

Electronica digital 2

Arquitectura pipeline procesador

Ferney Alberto Beltrán Molina



Agosto 2019

Contacto

Nombre: Ferney Alberto Beltrán Molina, Ing, MSc, PhD(c)
Email: fabeltranm@unal.edu.co
oficina:

Contenido

Recordando

Arquitectura Pipeline

Índice

Recordando

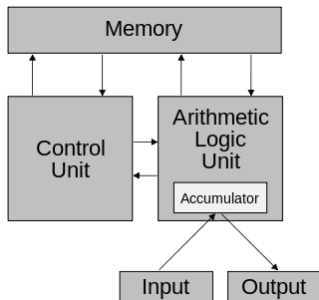
Arquitectura Pipeline

Antes..

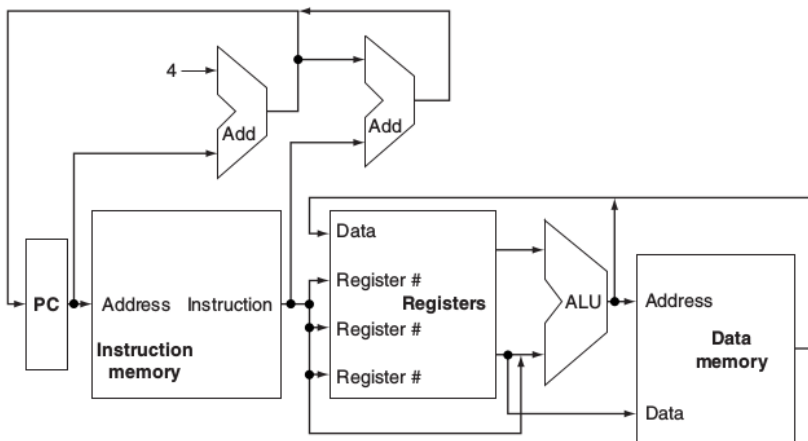
En la clase anterior vimos:

- ▶ Se hablar de la frontera Hw/Sw
- ▶ Se presenta la arquitectura básica del procesador
- ▶ Se presenta los componentes del procesador
- ▶ Introducción a la conexión SoC

