Medición de Performance

Organización del Computador 1
1er Cuat. 2018

¿Qué es el rendimiento?

Avión	Pasajeros	Autonomía (km)	Velocidad km/h	Productividad (pasajeros x km/h)
Boeing 777	375	7450	980	367.5
Boeing 747	470	6675	980	460.6
BAC/Sud Concorde	132	6435	2175	287.1
Douglas DC-8-50	146	14030	875	127.75

- ¿Cuál de estos aviones tiene mejor rendimiento?
  - ¿El que tiene mayor autonomía, o el más rápido?
  - ¿El más rápido o el que transporta mas pasajeros en menos tiempo?

Performance/Rendimiento

- Estudiar el rendimiento de una computadora me permite:
  - Comparar computadoras
  - Evaluar mejores al diseño de una computadora
  - Mejorar los programas que ejecuta la computadora

Rendimiento - Medición

- Necesitamos medir el rendimiento de una computadora
- ¿Cómo podemos medir el rendimento?
  - Cantidad de Instrucciones por segundo (IPS)
  - Cantidad de ciclos de reloj por segundo (CPS)
  - Cantidad de Instrucciones por ciclo de reloj (IPC)
  - Cantidad de ciclos de reloj por instrucción (CPI)

FLOPS

- FLOPS: Floating-point operations per second
- Se calcula multiplicando
  - # procesadores
  - # cores en cada procesador
  - # FLOP ejecutables por ciclo en cada core
  - # ciclos x segundo

FLOPS

- Megaflops = MFLOPS: 10^6 FLOPS
- Teraflops = TFLOPS: 10^12 FLOPS
- Petaflops = PFLOPS: 10^15 FLOPS
  - Intel 8087 (1980): 50 KFLOPS (10^3 FLOPS)
  - Roadrunner (2008): 1.026 PFLOPS
  - Thiane-2 (2013): 33.86 PFLOPS

IBM Roadrunner

Frecuencia de Reloj

- Pentium 200 Mhz vs. Pentium Pro 150 Mhz
- ¿Cuál ejecuta más instrucciones por segundo?
- ¿Es mejor la Pentium 200 Mhz que la Pentium Pro 150 Mhz?

