

# Clasificación de Computadores

## Clases de Computadores

Las categorías en las que se puede clasificar son las siguientes:

- Dispositivos móviles personales (PMD).
- Escritorio (*Desktop*).
- Servidores.
- *Clusters*.
- Embebidos (*Embedded*).

Característica	PMD	Escritorio	Servidor	Clusters	Embebido
Precio	\$100-\$1000	\$300-\$2500	\$5000-\$10.000.000	\$100.000-\$200.000.000	\$10-\$100.000
Precio μ	\$10-\$100	\$50-\$500	\$200-\$2000	\$50-\$250	\$0,01-\$100
Propósito	Energía, Tamaño	Precio-Rendimiento	Escalabilidad	Rendimiento	Específico

# Clasificación de Computadores

## Otras clasificaciones

Según generación:

- Primera generación (1946-1959), basado en tubos de vacío.
- Segunda generación (1959-1965), basado en transistores.
- Tercera generación (1965-1971), circuitos integrados.
- Cuarta generación (1971-1980), VLSI (20,000 transistores a 1,000,000).
- Quinta generación (1980-presente), ULSI (más de un millón de transistores).

Según propósito:

- General.
- Específico.

Según procesamiento de datos:

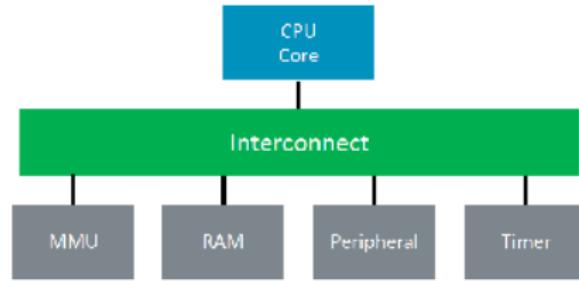
- Analógico.
- Digital.
- Híbrido.

# Clasificación de Computadores

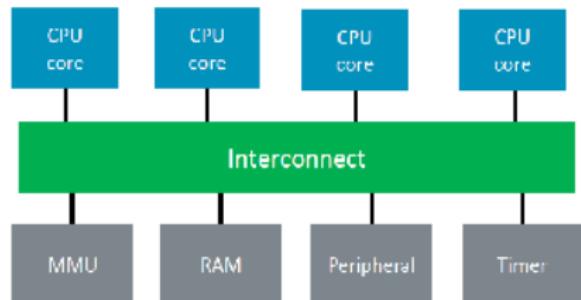
## Clasificación por Paralelismo

- Paralelismo a nivel de bit (BLP): Hacer el bus más grande o ancho.
- Paralelismo a nivel de Instrucción (ILP): Pipeline, VLIW, Superescalar, OoOE.
- Paralelismo a nivel de Hilo (TLP): Simultaneous Multithreading Processor (SMT).
- Paralelismo a nivel de datos: Arquitecturas vectoriales y GPUs.

La Era del *SingleCore*



Clasificación por Paralelismo



# ¿Arquitectura y Microarquitectura son lo mismo?

Arquitectura de un computador

La arquitectura de un procesador corresponde al Set de Instrucciones (ISA) que puede ejecutar dicho procesador.

**Arquitectura ⇒ Software**

Responde a la pregunta: **¿Qué hace/ejecuta/tiene el hardware?**.

Componentes de un ISA:

- Clase de ISA: *Register-Memory* o *Load-Store*.
- Direccionamiento de memoria: Endianness, alineamiento.
- Métodos de direccionamiento.
- Tipos y tamaños de operandos.
- Operaciones.
- Control de flujo.
- Encodificación.
- Costo (área, ley de Moore).
- Simplicidad: De diseño y verificación.
- Desempeño.
- Escalabilidad.
- Tamaño (*Memory footprint* ).
- Facilidad de programación.
- Seguridad.

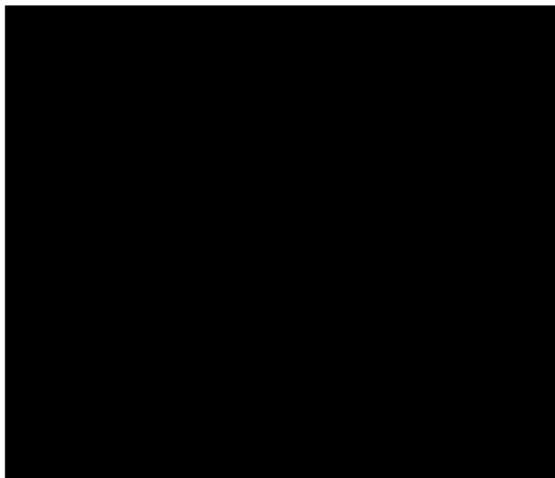
# ¿Arquitectura y Microarquitectura son lo mismo?

## Microarquitectura de un computador

La microarquitectura son los detalles de interconexión, implementación y optimización de una arquitectura. Aspectos de alto nivel de la implementación de un computador. También se conoce como organización.

**Microarquitectura ⇒ Hardware**

Responde a la pregunta: **¿Cómo hace/ejecuta/implementa el hardware?**



# Paralelismo, Arquitecturas

## Paralelas y Flynn

### Parallel Architectures by Flynn

- “...Parallel or concurrent operation has **many different forms** within a computer system...”
- “...A stream is a sequence of objects such as **data**, or of actions such as **instructions**. **Each stream is independent** of all other streams, and each element of a stream **can consist of one or more objects or actions**...”

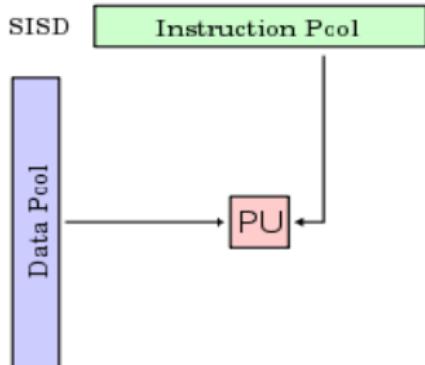
Las arquitecturas más comunes según la cantidad de *streams* son:

- SISD.
- SIMD.
- MISD.
- MIMD.

# Paralelismo, Arquitecturas Paralelas y Flynn

# SISD

- Significa *Single Instruction Single Data*.
- Arquitectura tradicional de un único procesador.
- Utiliza *pipelining*, realizando concurrentemente diferentes fases de procesamiento de una instrucción.
- Implementa *instruction level parallelism* (ILP) como superescalar o *very long instruction word* (VLIW).
- No se obtiene concurrencia de ejecución, pero si de procesos.



# Paralelismo, Arquitecturas Paralelas y Flynn

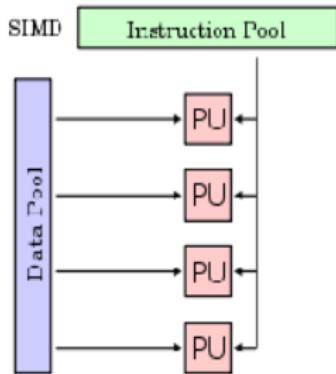
## SIMD

- Significa *Single Instruction Multiple Data*.
- Incluye procesadores de arreglo (array) y vectoriales.
  - Procesadores de arreglo: Instrucciones operan en múltiples elementos al mismo tiempo. Se conocen como *massively parallel processor*.
  - Procesadores vectoriales: Instrucciones operan en múltiples elementos en tiempos consecutivos.

### SIMD (Tiempo vs Espacio)

LD0	LD1	LD2	LD3
AD0	AD1	AD2	AD3
MU0	MU1	MU2	MU3
ST0	ST1	ST2	ST3

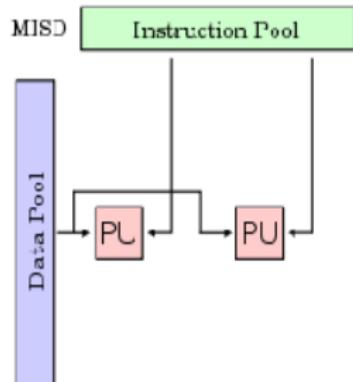
LD0			
LD1	AD0		
LD2	AD1	MU1	
LD3	AD2	MU2	ST0
	AD3	MU3	ST1
		MU4	ST2
			ST3



# Paralelismo, Arquitecturas Paralelas y Flynn

MISD

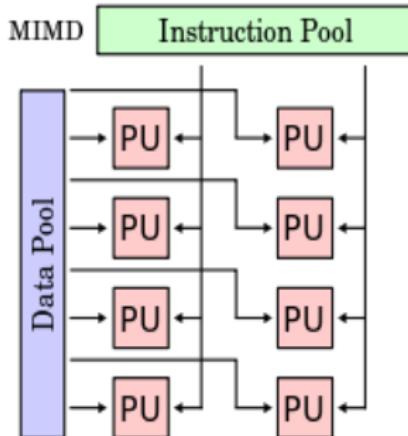
- Significa *Multiple Instruction, Single Data*.
- Se usa en sistemas aeroespaciales y arreglos sistólicos.
- También se puede usar para detectar y enmascarar errores



# Paralelismo, Arquitecturas Paralelas y Flynn

MIMD

- Significa *MultipleInstruction, MultipleData*.
- No necesariamente todos los procesadores deben ser idénticos, cada uno opera independientemente.
- Son: procesadores multinúcleo y superescalares.
- Cuando usan memoria compartida en este tipo de sistemas hay dos problemas:
  - Consistencia de memoria (se resuelve a través de combinación de técnicas de hardware y software).
  - Mantener la coherencia de caché (se resuelve mediante técnicas de hardware).



Según Flynn cómo se catalogan:

$$[f(x), g(y), h(z)] = \left[ \frac{x+1}{2}, \frac{\sin y}{y}, e^z \right] \quad (1)$$

$$[h(x, y)] = [e^{x+y}] \quad (2)$$

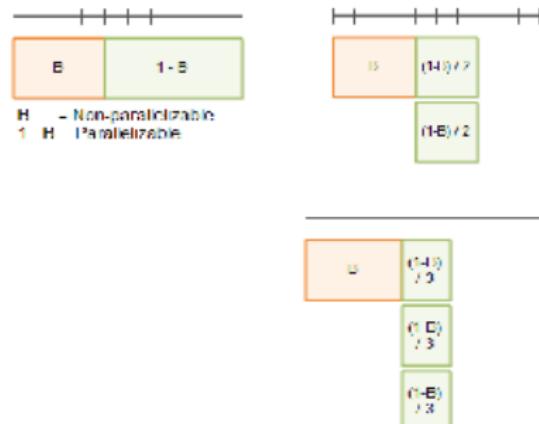
$$[f(x)] = a_0 + x(a_1 + x(a_2 + a_3 x)) = a_0 + a_1 x^2 + a_2 x^3 \quad (3)$$

$$[g(x, y, z)] = x^{a_0} + y^{a_1} + z^{a_2} \quad (4)$$

# Ley de Amdahl

¿Qué es?

- Permite obtener la ganancia en el desempeño debido a la mejora en una característica determinada.
- La ley de Amdahl puede servir como una guía para determinar la mejora real y como distribuir los recursos para tener mejor relación costo-desempeño.
- Gene Amdahl establece que todo programa se divide en:
  - Partes paralelizable.
  - Partes **no** paralelizables.



