



Tema 2. CPU: Introducción

Organización de Computadores

LUIS ENRIQUE MORENO LORENTE
RAÚL PÉRULA MARTÍNEZ
ALBERTO BRUNETE GONZALEZ
DOMINGO MIGUEL GUINEA GARCIA ALEGRE
CESAR AUGUSTO ARISMENDI GUTIERREZ
JOSÉ CARLOS CASTILLO MONTOYA

Departamento de Ingeniería de Sistemas y Automática





“NUEVAS” VS “VIEJAS” ARQUITECTURAS/ISA

Debate tradicional en arquitectura de computadores:
RISC vs. CISC

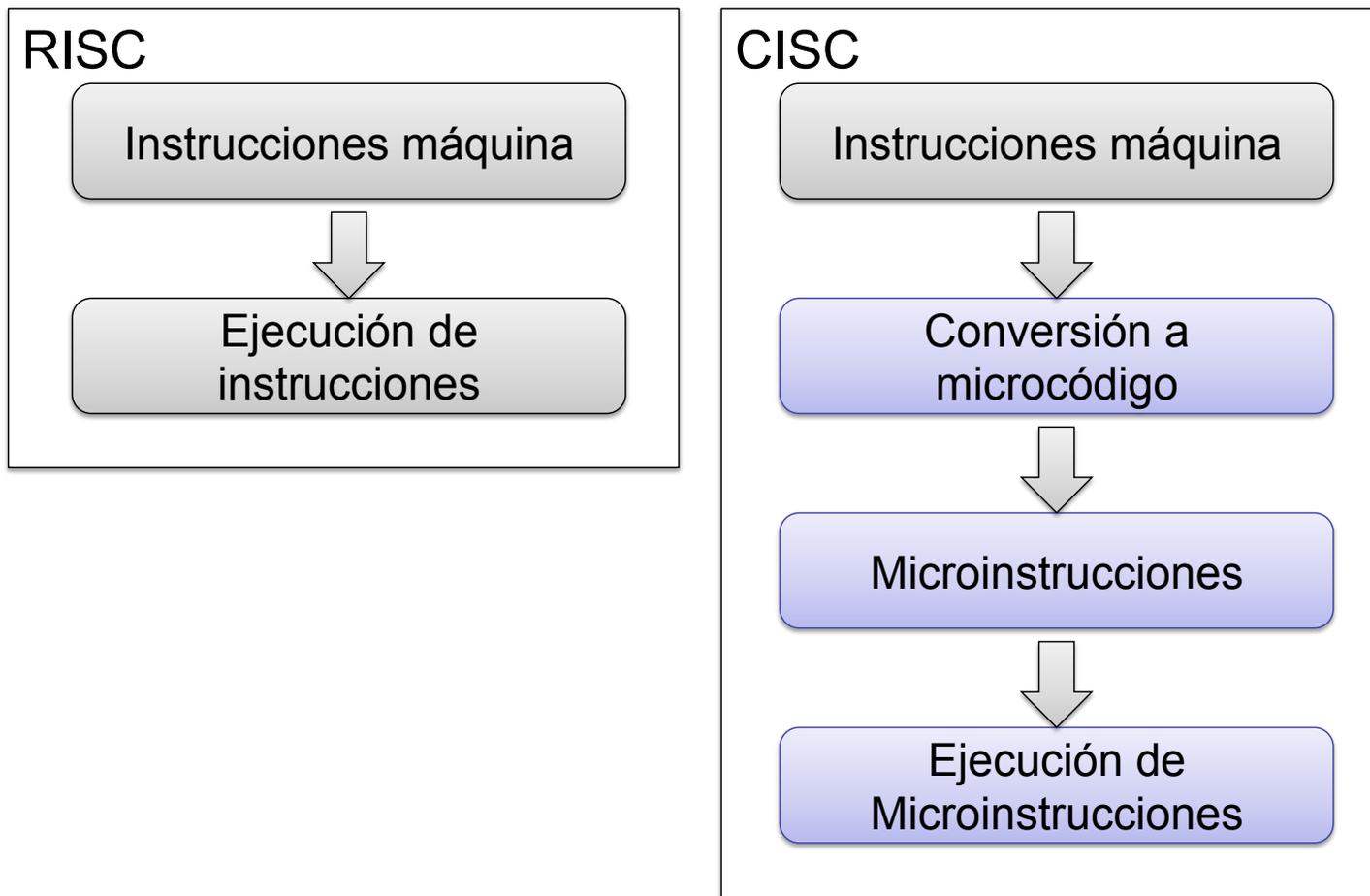
RISC
Reduced
Instruction Set
Computer

CISC
Complex
Instruction Set
Computer





“NUEVAS” VS “VIEJAS” ARQUITECTURAS/ISA





ARQUITECTURAS CISC

- Muchas instrucciones complejas.
- Instrucciones de longitud variable.
- Operaciones tipo Memoria a registro.
- Pocos registros “generales/implícitos” (8).
- Opcionalmente.
 - Arquitectura 32/64 bit.
 - 2-3 direccionamientos.
 - Varios modos de direccionamiento en las instrucciones load/store:
 - LD effective address
- No usa saltos retardados (Delayed branch).





ARQUITECTURAS RISC

- Número limitado de instrucciones simples.
- Instrucciones de longitud fija (32 bit) + codificación con campos fijos.
- Operaciones Registro a registro.
 - Arquitectura Load/Store.
- Alto número de registros de propósito general (32).
- Opcionalmente.
 - Arquitectura de 64bits.
 - 3 direccionamientos: registro, inmediato, desplazamiento.
 - Un único modo de direccionamiento para los load/store: base + desplazamiento.
 - Instrucciones aritméticas tipo reg-reg de 3-direcciones.
- **Delayed branch (salto retardado).**

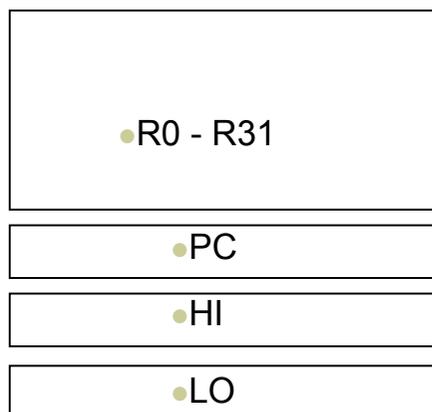




ARQUITECTURAS RISC: EJEMPLO MIPS

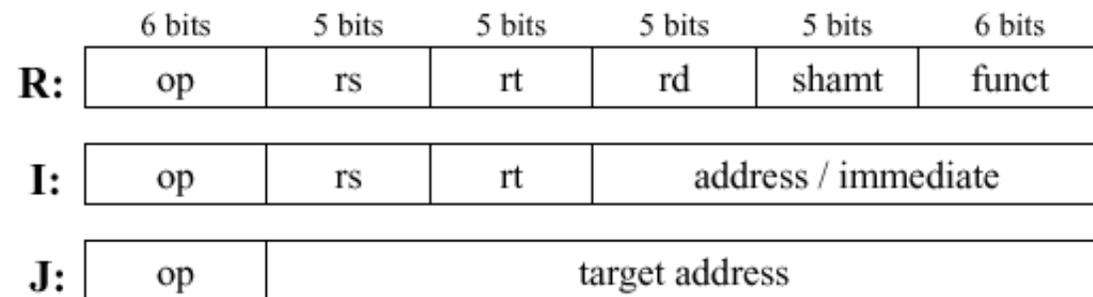
- Instrucciones (tipos)
 - Load/Store
 - Cómputo (ALU)
 - Jump / Branch
 - Coma flotante
 - coprocesador
 - Gestión de memoria
 - Especiales

Registros



Formato de instrucciones

- 3 formatos
- Longitud fija de 32 bit



op: basic operation of the instruction (opcode)
 rs: first source operand register
 rt: second source operand register
 rd: destination operand register
 shamt: shift amount
 funct: selects the specific variant of the opcode (function code)
 address: offset for load/store instructions ($\pm 2^{15}$)
 immediate: constants for immediate instructions

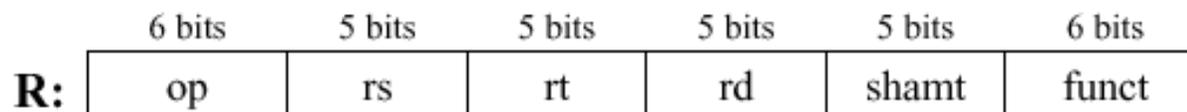
Imagen de Mafla, 2001
 Imagen de Mafla, 2001





ARQUITECTURAS RISC: EJEMPLO MIPS

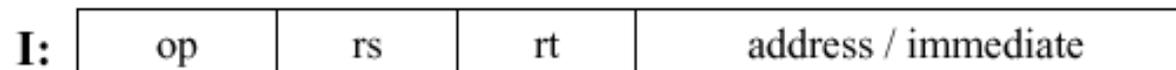
- Tipo R
 - Este tipo de instrucciones es el usado en las operaciones ALU registro - registro.
 - Tienen los primeros seis bits a 0, y los últimos seis bits (campo funct) codifica la operación aritmética.
 - Campos de la instrucción:
 - Op: Código de operación.
 - Rs: Primer registro operando fuente.
 - Rt: Segundo registro operando fuente.
 - Rd: Registro operando destino, donde se almacena el resultado de la operación.
 - Shamt (Shift Amount): Desplazamiento para las instrucciones de tipo Shift.
 - Funct: Función. Completa el OpCode para seleccionar el tipo de instrucción del que se trata.