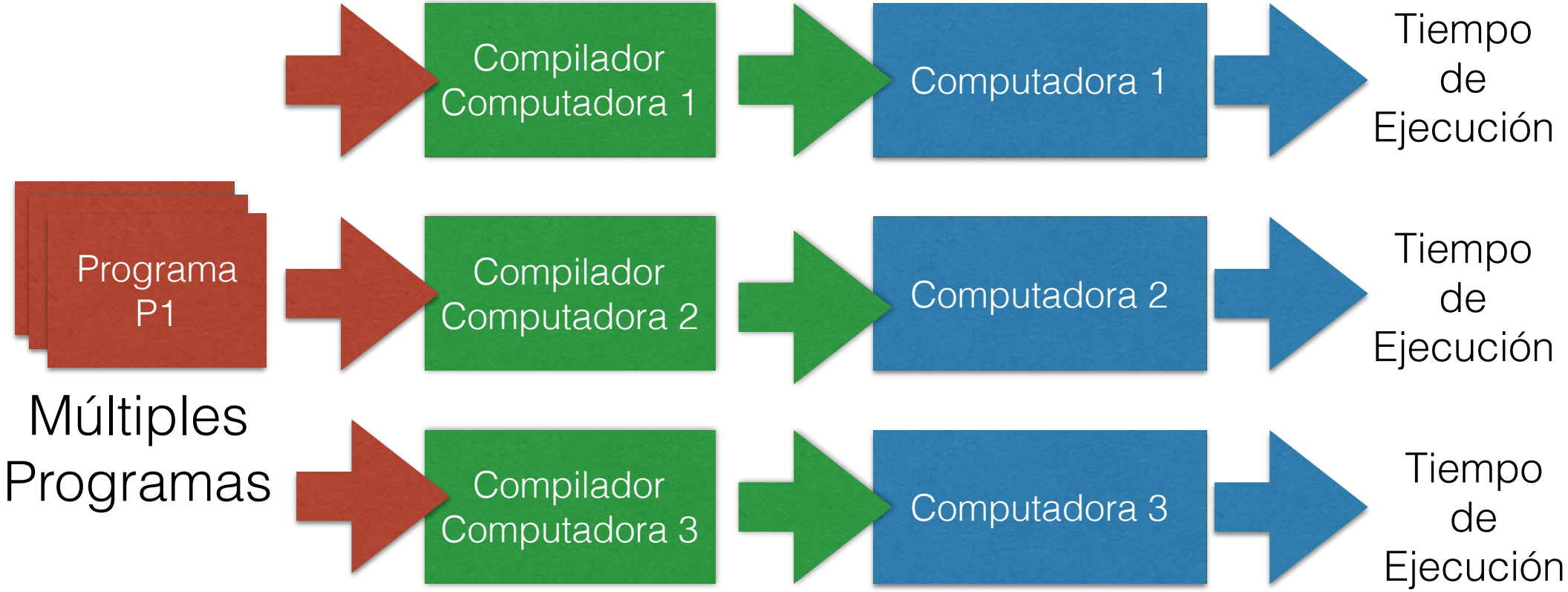
Benchmarks

- **Idea**: medir el rendimiento (tiempo de ejecución) ejecutar una tarea (conjunto de programas)
- Este conjunto de programas (o "*benchmark*") intenta aproximar el uso de la computadora
  - Benchmarks sintéticos
  - Benchmarks industriales
  - Benchmarks temáticos (gráficas, aritméticos, etc)

Benchmarks

- Para ejecutar un benchmark, las computadoras tienen que:
  - Compartir la misma arquitectura (=instrucciones y registros, memoria, etc.), o bien…
  - Disponer de compiladores/ensambladores que traduzcan una fuente común (C, FORTRAN)

Benchmarks


Programa
P1
Múltiples
Programas
Compilador
Computadora 1
Compilador
Computadora 2
Compilador
Computadora 3
Computadora 1
Computadora 2
Computadora 3
Tiempo
de
Ejecución
Tiempo
de
Ejecución
Tiempo
de
Ejecución

MIPS

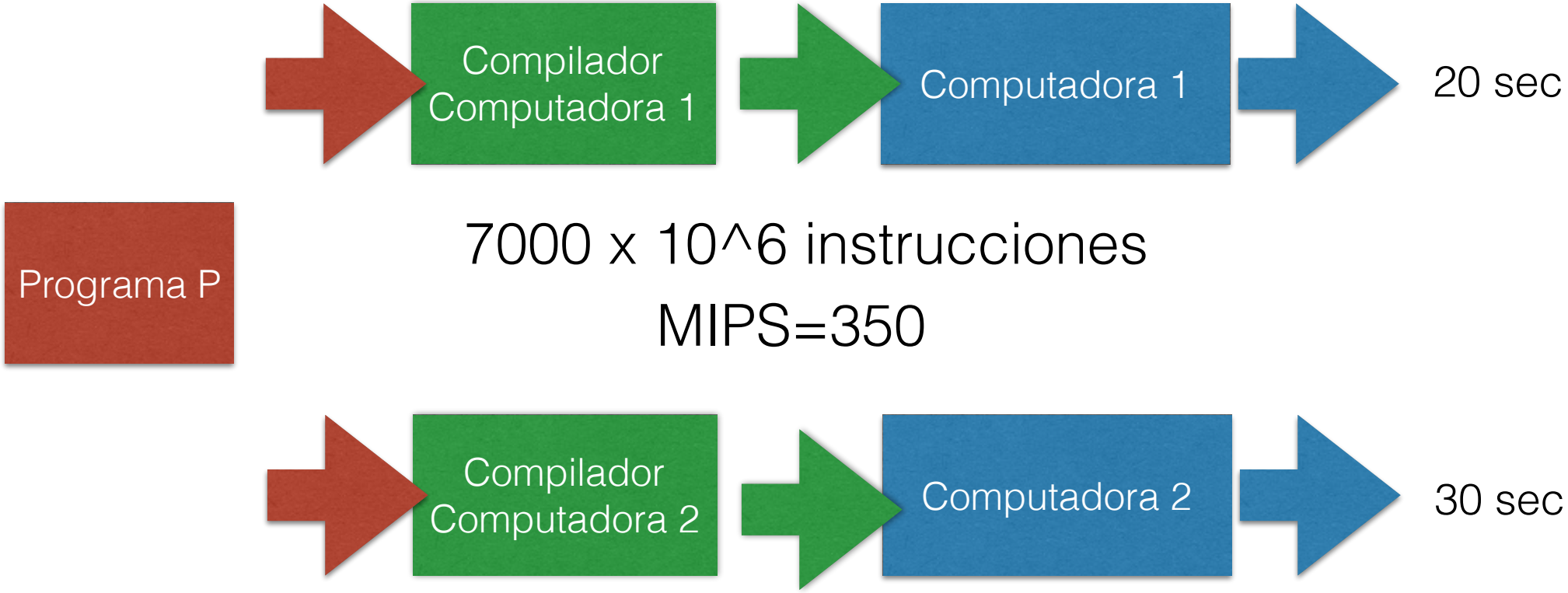
- Se calcula usando la frecuencia del reloj y la cantidad de instrucciones por segundo
- MIPS = frecuencia / (CPI * 10^6)
- **Pregunta:** ¿Qué pasa si la cantidad de ciclos por instrucción no es constante?