¿Qué es?

$$\frac{T-B}{N}$$

Donde:

$$T(N) = B + \frac{(T-B)}{N}$$

- $B$ : es la parte no paralelizable.

- $T$ : es el tiempo de duración de una tarea.

## Ley de Amdahl

La fracción paralelizable está dada por un factor  $N$

### Speedup

$$\text{Speedup} = \frac{\text{Tiempo de ejecución de una tarea sin mejora}}{\text{Tiempo de ejecución de una tarea con mejora}}$$

$$\text{Speedup} = \frac{T(1)}{T(N)} = \frac{T(1)}{B + \frac{(T(1)-B)}{N}}$$

Con  $T(1) = 1$  (sin mejora):

### Speedup Overall

$$\text{Speedup} = \frac{1}{B + \frac{(1-B)}{N}}$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1-\text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Donde

- $\text{Fraction}_{\text{enhanced}}$  es la fracción del tiempo de computación original que se puede ver beneficiado por la mejora.
- $\text{Speedup}_{\text{enhanced}}$  es la ganancia del producto de la ejecución en modo "mejorado". Esto es, qué tan rápido ejecutaría la tarea si la mejora se aplicara a todo el programa.

## Ley de Amdahl

### Ejemplo: Ley de Amdahl

#### Ejemplo

Supongamos que se desea mejorar un procesador utilizado para un servidor Web. El nuevo procesador es **10 veces más rápido** en tiempo de computación para la aplicación de servidor Web que el procesador antiguo. Asumiendo que el procesador original está **ocupado** un **40 %** del tiempo y el **60 %** del tiempo **esperando** por dispositivos de Entrada/Salida. ¿Cuál es la ganancia total producto de incorporar la mejora?

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

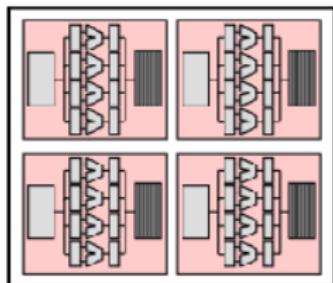
$$\text{Fraction}_{\text{enhanced}} = 40\% = 0,4 \quad \text{Speedup}_{\text{enhanced}} = 10$$

$$\text{Speedup}_{\text{overall}} = 1,56$$

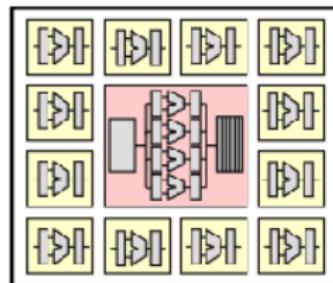
# Ley de Amdahl

Amdahl con unidades de procesamiento simétricas y no-simétricas

- La paralelización es uniformemente distribuida en las unidades de ejecución (procesadores).
- No aplica en sistemas heterogéneos y *multicore* (simétricos y no simétricos).
- Supone que no hay conflictos de recursos.



**Symmetric: Four 4-BCE cores**



**Asymmetric: One 4-BCE core  
& Twelve 1-BCE base cores**

## Confiabilidad

Probabilidad de que el sistema esté funcionando en el instante  $t$  dado que funcionaba en el instante  $t_0$ . Se tiene:

- Tiempo medio para una falla (MTTF).
- Fallos por tiempo ( $\lambda$ ):  $\lambda = \frac{1}{\text{MTFF}}$

$R(t)$ : probabilidad de que un sistema falle en  $t$  unidades de tiempo después de la última falla.

$$R(t) = e^{-\lambda(t-t_0)}$$

## Disponibilidad

Es el porcentaje del tiempo en el que el sistema estará disponible para brindar un servicio correcto.

Tiempo medio entre fallas (MTBF):  $\text{MTTF} + \text{MTTR}$

$$A = \frac{\text{MTTF}}{\text{MTBF}}$$

## Mantenibilidad

Tiempo requerido para que el sistema este funcionando luego de que se dio una falla.

- Tiempo medio para reparar (MTTR): Tiempo de la interrupción del servicio.
- Tasa de reparación ( $\mu$ ):  $\mu = \frac{1}{\text{MTTR}}$

$M(t)$ : probabilidad de que un sistema este funcionando en  $t$  unidades de tiempo después de que se presentó una falla.

$$M(t) = e^{-\mu t}$$



## Desempeño

## Benchmarking

El benchmark es un instrumento para comparar el desempeño de varios sistemas en aplicaciones reales.

Representa un recurso de software para evaluar un sistema y discriminar la mejor opción para el diseño.

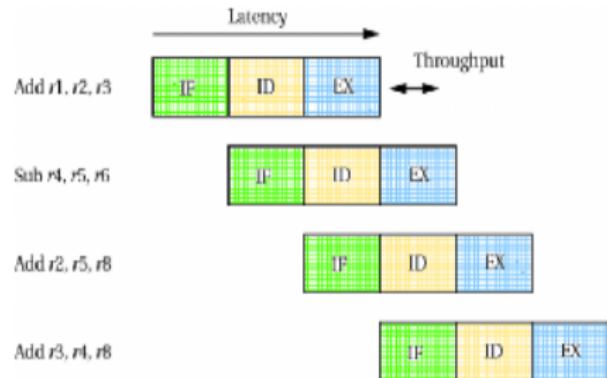
Varios tipos de benchmarks: SPEC, EEMBC, BDTi, Drystone, CoreMark.

Término aplica de manera diferente según el campo:

- ISP: Calidad de imagen.
- Memorias: Accesos a memoria por segundo.
- CPU's: Medida de la tasa en que los programas son ejecutados (IPC, CPI).

Punto de vista microscópico:

- Latencia: Tiempo requerido para ejecutar una instrucción desde inicio hasta finalización.
- Flujo de instrucciones (throughput): Tasa de finalización de instrucciones.



# Formas de Organización/Microarquitectura

- Arquitectura Von Neumann: Un único espacio de direccionamiento, y única ruta de acceso al CPU.
- Arquitectura Harvard: Memoria de datos y memoria de instrucciones tienen rutas de hardware diferentes hacia el CPU, además de espacios de direccionamiento separados.
- Arquitectura Harvard Modificada: rutas de hardware diferentes para el CPU Cache, y un espacio de direccionamiento único.

## Clasificación de los ISA

Otros ISAs

- ISA ortogonal: El código de operación y el operando son independientes.
- Cualquier instrucción puede usar cualquier operando.

Longitud fija vs longitud variable:

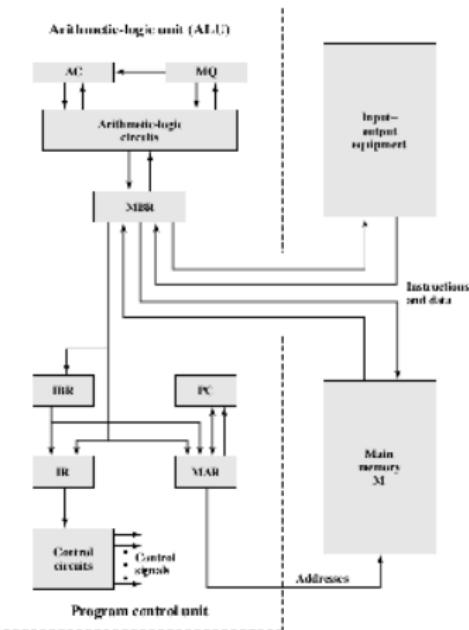
- Longitud fija: *Fetching y decoding* por hardware es rápido.
- Longitud variable: *Fetching y decoding* por hardware es lento.

# Arquitectura Von Neumann: Características VIEJA

- La información se representa por medio de direcciones.
- **Memoria unificada**, una única memoria para datos y programa.
- Las instrucciones almacenadas y ejecutadas secuencialmente: Program counter debe actualizarse ( $PC=PC+1$ ) para obtener siguiente instrucción.
- Ciclo de Fetch: Búsqueda, Decodificación, Ejecución, Almacenado.
- Cuenta con un ISA de 21 instrucciones.

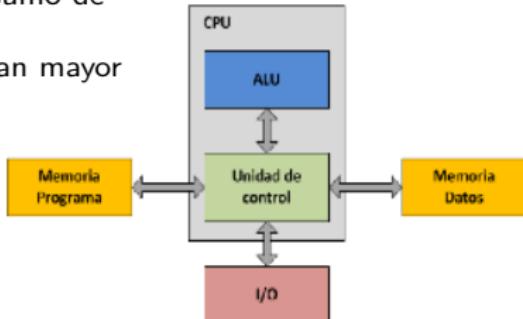
En la versión original se listaron las siguientes partes:

- *Central Arithmetic* (CA): Unidad encargada de llevar a cabo las operaciones aritméticas de suma, resta, multiplicación y división.
- *Central Control* (CC): Lógica de control del computador, encargado de llevar la secuencia correcta del programa.
- Memoria (M): Almacena largas cantidades de operaciones (programa). Se ejecuta secuencialmente.
- *I/O Equipment* (I,O): Periféricos del sistema.



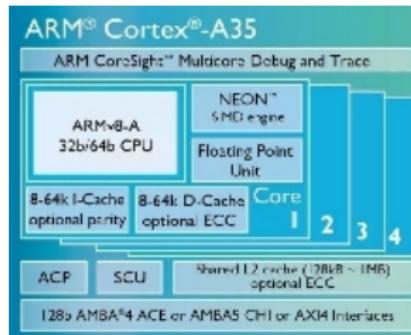
## Arquitectura Harvard: Características

- Memoria de datos y memoria de programa están **físicamente** separadas.
- Acceso a memoria de datos e instrucciones puede ser simultáneo.
- Ventaja: mayor rendimiento (paralelismo).
- La mayoría de DSP poseen una arquitectura tipo Harvard pues necesitan buscar datos y operaciones al mismo tiempo.
  
- Desventajas de la arquitectura de Harvard:
  - El espacio de direccionamiento separado implica 2 memorias físicas diferentes: mayor espacio, consumo de potencia.
  - Rutas diferentes (mayor ancho de banda) generan mayor consumo de potencia dinámica.



# Arquitectura Harvard Modificada

- Desventajas de la arquitectura de Harvard:
  - El espacio de direccionamiento separado implica 2 memorias físicas diferentes: mayor espacio, consumo de potencia.
  - Rutas diferentes (mayor ancho de banda) generan mayor consumo de potencia dinámica.
- Arquitectura de Harvard Modificada:
  - Disminuye el impacto de la separación de memoria.
  - Rutas separadas para instrucciones y datos, con único espacio de direccionamiento.
  - Provee instrucciones para acceder a los contenidos de la memoria de instrucciones como si fueran datos.
  - Una única memoria principal.
    - Utiliza dos memorias caché de CPU, para la separación de datos e instrucciones.
    - Una única memoria principal.
    - Desde el punto de vista macro se comporta como una arquitectura Von Neumann, pero internamente separa instrucciones y datos.
    - ¿Dónde está lo complicado?



La mayoría de las arquitecturas modernas-Harvard son en realidad Harvard Modificada.

# Clasificación de los ISA

Tipos de operando que existen:

Tipo de operando

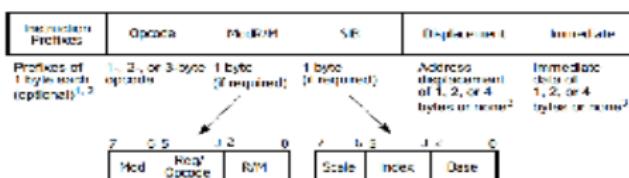
Load/Store → Divide las operaciones en dos categorías:

- Accesos a memoria (instrucciones: Load/Store en memorias y registros).
- Operaciones con ALU (solo entre registros).