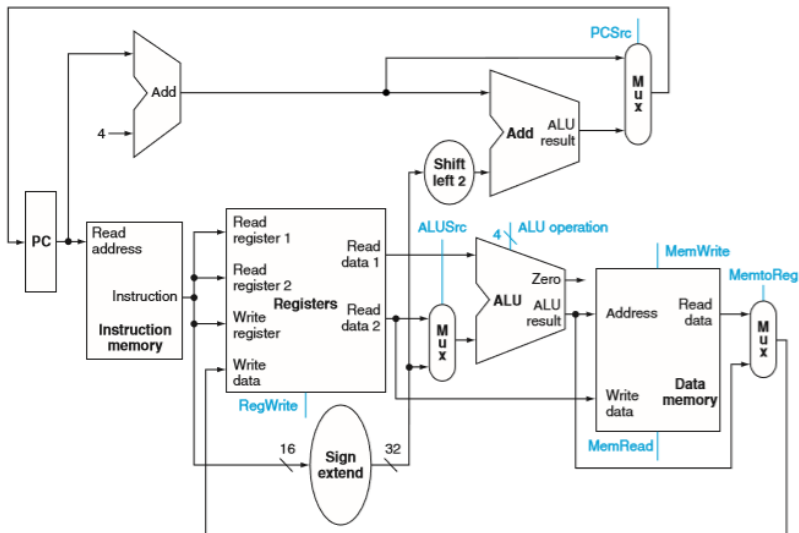
Hardware Software Interface



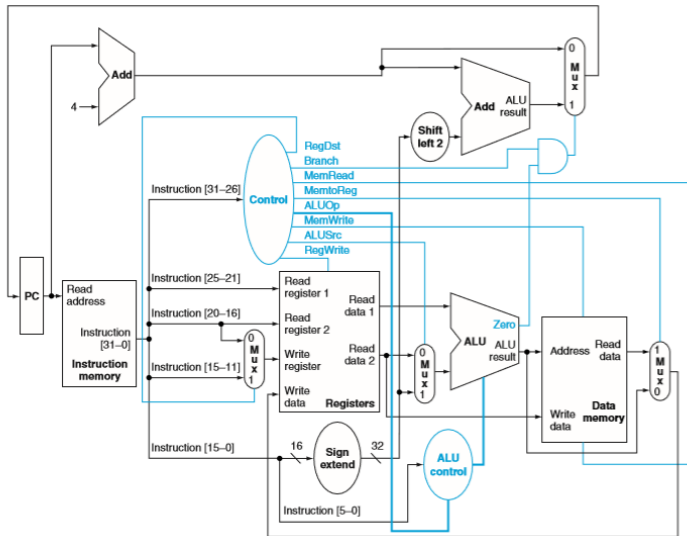
Datapath básico



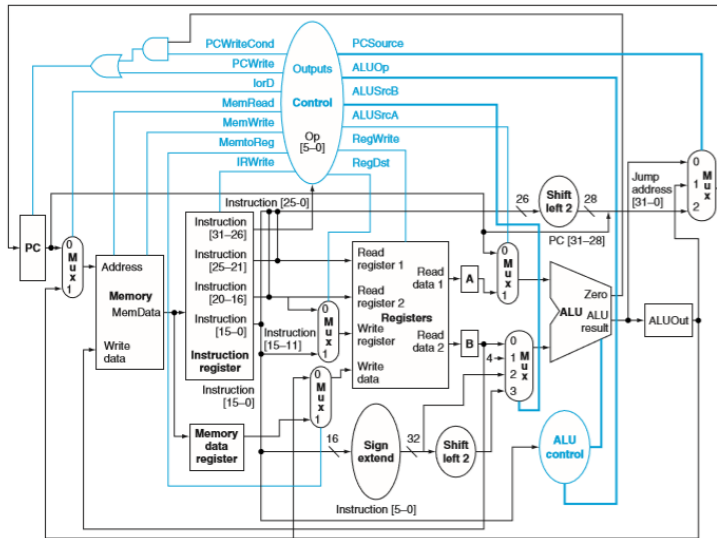
Unidad de control



Procesador



Procesador Multiciclo (Fetch Decode Execute)

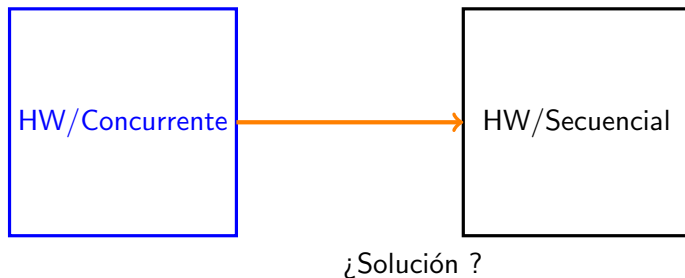


Índice

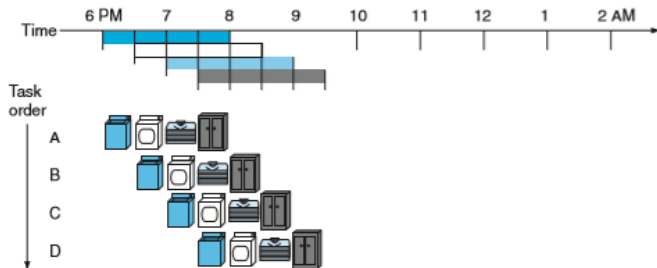
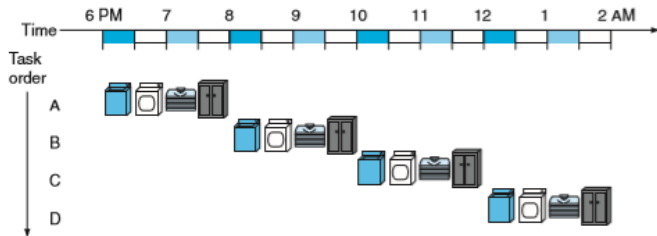
Recordando