MIPS

- Si no existe un valor uniforme de CPI, tenemos que utilizar un benchmark para calcular MIPS (pero será relativo al benchmark elegido)

- $$MIPS_p = \frac{\text{\# instrucciones para ejecutar p}}{\text{tiempo de ejecución de p x } 10^6}$$

MIPS


Programa P
Compilador Computadora 1
Computadora 1
20 sec
7000 x 10^6 instrucciones
MIPS=350
Compilador Computadora 2
Computadora 2
30 sec
11000 x 10^6 instrucciones
MIPS=400

MIPS - Ventajas

- Es intuitivo (más rápido debería implicar MIPS más alto)
- Sirve para comparar 2 arquitecturas **idénticas** (mismo set de instrucciones, memoria, codificación)
  - Ejemplo: comparar IBM 360 vs IBM 370
  - Es fácil de calcular si existe CPI (no necesita benchmarking)

MIPS - Desventajas

- Está relacionado con el número de instrucciones, pero no es representativo cuando hay instrucciones con diferente tiempo de ejecución
    - Puede variar inversamente al rendimiento (MIPS 400 tenía peor rendimiento que MIPS 350)

Benchmarks Sintéticos Clásicos

- **Whetstone** (1977): intensivo en operaciones de punto flotante, con llamados a rutinas trigonométricas y exponenciales
- **Linpack** (1984): LINear algebra PACKage. Resuelve sistemas lineales usando aritmética de doble precisión real
- **Dhrystone** (1984): operaciones de strings y enteros.

MIPS relativos

- MIPS: Se estandarizó la VAX-11/780 como la medida “1 MIPS”
    - Mide el tiempo de ejecución contra la VAX-11/780 en la ejecución de una tarea
    - También llamado VAX-MIPS
- Es una medida de rendimiento relativa (no absoluta)

VAX-11/780 (1977)

VAX-MIPS

- Comparación usando benchmark Dhrystone:
  - Intel 8086 (5Mhz) -> 0.33 VAX-MIPS
  - VAX-11/780 (5Mhz) -> 1 VAX-MIPS
  - Intel Core i7 (2.93Ghz) -> 82.30 VAX-MIPS
- Si el resultado fuera “80 VAX-MIPS” se lee como “80 veces más rápido que la VAX-11/780"

