

SEGMENTACIÓN Y PROCESADORES SEGMENTADOS

2.1 - SEGMENTACIÓN

- Introducción
- Segmentación
- Paralelismo

2.1.1 - Introducción a la segmentación

- Clasificación

2.1.2 - Diseño del control de entradas

- Tabla de reserva
- Latencia
- Secuencia de iniciaciones
- Secuencia del MAL
- Diagrama de estados
- Vector de colisión inicial (VCI)
- Como obtener el diagrama de estado
- Diagrama de estados simplificado (o modificado)
 - A partir del completo
 - Generación directa
 - Análisis del diagrama de estados
- Subconjunto de ciclos simples ---> Greedy
- Modificación de la tabla de reserva

2.1.3 - Hardware de control

- Datos siempre disponibles
- Datos no siempre disponibles

3 - PROCESADORES SEGMENTADOS

3.1 - Introducción a los procesadores segmentados

- Objetivo
- Notación
- Problemas
 - Riesgos estructurales
 - Riesgos de datos
 - Riesgos de control
- Problemas de procesadores segmentados
- Medida de las prestaciones de un procesador
 - Rendimiento
 - Velocidad de la CPU
 - Métricas comerciales
 - Ganancia y aceleración
 - Ley de Amdahl
- Problemas de la segmentación
 - Riesgos estructurales
 - Riesgos de datos
- Tipos de dependencias de datos
 - RAW
 - WAR
 - WAW
 - RAR
 - Soluciones
- Riesgos de control

3.2 - Procesadores segmentados lineales

- Características
- Arquitectura DLX
- Lenguaje máquina del DLX
- Segmentación en etapas
- Estudio de los riesgos estructurales
- DATA PATH segmentado
- Estudio de los riesgos de datos
- Cortocircuito para segmentar RAW
- Estudio de riesgos de control

3.3 - Procesadores segmentados con operaciones multiciclo

- Multiciclo simplificado
- Multiciclo general
- Extensión L.M.
- Data path
- Sincronización entre enteros/punto flotante
- Riesgos estructurales
- Riesgos de datos

3.4 - Procesadores con ordenación dinámica de instrucciones

- Procesadores con anticipación
- Implementaciones
 - Scoreboard ---> CDL 6600
 - Algoritmo de control
 - Tomasulo ----> IBM 360/91

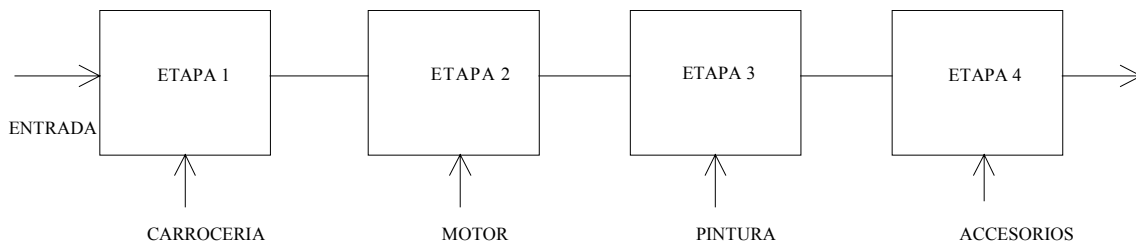
TEMA 2 - UNIDADES FUNCIONALES SEGMENTADAS

2.1 - SEGMENTACIÓN

2.1.1 - INTRODUCCIÓN A LA SEGMENTACIÓN

- La segmentación (pipelining) es una técnica de implementación por la cual se solapa la ejecución de múltiples instrucciones.

- Técnica clave para hacer CPU rápidas.
- Ejem: Línea de montaje de automóviles:



Z --> Tiempo por etapa

- Sin segmentar $T_{6\text{coches}} = 6 \cdot 4Z = 24Z$
- Segmentada $T_{6\text{coches}} = 1.4Z + 5Z = 9Z$

- Objetivo: Segmentar una función ----> Explotar el paralelismo a nivel temporal (solapamiento).

- Evaluar en cada momento, varias funciones con diferentes datos de entrada. Esto solo tiene sentido si:

- Evaluamos la función muchas veces con diferentes datos de entrada.
- Independencia entre las diferentes evaluaciones.

Ejemplo: Diagrama Espacio-Tiempo

E1	X	Y	Z			
E2		X	Y	Z		
E3			X	Y	Z	

-----> tiempo

Ei: Etapas

x,y,z: Datos de entrada

Es necesario un flujo continuo de datos, sino la segmentación no sirve para nada:

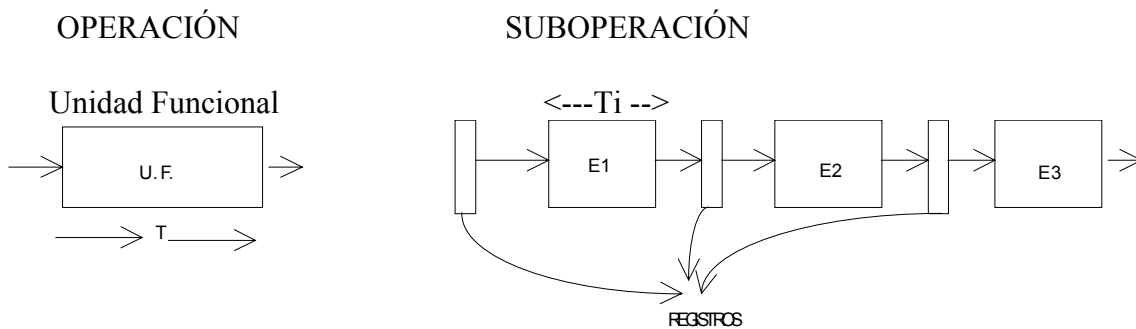
E1	X				Y		
E2		X				Y	
E3			X				Y

Solo segmentamos funciones muy usadas:

- Ejecución de una instrucción ---> Procesadores segmentados
- Multiplicación de 2 números en coma flotante ----> Coprocesadores matemáticos

Pasos a seguir:

- 1.- Descomposición en subfunciones.
- 2.- Secuencia de evaluación (posible paralelismo).
- 3.- Implementación de las subfunciones con hardware específico ---> ETAPAS.
- 4.- Interconexión y aislamiento de las etapas con registros. Añadir control.



RENDIMIENTO

Un procesador rinde más cuando es más rápido -----> menor tiempo

medida de rendimiento -----> Tiempo

Definiremos:

- Tiempo de ciclo: $T_c = \max(t_i)$
- Tiempo de evaluación de una función = nº de etapas * T_c
- T : Tiempo de evaluación de una función sin segmentar

Objetivo:

- $T_i = T_j \quad \forall i, j$

El tiempo de ciclo será el de la peor etapa. No sirve de nada mejorar una etapa sino se pueden mejorar igual todas.