Ejemplos: ARM, RISC-V, MIPS.

Tipo de operando: Register/Mem

Operaciones pueden ser entre registros y entre espacios de memoria.



1. The ISX prefix is optional, but it used must be immediately before the opcode; see Section 2.1, "ISX Prefixes" for additional information.

2. For VEX encoding information, see Section 2.3, "Intel® Advanced Vector Extensions (Intel® AVX)".

3. Some new instructions can take an 8-bit immediate or 24-bit displacement.

Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

# Clasificación de los ISA

## Complejidad de instrucciones: CISC

### Complex Instruction Set Computer:

- Enfoque inicial de arquitectura.
- El ISA contiene gran variedad de instrucciones: instrucciones poderosas y complicadas.
- Facilidad de programación.
- El compilador no realiza traducciones complejas: instrucciones son similares a lenguajes de alto nivel.

Ejemplo: x86.

#### Características típicas del set:

- Múltiples modos de direccionamiento (forma de acceder a datos en memoria).
- Formato de instrucciones variable.
- Duración de instrucciones variable.
- Bajo número de registros de propósito general. x86: RAX, RBX, RCX, RDX.
- Las instrucciones son capaces de ejecutar tareas complejas.
- Decodificación de instrucciones implica mayor lógica de hardware.

- Ventajas:
  - Facilidad de programación: Tareas complejas se realizan en menos tiempo.
  - Múltiples modos de direccionamiento simplifican las tareas.
  - Tamaño de código pequeño.
- Desventajas:
  - Instrucciones de tamaño variable: diferente tiempo de búsqueda y ejecución hacen muy complicado tener sistemas determinísticos. Hardware es complicado (área, dinero).
  - Muchas de las instrucciones especializadas no son utilizadas con frecuencia: El 98 % de las instrucciones en un programa típico corresponden al 20 % de las instrucciones del set.

# Clasificación de los ISA

## Complejidad de instrucciones: RISC

### Reduced Instruction Set Computer:

- Enfoque moderno: DSPs, CPUs para sistemas embebidos.
- El set está compuesto por pocas instrucciones con funcionalidad simple.
- La dificultad está en el programador (bajo nivel) o el compilador.

Ejemplos: MIPS, ARM.

### Reduced Instruction Set Computer: Características típicas del set:

- Ventajas:
  - Instrucciones de tiempo y tamaño fijo: simplifica hardware y brinda determinismo.
  - Mejor aprovechamiento de hardware.
  - Permite pipeline.
- Desventajas:
  - Tamaño de código mayor.
  - Carga pesada para el software (programa de bajo nivel o compilador).

- Pocos modos de direccionamiento (1-4).
- Las instrucciones tienen un tamaño fijo.
- El tiempo de ejecución de cada instrucción es el mismo.
- Alto número de registros de propósito general (16, +32).
- Decodificación de instrucciones simple.

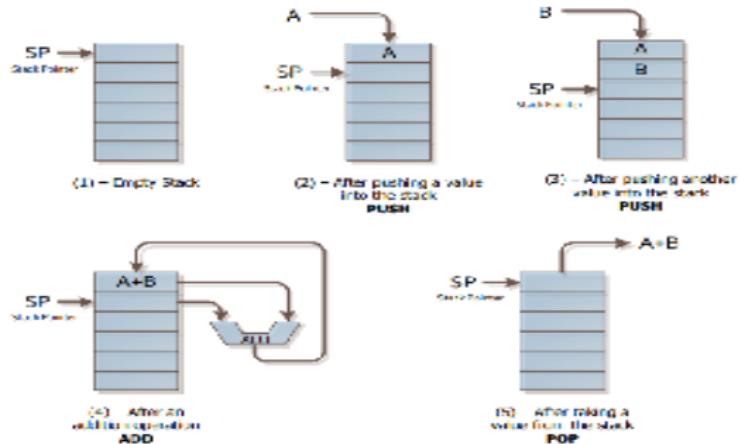
# Clases de ISA

El almacenamiento interno en un procesador es la forma más básica de diferenciación. Las cuatro clasificaciones más importantes son:

- Stack.
- Acumulador.
- Registro-Memoria.
- Registro-Registro.

# Stack

En la clase Stack, los operandos se encuentran en una única pila. Para realizar las operaciones se deben insertar los operandos (*push*) y luego extraer el resultado (*pop*). Los va extrayendo del *Top of Stack Register* (TOS).



## Stack: Ejemplo

Suponga la operación:  $A = B + C$ .

```
push B; // insertar operando 1 hacia latch ALU
push C; // insertar operando 2 hacia latch ALU
add;    //suma
pop A; //extraer resultado del TOS hacia A
```

Los operandos están **implícitos** en el TOS (no se definen/nombran explícitamente).

# Acumulador

En las arquitecturas con acumulador, un operando se encuentra **implícito** en el acumulador y el otro se encuentra **explícito** en memoria.

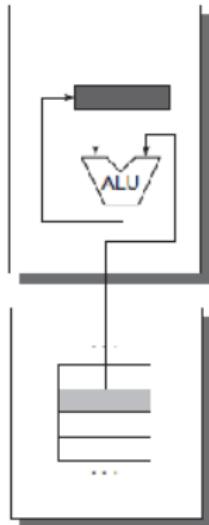
Suponga la operación:  $A = B + C$ .

Load B; // Carga B en el acumulador.

Add C; // Suma C al operando implícito en memoria.

Store A; //Almacena en memoria el resultado.

Uno de los operandos es explícito (**C**) y el otro implícito cargado previamente desde **B**.



## Registro (registro-memoria)

En esta arquitectura se cuenta con registros de propósito general (GPRs). Uno de los operando corresponde a un GPR y el otro proviene directamente de memoria.

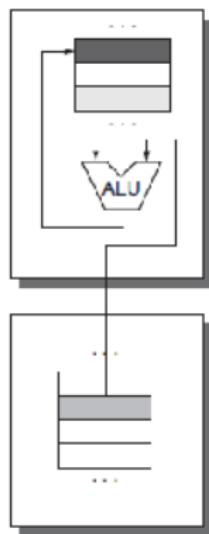
Suponga la operación:  $A = B + C$ .

Load R1, B; //Cargar B en GPR R1

Add R3, R1, C; //Sumar R1 con C (dir. memoria)

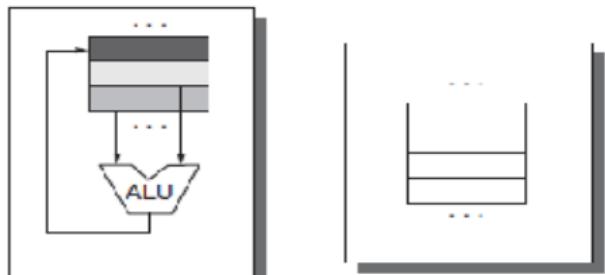
Store R3, A; //Almacenar resultado en A (dir. memoria).

Ambos operandos son explícitos. La mayoría de arquitecturas CISC utilizan esta clase.



## Registro-Registro

Una arquitectura de la clase registro-registro no realiza operaciones directamente desde memoria, sino que las realiza enteramente con operandos en registros de propósito general.



Suponga la operación:  $A = B + C$ .

Load R1, B; // Cargar B en GPR R1

Load R2, C; // Cargar C en GPR R2

Add R3, R1, R2; // Sumar R1 con R2 y almacena en R3.

Store R3, A; // Almacenar resultado en A (dir. memoria).

Ambos operandos son explícitos. La mayoría de arquitecturas CISC utilizan esta clase.

# Direccionamiento de memoria

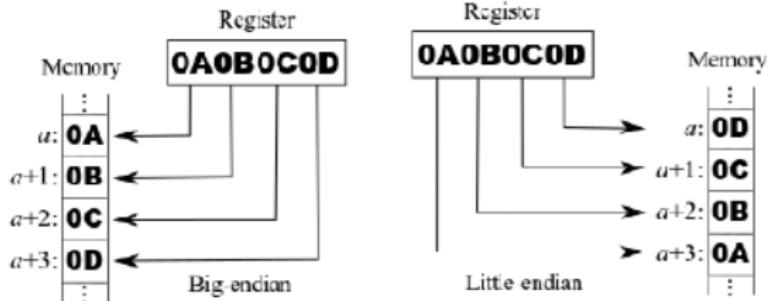
Independientemente de la arquitectura (reg-reg, reg-mem. etc) se debe tener una forma de interpretar y especificar las direcciones de memoria. Existen diferentes formas de acceder a una dirección:

## Endianness

Existen dos formas de ordenar los bytes en una dirección memoria (*Endianness*) (también se conoce *byte ordering*):

- *Little Endian*: Ordena el byte menos significativo en la dirección más pequeña de la palabra (doble palabra).
- *Big Endian*: Ordena el byte más significativo en la dirección más pequeña de la palabra (doble palabra).

Nivel de byte (8 bits)  
Nivel de media palabra (16 bits)  
Nivel de palabra / word (32 bits)  
Nivel de doble palabra (64 bits)



## Alineamiento de memoria

Surge como una limitación de los procesadores modernos.

CPUs son más eficientes cuando las direcciones de memoria son múltiplos del tamaño del dato (byte, KB, etc).

Algunos lenguajes de programación modernos esconden esta limitación al programador.

## Ejemplo

Represente el alineamiento en memoria para x86 de la siguiente estructura y mejore su orden para un mejor alineamiento.

```
struct:  
char a;  
short a1;  
char b1;  
float b;  
int c;  
char e;  
double f;
```

Data Type	32-bit [bytes]	64-bit [bytes]
char	1	1
short	2	2
int	4	4
long	8	8
float	4	4
double	8	8
long double	16	16
pointer	4	8

Table1: typical alignment requirements for data types on 32-bit and 64-bit Linux® systems as used by the Intel® C++ Compiler



Variable      Padding

# Modos de direccionamiento

Los modos de direccionamiento se refiere a la forma en que las arquitecturas **especifican** la dirección de un objeto que van a acceder.

- **Dirección efectiva:** El valor final de dirección especificado por el modo de direccionamiento.

A mayor cantidad de modos de direccionamiento → mayor complejidad (CISC).

A menor cantidad de modos de direccionamiento → menor complejidad (RISC).

Múltiples modos de direccionar datos dentro de una instrucción:

Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,\$3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] := Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables (+ simulates register/ indirect, direct addressing modes).
Register indirect	Add R4,(R1)	Regs[R4] := Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] := Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in array addressing; R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(100)	Regs[R1] := Regs[R1] + Mem[100]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indexed	Add R1,8(R1)	Regs[R1] := Regs[R1] + Mem[Mem[Regs[R1]]]	If R3 is the address of a pointer <i>p</i> , then mode yields <i>*p</i> .
Autodecrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] := Regs[R2] + d	Useful for skipping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, <i>d</i> .
Autodecrement	Add R1, -(R2)	Regs[R2] ← Regs[R2] - d Regs[R1] := Regs[R1] + Mem[Regs[R2]]	Same use as autodecrement. Autodecrement/ increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)(R3)	Regs[R1] ← Regs[R1] + Mem[100 - Regs[R2] + Regs[R3] + d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

# Operandos

## Operandos: Tipo y tamaño de operandos

El tipo de operandos normalmente se encuentra, en la mayoría de los casos, en el código de operación (opcode). Alternativamente los datos son anotados mediante etiquetas (*tags*) que son interpretadas por el HW (mnemónico).