ARQUITECTURAS RISC: EJEMPLO MIPS

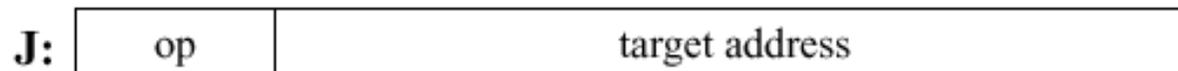
- Tipo I
 - Se trata de instrucciones que necesitan incorporar un operando inmediato, es decir, aritmético-lógicas, direccionamientos a memoria y saltos.
 - Campos de la instrucción:
 - Op: Código de la operación.
 - Rs: Registro fuente.
 - Rt: Registro destino.
 - Immediate: Operando inmediato o desplazamiento en direccionamientos a memoria u offset relativo al PC en los saltos.





ARQUITECTURAS RISC: EJEMPLO MIPS

- Tipo J
 - Usadas en las operaciones de salto incondicional.
 - Campos de la instrucción:
 - OpCode: Código de operación.
 - Offset: Offset relativo al PC.
 - Se trata de instrucciones que necesitan incorporar un operando inmediato, es decir, aritmético-lógicas, direccionamientos a memoria y saltos.





ARQUITECTURAS RISC: EJEMPLO MIPS64

Categoría	Nombre	Sintaxis de la instrucción	Significado	Formato/codop/codfunc		Notas	
Aritméticas	Suma	add \$1,\$2,\$3	$\$1 = \$2 + \$3$ (con signo)	R	0	20_{16}	suma dos registros
	Suma sin signo	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$ (sin signo)	R	0	21_{16}	
	Resta	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$ (con signo)	R	0	22_{16}	resta dos registros
	Suma inmediata	addi \$1,\$2,CONST	$\$1 = \$2 + \text{CONST}$ (con signo)	I	8_{16}		empleado para sumar constantes (y también para copiar de un registro a otro "addi \$1, \$2, 0")
	Suma inmediata sin signo	addiu \$1,\$2,CONST	$\$1 = \$2 + \text{CONST}$ (sin signo)	I	9_{16}		
	Multiplicación	mult \$1,\$2	$\text{LO} = ((\$1 * \$2) \ll 32) \gg 32$; $\text{HI} = (\$1 * \$2) \gg 32$;	R	0	18_{16}	Multiplica dos registros y guarda el resultado de 64 bits en dos puntos especiales de la memoria - LO y HI. De forma alternativa, uno puede decir que el resultado de esta operación es: (int HI,int LO) = (64 bits) $\$1 * \2 .
	División	div \$1, \$2	$\text{LO} = \$1 / \2 $\text{HI} = \$1 \% \2	R			Divide dos registros y guarda el resultado entero de 32 bits en LO y el resto en HI. ⁵

Imagen de Wikipedia, 2014





ARQUITECTURAS RISC: EJEMPLO MIPS64

Categoría	Nombre	Sintaxis de la instrucción	Significado	Formato/codop/codfunc	Notas
Lógicas	And	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	R	And bit a bit
	And con inmediato	andi \$1,\$2,CONST	$\$1 = \$2 \& \text{CONST}$	I	
	Or	or \$1,\$2,\$3	$\$1 = \$2 \$3$	R	Or bit a bit
	Or con inmediato	ori \$1,\$2,CONST	$\$1 = \2CONST	I	
	Or exclusivo	xor \$1,\$2,\$3	$\$1 = \$2 \wedge \$3$	R	
	Nor	nor \$1,\$2,\$3	$\$1 = \sim(\$2 \$3)$	R	Nor bit a bit
	Inicializar si menor que	slt \$1,\$2,\$3	$\$1 = (\$2 < \$3)$	R	Comprueba si un registro es menor que el otro.
Inicializar si menor que con inmediato	slti \$1,\$2,CONST	$\$1 = (\$2 < \text{CONST})$	I	Comprueba si un registro es menor que una constante.	

Imagen de Wikipedia, 2014





ARQUITECTURAS RISC: EJEMPLO MIPS64

Categoría	Nombre	Sintaxis de la instrucción	Significado	Formato/codop/codfunc		Notas
Transferencia de datos	Carga de dirección	la \$1, Etiqueta	\$1 = Dirección de memoria	I		Carga la dirección de memoria de una etiqueta.
	Carga de palabra	lw \$1,CONST(\$2)	\$1 = Memoria[\$s2 + CONST]	I	23 ₁₆	Carga la palabra almacenada desde (\$s2+CONST) en adelante (3 bytes más).
	Carga de media palabra	lh \$1,CONST(\$2)	\$1 = Memoria[\$s2 + CONST]	I	25 ₁₆	Carga la media palabra almacenada desde (\$s2+CONST) en adelante (1 byte más).
	Carga de byte	lb \$1,CONST(\$2)	\$1 = Memoria[\$s2 + CONST]	I		Carga el byte almacenado en (\$s2+CONST).
	Almacenamiento de palabra	sw \$1,CONST(\$2)	Memoria[\$s2 + CONST] = \$1	I		Almacena una palabra en (\$s2+CONST) y los siguientes 3 bytes. El orden de los operandos es una gran fuente de confusiones.
	Almacenamiento de media palabra	sh \$1,CONST(\$2)	Memoria[\$s2 + CONST] = \$1	I		Almacena la primera mitad de un registro (media palabra) en (\$s2+CONST) y el byte siguiente.
	Almacenamiento de byte	sb \$1,CONST(\$2)	Memoria[\$s2 + CONST] = \$1	I		Almacena el primer byte de un registro en (\$s2+CONST).
	Carga del inmediato superior	lui \$1,CONST	\$1 = CONST << 16	I		Carga un operando inmediato de 16 bits en los 16 bits del registro especificado. El valor máximo de la constante es 2 ¹⁶ -1
	Mover desde "high"	mfhi \$1	\$1 = HI	R		Mueve un valor de HI al registro. No se debe emplear una instrucción de multiplicación o división entre dos instrucciones mfhi (esta acción no está definida debido a la segmentación del MIPS).
	Mover desde "low"	mflo \$1	\$1 = LO	R	0	12 ₁₆ Mueve un valor de LO al registro. No se debe emplear una instrucción de multiplicación o división entre dos instrucciones mflo (esta acción no está definida debido a la segmentación del MIPS)

Imagen de Wikipedia, 2014





ARQUITECTURAS RISC: EJEMPLO MIPS64

Categoría	Nombre	Sintaxis de la instrucción	Significado	Formato/codop/codfunc		Notas
Desplazamiento de bits	Desplazamiento lógico a la izquierda	sll \$1,\$2,CONST	$\$1 = \$2 \ll \text{CONST}$	R		Desplaza el registro CONST bits a la izquierda (lo multiplica por 2^{CONST})
	Desplazamiento lógico a la derecha	srl \$1,\$2,CONST	$\$1 = \$2 \gg \text{CONST}$	R		Desplaza el registro CONST bits a la derecha rellenado con ceros (divide entre 2^{CONST}). Nótese que esta instrucción sólo funciona como división de un número en complemento a 2 si dicho número es positivo.
	Desplazamiento aritmético a la derecha	sra \$1,\$2,CONST	$\$1 = \$2 \ggg \text{CONST} + \left(\left(\sum_{n=1}^{\text{CONST}} 2^{31-n} \right) \cdot \$2 \ggg 31 \right)$	R		Desplaza el registro CONST bits rellenando con el bit de signo correspondiente (divide un número en complemento a 2 entre 2^{CONST})
Saltos condicionales	Salto si igual	beq \$1,\$2,CONST	if ($\$1 == \2) go to PC+4+CONST	I	04 ₁₆	Salta a la instrucción situada en la dirección especificada si ambos registros son iguales.
	Salto si no igual	bne \$1,\$2,CONST	if ($\$1 \neq \2) go to PC+4+CONST	I	05 ₁₆	Salta a la instrucción situada en la dirección especificada si ambos registros <i>no</i> son iguales.
Salto incondicional	Salto	j CONST	goto address CONST	J		Salta de forma incondicional a la instrucción almacenada en la dirección especificada.
	Salto a registro	jr \$1	goto address \$1	R		Salta a la dirección almacenada en el registro especificado.
	Salto y enlace	jal CONST	$\$31 = \text{PC} + 4$; goto CONST	J		Utilizada en las llamadas a subrutinas. Guarda en \$31 la dirección de retorno, a la que se vuelve con jr \$31

