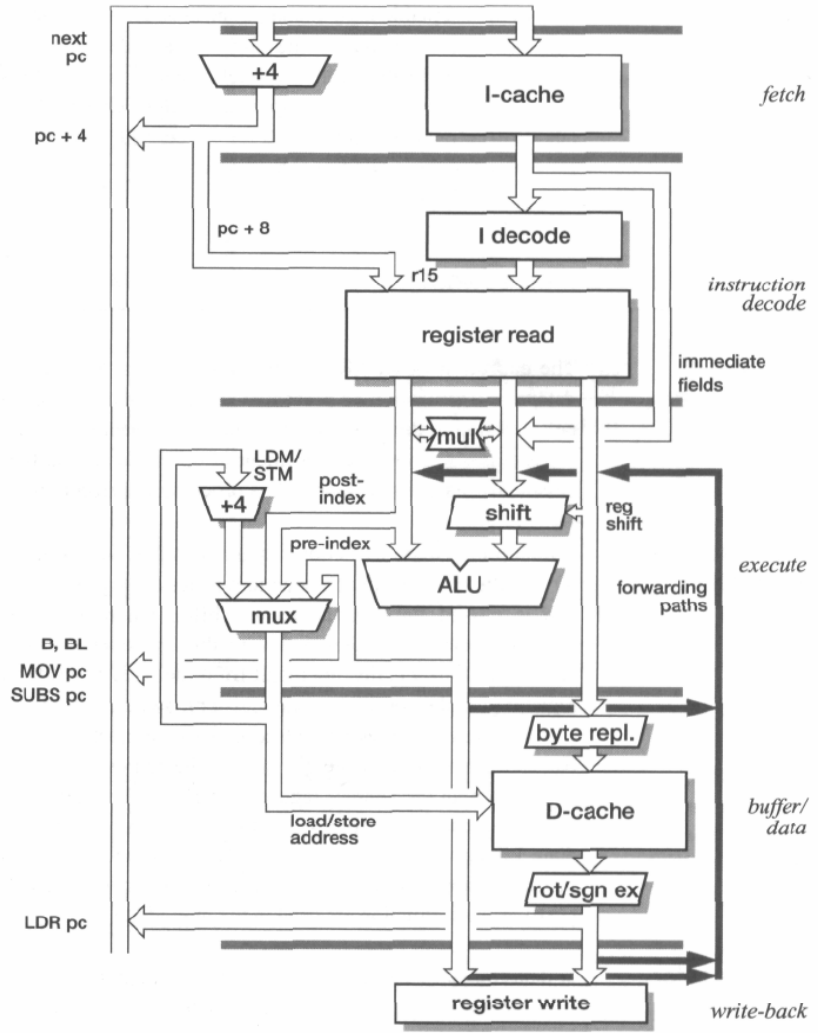
- *Fetch*: igual que en 3 etapas
- *Decode*:
  - La instrucción se decodifica y se leen los operandos desde del banco de registro → 3 puertos
- *Execute*:
  - Operación sobre el *Shifter*, la ALU o, en instrucciones *load/store* se calcula la EA
- *Buffer/Data*:
  - Acceso al dato en memoria. En procesamiento el resultado de la ALU se pasa al siguiente pipeline
- *Write-back*:
  - El dato, leído de memoria o resultante de la ALU, → RF



Organización con 5 etapas de pipeline: Avance de datos

- Dependencia de datos
  - La dependencia de datos aparece cuando una instrucción necesita el resultado de la instrucción previa
- Soluciones
  - *Pipeline stalling*
  - *Data forwarding*
    - El ARM posibilita el avance de datos, de cualquiera de sus tres operandos, desde cualquiera de las tres etapas intermedias de operandos
    - Aún así, existe un caso donde no es posible:

```
LDR     rN, {...}
ADD     r2, r1, rN
```

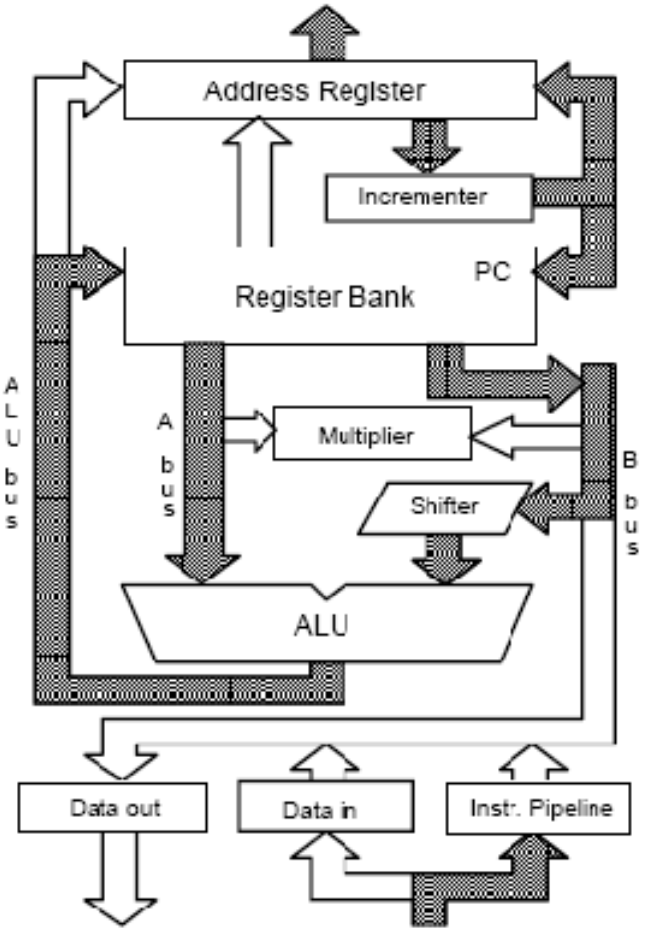


Organización con 5 etapas de pipeline: Comportamiento del PC

- Tratamiento del PC como registro de propósito general
  - La arquitectura de cinco etapas provoca la lectura de los operandos un ciclo antes que en la arquitectura de tres etapas
  - Este comportamiento provoca que el valor de PC leído sea PC+4 en lugar de PC+8, tal y como ocurría en aquella