- Minimizar T_i

Factores que reducen el rendimiento:

- Factores tecnológicos:

- Tiempo de carga de los registros ----> $\sum T_i > T$
- Desplazamiento del reloj (clock skew) (*)

- Factores de diseño y utilización:

- Partición en etapas inadecuadas
- Riesgo por dependencia de datos, surgen cuando una instrucción depende de los resultados de una instrucción anterior de forma que ambas no podrán ejecutarse de forma solapada.

(*) Clock skew: El clock debería llegar a todas las etapas en el mismo instante pero debido a diferentes factores físicos (cableado,carga,drivers....) se produce un desplazamiento temporal en la llegada del clock a cada etapa.

Riesgos estructurales

Surgen del conflicto de los recursos, cuando el hardware no puede soportar todas las combinaciones posibles de instrucciones en ejecuciones solapadas simultáneamente.

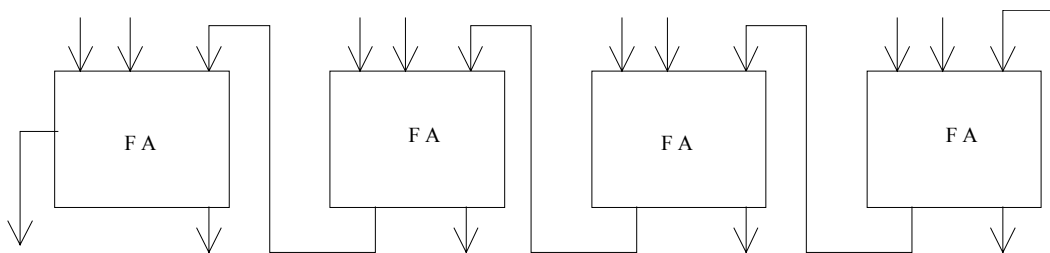
Ejem: pedir a una ALU que realice el cálculo de una dirección efectiva y una resta al mismo tiempo.

Control de entradas inadecuadas

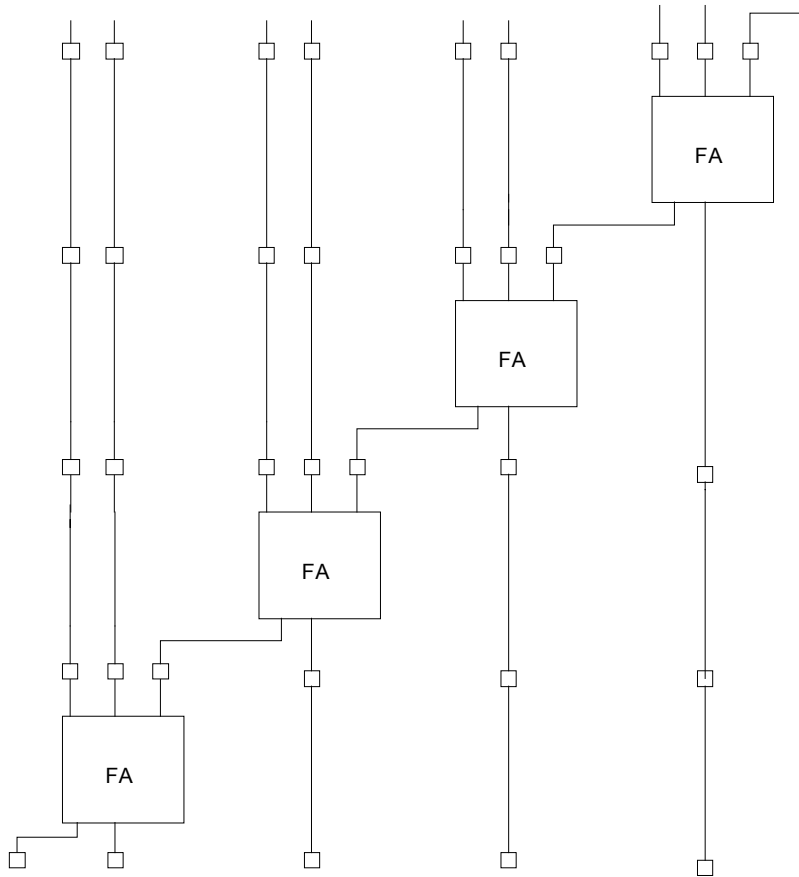
Surge de la segmentación de saltos y otras instrucciones que cambian el PC.

Ejemplos:

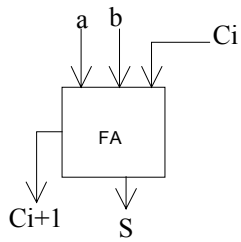
- Sumador segmentado: CPA (Sumador con propagación de Carry)



- Descomponer en etapas. Separar con registros



Calcular el tiempo que se tarda, para cada caso, en realizar 1 suma sabiendo que el tiempo de propagación de una puerta lógica de 2 entradas es "Z" y el de un registro "Z_L".

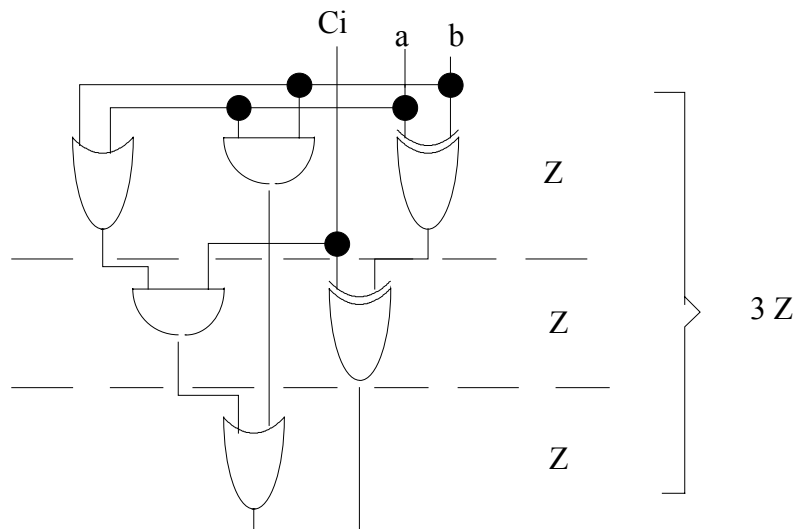


a	b	Ci	S	Ci+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = a \oplus b \oplus C_i$$

Ci \ a \ b	00	01	11	10
0			1	
1	1	0	0	1

$$C_{i+1} = ab + aC_i + bC_i = ab + C_i(a + b)$$



- Sumador sin segmentar: $T_{NS} = 3 Z * 4$

- Sumador segmentado: $T_S = (3 Z + Z_L) 4$

Si queremos realizar 100 sumas:

- $T_{NS} = (3 Z * 4) 100 = 1200 Z$

- $T_S = (3 Z + Z_L) 4 + 99(3 Z + Z_L) = (12 + 297) Z + (99 + 4) Z_L = 309 Z + 103 Z_L$

Si suponemos $Z = Z_L$

- $T_{NS}(100) = 1200 Z$

Aceleración = $T_{NS} / T_S = 1200 / 412 \approx 3$ veces

- $T_S(100) = 412 Z$

* Ejemplo 2:

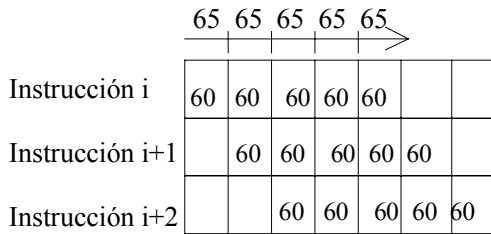
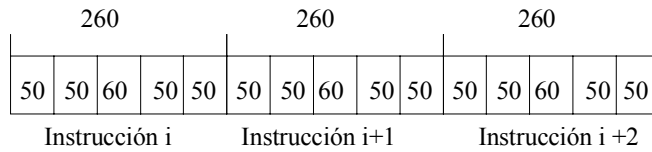
- Tenemos una máquina no segmentada con 5 pasos de ejecución cuya duraciones son 50ns, 50ns, 60ns y 50ns. Suponer que debido al tiempo de preparación y al desplazamiento del reloj (clock Skew), segmentar la máquina añade 5ns de gasto en cada etapa de ejecución.

Calcular la aceleración que conseguimos con la segmentación en la frecuencia de ejecución de las instrucciones.

-Solución:

- No segmentada: Tiempo medio de instrucción = $50 + 50 + 60 + 50 + 50 = 260$ ns

- Segmentada: El reloj debe ir a la velocidad de la etapa más lenta y debe tener en cuenta el gasto -----> $60 + 5 = 65$ ns



$$A = \frac{T \text{ medio por inst. no seg.}}{T \text{ medio por inst. seg.}} = \frac{260}{65} = 4$$

CLASIFICACIÓN DE LA SEGMENTACIÓN:

- Versatilidad

- UNIFUNCIÓN ----> Sumador
- MULTIFUNCIÓN ----> Procesador

- Utilización

- ESTÁTICA : Sólo un tipo de operación en un instante.
- DINÁMICA: Diferentes operaciones en ejecución a la vez.

Nosotros estudiaremos:

SEGMENTACIÓN MULTIFUNCIÓN ESTÁTICA

2.1.2 - DISEÑO DEL CONTROL DE ENTRADA

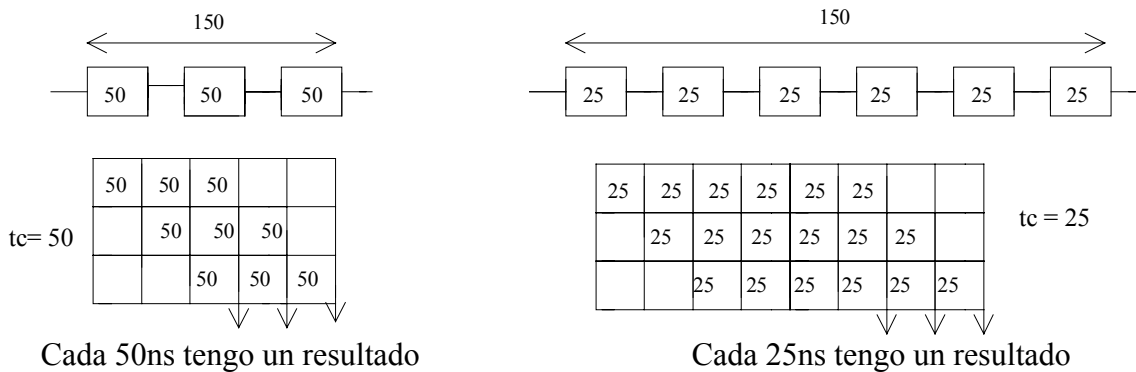
OBJETIVO: Pretendemos optimizar la productividad entendiéndose por productividad (throughput).

$th = 1 / tc$ recordar que tc ----> tiempo de ciclo

$$tc = \frac{\text{tiempo en evaluar una función}}{\text{nº de etapas}}$$

Si nº de etapas aumenta ----> tc disminuye ----> th aumenta

Ejemplo:



Esto se consigue realizando una política de "scheduling" ----> control o planificación de iniciaciones (principio de la evaluación de una función).

Hipótesis a considerar:

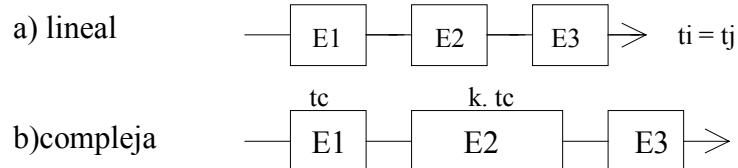
1.- El tiempo de ejecución de cada etapa es múltiplo del tiempo de ciclo.

$$T_{etapa} = K \cdot tc$$

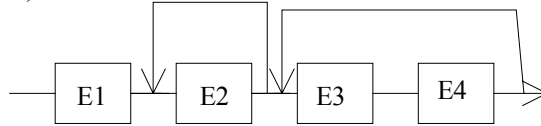
2.- El patrón de uso de las etapas se conoce a priori, es decir, se conoce cuantas veces se reutiliza una etapa, donde esta un dato en cada instante, cuanto tarda cada dato.

Definiciones

- Estructura

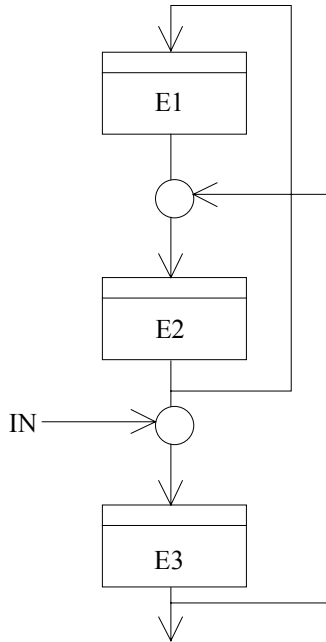


- Tabla de reserva (TR)



En un eje tenemos segmentos de la U.F. y en otro tenemos el tiempo. Nos indica el patrón de uso de las etapas en el tiempo. Es una abstracción del PIPELINE.

Ejem.