Byte (8 bits).

*Half-Word* (16 bits).

*Word* (32 bits) / punto flotante de precisión simple  
(SP-FP).

*DoubleWord* (64 bits) / punto flotante de precisión doble.

**Enteros:** Típicamente en representación complemento a 2.

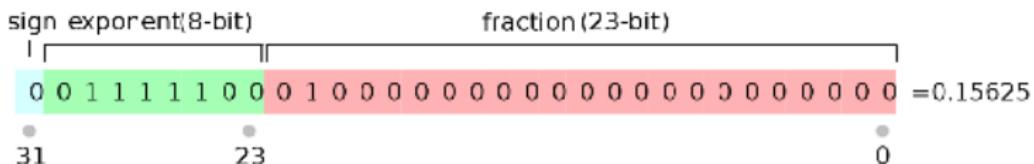
**Flotantes:** IEEE 754\*.

**Caracteres:** ASCII.

**Decimal:** BCD.

¿Cómo se representan decimales?

Ejemplo: Representar  $0,10_{10}$ , en base 2:  $0.\overline{00011}$



El exponente es desplazado mediante un sesgo para poder representar exponentes negativos.

- ① Convertir valor decimal a binario (solo número ignorar el signo).
- ② Colocar el número de la forma:  $\text{numero} \times 2^0$ .
- ③ Denotar el número de la forma  $1, a_1 a_2 \dots a_i \times 2^n$  (se corre la coma  $n$  espacios).
- ④ Determinar el signo: 0 si N es mayor que 0, 1 si N es menor que 0.
- ⑤ Determinar el exponente como  $E = n + 127$ , luego pasar a binario.
- ⑥ Determinar la mantisa F como  $F = a_1 a_2 \dots a_i$ .
- ⑦ Escribir el número según IEEE, completando con ceros a la derecha los 23 bits de la mantisa.

## IEEE 754

Ejemplo: Convertir el número 22,625 a flotante

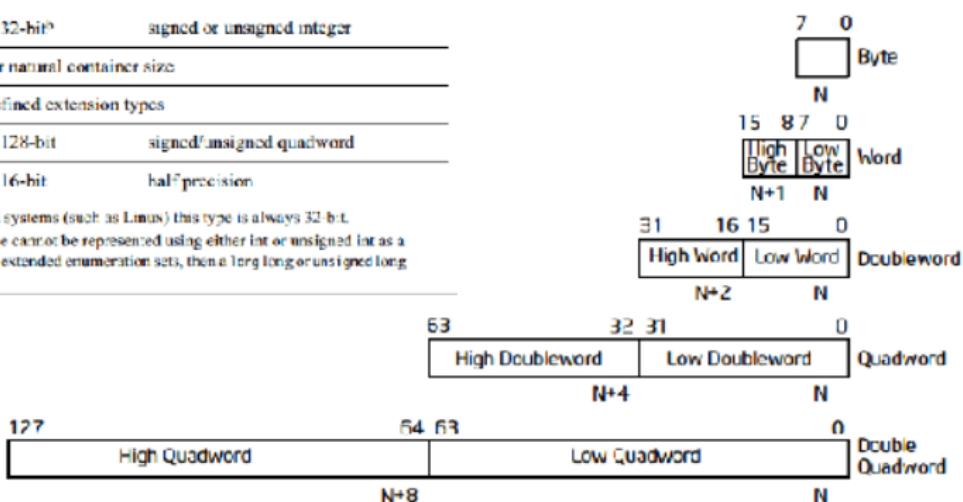
- ① Paso 1: Convertir  $22_{10}$  a binario:  $10110_2$ . Convertir la parte decimal  $0,625_{10}$  a binario:  $0,625_{10} = 101_2$ .
- ② Paso 2: Por lo tanto  $22,625_{10} = 10110,101_2$  en notación científica sería  $22,625_{10} = 10110,101_2 \times 2^0$ .
- ③ Paso 3: Se normaliza a  $1,0110101_2 \times 2^4$ . Por lo tanto  $n = 4$ .
- ④ Paso 4: Se obtiene el signo (0).
- ⑤ Paso 5: Se obtiene el exponente  
 $E = 127 + n \rightarrow E = 127 + 4 = 131_{10} = 10000011_2$ .
- ⑥ Paso 6: Se obtiene la mantisa  
 $F = 011010100000000000000000_2$ .
- ⑦ Paso 7: Se coloca en notación IEEE Signo + Exponente + Mantisa =  $01000001101101010000000000000000_2$ .

# Operando

## Operando: Ejemplo x86

Type	A32	A64	Description
int/long	32-bit	32-bit	integer
short	16-bit	16-bit	integer
char	8-bit	8-bit	byte
long long	64-bit	64-bit	integer
float	32-bit	32-bit	single-precision IEEE floating-point
double	64-bit	64-bit	double-precision IEEE floating-point
bool	8-bit	8-bit	Boolean
wchar_t <sup>a</sup>	16-bit unsigned	16-bit unsigned	short (compiler dependent)
	32-bit unsigned	32-bit unsigned	int (compiler dependent)
void* pointer	32-bit	64-bit	addresses to data or code
enumerated types	32-bit	32-bit <sup>b</sup>	signed or unsigned integer
bit fields	not larger than their natural container size		
ABI defined extension types			
int128/ uint128	128-bit	128-bit	signed/unsigned quadword
float	16-bit	16-bit	half precision

- a. Environment-dependent. In GNU-based systems (such as Linux) this type is always 32-bit.  
 b. If the set of values in an enumerated type can be represented using either int or unsigned int as a container type, and the language permits extended enumeration sets, then a long long or unsigned long long container may be used.



## Codificación

Al tener un tamaño de palabra finito y determinado para las instrucciones, es vital maximizar su uso para poder describir cada instrucción.

Respecto al tamaño de instrucción existen 3 formas comunes de longitud de instrucción:

Variable → CISC.

Fija → RISC.

Híbrido.

### Dirección relativa

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, Super11	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

# Codificación

Un set de instrucciones de **16 bits**, está constituido por instrucciones de **0, 1 y 2 operandos**, los operandos tienen un tamaño de 6 bits, si en él ya existen 5 instrucciones de 2 operandos y 33 instrucciones de 0 operandos. ¿Cuál es el número máximo de instrucciones de 1 operando que se pueden codificar con dicho ancho de palabra y como sería la codificación del ISA?

## Ejemplo

A	B	Y
0	0	0 op
0	1	1 op
1	0	2 op
1	1	X

Se necesita diferenciar entre si es un operando de 0, 1 o 2 operandos. Para identificarlos se usan 2 bits.

1 operandos → 6 bits → **256 instrucciones**

15	14	13	8	7	0
# Op	1	operandos (6 bits)	Id inst.	(256 instrucciones)	

0 operandos → 0 bits → 33 instrucciones

15	14	13	0
# Op	Id inst.	(14 bits o 16384 instrucciones)	

2 operandos → 12 bits → 5 instrucciones

15	14	3	2	0
# Op	2	operandos (12 bits = $2 \times 6$ bits)	Id inst.	

# Paralelismo

Técnica de programación e implementación en la que se pretende realizar operaciones simultáneamente, con el fin de reducir tiempos de ejecución, en un procesador.

Existen diferentes tipos de paralelismo:

- Paralelismo a nivel de bit.
- Paralelismo a nivel de instrucciones.
- Paralelismo a nivel de datos.
- Paralelismo a nivel de tareas.
- Paralelismo a nivel de hilos.

# Paralelismo a nivel de instrucción (ILP)

Técnica de **paralelismo** basada en la ejecución simultánea de instrucciones.

Posee dos enfoques:

Paralelismo por hardware (dinámico) - ejecución

Paralelismo por software (estático) - compilación

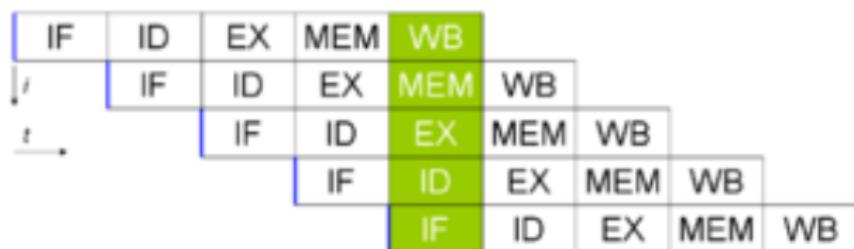
Tipos de ILP:

Segmentación Pipeline

Ejecución fuera de orden (OoOE)

VLIW

CPU's superescalares



# ILP - Bloque básico

Corresponde a una sección de código secuencial que no presenta ramificaciones (branches) hasta el final del mismo. En un bloque básico, el flujo de control es **secuencial** y no se detiene hasta terminar el bloque.

## Tiempo de ejecución en el peor de los casos (WCET)

Tiempo máximo que tarda en ejecutar un código en un hardware específico. Fundamental en sistemas de tiempo real.

- Análisis estático, típicamente.
- $x_i$  puede tener restricciones estructurales y/o dadas por el programador.

Dado un programa con  $N$  bloques básicos, donde cada bloque  $B_i$ , que posee un tiempo de ejecución máximo  $c_i$ , se ejecuta un número de veces  $x_i$ , el WCET es:

$$\sum_{i=1}^N c_i \cdot x_i$$

```
w = 0;  
x = x + y;  
y = 0;  
if( x > z )  
{  
    y = x;  
    x++;  
}  
else  
{  
    y = z;  
    z++;  
}  
w = x + z;
```

Source Code

```
w = 0;  
x = x + y;  
y = 0;  
if( x > z )
```

```
y = x;  
x++;
```

```
y = z;  
z++;
```

```
w = x + z;
```

Basic Blocks

# Mejoras ILP mediante Hardware

Técnicas que permiten hacer *hardware* capaz de procesar mas instrucciones simultáneamente

Segmentación Pipeline.

OoOE.

VLIW.

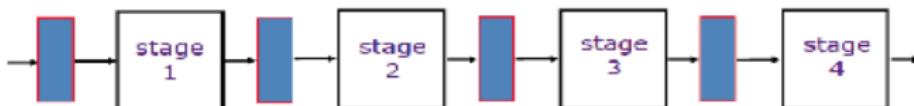
Superescalar.

Especulación\*.

Renombramiento de registros.

## Segmentación - Pipeline

Tecnica utilizada en el diseño de CPUs para aumentar el rendimiento, mediante la separacion de las etapas en el proceso de ejecución de una instrucción



El objetivo del diseñador del pipeline es balancear la longitud de cada etapa del pipeline.

Time per instruction on unpipelined machine

Number of pipe stages

T'De esta forma se obtiene un speedup teórico de N (número de etapas).

## Etapas básicas de un pipeline

- Búsqueda de instrucción (**IF**): Enviar el PC a memoria, traer nueva instrucción, actualizar el PC.
- Decodificación de instrucción (**ID**): "Traducción de instrucción", lectura de registros operandos.
- Ejecución /Dir efectiva (**Ex**): Operaciones en ALU: Memoria efectiva, R-R, R-I.
- Acceso a memoria (**MEM**): Instrucciones L/S.
- Escritura a registros (**WB**): Escritura de resultados R-R o instrucciones L/S a banco de registros.