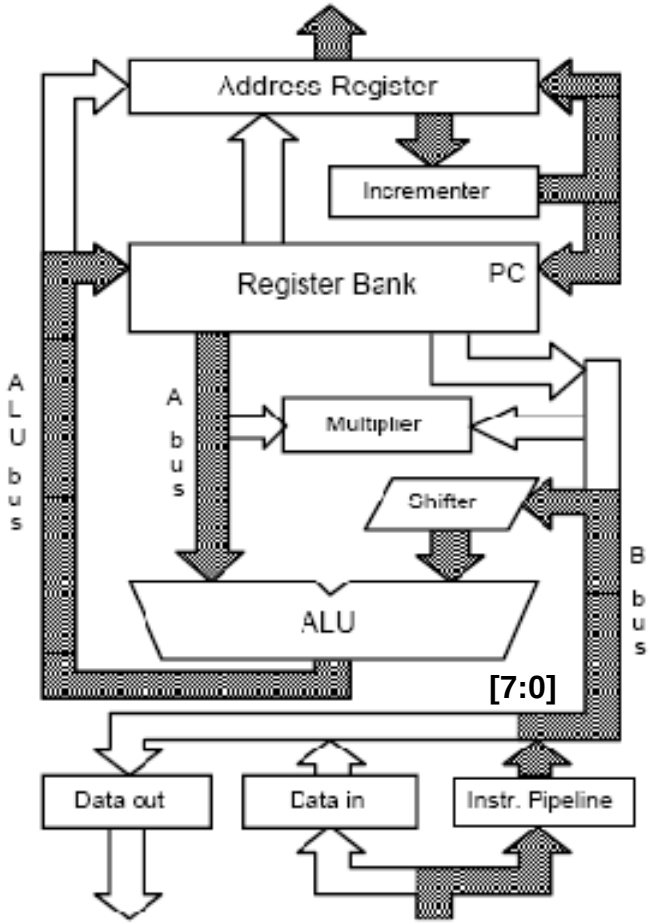
Ejecución de instrucciones: Instrucciones de Procesamiento de Datos I

- Operaciones registro-registro
- Se realiza el acceso a los dos operandos en el banco de registros
  - El operando en el bus B puede ser desplazado e introducido en la ALU en el mismo ciclo
- Al final del ciclo se escribe el resultado en el banco de registro a través del bus de la ALU
- El PC, almacenado en el registro de direcciones, es incrementado y volcado tanto sobre r15 como en el registro de direcciones. Esto permite la lectura consecutiva de la siguiente instrucción



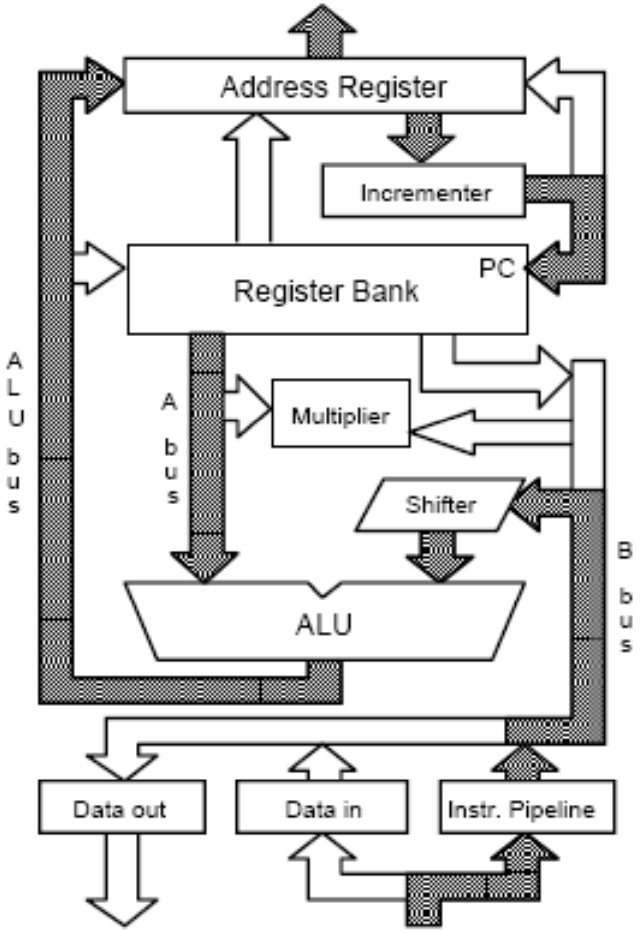
Ejecución de instrucciones: Instrucciones de Procesamiento de Datos II

- Operaciones registro-dato inmediato
- El dato inmediato, de tamaño *byte*, se localiza en el registro de instrucción
- El dato inmediato se deposita en el bus B de la arquitectura
- El dato inmediato se combina, a través de la ALU, con el operando en el bus A
  - El dato inmediato es de tamaño 8 bits y se puede desplazar en el *Shifter*
- Al final del ciclo se almacena el resultado en el banco de registros
- El procesamiento del PC es idéntico



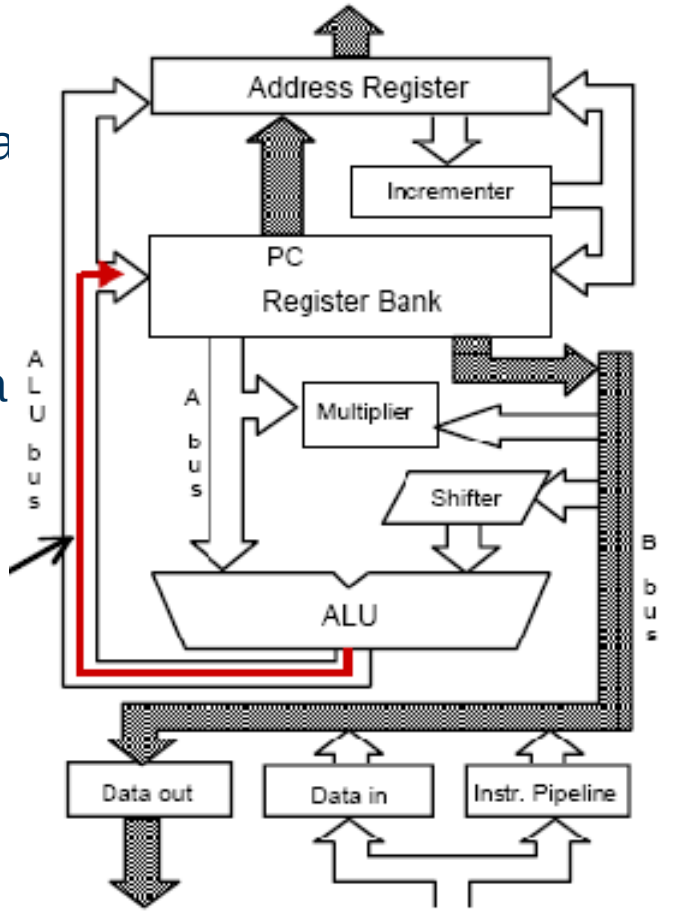
Ejecución de instrucciones: Instrucciones de Transferencia I

- *Load/Store*. Necesitan un ciclo adicional
- Primer ciclo: Cálculo de la EA
  - Partiendo de un registro base, se le suma un desplazamiento que proviene, bien del banco de registro, o bien como operando inmediato (ejemplo de la fig.)
  - El operando inmediato es de tamaño 12 bits y no puede ser desplazado
  - La operación se realiza en la ALU
  - El resultado se almacena en el registro de direcciones al final del ciclo
  - El valor actualizado del PC, contenido en el registro de direcciones, se salva en r15 para poder aceptar la dirección del dato