E1			A		A		
E2		A		A		A	
E3	A		A				A
	0	1	2	3	4	5	6

➤ Nos indica como funciona la U.F

De una tabla no podemos deducir directamente el pipeline. En este sentido la tabla pierde información.

Ejem.

E1	B	B					B	B
E2			B		B			
E3				B		B		
	0	1	2	3	4	5	6	7

Puede indicar:

- Realimentación
- Etapa que utiliza 2 ciclos

-----> Con la TR no conocemos todo el datapath

1 UF -----> Mas de una TR

1 TR -----> Mas de una UF

LATENCIA: número de unidades de tiempo que separan dos inicializaciones de la tabla de reserva.

COLISIÓN: cuando dos o más inicializaciones quieren usar la misma etapa al mismo tiempo.

LATENCIA PERMITIDA: cuando no hay colisiones en la utilización de las etapas.

LATENCIA PROHIBIDA: cuando provoca colisión.

- Para saber si hay colisiones, se superponen dos TR desplazadas según la latencia.

Ejem.

E1	B	B					B	B
E2			B		B			
E3				B		B		
	0	1	2	3	4	5	6	7

← tiempo de calculo = 8 →

Latencia 3 : Correcta

E1	B	B		B	B		B	B					
E2			B		B	B		B	B				
E3				B		B	B		B	B			
	0	1	2	3	4	5	6	7					

Colisión es secuencia <3,3>

Latencia = 4 ----> Correcta

- La latencia es variable. Hay una SECUENCIA DE LATENCIAS que será cíclica.

- Ejemplo anterior: secuencia de latencia: <3,8,3,8....>
<4,4,4.....>

- LATENCIA MEDIA: (de una secuencia) (\bar{L}): número de unidades de tiempo que separan dos inicializaciones de la tabla de reserva.

Ejem: Latencia media de (3,8) = 5.5

- VELOCIDAD DE INICIACIÓN MEDIA (o frecuencia): $\bar{V} = 1 / \bar{L}$

- UTILIZACIÓN (MEDIA) DE UNA ETAPA:

$$U = \bar{V} * \text{nº marcas en la TR de esta etapa} = \text{nº marcas} / \bar{L}$$

- MAL (Minimun Achievable latency) : latencia mínima posible.

* La salida de una estrategia de control es una secuencia de latencias. Esto define:

- CICLO DE LATENCIA: ocurre cuando una secuencia de latencia repite subsecuencias indefinidamente.

Ejem: secuencia 3,8,3,8,.....
 ciclo = <3,8>

- PERIODO DE UN CICLO: $P = \sum li$ li : latencias que forma el ciclo

- LATENCIA MEDIA DE UN CICLO: $\bar{L} = P / m$ m : n° de latencias

Ejemplo1: Calcular ciclo, periodo, \bar{L} y utilización de cada etapa (UE):

E1			A	A	A	A					A		A		
E2		A	A	A	A	A	A			A		A		A	
E3	A	A	A	A			A	A	A		A				A

$\leftarrow \underset{1}{\leftarrow} \leftarrow \xrightarrow{7} \rightarrow$

ciclo = <1,7>

$UE1 = n^\circ \text{ muestras} / \bar{L} = 2 / 4 = 0.5$

$P = 8$

$UE2 = 3 / 4$

$\bar{L} = 8 / 2 = 4$

$UE3 = 3 / 4$

- Supongamos que empezamos en 3:

E1			A		A	A		A			A		A		
E2		A		A	A	A	A		A	A		A		A	
E3	A		A	A		A	A		A	A	A				A

$\leftarrow \xrightarrow{3} \times \xrightarrow{5} \rightarrow$

ciclo = <3,5>

$P = 8$

-----> igual que antes

$\bar{L} = 8 / 2 = 4$

Ejemplo 2:

E1	B	B		B	B		B	B		B	B
E2			B		B	B		B			
E3				B		B	B			B	

$\leftarrow \begin{array}{c} 3 \\ \leftarrow \end{array} \right\rangle \leftarrow \begin{array}{c} 8 \\ \leftarrow \end{array} \right\rangle$

$$\text{ciclo} = \langle 3, 8 \rangle$$

$$UE1 = 4 / 5.5$$

$$P = 11$$

$$UE2 = 2 / 5.5$$

$$\bar{L} = 11 / 2 = 5.5$$

$$UE3 = 2 / 5.5$$

E1	B	B			B	B	B	B			B	B		
E2			B		B		B		B					
E3				B		B		B		B				

$\leftarrow \begin{array}{c} 4 \\ \leftarrow \end{array} \right\rangle \leftarrow \begin{array}{c} 4 \\ \leftarrow \end{array} \right\rangle$

$$\text{ciclo} = \langle 4, 4 \rangle$$

$$UE1 = 4 / 4 = 1 \rightarrow \text{Etapa completa no podemos ir mas rápido}$$

$$P = 8$$

$$UE2 = 2 / 4$$

$$\bar{L} = 8 / 2 = 4$$

$$UE3 = 2 / 4$$

COTA MÍNIMA DEL MAL

$$MAL \geq \text{MAX} (\text{número de marcas de una fila de la TR (etapa)})$$

Demo:

$M(i) \rightarrow$ n° de marcas de la etapa i

$1 / \bar{L} = \bar{V} \rightarrow$ operaciones por ciclo [velocidad (frecuencia) de iniciación]

ocupación de la etapa (utilización): $U = 1 / \bar{L} \cdot M(i) \leq 1 \quad \forall i$

Por tanto: $\bar{L} \geq M(i) \quad \forall i \rightarrow \bar{L} \geq \text{MAX} (M(i))$

Ejemplo:

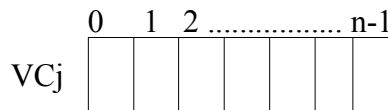
		A		A		
	A		A		A	
A		A				A

MAL ≥ 3
 ↳ n° max. de marcas de una fila

lo que nos viene a decir es: si cada iniciación ocupa n veces una etapa, en media, no podemos empezar una nueva operación hasta n ciclos.

DIAGRAMA DE ESTADOS

- Los estados determinan la ocupación de las etapas según la secuencia de iniciaciones seguida.
- La construcción de los diagramas de estado nos permiten diseñar políticas de control de entradas, intentando conseguir la optima.
- En cada ciclo el pipeline está en un estado concreto.
- Los arcos del grafo indican transacciones, nos llevan al nuevo estado después de un ciclo. Hay dos tipos de arcos, según si iniciamos o no una nueva operación.
- Las secuencias de latencias permitidas se encuentran siguiendo el grafo (secuencia correcta de iniciación).
- El objetivo es encontrar el camino con \bar{L} mínima e implementar el control que siga este camino.
- Los estados los representaremos como VECTORES DE COLISIÓN, que nos indican cual es la ocupación de las etapas según se inicie una operación o no. En un ciclo actualizamos el nuevo estado.
- Tamaño del vector: n° de columnas de la TR.