Número de instrucción	Ciclo de reloj								
	1	2	3	4	5	6	7	8	9
Instrucción i	IF	ID	EX	MEM	WB				
Instrucción i + 1	IF	ID	EX	MEM	WB				
Instrucción i+2		IF	ID	EX	MEM	WB			
Instrucción i+3			IF	ID	EX	MEM	WB		
Instrucción i+4				IF	ID	EX	MEM	WB	

# Ganancia en desempeño

La ganancia en el desempeño debido al pipeline está dada por

$$\begin{aligned}\text{Speedup from pipelining} &= \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}} \\ &= \frac{\text{CPI unpipelined} \times \text{Clock cycle unpipelined}}{\text{CPI pipelined} \times \text{Clock cycle pipelined}} \\ &= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}\end{aligned}$$

## Ventajas

Aumenta el rendimiento del CPU.

Brinda determinismo en ejecución de instrucciones.

## Desventajas

Etapas e instrucciones lentas afectan el rendimiento general

Mayor complejidad, más hardware.

Latencia es ligeramente mayor.

## Ventajas y desventajas

## Riesgos

# Riesgos en la segmentación

**Riesgo:** Situación que previene que la siguiente instrucción pueda ser ejecutada en el ciclo de reloj correspondiente.

Riesgos estructurales: conflictos de hardware entre instrucciones.

Riesgos de datos: Causado por dependencias **reales** entre datos de instrucciones

Riesgos de control: Saltos y branches.

Los riesgos reducen el desempeño ideal ganado por la técnica de pipeline.

## Stalls

Los riesgos provocan que el pipeline **se detenga** ( **stall** )

- Las instrucciones calendarizadas antes de la instrucción detenida deben terminar su ejecución.
- Las instrucciones calendarizadas después de la instrucción detenida deben ser detenidas igualmente.

Se debe tomar en cuenta el tiempo detención por instrucción:

$$\text{CPI}_{\text{pipelined}} = \text{IdealCPI} + \frac{\text{Pipeline stall clockcycles}}{\text{instruction}}$$

Si se toma IdealCPI = 1:

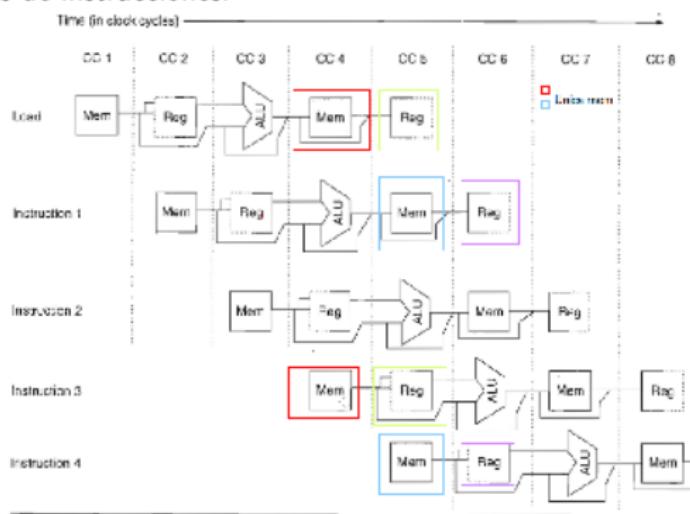
$$\text{Speedup} = \frac{\text{CPI}_{\text{unpipelined}}}{1 + \frac{\text{Pipeline stall clockcycles}}{\text{instruction}}}$$

# Unidades Funcionales - FU

Una unidad funcional es un elemento, dentro del hardware de un procesador, que realiza una función específica: Ejemplos:

## Riesgos Estructurales

En un procesador con pipeline se requieren unidades funcionales duplicadas para alojar recursos en todas las posibles combinaciones de instrucciones.



ALU Multiplicadores Contadores  
fpALU Comparadores Entre otros.

Cuando no hay recursos necesarios para evitar conflictos en uso de hardware se tiene un **riesgo estructural**

## Posibles Soluciones

Uso de stalls.

Solución simple

Disminuye el rendimiento -> 1 ciclo más

Duplicar hardware

Solución más compleja. Puede requerir lógica de control adicional

Puede ser cara (más hardware, más potencia)

No disminuye el desempeño

# Riesgos de datos

Los **riesgos de datos** ocurren cuando en el pipeline se cambia el orden de acceso a lectura/escritura de operandos, de forma que el orden difiere de la ejecución secuencial en un procesador sin pipeline.

DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9
XOR	R10,R1,R11

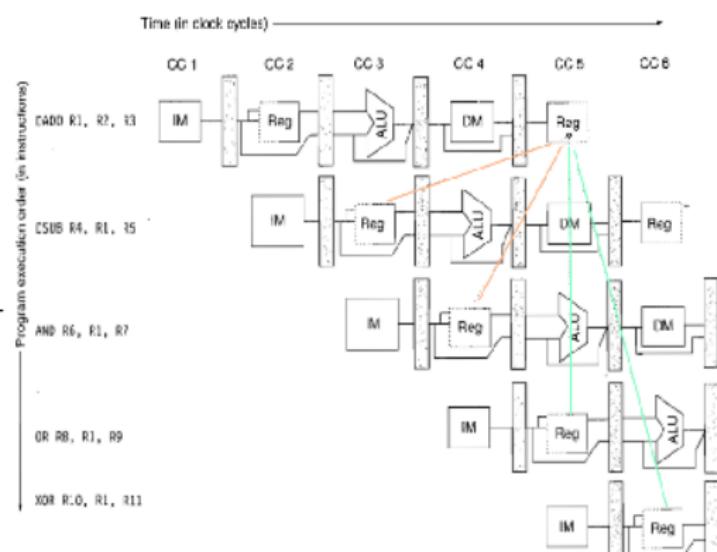
## Possibles Soluciones

Uso de stalls.

Solución simple

Disminuye el rendimiento -> requiere más de 1 ciclo adicio-

**Adelantamiento:** Consiste en mover el resultado de un registro directamente hacia la siguiente etapa donde se necesita, si esperar al WB.



# Riesgos de control

Los **riesgos de control** ocurren cuando al tener la ejecución de una instrucción *branch* se puede modificar o no el valor del PC, alterando el flujo de ejecución del programa.

- Dependiendo de si el salto se toma o no (etapa ID), la siguiente instrucción será o no la correcta.

## Possibles Soluciones

Stall de un ciclo.

- Después de cada salto se realiza dos IF.

**Predicción de salto:** Estrategia basada en métodos estadísticos o probabilistas para tratar de predecir si salto es tomado o no. En caso de fallar la predicción, se debe vaciar (flush) el pipeline.

Branch instruction	IF	ID	EX	MEM	WB	
Branch successor		IF	IF	ID	EX	MEM
Branch successor + 1				IF	ID	EX
Branch successor + 2					IF	ID

# Ejecución fuera de orden (OoOE)

Tipo de ILP en el que las instrucciones se ejecutan en un orden distinto al que fueron programadas.

## OoOE

1. SUB R1, R2, R3
2. ADD R4, R3, **R1**
3. ROR R2, R2, #4

En lugar de 1,2,3 (riesgo de datos), se puede cambiar el orden a 1,3,2 y ganar desempeño.

## Dependencias en pipeline

En una arquitectura que implementa pipeline (y/o otras formas de ILP) se pueden presentar 3 tipos de dependencias entre instrucciones:

- ① Dependencias de datos (reales).
- ② Dependencias de nombre.
- ③ Dependencias de control.

Las dependencias, en general, son **producto de los programas**

Si una dependencia lleva a un riesgo, su detección y corrección son propiedades de la **organización** del pipeline.

## Dependencias de datos (reales)

Una dependencia de datos entre instrucciones puede surgir en los siguientes casos:

- La instrucción  $i$  produce un resultado que puede ser utilizado por la instrucción  $j$ .
- La instrucción  $j$  depende de un dato de la instrucción  $k$ , y la instrucción  $k$  depende de un dato de la instrucción  $i$ .

L.D	F0,0(R1)	; F0=array element
ADD.D	F4,F0,F2	; add scalar in F2
S.D	F4,0(R1)	; store result
DADDUI	R1,R1,#-8	; decrement pointer 8 bytes
BNE	R1,R2,LOOP	; branch R1!=R2

### Ejemplo

1. ADD **R3**,R2, R1
2. SUB R1, **R3**, 1

Loop:

L.D	F0,0(R1)	; F0=array element
ADD.D	F4,F0,F2	; add scalar in F2
S.D	F4,0(R1)	; store result

## Componentes de una dependencia de datos

Al tratar con dependencia de datos se debe tomar en cuenta:

- La **posibilidad** de un riesgo.
- El orden en que las instrucciones deben ejecutarse (caso OoOE).
- Límite máximo de paralelismo que puede ser explotado.

## Soluciones a una dependencia de datos

Una dependencia no implica necesariamente un riesgo, pero deben ser atendidas.

- Mantener la dependencia, pero evitar el riesgo
- Eliminar la dependencia por la transformación del código\*\*

Pueden ser implementadas por software o por hardware.

# Dependencia de nombre

Ocurre cuando dos instrucciones usan el mismo registro ( o dirección de memoria), pero **NO** hay relación o flujo entre las instrucciones.

Dos tipos:

## Antidependencia

Ocurre cuando una instrucción  $j$  escribe a un registro o posición de memoria que una instrucción  $i$  lee.

### Ejemplo

1. ADD R3, **R2**, R1
2. SUB **R2**, R5, 1

## Dependencia de salida

Ocurre cuando una instrucción  $i$  y una instrucción  $j$  escriben al mismo registro o dirección de memoria.

### Ejemplo

1. ADD R3, R1, R2
2. SUB **R4**, R3, 1
3. ADD **R4**, R5, R5

## Solución dependencias de nombre

Dado que no hay transmisión entre las instrucciones, **no son dependencias** reales.

- Pueden ser ejecutadas en paralelo

Solución: **Renombramiento de registros**

- Por hardware: Calendarización dinámica de instrucciones. RRU: register renaming unit.
- Por software: Calendarización estática. Compilación.

## Dependencias de control

Una dependencia de control determina el orden de ejecución de una instrucción *i*, con respecto a una instrucción de salto previa.

```
if p1{  
    S1;  
}  
if p2{  
    S2;  
}
```

**No debe** invertirse el orden de ejecución cuando existen dependencia de control.

### Implicaciones

- ① Una instrucción dependiente de control en un salto NO puede moverse antes del salto.
- ② Una instrucción que NO es dependiente de control NO puede moverse justo después de un salto.

### Ejemplo I \_init :

**ADD** R1, R1, R2

**BEQZ** R1, T0, L1

**SUB** R1, R2, R3

L1:

done

# Riesgos de datos

Un riesgo de datos se puede tener cuando se presenta una dependencia de **nombre** o real de **datos** entre instrucciones lo suficientemente cercanas para que se pueda producir un cambio en el orden de acceso a los operandos.

Tres tipos de riesgos de datos:

## ReadAfter Write - RAW

Se presenta cuando una instrucción  $j$  trata de leer un operando antes de que la instrucción  $i$  lo escriba, obteniendo un valor antiguo.