Imagen de Wikipedia, 2014



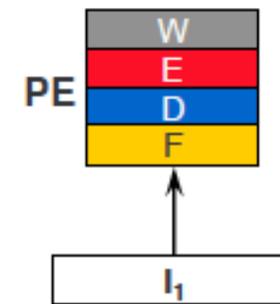
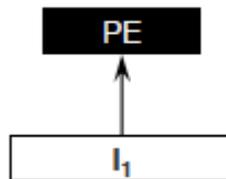
PARALELISMO A NIVEL DE INSTRUCCIÓN (ILP): EVOLUCIÓN

Etapas:

- Búsqueda (Fetch)
- Descodificación (Decode) y lectura de operandos
- Ejecución (Execute)
- Escritura del resultado (Writeback)

Esquema básico

- Processing element (PE)
- Contiene todas las etapas
- Procesa las instrucciones secuencialmente



Segmentación

- Solapamiento temporal de la ejecución de las instrucciones.
- La unidad de procesamiento se divide en varias unidades funcionales que operan en paralelo
- Cada etapa procesa una fase de la instrucción.
- Las instrucciones se ejecutan secuencialmente



UNIDAD CENTRAL DE PROCESO

- Se encarga de ejecutar los programas almacenados en la memoria, para lo cual, realiza repetitivamente los siguientes procesos:
 - Lectura de la instrucción (fetch)
 - Descodificación (interpretación)
 - Lectura operandos (datos)
 - Ejecución
 - Escritura del resultado
 - en memoria
 - en el banco de registros



UNA PRIMERA IDEA

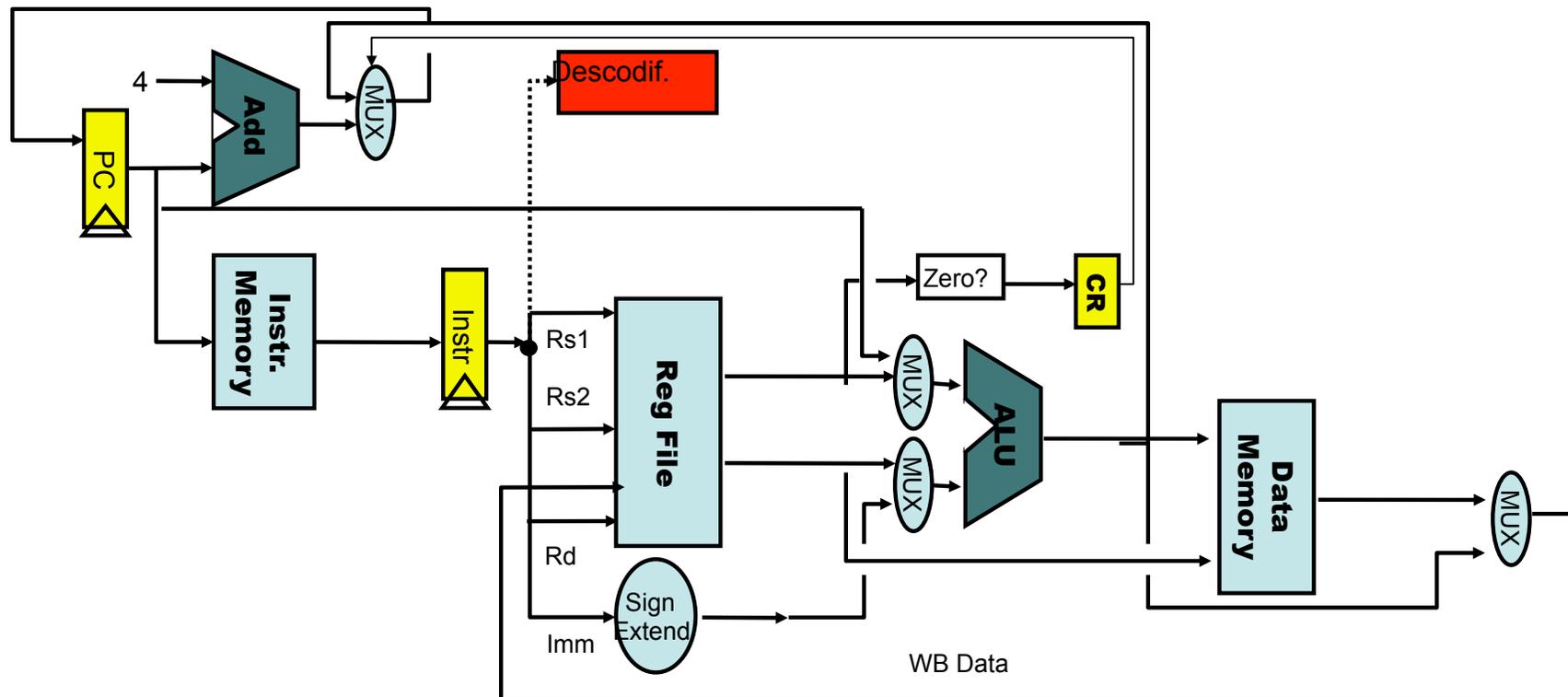


Imagen adaptada de Wikipedia, 2014

BÚSQUEDA DE INSTRUCCIÓN (IF)

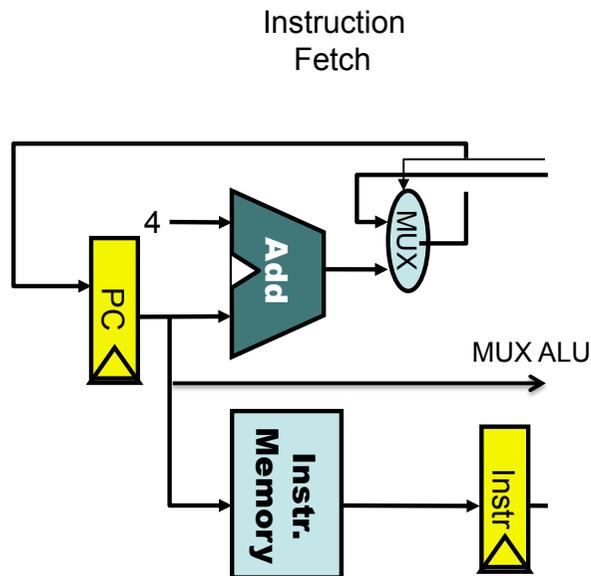


Imagen adaptada de Wikipedia, 2014

- Se busca la instrucción a la que apunta el PC en la caché de instrucciones.
 - Se lleva al registro de instrucción.
- Se lleva el PC al latch de interfaz entre etapas.
 - Se incrementa el PC en 4.
 - Se lleva de nuevo al PC.
 - Salvo que una instrucción previa de transferencia de control salto requiera su modificación, el multiplexor elije en este caso la dirección objetivo del salto.

DESCODIFICACIÓN Y LECTURA DE OPERANDOS (ID)

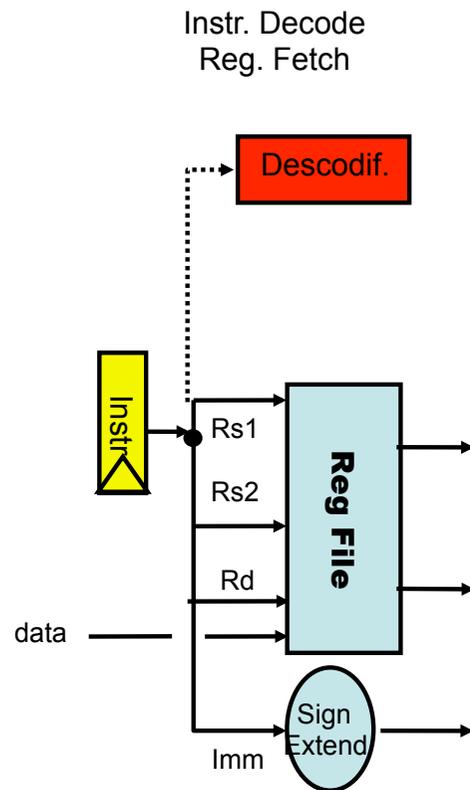


Imagen adaptada de Wikipedia, 2014

- Se decodifica la instrucción en la 1ª mitad del ciclo.
- Dependiendo del tipo de instr. se realiza lo siguiente en la 2ª mitad del ciclo:
- Registro-registro (aritméticas/lógicas)
 - Se leen los registros Rs1 y Rs2 del banco de registros que se llevarán a la ALU.
- Referencia a memoria (load/store)
 - Se lee el registro base de la dirección de memoria y se lleva a la entrada 1 de la alu.
 - Se extiende el signo al desplazamiento y se lleva a la ALU.
 - Si es un store se lee el registro a almacenar en memoria y se lleva a la entrada 2 de la alu.
- Transferencia de control (branch)
 - Se extiende el signo al desplazamiento
 - Se lee el registro que determina la dirección base del salto.