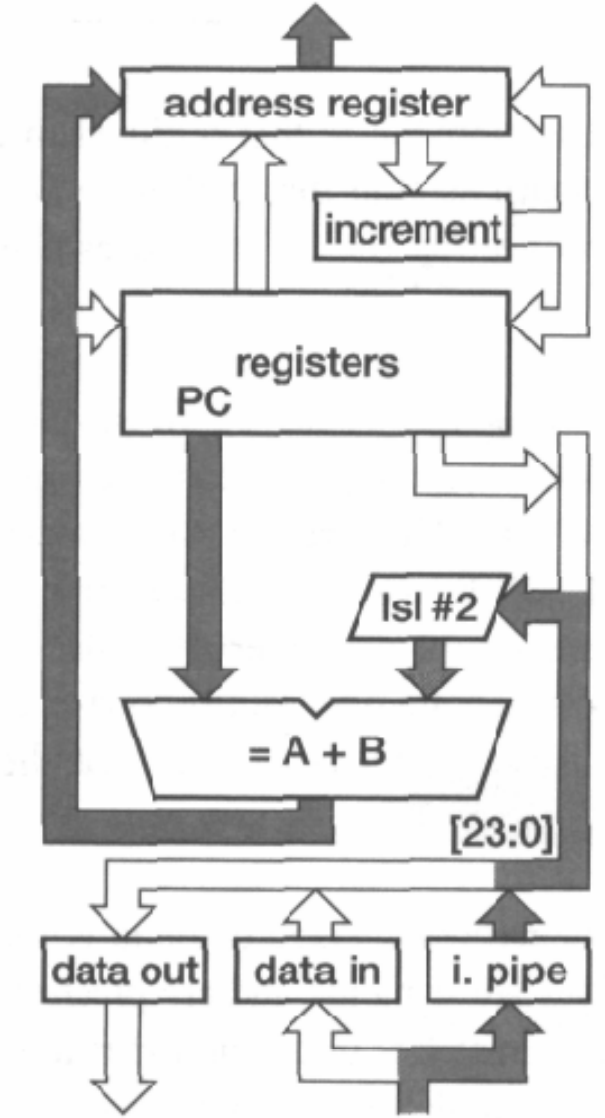
Ejecución de instrucciones: Instrucciones de Transferencia II

- Segundo ciclo: Transferencia de datos
  - El dato a almacenar en memoria es enviado al "bloque" *Data out* a través del bus B y, al mismo tiempo, se envía a la memoria (comportamiento *latch*)
  - Cuando la transferencia es de tamaño *byte/halfword* el bloque *Data out* realiza una copia del mismo en el bus de salida
  - Durante el ciclo de transferencia, el *Datapath* mantiene la dirección lo que posibilita la auto-indexación durante el mismo ciclo
- La instrucción *Load* sigue un procedimiento similar, excepto:
  - Segundo ciclo: dato proveniente de memoria se deposita en *Data in*
  - Necesidad de un tercer ciclo: *Bank register* ← *Data in*



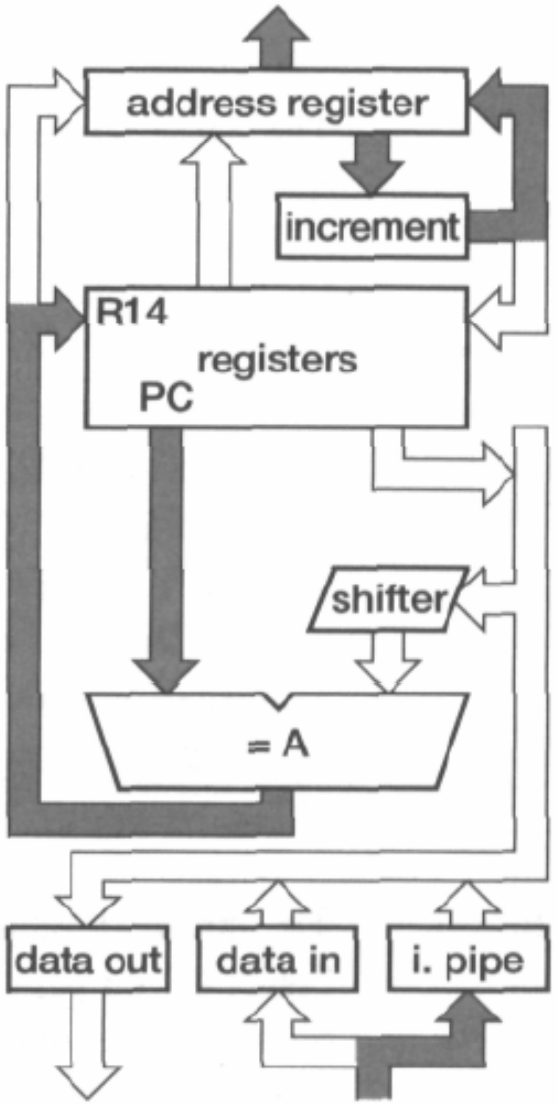
Ejecución de instrucciones: Instrucciones de Control I

- Se ejecutan en tres ciclos
- Primer ciclo: Cálculo de la EA de salto
  - En el primer ciclo se calcula la EA, de la misma manera que en transferencia
  - En este caso, se obtiene un *offset* de 24 bits de la propia instrucción, que es desplazado (LSL) dos bits a la izquierda para alinear a nivel de palabra
  - El resultado es tratado como una dirección de instrucción → *address register*



Ejecución de instrucciones: Instrucciones de Control II

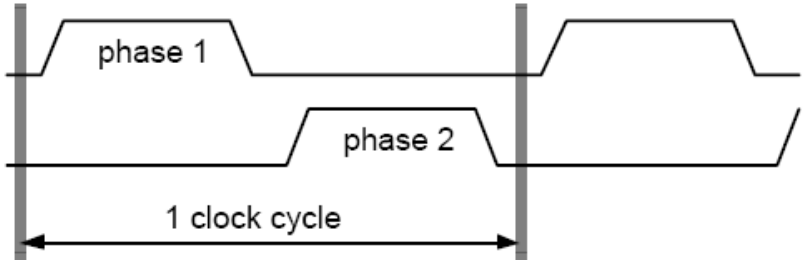
- Segundo ciclo: Salva la dirección de retorno, en caso de *BL*
  - En caso de una instrucción *Branch and Link*, se copia la dirección de retorno en r14
- Tercer ciclo: Corrección de la dirección de retorno
  - Completa la carga del pipeline, realización de un *flush* del pipeline
  - Realiza una corrección en la dirección de retorno almacenada en r14 (PC+8) para ajustarla a la siguiente instrucción (PC+4) después del salto



Implementación del ARM: *Clocking scheme*

Clocking scheme

- El ARM no opera con registros sensibles al flanco. En su caso, utiliza *latches* habilitados por nivel (transparencia)
- Se basa en un esquema de reloj de dos fases no solapadas, obtenidas a partir de una única señal de reloj
- La transferencia de datos se consigue de la siguiente manera:
  - *Phase 1*. Los datos se cargan en los *latches* de entrada durante la fase 1
  - *Phase 2.* Los datos se ponen en la salida durante la fase 2