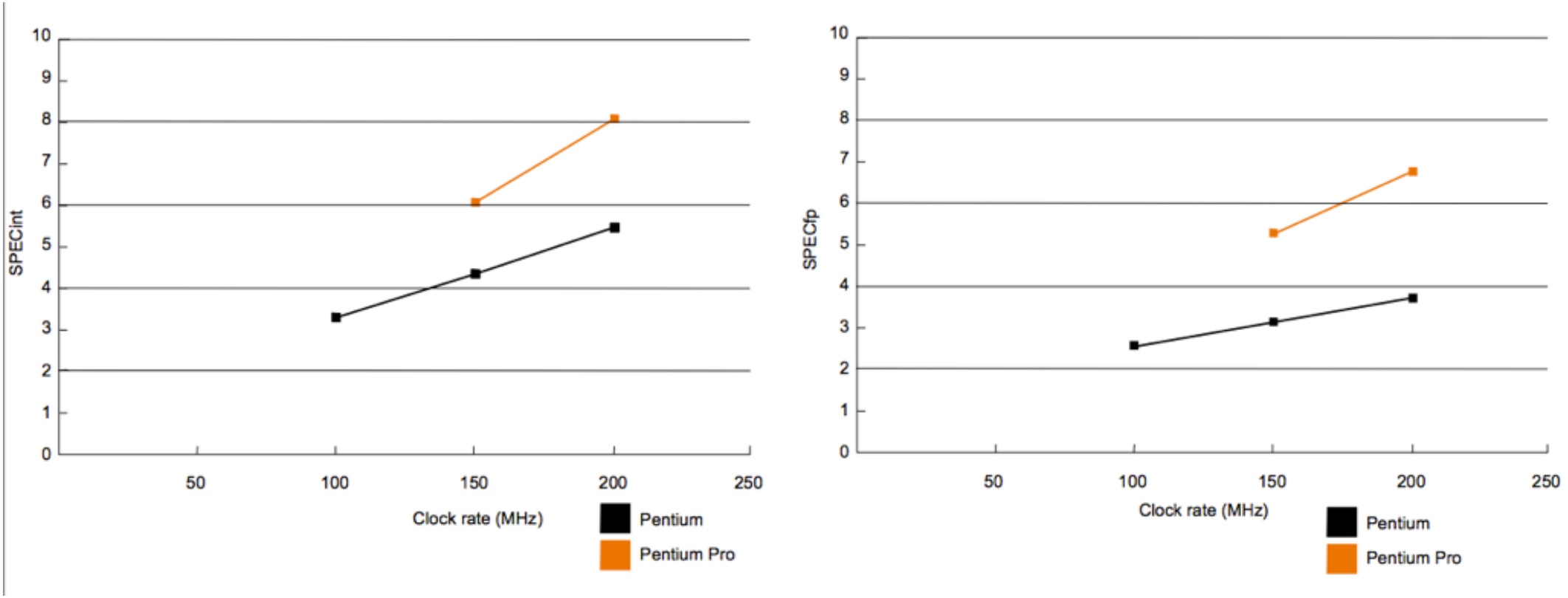
SPEC benchmarks

- **SPEC**: Standard Performance Evaluation Corporation (>60 miembros: Microsoft, Intel, Google, HP, etc. + Universidades)
- **CPU2006** (2006): benchmark para rendimiento de CPU
  - SPECint2006: 12 programas aritmética entera
  - SPECfp2006: 19 programas aritmética punto flotante
- SPECratio: Es una medida **relativa** a una arquitectura de referencia

Intel Core i7 920 (2.66 Ghz) SPECInt2006

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Rendimiento Relativo



- ¿Es mejor la Pentium 200 Mhz que la Pentium Pro 150 Mhz? **Respuesta:** El reloj no es determinante del rendimiento

Rendimiento - Mejora

- Supongamos que somos los diseñadores de una nueva versión de una computadora
- Un benchmark tarda 100 sec en ejecutarse, deseamos que se ejecute 5 veces más rápido
  - Para lograrlo, se identifica que 80 sec se gastan en multiplicaciones
  - **Pregunta** ¿Cuánto debemos mejorar la velocidad de la multiplicación?

Rendimiento - Mejora

- De los 100 sec de ejecución del benchmark
  - 80 sec lleva ejecutar multiplicaciones
  - 20 sec lleva ejecutar otras instrucciones
- Si el tiempo de ejecución de 100 sec se desea mejorar 5 veces, eso significa que debemos ejecutar el benchmark en 20 sec
- ¿Cuánto tiempo debe tardar ejecutar la multiplicación?

Rendimiento - Mejora

- 20 sec = T_mult + 20 sec
- Por lo tanto, T_mult=0
- Por lo tanto, no es posible realizar la mejor
- En general, no podemos infringir la *"Ley de Amdahl"*

Ley de Amdahl (1967)

- *"El speedup o incremento de rendimiento está limitado por la proporción en que se utilice la prestación mejorada."*