EJECUCIÓN O CÁLCULO DE DIRECCIÓN (EX)

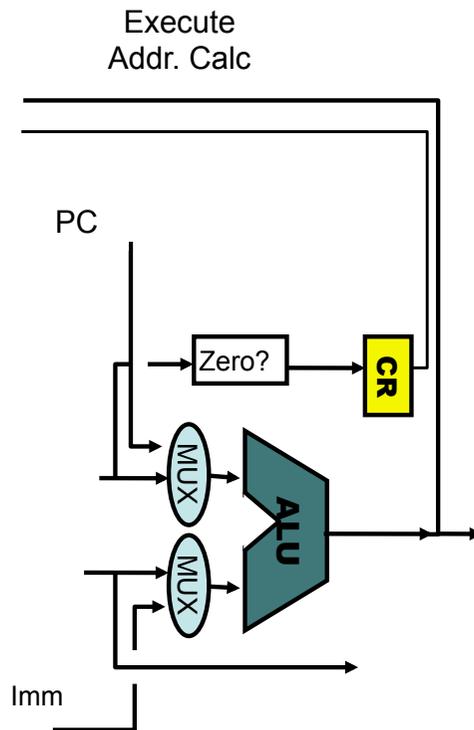


Imagen adaptada de Wikipedia, 2014

- La ALU ejecuta la operación sobre los datos:
 - Las entradas a la alu (arit./log)
 - El PC y el Imm (direc. de salto relativa a PC)
 - El Reg base y el Imm (direc de salto relativa a reg o direc de memoria)
- Dependiendo del tipo de instr.:
- Registro-registro (aritméticas/lógicas)
 - La salida de Alu contiene el resultado de la operación sobre los registros Rs1 y Rs2, y se lleva la salida.
- Referencia a memoria (load/store)
 - La salida de Alu contiene la dirección efectiva de memoria (a leer o escribir).
 - Si es una instr store, el contenido del Rs 2 (reg a almacenar en mem) se lleva en paralelo.
- Transferencia de control (branch)
 - La salida de Alu contiene la dirección efectiva del salto.
 - Se determina si la condición de salto se verifica o no y se lleva al condition reg (CR).

ACCESO A MEMORIA/ FINAL. DE SALTO (MEM)

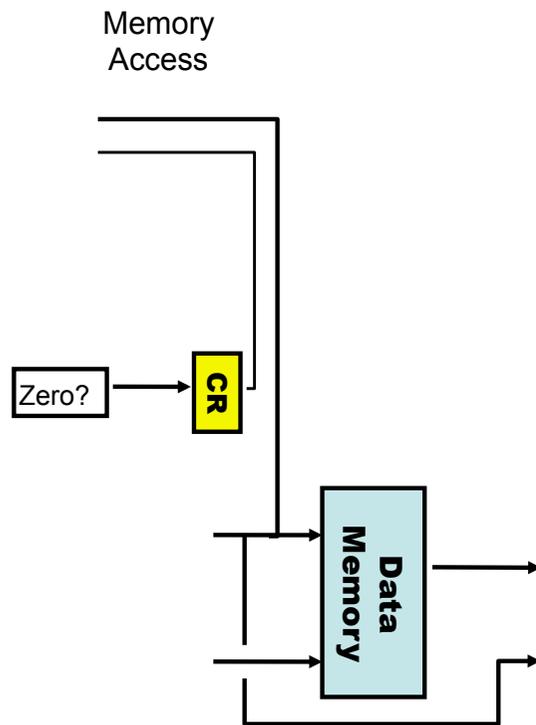


Imagen adaptada de Wikipedia, 2014

- Se realizan operaciones sólo en el caso de instrucciones load, store y branch:
 - Registro-registro (aritméticas/lógicas)
 - La salida de ALU se transfiere.
- Dependiendo del tipo de instr.:
- Referencia a memoria (load/store)
 - Load: se lee el dato de la caché de datos
 - Store: se escribe en la memoria (en la dirección efectiva calculada) el dato leído del banco de registros.
 - Transferencia de control (branch)
 - Tanto para los saltos incondicionales como para los condicionales tomados se modifica el PC de la etapa inicial IF por el contenido del registro de salida de la alu.
 - Para los saltos condicionales no tomados, el PC no se modificará. En el mux de la etapa IF se elije el Pc+4.

FASE DE ESCRITURA DEL RESULTADO (WB)

Write-Back

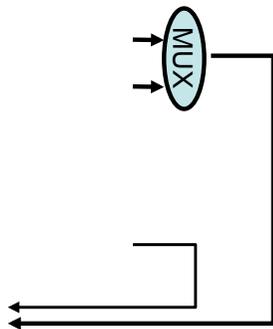
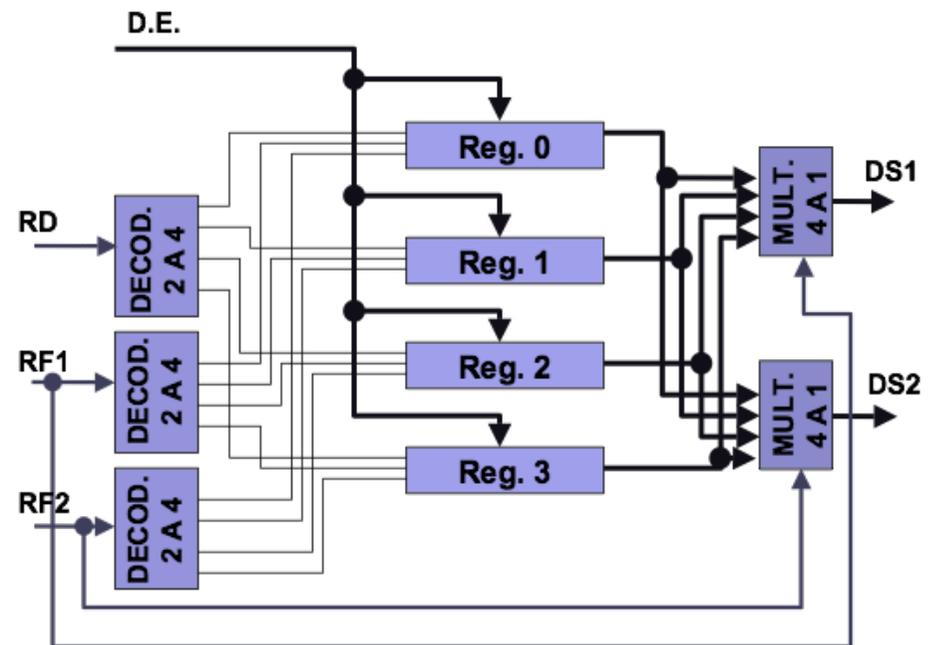
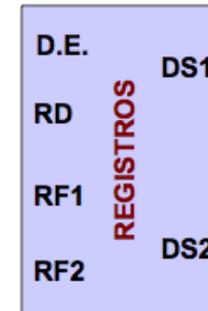


Imagen adaptada de Wikipedia, 2014

- Durante esta fase el resultado de la ejecución de una instrucción registro-registro o load a memoria es almacenado en el banco de registros de la etapa ID.
- Dependiendo del tipo de instr.:
- Registro-registro (aritméticas/lógicas)
 - El resultado de la ALU se escribe en el banco de registros de la etapa ID.
- Referencia a memoria (load)
 - El dato leído de la memoria cache se escribe en el banco de registros de la etapa ID.

BANCO DE REGISTROS

- El banco de registros requiere:
 - Poder acceder a dos registros simultáneamente (2 operandos fuente) en lectura.
 - Poder escribir en un registro en la escritura.
- Tiene tres entradas de direccionamiento de registros, una entrada para los datos a escribir y dos salidas para los registros leídos.