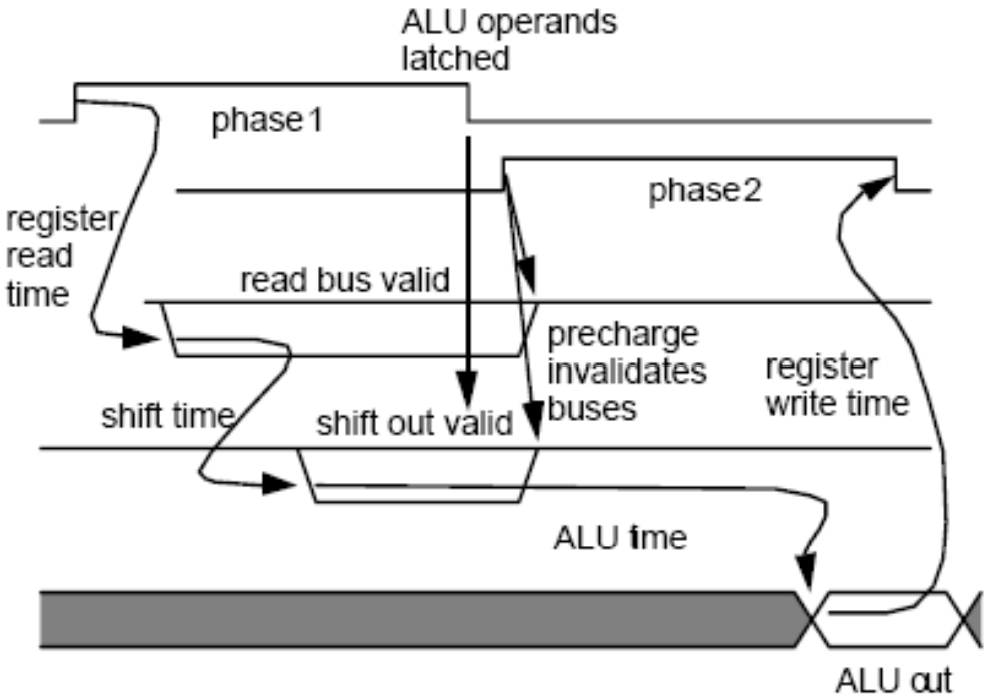
- El no solapamiento de las fases evita la aparición de carrera en los *latches*

Implementación del ARM: *Datapath timing* I

- *Register read*
  - Los buses de lectura del banco de registros se “precargan” durante la fase 2
  - Durante la fase 1 los buses de lectura se descargan
- *Barrel shifter*
  - Indicar que el segundo operando pasa a través de desplazador
- *ALU operation*
  - La ALU dispone de *latches* en su entrada que están abiertos durante la fase 1, lo que permite que la operación se vaya ejecutando tan pronto los operandos sean válidos, y cierran durante la fase 2, no pasando a través de la ALU
  - La ALU continúa procesando durante la fase 2, dando un resultado válido al final de la misma
  - Al final de la fase 2, el resultado es almacenado en el registro destino

Implementación del ARM: *Datapath timing* II

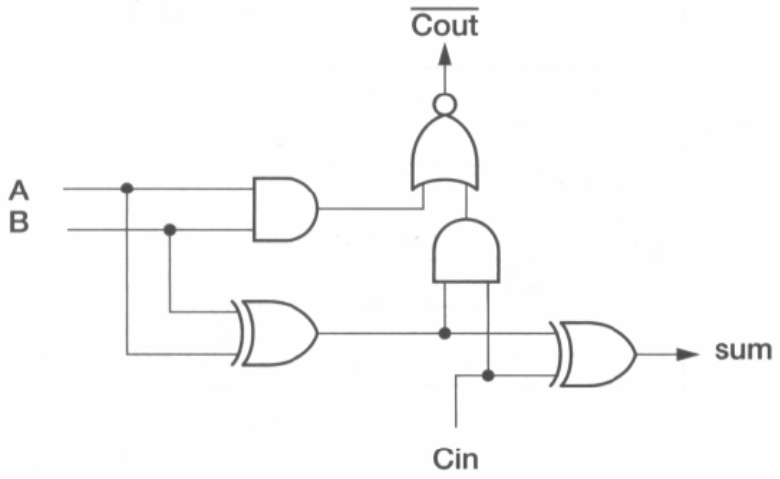
- *Datapath timing* (tres etapas de pipeline)
  - Tiempo mínimo de ciclo de *Datapath*
    - *Register read time*
    - *Shifter delay*
    - *ALU delay*
    - *Register write-setup time*
    - *Non-overlapping time phase*



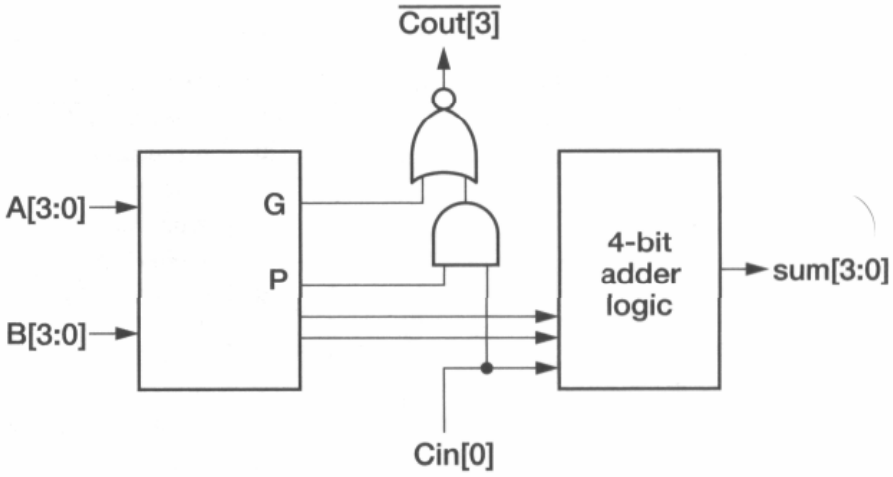
  - Fase crítica: retardo en la ALU

Implementación del ARM: Organización de la ALU I

- ARM1: *Ripple-carry adder*
  - CMOS AOI *gates*
  - Peor caso: retardo de 32 puertas