Arquitectura Pipeline

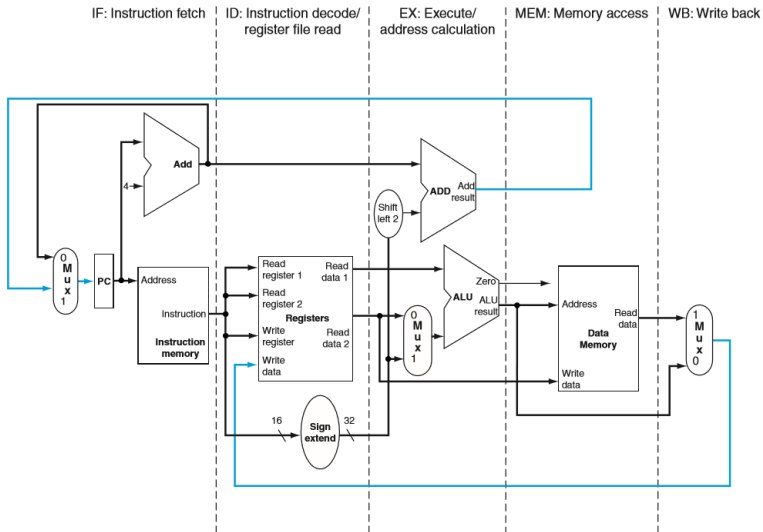
Concurrente a Secuencial



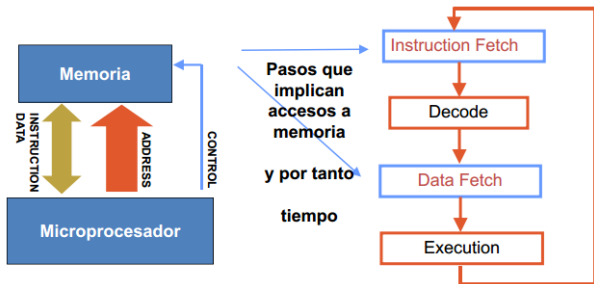
concepto



concepto

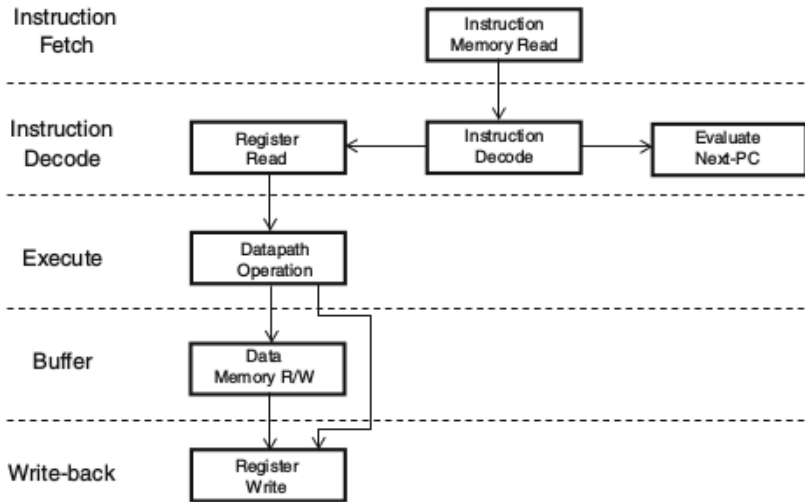


Ciclo de instrucción

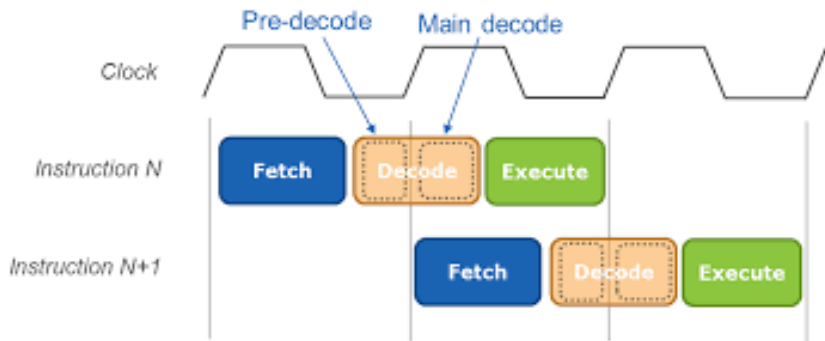


- ▶ Buscar la instrucción en la memoria principal
- ▶ Decodificar la instrucción
- ▶ Ejecutar la instrucción
- ▶ Almacenar o guardar resultados