- Speed-Up = $\dfrac{1}{(1-f) + f/k}$

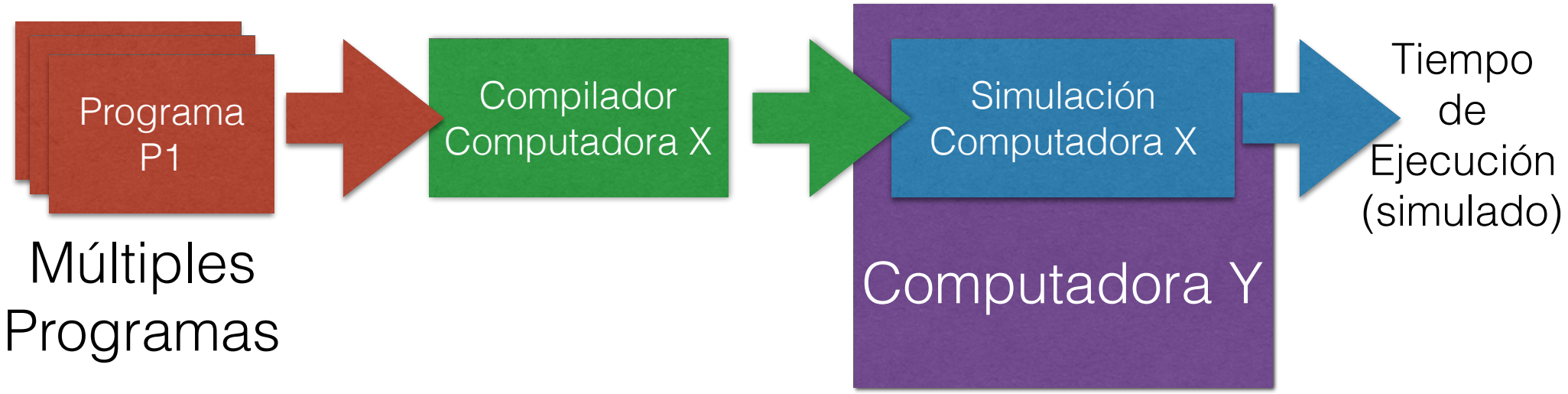
- Donde f es la fracción del trabajo realizada por el componente mejorable, y

- k es el Speed-Up del nuevo componente

Rendimiento - Diseño

- Se puede estudiar el rendimiento no sólo de computadoras reales, sino también de **diseños** de computadoras
  - Se realiza una **simulación** de la computadora a partir del diseño
  - Se simula la ejecución del benchmark en el diseño
  - Permite comparar distintos diseños posibles

Rendimiento - Simulación



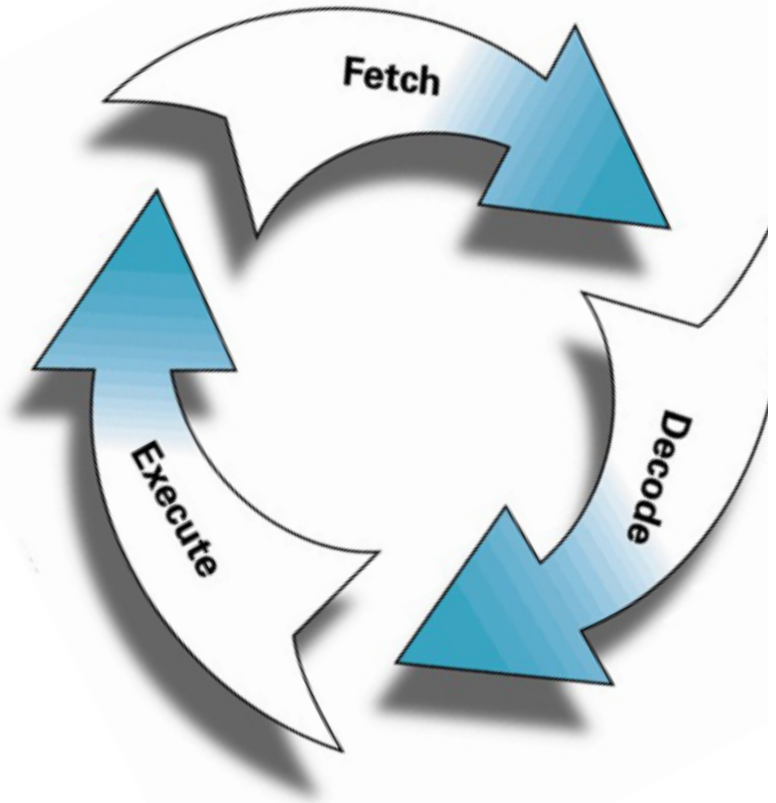
Optimización de Performance

Optimización de Performance

- La memoria caché nos permite optimizar el acceso a memoria de un programa.
- ¿Qué otras formas tenemos de diseñar una computadora para optimizar su ejecución?