Ejemplo

1. ADD **R3,R2, R1**
2. SUB **R1, R3, 1**

## Escritura después de lectura(WAR)

Se presenta cuando la instrucción  $j$  trata de escribir un destino **antes** de que sea leído por la instrucción  $i$ , lo que provoca que  $i$  lea el nuevo valor (incorrecto).

Ejemplo

- i ADD R4,R2, **R1**  
j SUB **R1, R3, 1**

## Escritura después de escritura (WAW)

Se presenta cuando la instrucción  $j$  trata de escribir un operando **antes** de que se escrito por la instrucción  $i$ . Las escrituras se realizan en el orden incorrecto.

Ejemplo

- i ADD **R1,R2, R3**  
j SUB **R1, R3, 1**

## Técnicas de software para mejorar ILP

Se consideran estáticas.

Se realizan durante tiempo de compilación.

- 'Información' para branch prediction.
- Reordenamiento de código (memoria).

Durante compilación se puede reorganizar el código de forma que se optimice el uso de procesador.

- Renombramiento de registros.

Durante compilación se puede detectar falsas dependencias de datos y renombrar registros para aumentar el ILP.

- *Loop unrolling*.

La idea principal es reducir la cantidad de iteraciones y lógica de control (instrucciones condicionales) en un bloque de código para mejorar el ILP.

Puede darse como optimización del compilador o en el código fuente directamente.

---

## Práctica 1

### Enfoque cuantitativo a la Introducción a Computadores

---

Fecha de asignación: 9 de septiembre de 2020

| Profesor: Luis Chavarría Zamora

---

Para cada uno de los ejercicios a continuación, realice los cálculos y demostraciones necesarias, para llegar a la solución correcta.

1. La disponibilidad es la consideración más importante a la hora de diseñar servidores, seguido por la escalabilidad y el flujo de instrucciones.
  - a) Se tiene un procesador con un promedio de 100 fallas por año. ¿Cuál es el tiempo medio para falla (MTTF), en días y horas, del sistema? Asuma una distribución uniforme de los fallos. (Respuesta: 3 días y 15 horas.)
  - b) Si toma un día para tener el sistema funcionando de nuevo, ¿cuál es la disponibilidad del sistema? (Respuesta:  $A = 0,785$ ).
  - c) Suponga que el gobierno, para recortar gastos, va a construir un supercomputador de componentes económicos, en lugar de componentes más caros y confiables. ¿Cuál es el MTTF para el sistema con 1000 de los procesadores mencionados arriba? Asuma que si uno falla, todos fallan.
2. En granjas de servidores usadas por Amazon o eBay, una sola falla no causa que el sistema entero falle. En su lugar, causa la reducción del número de peticiones que pueden ser atendidas simultáneamente.
  - a) Si una compañía tiene 1000 computadores, cada uno con un MTTF de 35 días, y experimenta una falla catastrófica solamente si 1/3 de los computadores fallan, ¿cuál es el MTTF del sistema? (Respuesta: 31 años, 350 días, 9 horas y 36 minutos).
  - b) Si cuesta \$1000 extra, por computador, doblar el MTTF, ¿sería esta una buena decisión de negocio? Justifique su respuesta desde el punto de vista financiero.
3. Su compañía acaba de comprar un nuevo procesador Intel Core i5, de doble núcleo, y se le ha asignado a usted optimizar el software para ese procesador. Usted ejecutará dos aplicaciones en el procesador, pero los recursos requeridos no son iguales. La primera aplicación requiere 80 % de los recursos y la otra el 20 % restante. Asuma que cuando paraleliza una porción del programa, la mejora de esa porción es de 2.
  - a) Dado que el 40 % de la primera aplicación es paralelizable ¿Qué mejora observaría el sistema **general** (ambas aplicaciones se ejecutan)? (Respuesta: 1.1904)

- b) Dado que el 99 % de la segunda aplicación es paralelizable ¿Qué mejora observaría el sistema **general** (ambas aplicaciones se ejecutan)? (Respuesta: 1.1098)
- c) Si se aplican las mejoras listadas en (a) y (b), ¿cuál sería la mejora **general** del sistema (ambas aplicaciones se ejecutan)? (Respuesta: 1.3495)
4. Al paralelizar una aplicación, la mejora ideal es equivalente al número de procesadores. Esto es limitado por dos aspectos: porcentaje de la aplicación paralelizable y costo de comunicación. La ley de Amdahl toma en cuenta el primer factor, pero no el segundo.
- a) ¿Cuál es la mejora con 8 procesadores si el 80 % de la aplicación es paralelizable, ignorando la comunicación? (Respuesta: 3.33).
- b) ¿Cuál es la mejora con 8 procesadores si, para cada procesador, el *overhead* de comunicación es de 0.5 % del tiempo de ejecución original? (Respuesta: 2.94)
- c) ¿Cuál es la mejora con N procesadores si, por cada vez que se dobla la cantidad de procesadores, el *overhead* de comunicación aumenta un 0.5 % del tiempo de ejecución original? (**Pista:** ¿qué función sigue la distribución:  $f(2) = 1$ ,  $f(4) = 2$ ,  $f(8) = 3$ )
- d) Determine una ecuación para la cantidad de procesadores, con la máxima mejora, en una aplicación con P % de la aplicación paralelizable y, por cada vez que se dobla la cantidad de procesadores, el overhead de comunicación aumenta un 0.5 % del tiempo de ejecución original.

## Práctica 1 - Solución

① a. ~~100 fallos~~  $\Rightarrow \frac{365}{100} = 3,65$  días

$$0,65 \text{ días} \cdot 24 \text{ horas} = 15,6 \text{ horas}$$

$$0,6 \text{ horas} \cdot 60 \text{ minutos} = 36 \text{ minutos}$$

R/ 3 días 15 horas y 36 minutos

b. MTTF =  $\frac{3,65 \text{ días}}{3,65 \text{ días} + 1 \text{ día}} = 0,785$

c.  $\frac{3,65}{1000} = 0,00365$

② a. MTTF = 35 días  $\cdot 1000$  computadoras  $= \frac{35000}{3} = 11\,666,66$  días

$$11\,666 \text{ días} \cdot \frac{1 \text{ año}}{365 \text{ días}} = 31,96 \text{ años}$$

$$0,96 \text{ años} \cdot \frac{365 \text{ días}}{1 \text{ año}} = 350,4 \text{ días}$$

$$0,4 \text{ días} \cdot \frac{24 \text{ h}}{1 \text{ d}} = 9,6 \text{ horas}$$

$$0,6 \text{ h} \cdot \frac{60 \text{ m}}{1 \text{ h}} = 36 \text{ minutos}$$

R/ 31 años 350 días 9 horas y 36 minutos.

b. No es necesario, mejor ahorrar el dinero.

③ a.  $\frac{1}{(1-0,4 \cdot 0,8) + \frac{0,4 \cdot 0,8}{2}} = 1,19048$

b.  $\frac{1}{(1-0,99 \cdot 0,2) + \frac{0,99 \cdot 0,2}{2}} = 1,10988$

c.  $\frac{1}{(1-0,4 \cdot 0,8 - 0,99 \cdot 0,2) + \frac{0,4 \cdot 0,8 + 0,99 \cdot 0,2}{2}} = 1,3495$

④ a.  $\frac{1}{(1-0,8) + \frac{0,8}{8}} \approx 3,33$

b.  $\frac{1}{(1-0,8) + 0,005 \cdot 8 + \frac{0,8}{8}} \approx 1,9412$

c.  $\frac{1}{(1-0,8) + 0,005 \cdot \log_2(N) + \frac{0,8}{N}}$

d.  $\frac{d}{dN} \frac{1}{(1-p_N) + 0,005 \cdot \log_2(N) + \frac{p_N}{N}} = 0$

## Práctica 2

### Principios del ISA, Amdahl y Flynn

---

Fecha de asignación: 6 de marzo de 2020

| Profesor: Luis Chavarría Zamora

---

Para cada uno de los ejercicios a continuación, realice los cálculos y demostraciones necesarias, para llegar a la solución correcta.

1. La siguiente es una descripción de una implementación paralela:

*Este arreglo corresponde a una distribución homogénea de unidades de procesamiento de datos llamadas células o nodos. Cada nodo calcula independientemente un resultado parcial como una función de los datos recibidos, los nodos operan sobre un único set de datos, a la vez*

Con base en la descripción anterior como se puede clasificar la implementación según la taxonomía de Flynn. Justifique, su respuesta.



2. Mencione las limitaciones y simplificaciones que toma en cuenta la ley de Amdahl.
3. Suponga que en un programa el 0.2% no es paralelizable, ahora, dicho programa corre en una super computadora que posee 2 600 000 núcleos, y que estos funcionan a la misma velocidad sin ninguna penalidad ¿Cuál es la mejora (*Speedup*) empleando 24, 1024, 1048576, 2097152 núcleos?
4. Respecto al programa anterior suponga que ahora existen 2 implementaciones que  $I_1$  e  $I_2$  que implican una penalización en la paralelización de  $P_1(n) = 0,002n$  y  $P_2(n) = 0,004 \log_2(n)$  respectivamente, donde n corresponde al número de núcleos. Determine el número de núcleos en el cual se hace máximo la mejora (*Speedup*).
5. Suponga que la sección paralelizable de un programa puede ser mejorado de tres diferentes maneras:
  - a) Mejora X brinda *Speedup* de 30.
  - b) Mejora Y brinda *Speedup* de 20.
  - c) Mejora Z brinda *Speedup* de 15.

En la arquitectura empleada sólo es posible tener una mejora activa a la vez.

Basado en lo anterior cómo es posible plantear la Ley de Amdahl para manejar múltiples mejoras activas una a la vez.

Empleando esta generalización como se podría obtener un *Speedup* total de 15, si las mejoras X y Z sólo pueden estar activas un 30 % del tiempo.



6. Un set de instrucciones está constituido por un ancho de palabra de 32 bits, existen instrucciones de 0, 1 y 2 operandos, los operandos tienen un tamaño de 6 bits, si ya existen 5 instrucciones de 2 operandos y 33 instrucciones de 0 operandos. ¿Cuál es el número máximo de instrucciones de 1 operando que se pueden codificar con dicho ancho de palabra? 
7. En el siguiente diagrama se muestran los contenidos de la memoria:

0xDEAD	16
0xDEAF	32

Si se requiere hacer un programa de ensamblador que pueda sumar, y multiplicar los contenidos en dichas posiciones de memoria, y guardar los resultados de la suma en 0xDEAD y de la multiplicación en 0xDEAF, proponga:

- a) ¿Cómo sería tal programa en una arquitectura LOAD/STORE? 
- b) ¿Cómo sería tal programa en una arquitectura REG/MEM?



## Práctica 2

### 1. MISD

- La paralelización es uniformemente distribuida en las unidades de ejecución.
- No aplica en sistemas heterogéneos y multicore.
- Supone q' no hay conflictos de recursos.