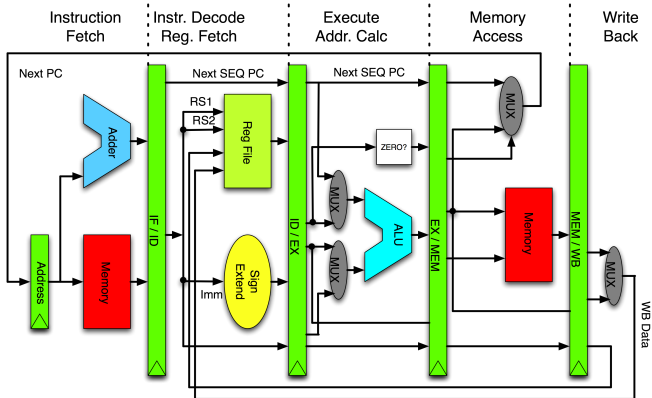
RISC Pipeline



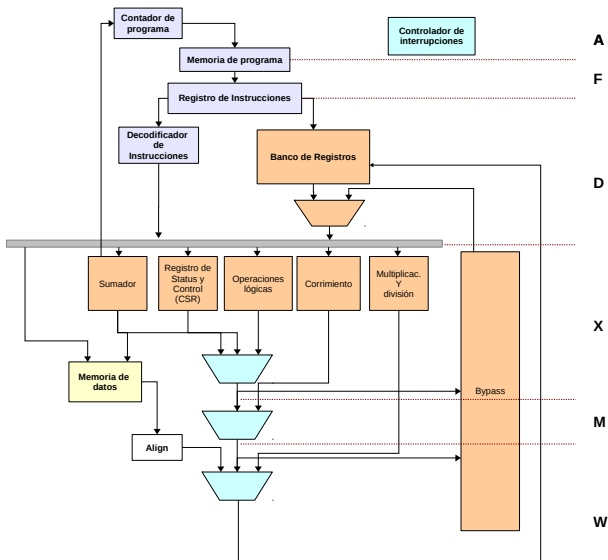
Pipeline Cortex M0 3 Ciclos



MIPS Pipeline 5 ciclos



LM32 Pipeline



Pipeline

- A *Address*: Se calcula la dirección de la instrucción a ser ejecutada y es enviada al registro de instrucciones.
- F *Fetch*: La instrucción se lee de la memoria.
- D *Decode*: Se decodifica la instrucción y se toman los operandos del banco de registros o tomados del bypass.
- X *Execute*: Se realiza la operación especificada por la instrucción. Para instrucciones simples (sumas y operaciones lógicas), la ejecución finaliza en esta etapa, y el resultado se hace disponible para el bypass.
- M *Memory*: Para instrucciones más complejas como acceso a memoria externa, multiplicación, corrimiento, división, es necesaria otra etapa.
- D *Write back*: Los resultados producidos por la instrucción son escritas al banco de registros.

Data Hazards

Program

```
start: mov    r0, #5
        ldr    r1, [r0]
        add    r2, r1, #3
        mov    r3, #0
        nop
```

Cycle	Fetch	Decode	Execute	Buffer	Writeback
0	mov r0, #5				
1	ldr r1, [r0]	mov r0, #5			
2	add r2, r1, #3	ldr r1, [r0]	mov r0, #5		
3	mov r3, #0	add r2, r1, #3	ldr r1, [r0]	mov r0, #5	
4	wait	wait	wait	ldr r1, [r0]	mov r0, #5
5		mov r3, #0	add r2, r1, #3	unused	ldr r1, [r0]
6			mov r3, #0	add r2, r1, #3	unused
7				mov r3, #0	add r2, r1, #3
8					mov r3, #0

Structural Hazards

Program

```
mov    r0, #5
ldmia  r0, {r1,r2}
add    r4, r1, r2
add    r4, r4, r3
```

Cycle	Fetch	Decode	Execute	Buffer	Writeback
0	mov r0, #5				
1	ldmia r0,{r1,r2}	mov r0, #5			
2	add r4,r1,r2	ldmia r0,{r1,r2}	mov r0, #5		
3	wait	wait	ldmia r0,{r1,r2}	mov r0, #5	
4	add r4,r4,r3	add r4,r1,r2	ldmia r0,{r1,r2}	load r1	mov r0, #5
5		add r4,r4,r3	add r4,r1,r2	load r2	update r1
6			add r4,r4,r3	add r4,r1,r2	update r2
7				add r4,r4,r3	add r4,r1,r2
8					add r4,r4,r3

Hazards (Control Hazards)

Program

```
start: mov    r0, #5
      cmp    r0, #5
      ble    TGT
      mov    r0, #0
      nop
TGT:   add    r0, #10
```

Cycle	Fetch	Decode	Execute	Buffer	Writeback
0	cmp r0, #5				
1	ble TGT	cmp r0, #5	<i>interlock</i>		
2	mov r0, #0	ble TGT	cmp r0, #5		
3	TGT: add r0, #10	mov r0, #0		cmp r0, #5	
4		TGT: add r0, #10	unused		cmp r0, #5
5			TGT: add r0, #10	unused	
6				TGT: add r0, #10	unused

Referencias bibliográficas)

1. Computer Organization and Design 5th Edition Patterson
Hennessy