Ciclo de Instrucción

- El CPU realiza continuamente:
  - Fetch de la instrucción
  - Decode de la instrucción
  - Execute de la instrucción



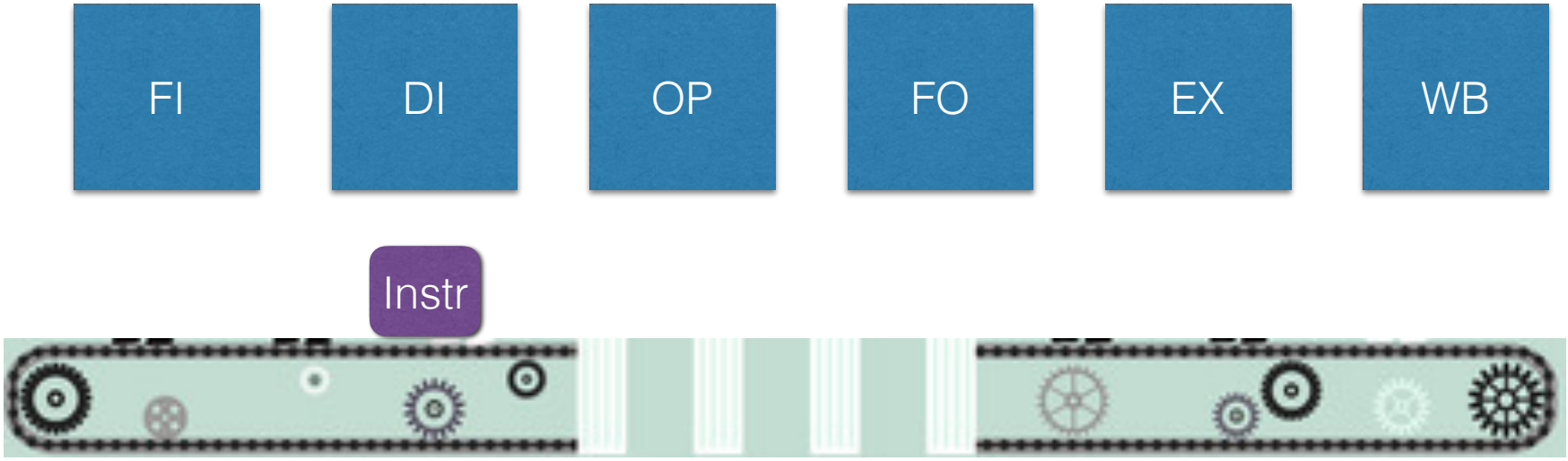
Etapas del Ciclo de Instrucción

1. Fetch instrucción (FI)
2. Decodificar opcode (DI)
3. Calcular dirección de los operandos (OP)
4. Obtener operandos (FO)
5. Ejecutar la instrucción (EX)
6. Almacenar resultado (WB)

Etapas del Ciclo de Instrucción

- A nivel micro-arquitectura, cada paso etapa del ciclo de instrucción puede ser resuelto por un componente distinto.