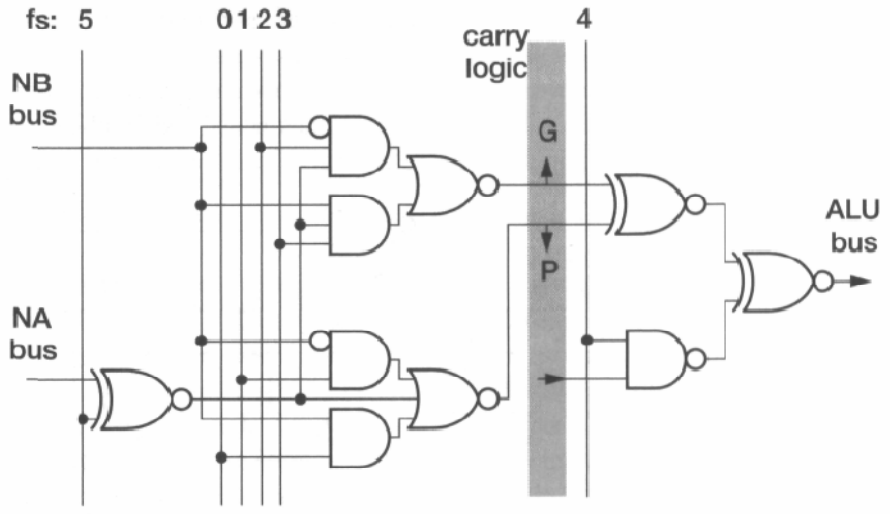


- ARM2: *Carry Look-ahead adder*
  - Se basa en el cálculo de los términos G (*Carry Generate*) y P (*Carry Propagate*)
  - Peor caso: retardo de 8 puert
  - Operaciones:
    - Aritméticas, lógicas
    - Cálculo de la dirección acceso a memoria
    - Cálculo de la dirección salto



Implementación del ARM: Organización de la ALU II

- ARM2: *Carry Look-ahead adder*
  - Implementación de una unidad de bit

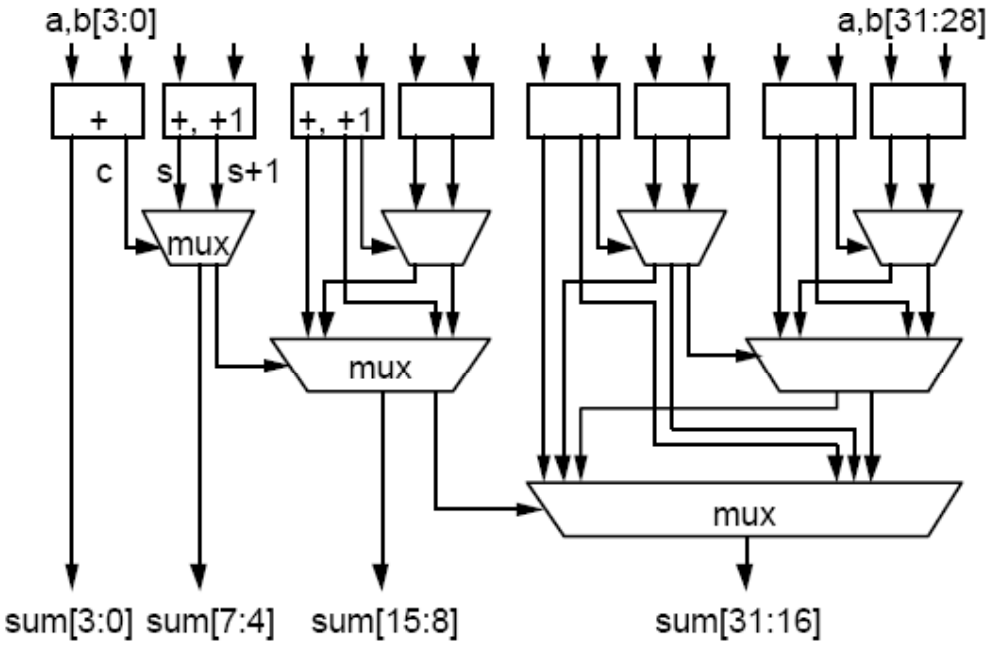


  - Tabla de verdad de códigos de operación

fs5	fs4	fs3	fs2	fs1	fs0	ALU output
0	0	0	1	0	0	A and B
0	0	1	0	0	0	A and not B
0	0	1	0	0	1	A xor B
0	1	1	0	0	1	A plus not B plus carry
0	1	0	1	1	0	A plus B plus carry
1	1	0	1	1	0	not A plus B plus carry
0	0	0	0	0	0	A
0	0	0	0	0	1	A or B
0	0	0	1	0	1	B
0	0	1	0	1	0	not B
0	0	1	1	0	0	zero

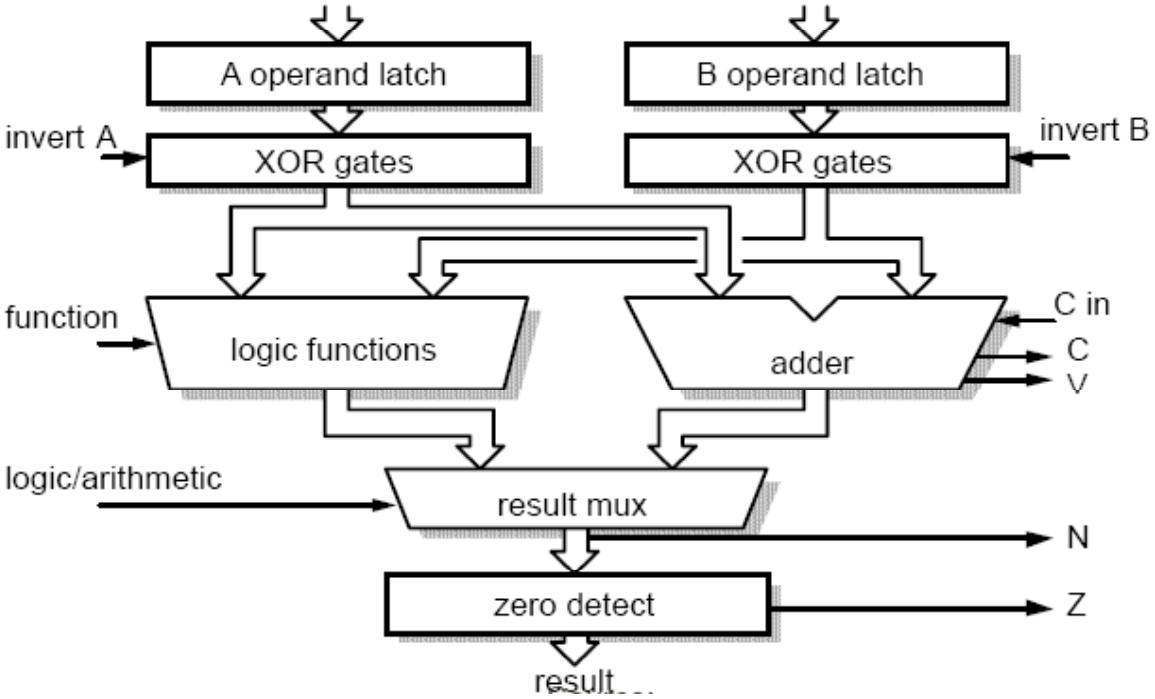
Implementación del ARM: Organización de la ALU III