- Interpretación de las entradas:
 - VCj(i) = ---> 0 ---> De aquí a 'i' ciclos se puede empezar una operación sin colisión.
 - VCj(i) = ---> 1 ---> No se puede empezar una operación en estos 'i' ciclos (hay colisión).

VECTOR DE COLISIÓN INICIAL (VCI)

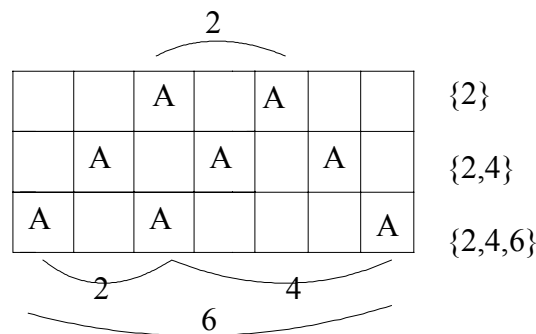
- Es un estado "especial", indica como queda la U.F. después de iniciar la primera operación.

- Algoritmo para obtener VCI:

1.- Iniciamos una operación con la UF descargada ----> TR

2.- Buscamos todas las posiciones que provocan la colisión al desplazar la TR sobre si misma ----> LATENCIAS PROHIBIDAS (Buscar las distancias entre marcas de una misma etapa).

- Ejem:



- Como las UF a estudiar son estáticas ---> latencia '0' prohibida por tanto, latencias prohibidas: {0,2,4,6}

VCI =

1	0	1	0	1	0	1
---	---	---	---	---	---	---

└─ A partir de aquí siempre podemos empezar

GENERACIÓN DEL DIAGRAMA DE ESTADOS

- Partir del VCI.

- Pasado un ciclo el vector se desplaza 1 posición a la izquierda y se añade un '0'.

— $X_0X_1X_2X_3X_4X_5$ ----> $X_1X_2X_3X_4X_50$

- Si el primer bit (X_1) es 1 ----> nuevo VC

Sino ($X_1=0$) podemos iniciar una nueva operación ---> 2 VC ----> 2 arcos.

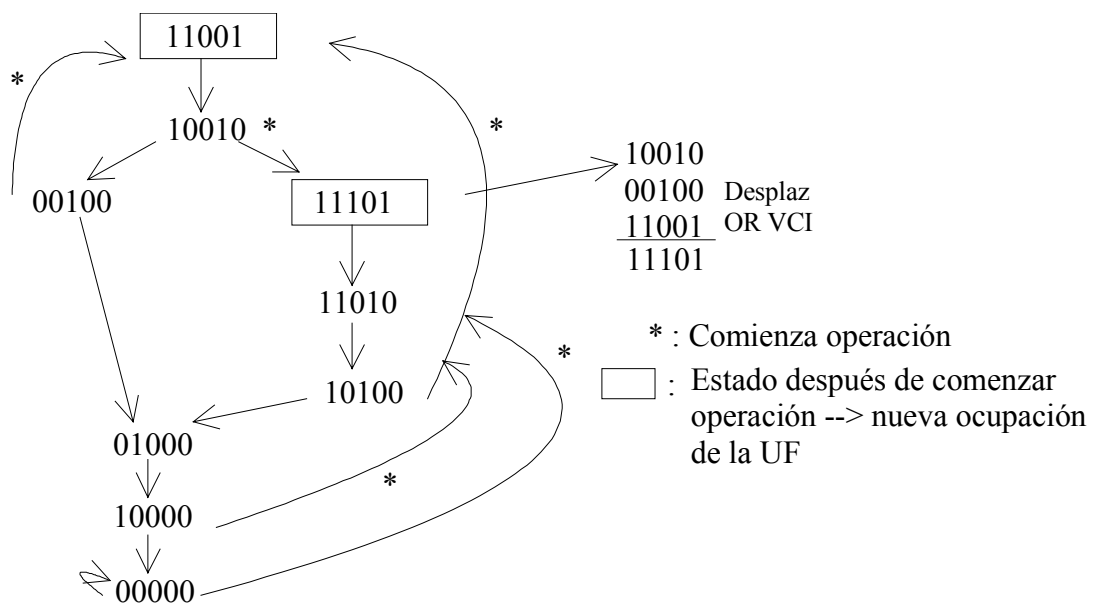
casos:

(1): No inicia operación ---> $0 X_2 X_3 X_4 X_5 0$ Nuevo VC

(2): Inicia operación ----> hacer una OR con el VCI ---> nos da el nuevo estado (nueva ocupación de la UF).

- Repetir hasta que no salga ningún estado nuevo.

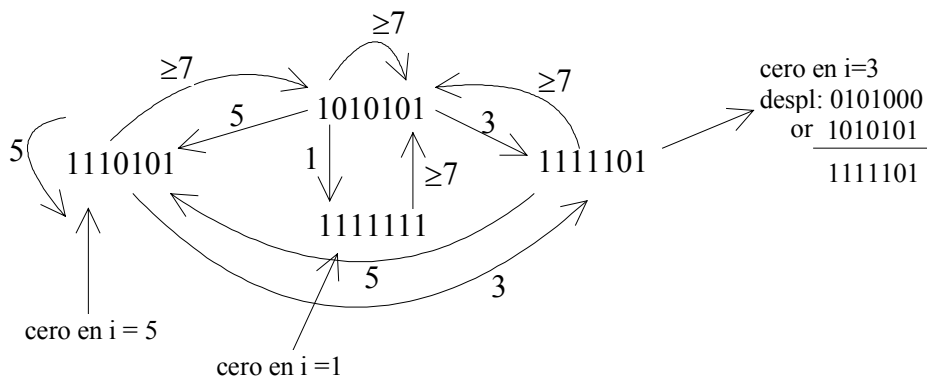
Ejemplo: VCI = 11001



Ejemplo: VCI = 1010101

- Partimos del VCI.
- Para cada $VC_j(i) = 0 \rightarrow$
 - 1.- Desplazamos i posiciones a la izquierda el vector de colisión.
 - 2.- Añadimos i ceros a la derecha.
 - 3.- Hacemos OR con VCI.
 - 4.- Etiquetamos el arco con i .
- Repetir
- Añadir un arco hacia el inicial con etiqueta (" $\geq n^\circ$ bits del VCI).