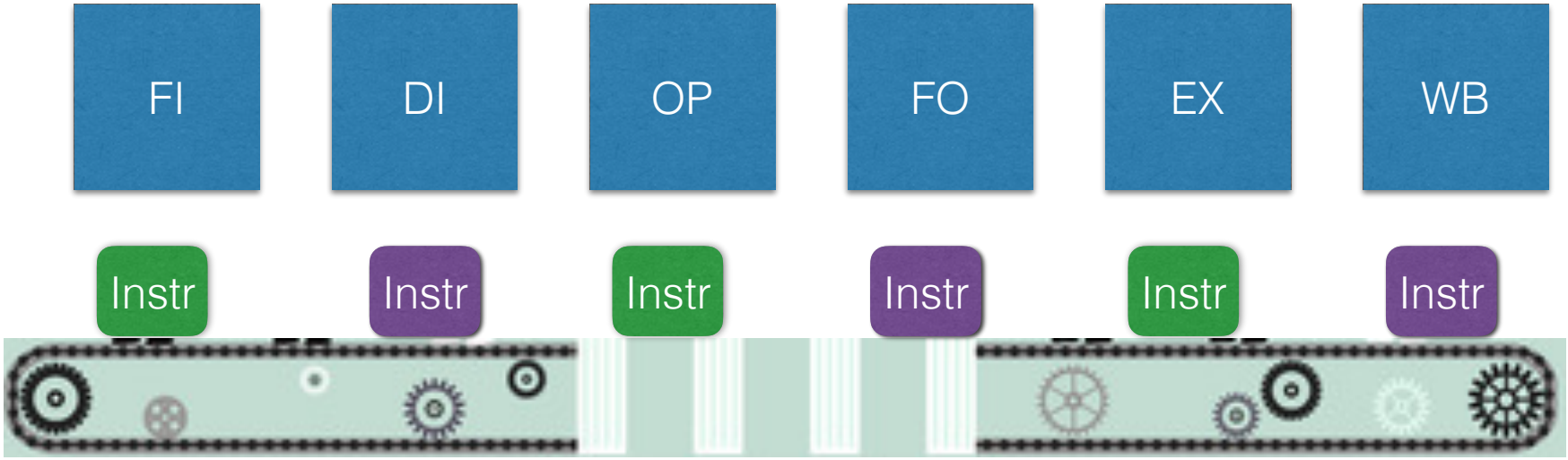
- Podemos pensar entonces, en la ejecución de una instrucción como una cinta transportadora



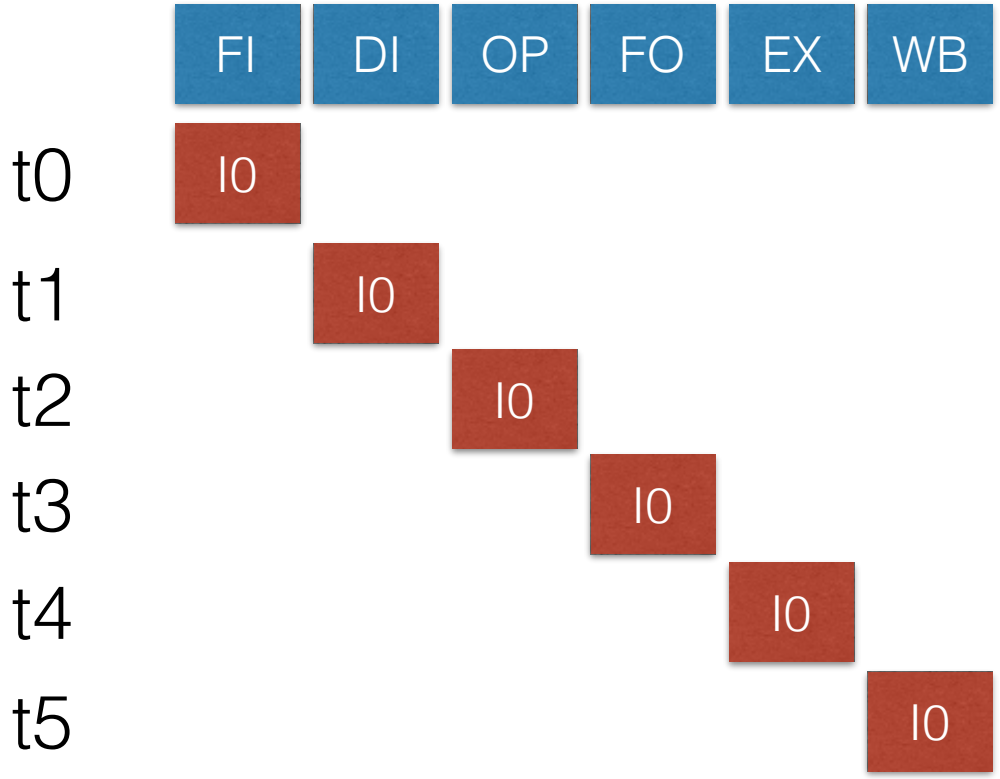
Instruction-Level Pipelining

- Si se trata de componentes distintos, por lo tanto se podría ejecutar simultáneamente múltiples instrucciones.