RUTA DE DATOS BÁSICA

- Ruta de datos monociclo: La frecuencia de reloj se debe seleccionar para poder ejecutar la instrucción de mayor coste temporal.
 - Se retrasa la ejecución del resto de operaciones.
- Dividiendo la ejecución de una instrucción en varias fases independientes, se mejora el rendimiento del procesador.
 - Cada instrucción solo ejecuta los ciclos que requiere.
 - La frecuencia de reloj es mayor.
 - → conduce a la **segmentación**



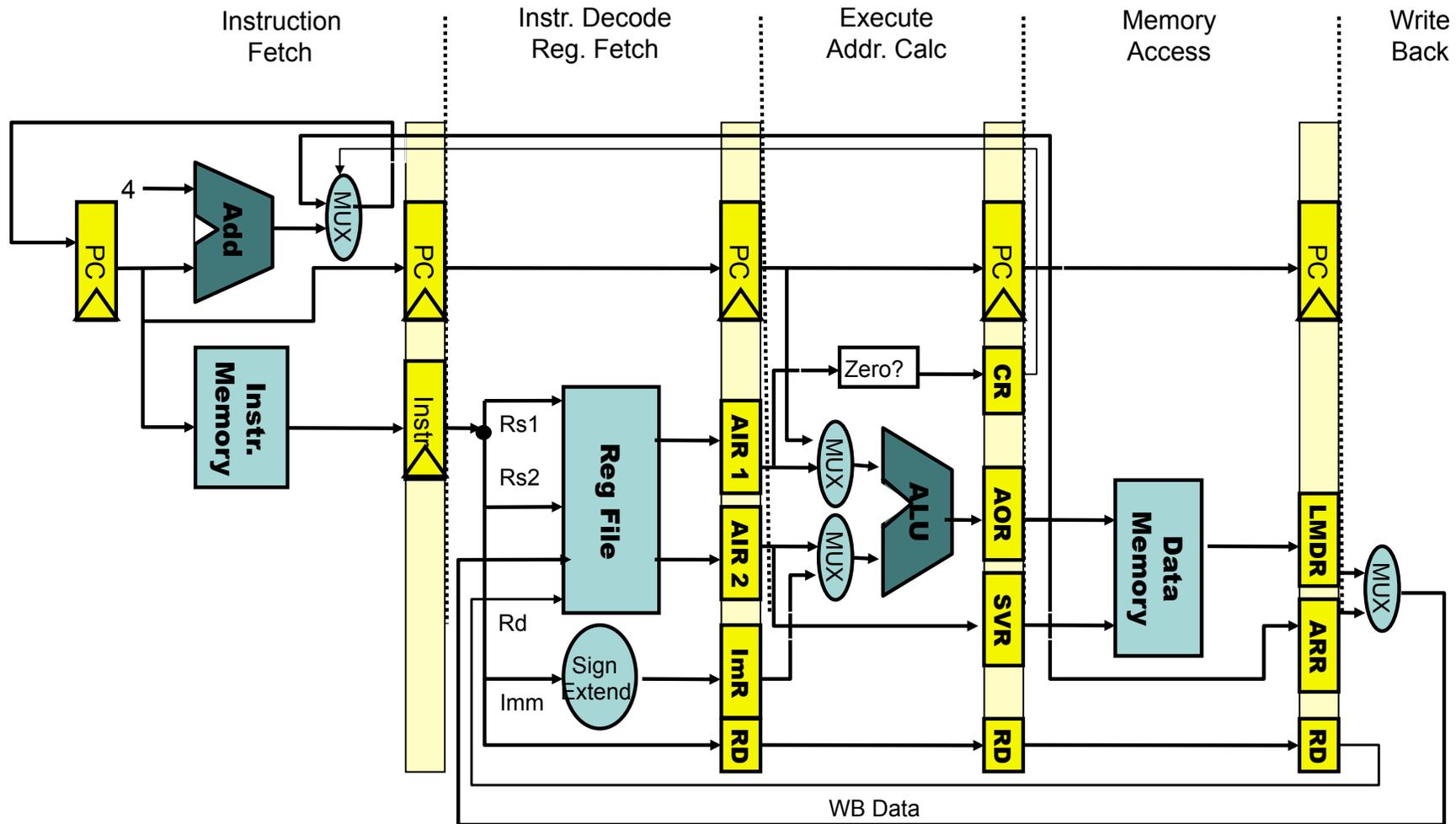


SEGMENTACIÓN

- **Objetivo:**
 - Incrementar la velocidad de ejecución utilizando múltiples unidades concurrentemente pero que preservan el orden original de las instrucciones. Puede ser invisible al programador y al compilador.
- **Método básico:**
 - Dividir la ejecución de la instrucción en varias etapas y ejecutar cada etapa en una unidad diferente.
- **Observaciones:**
 - Se suele utilizar un modelo “síncrono” para el control de los cauces (igual tiempo para cada etapa).
 - Entre las etapas se sitúan registros
 - Se suele utilizar un mecanismo de parada cuando resulta bloqueada alguna instrucción.
 - Ciclo de máquina: es el tiempo de ejecución en cada etapa del cauce (valores típicos: 1-2 CPU Cycles)



EJEMPLO DE EJECUCIÓN SEGMENTADA





GANANCIA DE “PRESTACIONES”

- A igualdad de tecnología, ¿qué ventaja se obtiene al usar una CPU segmentada?
 - Supongamos que la ejecución de una instrucción se divide en k etapas.
 - Cada etapa tarda un ciclo en ejecutarse.
 - Supongamos que el programa consta de n instrucciones (n grande)
 - La 1ª instrucción tarda en ejecutarse k ciclos (k etapas x 1 ciclo)
 - Las instrucciones posteriores finalizan en los ciclos siguientes

ciclo\Etapa	E1	E2	E3	E4	E5	
1	i1					
2	i2	i1				
3	i3	i2	i1			
4	i4	i3	i2	i1		
5	i5	i4	i3	i2	i1	i1 termina
6	i6	i5	i4	i3	i2	i2 termina
7	i7	i6	i5	i4	i3	...





TIEMPO DE RESPUESTA Y RENDIMIENTO

- Tiempo de respuesta (latencia)
 - Tiempo entre el comienzo y la finalización de una tarea (observado por el usuario).
 - $\text{Response Time} = \text{CPU Time} + \text{Waiting Time (I/O, OS scheduling, etc.)}$.
- Throughput
 - Rendimiento.
 - Número de tareas por unidad de tiempo.





TIEMPO DE RESPUESTA Y RENDIMIENTO

- Reducir el tiempo de ejecución mejora el rendimiento.
 - Ejemplo: usar una versión más rápida de un procesador.
 - Menos tiempo para realizar una tarea, más tareas por unidad de tiempo.
- Mejorar el rendimiento mejora el tiempo de respuesta.
 - Ejemplo: aumentar el número de procesadores en un sistema multiprocesador.
 - Más tareas ejecutadas en paralelo.
 - No cambia el tiempo de ejecución de las tareas secuenciales individuales.
 - Pero el tiempo de espera es menor, por lo que el tiempo de respuesta se reduce.





DEFINICIÓN DE PRESTACIONES

- Para un programa corriendo en X:

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

- X es n veces más rápida que Y si:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$





¿QUÉ ES EL TIEMPO DE EJECUCIÓN?

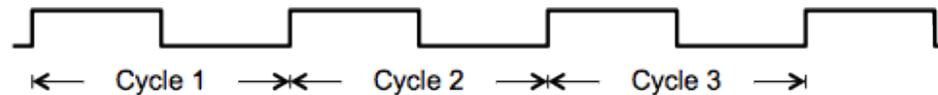
- Tiempo transcurrido real:
 - No
 - I/O, S.O.,etc. no interesan.
- Tiempo de ejecución de la CPU
 - Tiempo transcurrido ejecutando instrucciones del programa.
 - No cuentan las I/O ni S.O.
 - Medido en segundos o ciclos de reloj de la CPU.





CICLOS DE RELOJ

- Clock cycle = Clock period = $1 / \text{Clock rate} = \text{cycle time}$



- Clock rate = Clock frequency = Cycles per second
 - 1 Hz = 1 cycle/sec
 - 1 MHz = 10^6 cycles/sec
 - 2 GHz clock has a cycle time = $1/(2 \times 10^9) = 0.5$ nanosecond (ns)
 - 1 KHz = 10^3 cycles/sec
 - 1 GHz = 10^9 cycles/sec

$$\text{CPU Execution Time} = \text{CPU cycles} \times \text{cycle time} = \frac{\text{CPU cycles}}{\text{Clock rate}}$$





MEJORAR LAS PRESTACIONES

- Dos opciones
 - Reducir el número de ciclos de reloj
 - Reducir el periodo del reloj
- Ejemplo:
 - Un programa corre en 10 segundos en el ordenador X a 2 GHz
 - ➔ ¿Cual es el número de ciclos de reloj?
 - Queremos que el ordenador Y corra el mismo programa en 6 segundos, pero Y necesita un 10% más de ciclos
 - ➔ ¿Cual debe ser la frecuencia de reloj de Y?