Ejem: 1010101



ANÁLISIS DEL DIAGRAMA DE ESTADOS

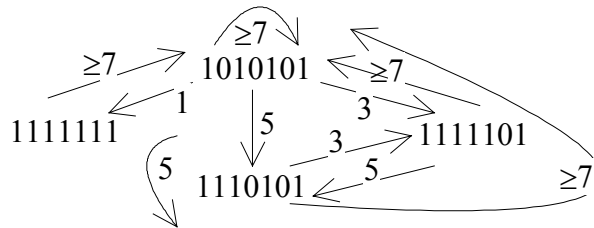
- El diagrama de estados representa todas las posibles secuencias de iniciación válidas.
- OBJETIVO: Encontrar la secuencia óptima (MAL).
- Con un número finito de estados, todas las secuencias de iniciación forman ciclos.
- Si hay pocos estados se puede hacer una búsqueda exhaustiva de todos los ciclos y coger el de \bar{L} MIN.
- Podemos distinguir dos tipos de ciclos:
 - (i) SIMPLES : Sólo se pasa una vez por un estado.
 - (ii) COMPUESTOS: Se pasa más de una vez por algún estado.

LEMA: Si en un diagrama de estados reducido existe un ciclo con latencia media \bar{L} , existe, como mínimo, un ciclo simple con latencia media $\leq \bar{L}$.

Demo:

- Un ciclo o es simple o está formado por ciclos simples; si la velocidad media es $V = \bar{L} / n^\circ$ iniciaciones, o todo ciclo ha ido a esta velocidad o algún ciclo simple ha ido a menos velocidad y otros a mas. -----> sólo hace falta buscar los ciclos simples.

- Ejemplo



ciclos simples:	\bar{L}	
<7>	7	
<1,7>	4 ----->	
<5>	5	MAL = 4
<3,7>	5	
<3,5>	4 ----->	
<5,3,7>	5	
<5,7>	6	

		A		A		
	A		A		A	
A		A				A

MAL \geq 3
 4
 ↓
 MAL > 3
 ↓
 No estamos aprovechando al 100 % las etapas

SUBCONJUNTO DE CICLOS SIMPLES -----> GREEDY

- Entre los ciclos simples hay un subconjunto muy sencillo de conseguir: CICLOS GREEDY.

- Son ciclos donde el cambio de estado se hace siempre siguiendo el arco de menor lantecia.

- Como obtener todos los posibles ciclos de Greedy:

- 1.- Escoger un estado del grafo modificado (p.ej. el VCI).

$$X(1) \leq X(m) - L(m) + \alpha \quad \text{--->} \quad X(1) \leq X(1) - \sum_{i=1}^m L(i) + m\alpha \quad \text{----->}$$

$$\alpha \geq \frac{\sum_{i=1}^m L(i)}{m} \quad \text{<----->} \quad \alpha \geq \bar{L} \text{ ciclo de Greedy}$$

Por definición:

$$\text{MAL} = \text{MIN} (\bar{L} \text{ ciclos simples})$$

$$\text{MAX} (\text{n}^\circ \text{ marcas etapa}) \leq \text{MAL} \leq \bar{L} \text{ ciclos greedy} \leq \text{n}^\circ \text{ 1's en el VCI}$$

Ejem:

Tabla A: $3 \leq \text{MAL} \leq 4 \leq 4$

Tabla B: $4 \leq \text{MAL} \leq 4 \leq 6$

MODIFICACIÓN DE LA TABLA DE RESERVA

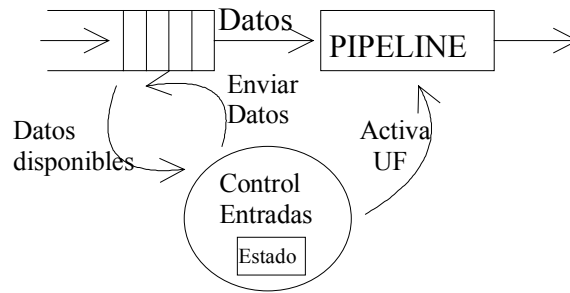
- Si a partir de una TR no se llega a la cota mínima (a usar al 100% alguna etapa) se puede intentar modificar la TR añadiendo retardos (etapas que no hacen nada) que no modifiquen el número de marcas por fila (--> no modifica la cota inferior) pero modifican el VCI -----> puede salir un ciclo óptimo mejor.

- Para no modificar el cálculo las etapas se tienen que usar en el mismo orden.

- Conseguimos el mínimo a cambio de hacer más largo el tiempo de cálculo total de un resultado ----> si el cálculo se hace pocas veces no vale la pena modificar la TR.

2.1.3 - HARDWARE DE CONTROL

El objetivo final es diseñar el controlador de una U.F. segmentada para conseguir el máximo rendimiento.



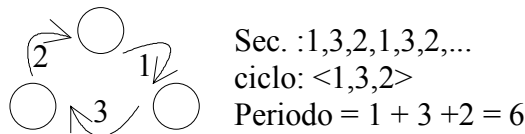
El diseño del control dependerá de:

- Tabla de reserva de la UF y estado en que se encuentra.
- Disponibilidad de datos en su entrada.
 - **A** ----> Datos siempre disponibles.
 - **B** ----> Datos no siempre disponibles.

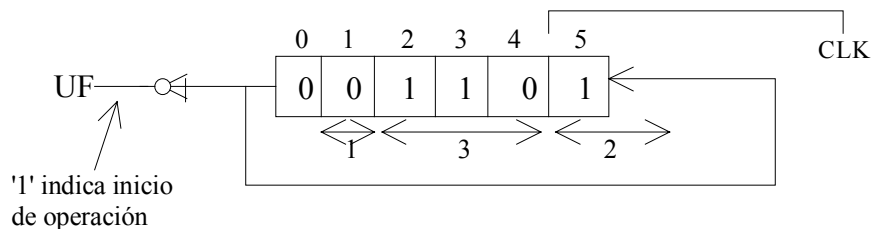
A ----> Datos siempre disponibles (petición continua)

- Buscar ciclo óptimo -----> \bar{L}_{MIN} (MAL)
- Siempre seguiremos este ciclo, por eso sólo necesitamos un registro que rotará una posición por ciclo. El primer bit indicará si se puede iniciar una operación o no.