- ARM6: *Carry Select adder*
  - Utiliza bloque de sumadores *ripple-carry* de 4 bits que calculan en paralelo con acarreo de entrada a zero y a uno
  - El resultado final se selecciona mediante multiplexores dependienc del valor del acarreo en la etapa anterior
  - Peor caso:
    - O(log2[*word width]) gates long*
  - Incremento significativo del área



Implementación del ARM: Organización de la ALU IV

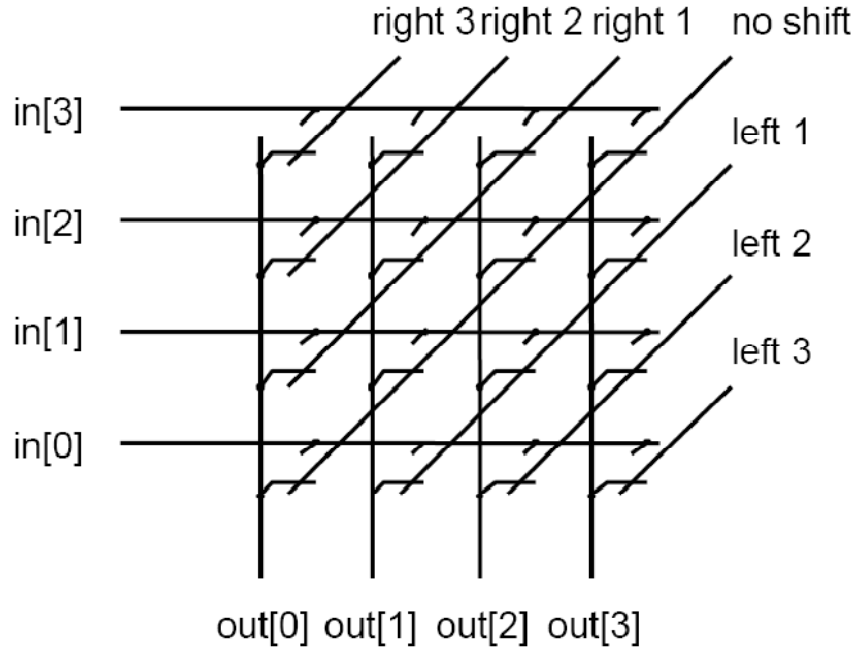
ARM6: *Carry Look-ahead adder*, implementación



- Posibilidad de invertir los operandos antes de su procesamiento
- Dos unidades en paralelo, *lógica* y *aritmética*
- Multiplexación de las salidas

Implementación del ARM: *Barrel Shifter*

- Su retardo contribuye directamente al retardo de la ruta crítica
  - Utilización de un *cross-bar switch*
  - El ARM utiliza una arquitectura de 32x32
  - La conmutación se realiza con transistores NMOS
  - Utiliza lógica de precarga
  - Durante la precarga todas las salidas se ponen a cero y aquellas que no están conectadas a ninguna entrada permanecen a cero
  - La rotación se consigue mediante combinación:
    - Rotación derecha: *right 1 + left 3*



Implementación del ARM: Multiplicador I

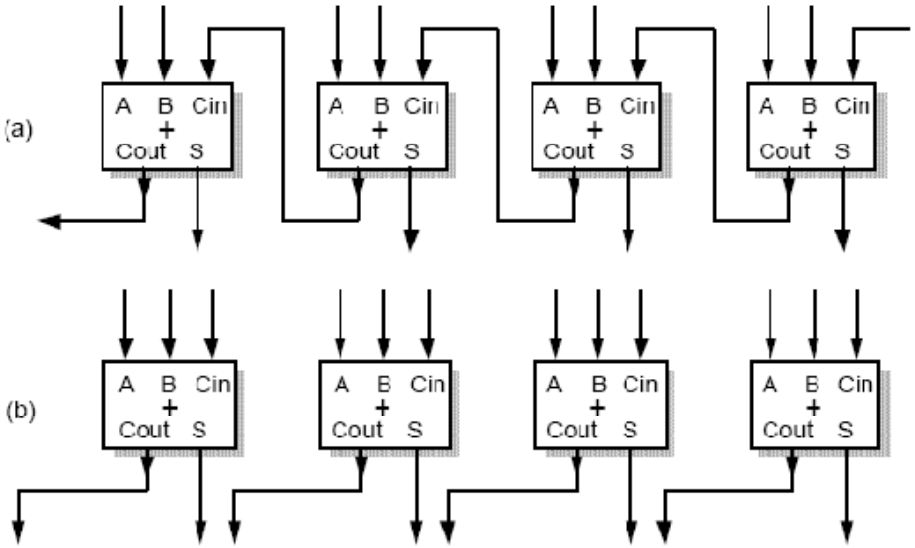
- Uso del algoritmo de *Booth* modificado
  - Todas las posibles combinaciones del multiplicador de dos bits pueden obtenerse mediante desplazamiento y sumas o restas
  - Algoritmo de multiplicación de dos bits

Carry-in	Multiplier	Shift	ALU	Carry-out
0	x 0	LSL #2N	A + 0	0
	x 1	LSL #2N	A + B	0
	x 2	LSL #(2N + 1)	A – B	1
	x 3	LSL #2N	A – B	1
1	x 0	LSL #2N	A + B	0
	x 1	LSL #(2N + 1)	A + B	0
	x 2	LSL #2N	A – B	1
	x 3	LSL #2N	A + 0	1

  - Uso del desplazador para obtener el producto
    - x3 → x(-1) + x4
  - Uso eficiente de recursos hardware

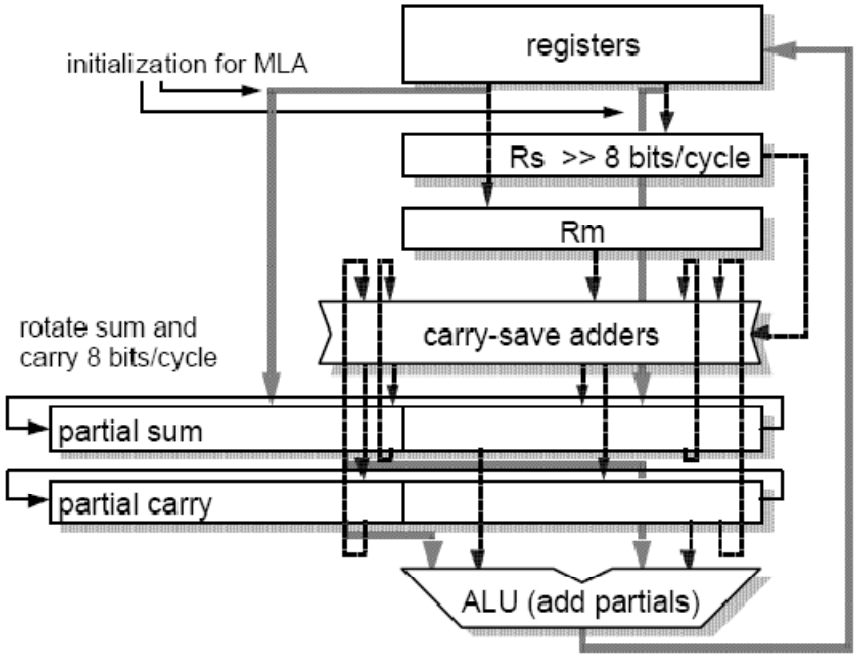
Implementación del ARM: Multiplicador II