MEJORAR LAS PRESTACIONES

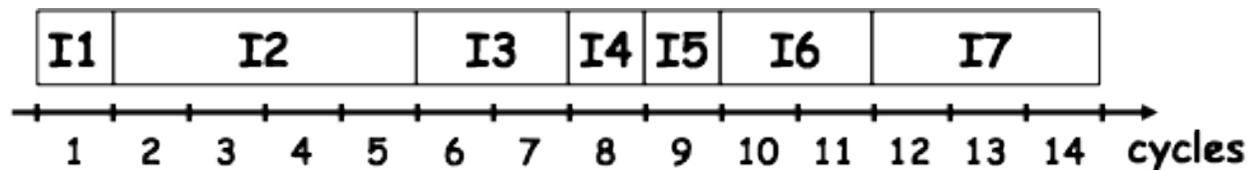
- Solución:
 - CPU cycles on computer X = $10 \text{ sec} \times 2 \times 10^9 \text{ cycles/s} = 20 \times 10^9$ cycles
 - CPU cycles on computer Y = $1.1 \times 20 \times 10^9 = 22 \times 10^9$ cycles
 - Clock rate for computer Y = $22 \times 10^9 \text{ cycles} / 6 \text{ sec} = 3.67 \text{ GHz}$





CICLOS POR INSTRUCCIÓN (CPI)

- Las instrucciones necesitan un número diferente de ciclos para ejecutarse.
 - Multiplicación > suma
 - Floating point > integer
 - Accesos a memoria > accesos a registros
- CPI: media del número de ciclos de reloj por instrucción.
- Ejemplo:



$$\text{CPI} = 14/7 = 2$$





ECUACIÓN DE PRESTACIONES

$$\text{CPU cycles} = \text{Instruction Count} \times \text{CPI}$$

Tiempo de ejecución CPU

$$\text{Time} = \text{Instruction Count} \times \text{CPI} \times \text{cycle time}$$





INFLUENCIA DE FACTORES EN LAS PRESTACIONES

$$\text{Time} = \text{Instruction Count} \times \text{CPI} \times \text{cycle time}$$

	I-Count	CPI	Cycle
Program	X		
Compiler	X	X	
ISA	X	X	X
Organization		X	X
Technology			X





EJEMPLO

- Supongamos que tenemos dos implementaciones de la misma ISA
- Para un programa dado
 - La máquina A tiene un rejoj con período 250 ps y CPI de 2.0
 - La máquina B tiene un rejoj con período 500 ps y CPI de 1.2
 - ¿Qué máquina es más rápida y por cuanto?





SOLUCIÓN

$$\text{Time} = \text{Instruction Count} \times \text{CPI} \times \text{cycle time}$$

- En número de instrucciones es el mismo = I
- CPU execution time (A) = $I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$
- CPU execution time (B) = $I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$
- La máquina A es más rápida que la B por un factor = $600 * I / 500 * I = 1.2$





CÁLCULO DEL CPI

- CPI_i = ciclos por instrucción del tipo i
- C_i = número de instrucciones del tipo i

$$\text{CPU cycles} = \sum_{i=1}^n (CPI_i \times C_i)$$

$$CPI = \frac{\sum_{i=1}^n (CPI_i \times C_i)}{\sum_{i=1}^n C_i}$$





EJEMPLO

- Un diseñador de código decide entre dos secuencias de código para una máquina. Hay tres tipos de instrucciones
 - Clase A: requiere 1 ciclo/instr
 - Clase B: requiere 2 ciclo/instr
 - Clase C: requiere 3 ciclo/instr
- El primer código tiene 5 instrucciones: 2 de A, 1 de B, y 2 de C
- El segundo código tiene 5 instrucciones: 4 de A, 1 de B, y 1 de C
- ➔ Calcular los ciclos de CPU para cada código. ¿Cual es más rápido?
- ➔ ¿Cual es la CPI de cada programa?





SOLUCIÓN

- CPU cycles (1^{er} programa) = $(2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10$ cycles
- CPU cycles (2^o programa) = $(4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9$ cycles
- El segundo programa es más rápido, aún ejecutando una instrucción más.

- CPI (1^{er} programa) = $10/5 = 2$
- CPI (2^o programa) = $9/6 = 1.5$





EJEMPLO

- Dada una ISA para un procesador RISC, ¿cual es la CPI?
- ¿Cual es el porcentaje de tiempo usado por cada clase de instrucción?

Class _i	Freq _i	CPI _i	CPI _i x Freq _i	% Time
ALU	50%	1		
Load	20%	5		
Store	10%	3		
Branch	20%	2		





EJEMPLO

- Dada una ISA para un procesador RISC, ¿cual es el CPI?
- ¿Cual es el porcentaje de tiempo usado por cada clase de instrucción?

Class _i	Freq _i	CPI _i	CPI _i x Freq _i	% Time
ALU	50%	1	$0.5 \times 1 = 0.5$	$0.5 / 2.2 = 23\%$
Load	20%	5	$0.2 \times 5 = 1.0$	$1.0 / 2.2 = 45\%$
Store	10%	3	$0.1 \times 3 = 0.3$	$0.3 / 2.2 = 14\%$
Branch	20%	2	$0.2 \times 2 = 0.4$	$0.2 / 2.2 = 18\%$

- $CPI = 0.5 + 1.0 + 0.3 + 0.4 = 2.2$





EJEMPLO: CÁLCULO DEL IMPACTO DE LOS SALTOS

- Supongamos inicialmente que todas las instrucciones tardan en ejecutarse 1 ciclo, $CPI = 1.0$ (ignorando los saltos).
- Supongamos que en cada salto se origina una penalización adicional de 3 ciclos.
- Si el 30% de las instrucciones del programa fuesen saltos como afectaría al CPI.





EJEMPLO: CÁLCULO DEL IMPACTO DE LOS SALTOS

- Supongamos inicialmente que todas las instrucciones tardan en ejecutarse 1 ciclo, $CPI=1.0$ (ignorando los saltos).
- Supongamos que en cada salto se origina una penalización adicional de 3 ciclos.
- Si el 30% de las instrucciones del programa fuesen saltos como afectaría al CPI.

Operación	Frecuencia	Ciclos	CPI(i)	%tiempo
Otras Instrucciones	70%	1	0.7	37%
Saltos	30%	1+3	1.2	63%

- El nuevo CPI sería: $CPI=1.9$
 - Es decir se ha reducido casi a la mitad las prestaciones reales de la CPU.





MIPS COMO MEDIDA DE PRESTACIONES

- MIPS: Millions Instructions Per Second
- MIPS especifica la tasa de ejecución de instrucciones

$$\text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6} = \frac{\text{Clock Rate}}{\text{CPI} \times 10^6}$$

- De aquí se obtiene:

$$\text{Execution Time} = \frac{\text{Inst Count}}{\text{MIPS} \times 10^6} = \frac{\text{Inst Count} \times \text{CPI}}{\text{Clock Rate}}$$





DESVENTAJAS DEL MIPS

- No tiene en cuenta las características de las instrucciones.
- Varía entre programas del mismo ordenador.
- Puede variar inversamente al rendimiento (ver ejemplo siguiente).





EJEMPLO MIPS

- Dos computadores ejecutan el mismo programa en una máquina a 4 GHz con tres tipos de instrucciones
 - Clase A: requiere 1 ciclo/instr
 - Clase B: requiere 2 ciclo/instr
 - Clase C: requiere 3 ciclo/instr
 - El número de instrucciones producido por la máquina 1:
 - 5 billones Clase A
 - 1 billones Clase B
 - 1 billones Clase C
 - El número de instrucciones producido por la máquina 2:
 - 10 billones Clase A
 - 1 billones Clase B
 - 1 billones Clase C
- ➔ ¿Qué compilador produce mejor tiempo de ejecución?
- ➔ ¿Y mejor MIPS?





SOLUCIÓN

- Ciclos de CPU
 - CPU cycles (compiler 1) = $(5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9$ cycles
 - CPU cycles (compiler 2) = $(10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9$ cycles
- Tiempo de ejecución
 - Execution time (compiler 1) = 10×10^9 cycles / 4×10^9 Hz = 2.5 sec
 - Execution time (compiler 2) = 15×10^9 cycles / 4×10^9 Hz = 3.75 sec
- El compilador 1 genera un programa más rápido (menor tiempo de ejecución)





SOLUCIÓN

- MIPS rate
 - $\text{MIPS} = \text{Instruction Count} / (\text{Execution Time} \times 10^6)$
 - $\text{MIPS (compiler 1)} = (5+1+1) \times 10^9 / (2.5 \times 10^6) = 2800$
 - $\text{MIPS (compiler 2)} = (10+1+1) \times 10^9 / (3.75 \times 10^6) = 3200$
- El compilador 2 tiene mejor MIPS!