0 -----> iniciar
 1 -----> no iniciar



- El tamaño del registro será el periodo.
- El bit '0' del registro siempre será '0' -----> se puede iniciar siempre operación



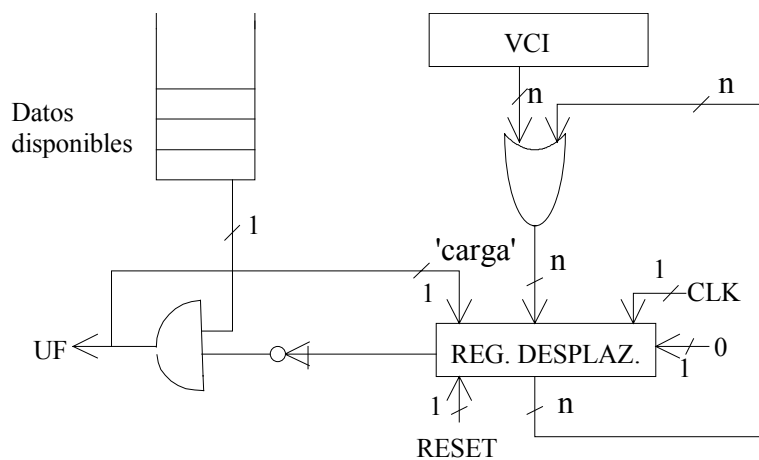
B ----> Datos no siempre disponibles

- El control anterior no será eficiente ya que sólo deja entrar datos en instantes fijos, independientemente de si hay datos o no.

- Como no sabemos cuando pueden llegar datos, seguiremos una política de greedy, intentaremos satisfacer una petición lo antes posible (ya que no sabemos si llegarán más datos).

- El controlador es simular al proceso seguido para generar el grafo de estados originales. Parte del VCI y desplaza un bit por ciclo, según si tiene datos disponibles y puede empezar. Hace una OR con el VCI.

NOTA: si se puede realizar una iniciación (bit a '0') pero no hay datos disponibles el registro de desplazamiento no se modifica.



Secuencia: (Inicialmente RESET y se pone a '0' ---> puede iniciar)

- (i) Al inicio del ciclo se hace el desplazamiento a la izquierda
- (ii) Al final se cargan los 'n' bits si se ha iniciado una operación.

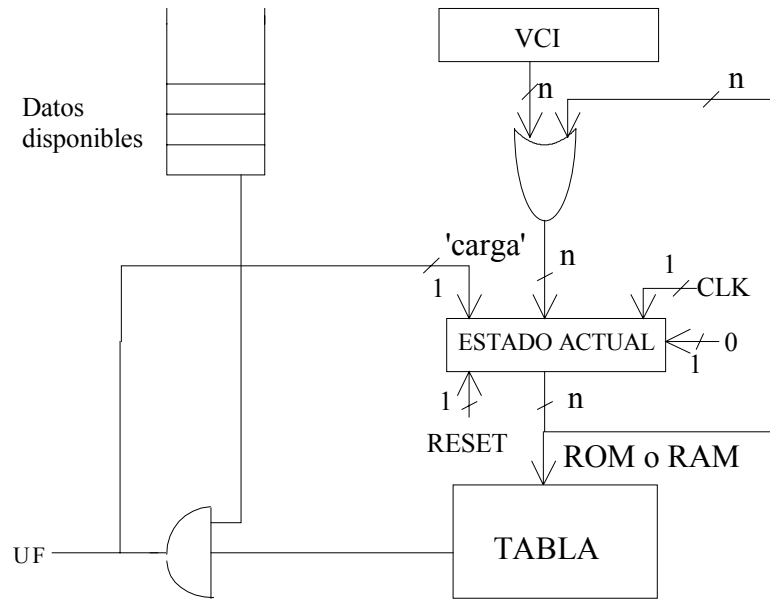
- Presenta un problema: se puede entrar en ciclos greedy muy malos.

- No garantiza un buen rendimiento, sólo será óptimo si $n^{\circ} \text{ marcas} = \text{ciclo greedy}$
 fila max

- SOLUCIÓN:

- (1) - Modificar la TR para que todos los ciclos greedy sean óptimos
- (2) - Mejorar el controlador con una tabla que indique si interesa seguir un ciclo o no. Así aunque se pueda empezar una operación, evitaremos los ciclos malos. -----> GUÍA A CICLOS ÓPTIMOS.

- El vector de colisión actual servirá como dirección a una tabla que indicará si iniciar o no. Intentará (y conseguirá) mantener el estado dentro de la rama donde está el ciclo óptimo.



la decisión la toma la tabla y no el estado.

Ejercicios: 1,2,3(1),4,22,23.

TEMA 3 - PROCESADORES SEGMENTADOS

3.1 - INTRODUCCIÓN A LOS PROCESADORES SEGMENTADOS

Aplicación de la segmentación al proceso de ejecución de una instrucción.

OBJETIVO: Solapar (conurrencia) en el espacio la ejecución de un algoritmo secuencial para poder ejecutar 1 instrucción por ciclo.

- programación secuencial.
- Ejecución concurrente transparente al programador.

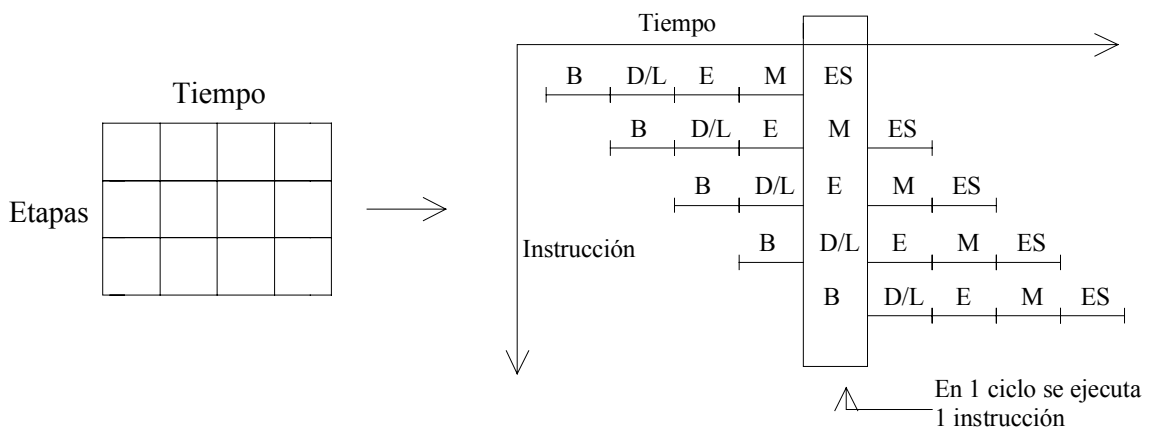
Identificación de las etapas: (pensado para máquina de 3 direcciones)

- Búsqueda de la instrucción.
- Decodificación de la instrucción y búsqueda de registros (lectura).
- Ejecución (y cálculo de dirección efectiva).
- Memoria (acceso) --> accede a memoria si es necesario. Carga o almacena.
- Escritura (o postescritura write-back) --> Escribe el resultado en el registro destino.