- Optimización a nivel de velocidad (*high performance multiplier*)
  - *Redundant binary representation*
    - Necesidad de almacenar sumas y acarreos parciales
  - *Ripple Carry Adder*
    - Entrada: Datos a sumar
    - Salida: Suma binaria
  - *Carry Save Adder*
    - Entrada: *Partial sum, Partial carry, Partial product*
    - Salida: Suma binaria en representación redundante



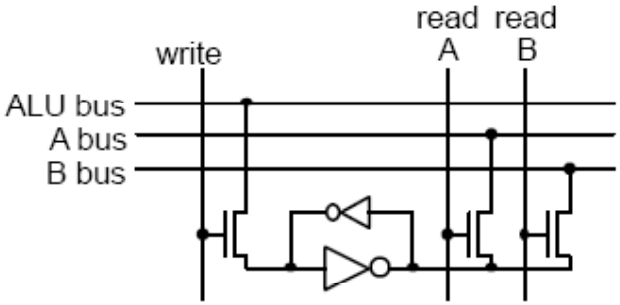
Implementación del ARM: Multiplicador III

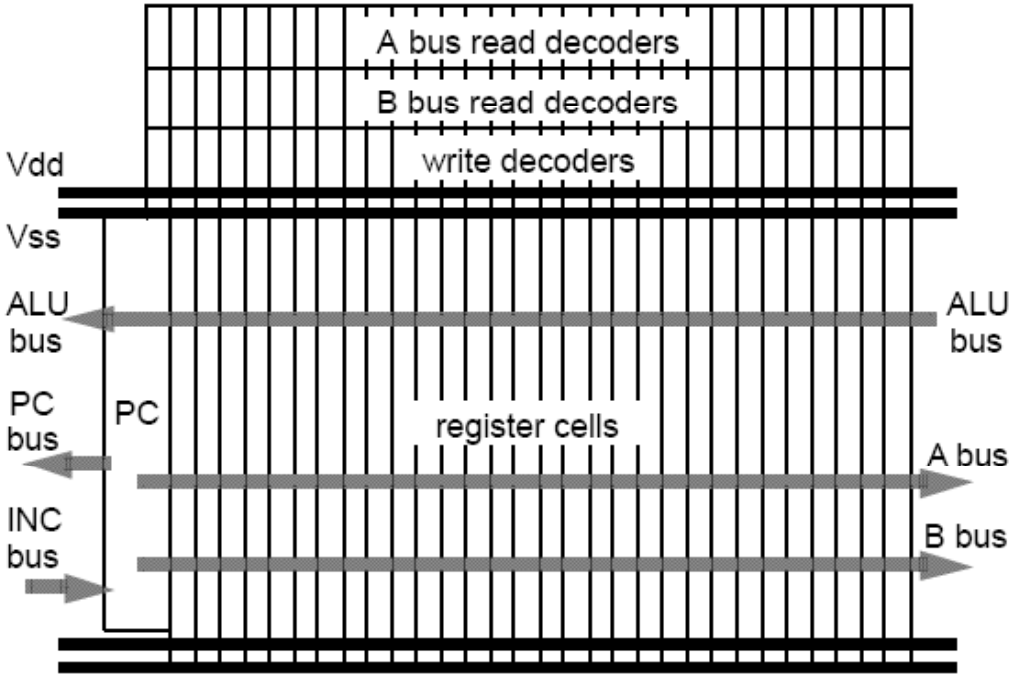
- *High performance multiplier*
  - Evita el uso de sumadores con propagación de acarreo
  - Utiliza una estructura con cuatro etapas. Cada etapa puede realizar el cómputo de 2 bits → 8 bits por ciclo de reloj
  - Las sumas y acarreos parciales se ajustan en una etapa final basada en sumadores
  - Rs → Desplaza a derecha el multiplicador 8 bits por ciclo y los resultados parciales son rotados a derecha
  - El ciclo se ejecuta un máximo de 4 veces (*early termination*)



Implementación del ARM: Banco de registros

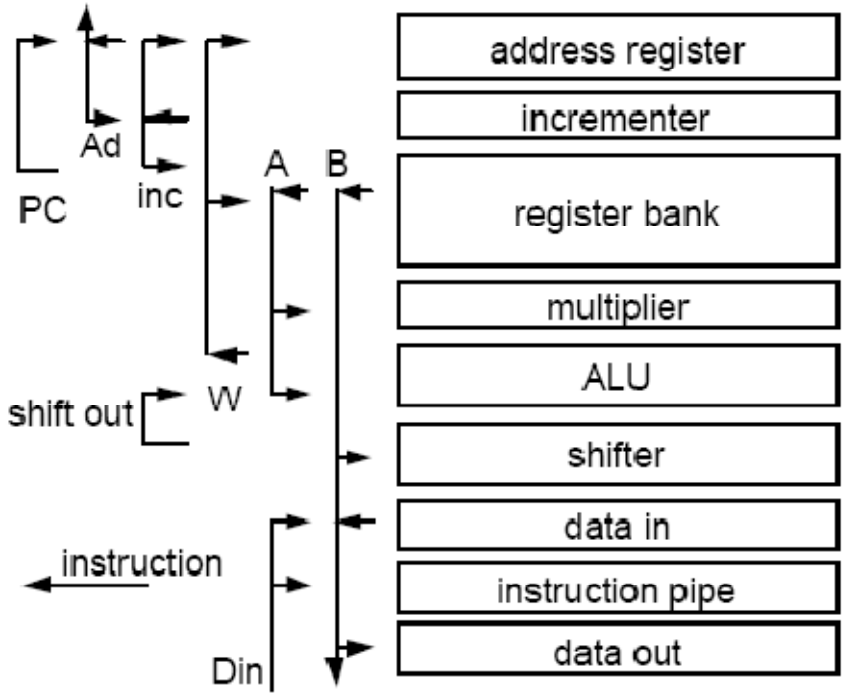
- Contiene 31 registros de propósito general de 32 bits
  - Unidad básica formada por una pareja de inversores CMOS asimétrica
  - *Program Counter*
    - Implementado físicamente en el banco de registros
    - Dispone de dos puertos de escritura y tres de lectura
    - Dispuesto al final de la matriz
  - Alta densidad de transistores