3. 1

$$0,002 + \frac{1-0,002}{N}$$

Para  $N=2$  : 22,94

Para  $N=1024$  : 336,1786

Para  $N=10^48576$  : 499,7622

Para  $N=2097152$  : 499,88

4. Para  $P_1$

$$\frac{d}{dN} \frac{1}{0,002 + 0,002N + \frac{1-0,002}{N}} = 0$$

$$N = \sqrt{499} \approx 22,34 \approx 22$$

Para  $P_2$

$$\frac{d}{dN} \frac{1}{0,002 + 0,004 \log_2(N) + \frac{1-0,002}{N}} = 0$$

$$N = 172,94 \approx 173$$

5.

$$15 = X + Y + Z = 30 \cdot 30\% + 20 \cdot X + 15 \cdot 30\%.$$

$$X = 0,075 = 7,5\%$$

6. Se debe verificar las instrucciones de 2 operandos y 0 operandos

Para identificar el operando:

00	0op
01	1op
10	2op
11	X

2 operandos  $\rightarrow$  cada operando 6 bits = 12 bits

31 30 29	28 27	0
#op	12 bits	8 bits

operando  $\rightarrow$  Sobran 18 bits para identificar instrucciones  
 $2^{18} = 262144 \gg 5$

Este resultado verifica que se pueden poseer 5 instrucciones.

0 operandos

31 30 29	0
#op	30 bits

$\rightarrow$  Sobran 30 bits para identificar

$$2^{30} = 1073741824 \gg 33$$

Este resultado verifica que se pueden poseer 33 instrucciones.

Ahora se calculan cuantas inst. de 1 op se pueden tener

31 30 29 28 27	0	
#op	6 bits	24 bits

operando  $\rightarrow$  Sobren 24 bits

$$2^{24} = 16777216$$

Como cubre el bit 11 del #op se puede utilizar para más inst de 1 op.  $2^4 + 2^{24} = 33554432$

---

Carné: \_\_\_\_\_ Nombre: \_\_\_\_\_ Nota: \_\_\_\_\_

## INSTRUCCIONES GENERALES.

- Debido al estado de la crisis se recalca la importancia de mantener una comunicación continua y pronta con el profesor para atender cualquier eventualidad lo más rápido posible.
- Esta evaluación se realizará durante horas de clase para no interferir con el proceso de aprendizaje en otros cursos.
- Si presenta problemas con el quiz contacte al profesor por correo electrónico o al teléfono 8331-0211.
- Se realizará una videollamada durante la realización de la evaluación para atender las dudas inmediatamente.
- Esta evaluación es **individual** y tiene una duración máxima de 45 minutos. El control del tiempo debe tener la siguiente distribución:
  - En los primeros 5 minutos debe ingresar la contraseña del enunciado. Si tiene problemas debe escribir al profesor inmediatamente. El enunciado se subirá previamente a la plataforma TEC-Digital (por lo menos 24 horas antes de la realización).
  - Del minuto 5 al minuto 50 debe realizar el examen corto, si tiene dudas realicela durante la llamada.
  - Del minuto 50 al 55 debe unir los archivos en un pdf.
  - Del minuto 55 al 60 debe subir el archivo a la plataforma de TEC-Digital.
- La entrega está habilitada desde el minuto 0 del quiz.
- Despues del minuto 60 se rebajará un punto por minuto para la base de calificación. Se revisará la última versión enviada. Si tiene problemas subiendo contenido adjunte un screenshot y la justificación al correo electrónico del profesor **inmediatamente, recuerde la importancia de la comunicación oportuna.**
- Responda de forma clara, ordenada y legible en un pdf. **El quiz debe ser escrito a mano, no se permite editores de texto.**
- El documento será sometido a control de plagios, se prohíben copias textuales de otros estudiantes o sitios en Internet.
- El documento debe reflejar el entendimiento del concepto, por esta razón tiene que ser explicado en sus propias palabras, sin recurrir a citas bibliográficas.
- Este examen corto es de 70 puntos.

Conteste los siguientes problemas de manera adecuada. Realice el planteamiento de todos los procedimientos necesarios para llegar a la solución correcta.

1. Explique en qué tipo de instrucciones no funciona una mejora dinámica por unidad de adelantamiento en una arquitectura pipeline de 5 etapas. Use un dibujo como ayuda a su explicación (10 pts).
2. ¿Cuál es la diferencia entre dependencia y riesgo en una arquitectura pipeline? (10 pts).
3. Explique la funcionalidad del loop unrolling como una técnica de mejora de ILP (10 pts).

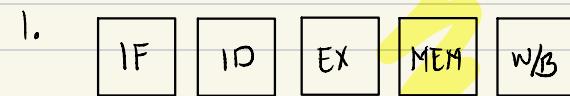
4. Explique en qué consiste el fenómeno de cuello de botella de Von Neumann (10 pts).
5. Dado el siguiente código de ARM en ASM:

```
MOV R0,#64
ROR R0,R0,#2
MOV R2,#0
B LOOP
LOOP
LRDB R1,[R0,R2]
ADD R1,R1,R2
SLR R1,R1,#2
ADD R2,R2,#1
CMP R2,#8
BEQ OFFSET
B LOOP
OFFSET
ADD R1,R1,#20
```

Suponiendo que el programa va correr en una micro-arquitectura segmentada de 5 etapas, sin hardware replicado ni capacidad para detección y solución de riesgos:

- 5.1. Identifique los riesgos/dependencias presentes (10 pts).
- 5.2. Soluciones mediante el uso de stalls los riesgos (10 pts).
- 5.3. Determine el WCET del nuevo código en 5.2 (10 pts).

## Quiz 2



2. Las dependencias son causadas por el programa  
Los riesgos son causados por la arquitectura

3. Cuada realizar el Branch q' realiza dos IF, donde este branch  
implica un riesgo de control

4. Sucede que pasan datos o instrucciones, no pueden pasar ambos a la vez

## 5. Dependencias

1. Dependencia de datos por R0 en líneas 1 y 2.

2. " datos R0 " 2 y 6.

3. " control en linea 4.

4. " datos por R2 en líneas 3 y 6.

5. " datos R1 " 6 y 7.

6. " datos R1 " 6 y 8.

7. " datos R1 " 7 y 8.

8. " datos R2 " 9 y 10.

9. " control en linea 11,

10. " control " 12.

## Riesgos

Puesto que la ejecución es en orden solo se consideran los RAW  
Además del CMP puesto que la linea anterior modifica R2

## Solución

- 2 stalls entre 1 y 2
- 2 stalls entre 6 y 7
- 2 stalls entre 7, 8
- 2 stalls entre 9 y 10

Bloques	1-4	6-11	12	14
se repite	1	8	7	1
Contidad inst	4+2 stalls	6+6 stalls	1	1

$$WCET = \underset{\text{Contd inst ad}}{(4+2) \times 1} + \underset{\text{6 veces que se repite}}{(6+6) \times 8} + 1 \times 7 + 1 \times 1 = 110$$

Carné: \_\_\_\_\_ Nombre: \_\_\_\_\_ Nota: \_\_\_\_\_

## INSTRUCCIONES GENERALES.

- Esta evaluación es individual y tiene una duración máxima de 2 horas.
- Responda de forma clara y ordenada.
- No se aceptarán reclamos en respuestas hechas en lápiz o lapicero borrible.
- Puede usar material impreso o escrito.
- No se permite el uso del celular o cualquier aparato digital para el desarrollo de este examen a excepción de la calculadora.
- Este examen es de 28.5 puntos.

Desarrollo. 28.5 puntos.

A continuación se le presentan una serie de preguntas que debe contestar de forma clara y concisa. Debe mostrar todo su procedimiento para obtener el puntaje en cada punto.

1. **(45 minutos)** Dado el siguiente programa en ensamblador para RV32I (Sugerencia: no copie todo el código):

```
1 .start:
2     ADDI t4 ,x0,#0xfffff800
3     LW    t1 ,0( t4 )
4     LI    t3,#0x12345      // Load Inmediate
5     BNE   t3 ,$zero , ciclo
6     J     salida
7 ciclo :
8     BEQ   t3 ,$zero , salida
9     ADDI t4 ,t4 ,0x004
10    LW    t5 ,0( t4 )
11    ADD   t6 ,t1 ,t5
12    ADD   t5 ,t1 ,t6
13    SW    t5 ,0( t4 )
14    SRLI  t3 ,t3 ,2      // Shift right logical Immediate
15    J     ciclo
16 salida :
17    LI    t4,#0x11        // Load Inmediate
18    SW    t5 ,0( t4 )
19    .end
```

Explique lo siguiente:

- 1.1. **Identifique y clasifique las dependencias presentes en este programa (3 pts).** Solo identifique dependencias que se encuentren dentro del rango de cinco líneas después de la actual. Use el siguiente formato para su respuesta:

**Dep., Subtipo Dep. (opcional), REG(opcional), línea 1, línea 2 (opcional)**

- 1.2. **En un contexto de ejecución en orden, identifique y clasifique los riesgos (3 pts).** Use el siguiente formato para su respuesta:  
**Riesgo, Subtipo riesgo (opcional), REG(opcional), línea 1, línea 2 (opcional)**
- 1.3. **De forma estática con STALLS resuelva los riesgos en un pipeline balanceado de 5 etapas (3 pts).** La lectura y escritura se da en el flanco de subida. **Considere los efectos de latencia para asegurar una escritura y lectura exitosa.** NO hay unidad de adelantamiento y predicción de saltos. Indique entre cual y cual línea se hace la mejora. Sea eficiente y eficaz con su propuesta, se penalizarán ciclos de más que reduzcan el rendimiento.
- 1.4. **Calcule el WCET después de la mejora anterior (asuma un pipeline vacío al inicio) (1.5 pts).**
- 1.5. **Calcule el WCET si se aplica una política de salto tomado en el bloque básico ciclo. Indique el porcentaje de mejora con respecto al WCET de 1.4 (es un porcentaje, no calcule Amdahl). (1.5 pts).**
2. **(30 minutos)** Se desea mejorar la micro-arquitectura de un procesador DSP **muy viejo**. Las instrucciones de enteros y control de flujo están en el mismo hardware y usan los mismos aceleradores, por esa razón las mejoras son iguales **X** veces en ambos casos. Cuenta con un acelerador para procesamiento gráfico. Después de realizar un análisis del tipo de instrucciones y su frecuencia mediante un *benchmark* determinado se obtuvieron los siguientes datos:
- Las instrucciones de enteros son 25 % no paralelizables y son **X** veces más rápidas. Se ejecutan un 15 % del tiempo.
  - Las instrucciones de control de flujo son 15 % paralelizables y tiene una mejora de **X** veces. Se ejecutan un 15 % del tiempo.
  - Las instrucciones de acceso a memoria se ejecutan un 20 % del tiempo, son no paralelizables y no pueden mejorarse.
  - Las instrucciones de procesamiento gráfico son 90 % paralelizables y tiene una mejora de 100 veces. Se ejecuta un 50 % del tiempo.
- Ninguna de las instrucciones se ejecutan simultáneamente (a pesar de que pueden), es decir, solo se ejecuta una a la vez.
- 2.1. **¿Cuántas veces deben mejorar las instrucciones de control de flujo para obtener un Speedup total del sistema de 5.25 (100 % del tiempo)? (6 pts).**
- 2.2. Se le contrata a usted como ingenier@ en computadores para analizar el sistema con estos datos presentados. Con base en esto conteste lo siguiente:
- **El sistema al ser muy viejo no es óptimo a nivel de consumo de potencia, sin embargo, a nivel de memoria es óptimo. Proponga usted alguna mejora al sistema, sabiendo que no puede proponer una ejecución simultánea de instrucciones y no puede cambiar el código de software (1.5 pts).**

3. **(30 minutos)** Dada la siguiente aplicación en un procesador basada en **RV64I**, tiene 64 registros que permite operaciones entre registros e inmediatos, el ISA tiene 4 tipos de instrucciones organizados de la siguiente manera:

- Tipo A: Instrucciones que emplean un registro de *source*, un registro de *destination* y un valor inmediato.
- Tipo B: Instrucciones que emplean 2 registros de *source*, un registro de *destination*.
- Tipo C: Instrucciones que emplean un valor inmediato de *source* y un registro de *destination*.
- Tipo D: Instrucciones que emplean un registro de *source* y un registro de *destination*.