(separación de etapas orientadas a RISC)

NOTACIÓN:

Pasaremos de:



PROBLEMAS : (que hacen que no se llegue a una instrucción por ciclo)

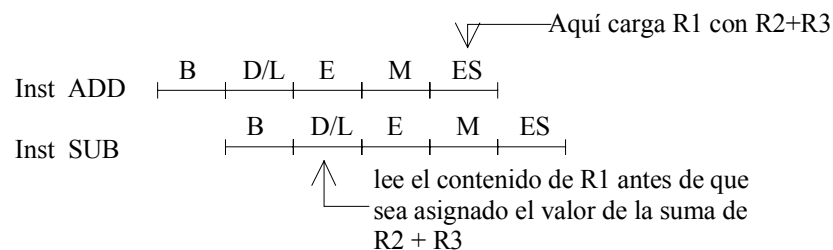
- Riesgos estructurales: surgen cuando alguna combinación de instrucciones no se puede acomodar debido a conflictos de recursos.

Ejem: si tenemos una única memoria para datos e instrucciones no se podrá iniciar una búsqueda de un dato y una instrucción en el mismo ciclo.

- Riesgos de datos: se presentan cuando el orden de acceso a los operandos los cambia la segmentación, con relación al orden normal que se sigue en las instrucciones que se ejecutan secuencialmente.

Ejem:

ADD R1,R2,R3
SUB R4,R1,R5



- Riesgos de control: surgen de la segmentación de los saltos y otras instrucciones que cambian el PC.

TIPOS DE PROCESADORES SEGMENTADOS

- 1.- Procesadores segmentados lineales.
 - 2.- Procesadores con operaciones multiciclo.
 - 3.- Procesadores avanzados - SCOREBOARD
- TOMASULO
- * Evolución RISC ----> CISC ----> RISC

MEDIDA DE LAS PRESTACIONES DE UN PROCESADOR

- Nos interesa poder medir el rendimiento de un sistema, así como poder comparar sistemas diferentes, por eso introduciremos unos pocos conceptos que nos permitirán hablar de todo esto.

- Tiempos (usuarios)
- Tiempo de respuesta
- Tiempo de ejecución
- Tcpu

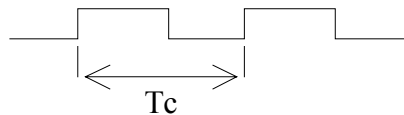
Rendimiento

- Throughput -----> (Productividad th = 1 / Tc)
(centro de cálculo)

- Desde el punto de vista de AC nos interesa reducir el tiempo de ejecución de un programa.

VELOCIDAD DE LA CPU

- Típica medida; se mide por el tiempo de ciclo de reloj o por la frecuencia.



$$\text{Frec (Mhz)} = \frac{1}{Tc * 10^6}$$

$$\text{Frec} = \frac{1}{Tc(\text{seg})} \text{ Hz} \text{ -----> } 1\text{Hz} = \frac{1\text{ciclo}}{\text{seg}} \text{ -----> } 1\text{Mhz} = \frac{10^6 \text{ ciclos}}{\text{seg}} = \frac{1\text{ciclo}}{1\text{Mseg}}$$

lo que realmente queremos minimizar es:

$$T_{cpu} = n^{\circ} \text{ instrucciones} * \text{ciclos por instrucción} * Tc(\text{seg})$$

- Es una medida dinámica; depende de cada programa y cada ejecución.
- Es difícil de optimizar, reduciendo cualquiera de los componentes podemos empeorar otra: las componentes están ligadas.

$$T_{cpu} = I * CPI * Tc$$

- I: f(compiladores, lenguaje máquina)
- CPI: f(lenguaje máquina y organización)
- Tc: f(organización. Hardware ---> Tecnología)

COMPARACIÓN RISC - CISC

	RISC	CISC
CPI	1.25	4.5
I	1.2M	M

MÉTRICAS COMERCIALES

MIPS: Millones de instrucciones por segundo.

$$\text{MIPS} = \frac{n^{\circ} \text{instruc. totales ejecutadas}}{T_{\text{cpu.seg}} * 10^6} = \frac{I}{I * \text{CPI} * T_{\text{ciclo}} * 10^6} \frac{\text{Frec. Mhz}}{\text{CPI}}$$

- Ejem: un procesador ideal (CPI = 1) con un reloj de 25 Mhz -----> 25 MIPS

- ¡No permite comparar máquinas diferentes !

MFLOPS: Millones de operaciones en punto flotante por segundo.

$$\text{MFLOPS} = \frac{n^{\circ} \text{operac. punto. flotante de un programa}}{\text{Tiempo de ejecución} * 10^6}$$

- Las dos medidas son dinámicas, dependen del programa y de la máquina.

- Para comparar máquinas diferentes se deben ejecutar los mismos programas en las máquinas. Programas utilizados: - linpack y -Bechmarks

ACELERACIÓN (SPEEDUP)

- "X es un n % más rápido que Y".

$$\text{Velocidad X} = \frac{1}{\text{Tiempo. Ejec. X}}$$

$$V_x = V_y + \frac{n}{100} V_y \quad \text{----->} \quad n = \frac{V_x - V_y}{V_y} * 100$$

$$V_x = 1 / T_x = \frac{1}{T_y} + \frac{n}{100} \frac{1}{T_y} \quad \text{---->} \quad 1 / T_x = \frac{1}{T_y} \left(1 + \frac{n}{100} \right)$$

$$\frac{T_y}{T_x} = 1 + \frac{n}{100} = S \quad \text{----->} \quad \text{SPEED UP}$$

$$V_x = S \cdot V_y \text{ -----} \rightarrow S = \frac{V_x}{V_y} = \frac{T_y}{T_x}$$

$$\frac{n}{100} = \frac{V_x - V_y}{V_y} = \frac{V_x}{V_y} - 1 = S - 1 \rightarrow S = 1 + \frac{n}{100}$$

LEY DE AMDAHL

$$S = \frac{T_{original}}{T_{con.mejora}}$$

- Nos da el Speed-up total cuando modificamos (mejoramos) una parte de un computador/procesador.

- Depende de:

- Fracción de tiempo de ejecución donde se puede aplicar la mejora (fm).

- Sm : Speed-up que conseguiremos si todo el tiempo vamos con la parte mejorada.

$$S_{total} = \frac{1}{(1 - fm) + \frac{fm}{S_m}}$$

Límites:

$$S_{total} = 1 / 1 - fm \quad \text{-----} \rightarrow \quad S_{total} = 1 / 1 - fm$$

$$S_m \rightarrow \infty$$

- El modo más lento limita el rendimiento total aunque el modo más rápido sea ∞ entre más rápido.

$$S_{total} = S_m = T_s / T_m$$

$$fm \rightarrow 1$$