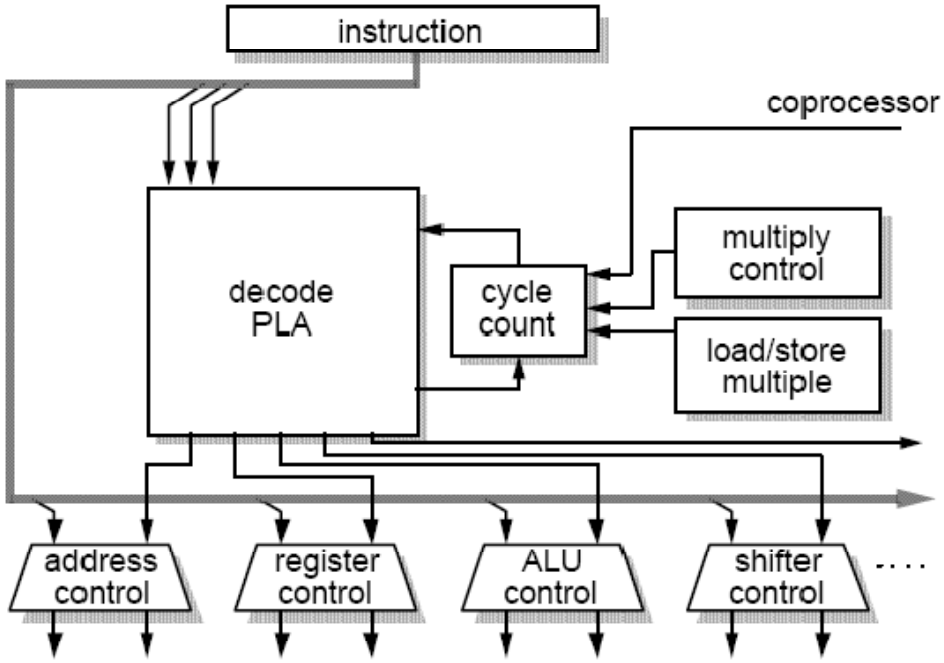
Implementación del ARM: *Datapath layout*

- Útil en el diseño del procesador
- Da una idea de la localización de buses y funciones
- El orden de los bloques se selecciona de tal manera que se disminuya el número de buses que atraviesan las funciones más complejas
  - Localización de *feedtrhough* en el diseño
  - Bus B:
    - atraviesa la ALU si bien no interviene en las funciones implemen-tadas



Implementación del ARM: Estructuras de control

- *Decode Programmable Logic Array*
  - Utiliza parte de la decodificación de la instrucción así como un contador de ciclos. Este último permite definir las señales de control en el siguiente ciclo
- Bloques de lógica distribuida
  - Actúan sobre cada bloque del *Datapath*
- *Decentralized control units*
  - Actúan en instrucciones con múltiples ciclos de ejecución
    - *Load/StoreM*
    - *Mul and Co-pro*



Interfaz de Co-procesador I

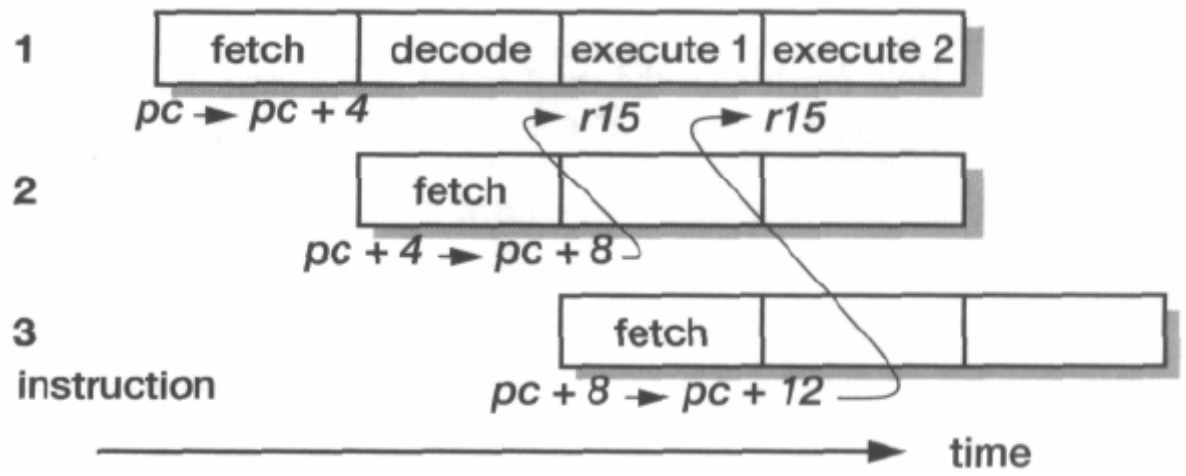
- Extensión del juego de instrucciones mediante
  - Emulación por software mediante instrucciones no definidas
  - *Hardware coprocessors: Floating point* o *ASIP*
- Características
  - Soporta hasta 16 coprocesadores
  - Cada uno con un máximo de 32 registros
  - Arquitectura de *Load/Store*, donde el ARM es quien controla la transferencia de instrucciones
  - El procesamiento de datos es interno
  - En caso de acceso a memoria es el ARM quien genera la dirección y el co-procesador quien controla el flujo de datos
    - El ARM incrementará la dirección en caso de *load/storeM*
    - Longitud máxima de transferencia limitada a 16 palabras
  - Mecanismo de *hadshake*

Interfaz de Co-procesador II

- Señales de comunicación para el ARM7TDMI
  - *cpi** (*Co-Procesor Instruction*, ARM →): El ARM ha identificado una instrucción de co-procesador y va a proceder a ejecutarla
  - *cpa* (*Co-Procesor Absent,* ARM ←): Indica la No Disponibilidad de un co-procesador para la ejecución de la instrucción
  - *cpb* (*Co-Procesor Busy,* ARM ←): Indica que el co-procesador no puede comenzar la ejecución de la instrucción
- Existen cuatro formas de procesar una instrucción de co-procesador, dependiendo de las señales de *handshake:*
  - No ejecutarla: E.g. debido a que no se cumple la condición, *cpi** no se activa
  - Emularla: *cpi** se activa pero no hay co-procesador, *cpa* activa. Se emula mediante instrucción no definida
  - Espera: *cpi**se activa, *cpa* desactivado pero *cpb* está activado. *Pipeline stalling*
  - Ejecutarla: *cpi**, *cpa* y *cpb* están a nivel bajo

Ejemplos I

- Demostración del uso del PC como registro de propósito general
  - En instrucciones con ejecución de un ciclo, el valor que se carga en el PC es de pc+8
  - En instrucciones multiciclos, y debido a que el PC se incrementa en cada ciclo de fetch, el valor en el segundo ciclo de ejecución devuelve PC+12. Al tener un pipeline de tres etapas, éste será el máximo valor que tenga el PC



Ejemplos II

- Dibujar el flujo de ejecución, a nivel de pipeline, del siguiente código

```
LOOP1   LDR     r0, [r1], #4        ; r0 ← word
        STR     r0, [r2], #4        ; Copia palabra
        CMP     r1, r3              ; Condición de finalización
```

- Dar la secuencia completa para realizar la multiplicación de 15 x -3 usando el algoritmo de *Booth*