MEDIDAS DE PRESTACIONES: MFLOPS

- Millones de instrucciones de coma flotante por segundo.

$$MFLOPS = \frac{(n^{\circ} \text{ operaciones FP})}{(T_{ejec} \times 10^6)}$$

- Sólo puede aplicarse a las instrucciones FP.
- Las operaciones de FP no son consistentes entre máquinas.
- Ejemplo:
 - Pentium tiene “sqrt,sin,cos” y CRAY C90 no
- La medida cambia mucho dependiendo de la instrucciones FP utilizadas ya que las hay muy lentas.





ACELERACIÓN: “Speed up”

- Es la relación entre los tiempos de ejecución de un cierto programa en dos máquinas diferentes.
 - Aceleración asintótica: es la relación entre los tiempos de ejecución en un sistema monoprocesador y uno con infinitos procesadores.

$$S_{\infty} = \frac{(T_e(1))}{(T_e(\infty))}$$

- Aceleración, ejemplo para un caso con dos procesadores frente al monoprocesador

$$S_2 = \frac{(T_e(1))}{(T_e(2))}$$





ACELERACIÓN EN UN PROCESADOR SEGMENTADO

- Supongamos que un cierto programa tiene N instrucciones y supongamos que se realizan k etapas en la ejecución de cada instrucción en el procesador.
 - El tiempo de ejecución en un procesador no segmentado es:

$$T_1 = k \cdot N \cdot T_c$$

- El tiempo de ejecución en un procesador segmentado en k etapas es:

$$T_k = [k + (N - 1)] \cdot T_c$$

- La aceleración es por lo tanto:

$$S_k = \frac{T_1}{T_k} = \frac{(k \cdot N \cdot T_c)}{([k + (N - 1)] \cdot T_c)}$$

- Expresión que tiende a k cuando N tiende a infinito.





EVALUACIÓN MEDIANTE PROGRAMAS

- Cuatro niveles:
 - Programas reales: compiladores de C, Tex, CAD y Spice
 - Kernels: extractos pequeños de programas (Livermore loops, Linpack)
 - Benchmarks de juego: código con resultado conocido (criba de Eratóstenes)
 - Benchmarks sintéticos: están especialmente diseñados para medir el rendimiento de un componente individual de un ordenador, normalmente llevando el componente escogido a su máxima capacidad.





BENCHMARK SUITES: SPEC

- SPEC: System Performance Evaluation Cooperative
 - SPEC89
 - Ejecuta 10 programas, devuelve un único valor.
 - SPEC92
 - 6 programas con números enteros
 - 14 programas con números en fp
 - Sin límites en las opciones de compilación
 - SPEC95
 - 10 programas con números enteros
 - 10 programas con números en fp
 - Dos opciones de compilación
 - La mejor posible para cada programa
 - La misma para todos





“ ”

- CINT2000 (Integer Components of SPEC CPU 2000)

Benchmark	Language	Category
164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator





SPEC2000 “CFP 2000”

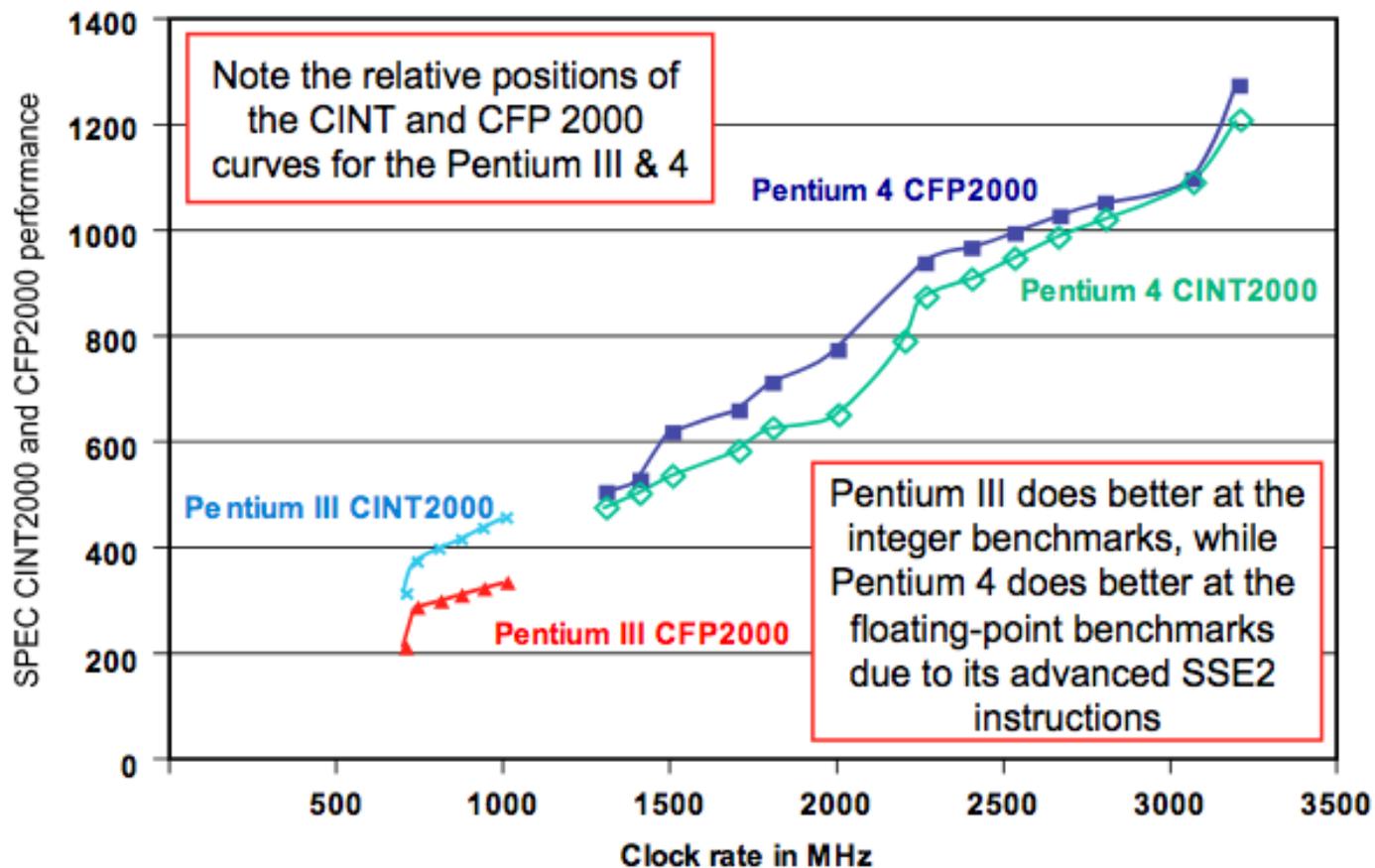
- CFP 2000 (Floating point Components of SPEC CPU 2000)

Benchmark	Language	Category
168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Partial Differential Eq.
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.quake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.amp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Nuclear Phy. Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution



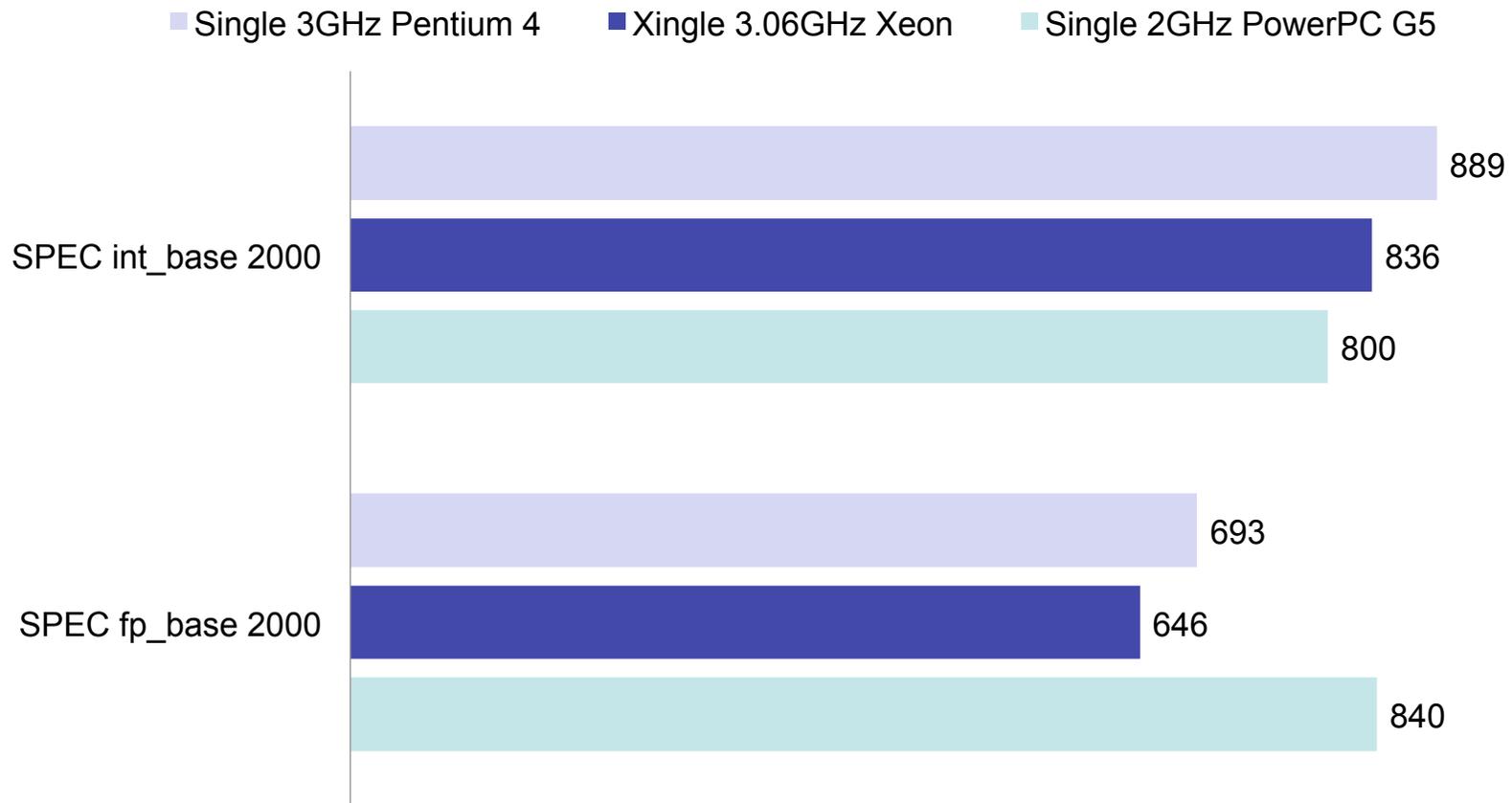
SPEC PARA PENTIUM III Y PENTIUM IV

SPEC ratio = Execution time is normalized
relative to Sun Ultra 5 (300 MHz)
SPEC rating = Geometric mean of SPEC ratios



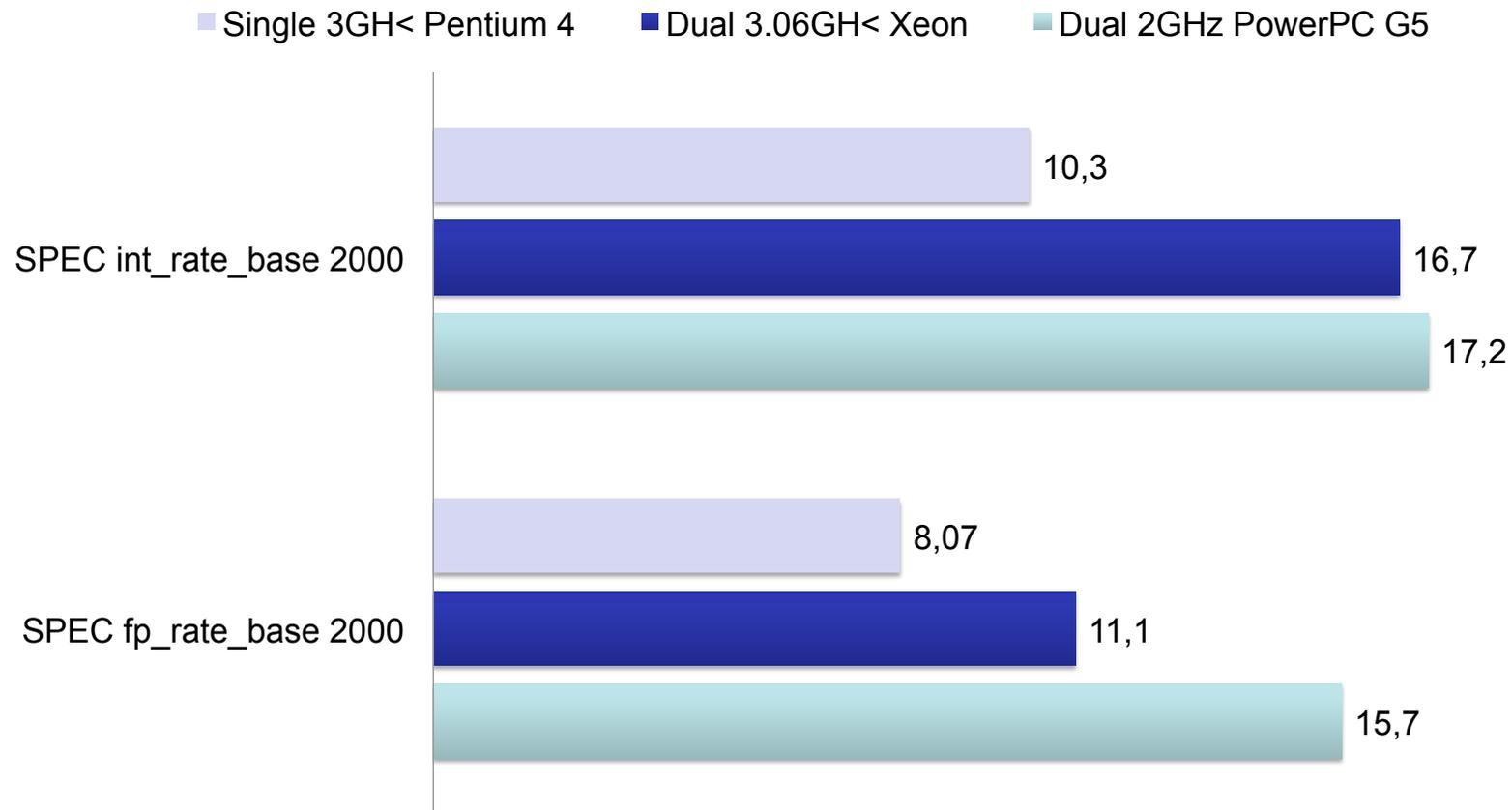


SPEC 2GH G5, 3.06GHz XEON, 3GHz PENTIUM 4: SPEED





SPEC 2GH G5, 3.06GHz XEON, 3GHz PENTIUM 4: THROUGHPUT





PROBLEMAS DE LOS PROGRAMAS DE PRUEBA

- Problemas:
 - Los benchmarks reducidos (toys) y los sintéticos no cargan la memoria principal del sistema de forma realista (todo el programa cabe en la memoria cache).
 - Una vez que el benchmark se estandariza inmediatamente aparecen mejoras específicas para el mismo elevando los resultados de rendimientos.
 - Si los benchmarks fuesen nuestros programas nos veríamos beneficiados ya que los interesados harían que nuestra aplicación fuese más rápida, pero no es este el caso.
 - Los benchmarks reales son difíciles de realizar, situación que se agudiza en los casos de:
 - Máquinas no construidas, simuladores más lentos.
 - Benchmarks no portables.
 - Compiladores no disponibles.





MEDIDA “HOMOGÉNEA” DE PRESTACIONES

- Media aritmética

$$\frac{1}{n} \sum_{i=1}^n Time_i$$

- Media armónica

$$\frac{n}{\left(\sum_{i=1}^n \left(\frac{1}{Rate_i} \right) \right)}$$

- Tiempos ponderados
- Media geométrica ponderada

$$\sqrt[n]{\prod \left(\frac{T_i}{w_i} \right)}$$





REFERENCIAS

[Mafla, 2001] Mafla, E. (2001) *Course Notes, CDA3101*, at URL <http://www.cise.ufl.edu/~emafla/> (as-of 11 Apr 2001)

[Wikipedia, 2014] Procesador MIPS, [http://es.wikipedia.org/wiki/MIPS_\(procesador\)](http://es.wikipedia.org/wiki/MIPS_(procesador))

[Mudawar, 2014] Mudawar, M. (2014) Computer Architecture course slides, Computer Engineering Department King Fahd University of Petroleum and Minerals, http://opencourseware.kfupm.edu.sa/colleges/ccse/coe/coe308/files%5C2-Lecture_Notes_07-Performance.pdf