Esta arquitectura tiene 250 instrucciones con la siguiente distribución: 20 % instrucciones tipo A, 20 % tipo B, 25 % tipo C y 35 % tipo D. El ISA pide que en instrucciones variables, estas tengan tamaño de múltiplo de 1 byte. Teniendo esto y lo anterior en cuenta, se le pide:

- 3.1. **Determinar el tamaño de cada tipo de instrucción si se tiene un tamaño variable (3 pts).**
- 3.2. **Determinar el tamaño mínimo si tiene que usar tamaño fijo (no aplica la restricción de múltiplo de 1 byte) (3 pts).**
4. **(15 minutos)** De las siguientes direcciones cuales están alineadas en memoria. Evalúe dos escenarios por aparte para *double word* y *quad word*. En caso de que estén desalineadas coloque la siguiente dirección correcta:
  - 4.1. 0x55938A5B (1.5 puntos).
  - 4.2. 0x48F04B6F (1.5 puntos).

## Solución de examen

### 1. RV32I

#### 1.1. Dependencias:

- 1.1.1. Dependencia de datos, t4, línea 2, línea 3.
- 1.1.2. Dependencia de datos, t3, línea 4, línea 5.
- 1.1.3. Dependencia de nombre, dependencia de salida, t4, línea 2, línea 9.
- 1.1.4. Dependencia de control, línea 5.
- 1.1.5. Dependencia de control, línea 6.
- 1.1.6. Dependencia de control, línea 8.
- 1.1.7. Dependencia de datos, t3, línea 4, línea 8.
- 1.1.8. Dependencia de datos, t4, línea 9, línea 10.
- 1.1.9. Dependencia de datos, t5, línea 10, línea 11.
- 1.1.10. Dependencia de nombre, dependencia de salida, t5, línea 10, línea 12.
- 1.1.11. Dependencia de datos, t6, línea 11, línea 12.
- 1.1.12. Dependencia de datos, t4, línea 9, línea 13.
- 1.1.13. Dependencia de datos, t5, línea 12, línea 13.
- 1.1.14. Dependencia de datos, t3, línea 14, línea 8.
- 1.1.15. Dependencias de nombre, antidependencia, t5, línea 11, línea 12.
- 1.1.16. Dependencia de nombre, antidependencia, t4, línea 13, línea 9.
- 1.1.17. Dependencia de control, línea 15.
- 1.1.18. Dependencia de nombre, antidependencia, t4, línea 13, línea 17.
- 1.1.19. Dependencia de datos, t4, línea 17, línea 18.

#### 1.2. Riesgos:

- 1.2.1. Riesgo de datos, RAW, t4, línea 2, línea 3.
- 1.2.2. Riesgo de datos, RAW, t3, línea 4, línea 5.
- 1.2.3. Riesgo de control, línea 5.
- 1.2.4. Riesgo de control, línea 8.
- 1.2.5. Riesgo de datos, RAW, t3, línea 4, línea 8.
- 1.2.6. Riesgo de datos, RAW, t4, línea 9, línea 10.
- 1.2.7. Riesgo de datos, RAW, t5, línea 10, línea 11.
- 1.2.8. Riesgo de datos, RAW, t6, línea 11, línea 12.
- 1.2.9. Riesgo de datos, RAW, t4, línea 9, línea 13.
- 1.2.10. Riesgo de datos, RAW, t5, línea 12, línea 13.
- 1.2.11. Riesgo de datos, RAW, t3, línea 14, línea 8.
- 1.2.12. Riesgo de datos, RAW, t4, línea 17, línea 18.

#### 1.3. Stalls:

- 1.3.1. 3 stalls entre líneas 2 y 3.
- 1.3.2. 3 stalls entre líneas 4 y 5.
- 1.3.3. 2 stalls entre líneas 5 y 6.
- 1.3.4. 2 stalls entre líneas 8 y 9.
- 1.3.5. 3 stalls entre líneas 9 y 10.
- 1.3.6. 3 stalls entre líneas 10 y 11.
- 1.3.7. 3 stalls entre líneas 11 y 12.
- 1.3.8. 3 stalls entre líneas 12 y 13.

1.3.9. 2 stalls entre líneas 14 y 15. Para riesgo entre 14 y 8.

1.3.10. 3 stalls entre líneas 17 y 18.

1.4. WCET:

$$\begin{aligned}
 WCET = & \underbrace{\left(4 + \underbrace{3+3+2}_{\text{Stalls}}\right) \times \frac{1}{\text{Veces}}}_{BB1} + \underbrace{(1) \times \frac{0}{\text{Veces}}}_{BB2} + \underbrace{\left(1 + \underbrace{2}_{\text{Stalls}}\right) \times \frac{10}{\text{Veces}}}_{BB3} \\
 & + \underbrace{\left(7 + \underbrace{3+3+3+3+2}_{\text{Stalls}}\right) \times \frac{9}{\text{Veces}}}_{BB4} + \underbrace{\left(2 + \underbrace{3}_{\text{Stalls}}\right) \times \frac{1}{\text{Veces}}}_{BB5} \\
 = & 236 \text{ ciclos} + 4 \text{ ciclos} = 240 \text{ ciclos}
 \end{aligned}$$

$$\begin{aligned}
 1.5. WCET = & \underbrace{\left(4 + \underbrace{3+3+2}_{\text{Stalls}}\right) \times \frac{1}{\text{Veces}}}_{BB1} + \underbrace{(1) \times \frac{0}{\text{Veces}}}_{BB2} + \underbrace{(1) \times \frac{10}{\text{Veces}}}_{BB3} + \underbrace{2}_{\text{Stalls fallo}} + \\
 & \underbrace{\left(7 + \underbrace{3+3+3+3+2}_{\text{Stalls}}\right) \times \frac{9}{\text{Veces}}}_{BB4} + \underbrace{\left(2 + \underbrace{3}_{\text{Stalls}}\right) \times \frac{1}{\text{Veces}}}_{BB5} = 218 \text{ ciclos} + 4 \text{ ciclos} = \\
 & 222 \text{ ciclos}
 \end{aligned}$$

Es un 8,1% (241/223) más rápido.

2. Amdahl.

2.1. Speedup:

2.1.1.Opción 1:

- Instrucciones de enteros (15% del tiempo):  $\frac{0.15}{0.25 + \frac{0.75}{X}}$
- Instrucciones de control de flujo (15% del tiempo):  $\frac{0.15}{1 - 15\% + \frac{0.15}{X}} = \frac{0.15}{0.85 + \frac{0.15}{X}}$
- Instrucciones de acceso a memoria (20% del tiempo):  $\frac{0.2}{1} = 0.2$
- Instrucciones de procesamiento gráfico (50% del tiempo):  $\frac{0.5}{1 - 0.9 + \frac{0.9}{100}} = \frac{500}{109} = 4.587$

$$\text{Amdahl: } \frac{0.15}{0.25 + \frac{0.75}{X}} + \frac{0.15}{0.85 + \frac{0.15}{X}} + 0.2 + 4.587 = 5.25 \rightarrow X = 2.93 \approx 3$$

3 veces

2.1.2.Opción 2 (2 puntos menos pues esto implica una mejora simultánea):

$$\begin{aligned}
 \text{Amdahl: } & \frac{1}{0.25 * 0.15 + 0.85 * 0.15 + 1 * 0.2 + 0.1 * 0.5 + \frac{0.75 * 0.15}{X} + \frac{0.15 * 0.15}{X} + 0 + \frac{0.9}{100}} = 5.25 = \frac{1}{0.415 + \frac{0.135}{X} + 0.09} = \frac{1}{0.424 + \frac{0.135}{X}} \\
 & 0.424 + \frac{0.135}{X} = \frac{4}{21} \rightarrow \frac{0.135}{X} = -0.2335 \rightarrow X = -0.5781
 \end{aligned}$$

2.2. Pasar a una arquitectura de Harvard Modificada.

3. Instrucciones:

3.1. Tamaño variable:

Definición de tipo:

A	B	OUT
0	0	Tipo A
0	1	Tipo B
1	0	Tipo C
1	1	Tipo D

Son 2 bits.

Registros:

- Tipo A: Registro source (6 bits para 64), registro destination (6 bits para 64), inmediato (64 bits): 76 bits.
- Tipo B: Registro source 1 (6 bits para 64), registro source 2 (6 bits para 64), registro destination (6 bits para 64): 18 bits.
- Tipo C: Inmediato (64 bits), registro de destination (6 bits para 64): 70 bits.
- Tipo D: Registro source (6 bits para 64), registro de Destination (6 bits para 64): 12 bits.

Instrucciones:

- Tipo A:  $250 * 20\% = 50$  instrucciones => 6 bits.
- Tipo B:  $250 * 20\% = 50$  instrucciones => 6 bits.
- Tipo C:  $250 * 25\% = 63$  instrucciones => 6 bits.
- Tipo D:  $250 * 35\% = 87$  instrucciones => 7 bits.

Final:

- Tipo A:

Tipo	Registros e inmediato	ID
2 bits	76 bits	6 bits

84 bits => 88 bits para que sea de 8 bits o un byte.

- Tipo B:

Tipo	Registros	ID
2 bits	18 bits	6 bits

26 bits => 32 bits para que sea de 8 bits o un byte.

- Tipo C:

Tipo	Registros e inmediato	ID
2 bits	70 bits	6 bits

78 bits => 80 bits para que sea de 8 bits o un byte.

- Tipo D:

Tipo	Registros	ID
2 bits	12 bits	7 bits

21 bits => 24 bits para que sea de 8 bits o un byte.

Sería:

- Tipo A: 88 bits u 11 bytes.
- Tipo B: 32 bits o 4 bytes.
- Tipo C: 80 bits o 10 bytes.
- Tipo D: 24 bits o 3 bytes.

### 3.2. Tamaño fijo:

Solo sería necesario eliminarle los 2 bits y mantener constante la cantidad de instrucciones:

Para 250 instrucciones se ocupan 8 bits o un byte.

Se debe tomar en cuenta que debe soportar entre 76 y 12 bits, por esa razón, debe ser de al menos 76 bits.

Registros e inmediatos	Opcode
76 bits	8 bits

En total sería 84 bits para los cuatro tipos.

## 4. Alineamiento.

- Double word: 8 bytes. Para que esté alineado con 8 bytes debe ser múltiplo, entonces, tiene que terminar en 8 o 0.
- Quad word: 16 bytes. Para que esté alineado con 16 bytes debe ser múltiplo, entonces, tiene que terminar en 0.

### 4.1. 0x55938A5B

4.1.1.Double word: No alineado, sería 0x55938A60.

4.1.2.Quad word: No alineado, sería 0x55938A60.

### 4.2. 0X48F04B6F

4.2.1.Double word: No alineado, sería 0x48F04B70.

4.2.2.Quad word: No alineado, sería 0x48F04B70.