- A esto se lo denomina **pipelining** de nivel de instrucción

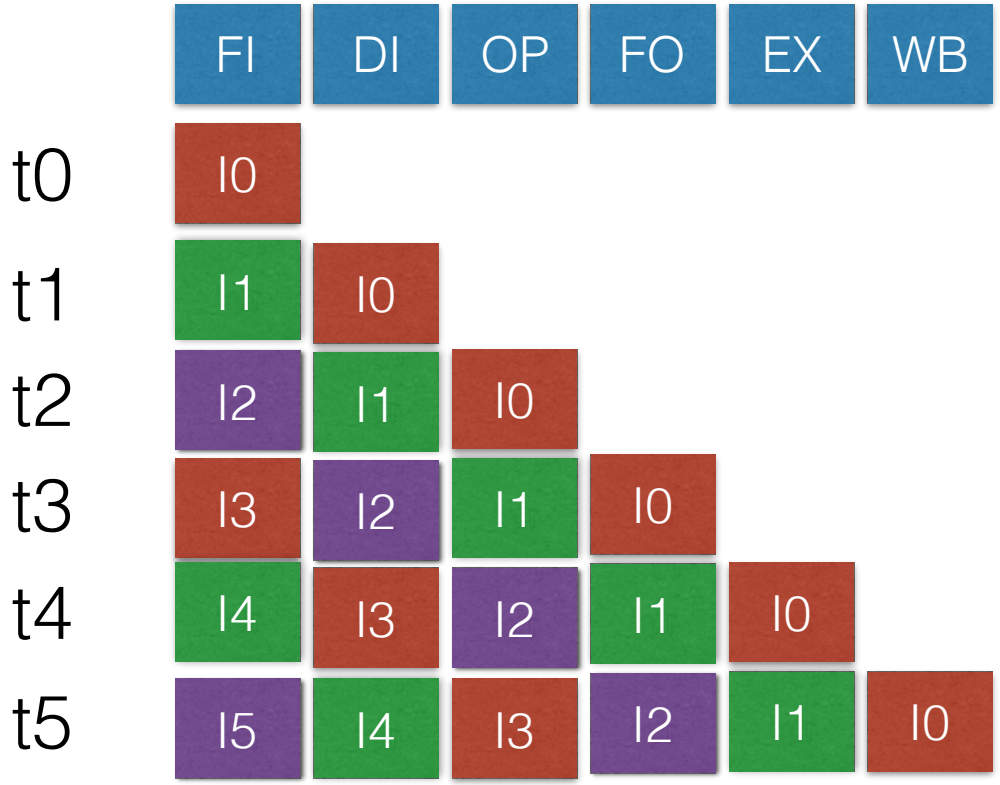


Sin Pipelining



- Ejecutar la instrucción “I0" consumió 6 ciclos de reloj

Máximo Pipelining



...

- Una vez lleno el pipeline, podemos ejecutar 1 instrucción por cada ciclo de reloj

Instruction-Level Pipelining

- El pipelining puede mejorar notablemente la performance de una computadora
  - Sin Pipelining: 6 ciclos por instrucción (CPI=6)
  - Máximo Pipelining: 1 ciclo por instrucción (CPI=1)
- ¿Qué puede limitar o entorpecer el máximo pipelining?

Conflictos de Recursos

- ¿Qué pasa si 2 etapas quieren usar el mismo recurso simultáneamente?
  - Por ejemplo: leer la memoria o sumar usando la ALU
  - El acceso al recurso limita el pipelining

Conflictos de Recursos

- Posibles soluciones:
  - Incorporar redundancia (más caminos de datos, duplicar componentes)
  - Uso de memoria caché
  - Separación memoria datos/memoria de programa

Dependencia de datos

- ¿Qué pasa si la siguiente instrucción necesita datos creados por la instrucción actual?
  - Por ejemplo:
    - ADD R0,R1
    - MOV R2,R0 # necesita leer R0

Dependencia de datos

- Posibles soluciones:
  - Detectar la dependencia de datos
    - Luego, forzar el retraso de la ejecución de la instrucción
    - Transmitir internamente el dato (sin necesidad de pasar por memoria o registros)

Saltos Condicionales

- Un salto condicional es un cambio en el flujo secuencial del programa
- Por lo tanto, las instrucciones siguientes no deben haber sido siquiera leídas

Saltos Condicionales

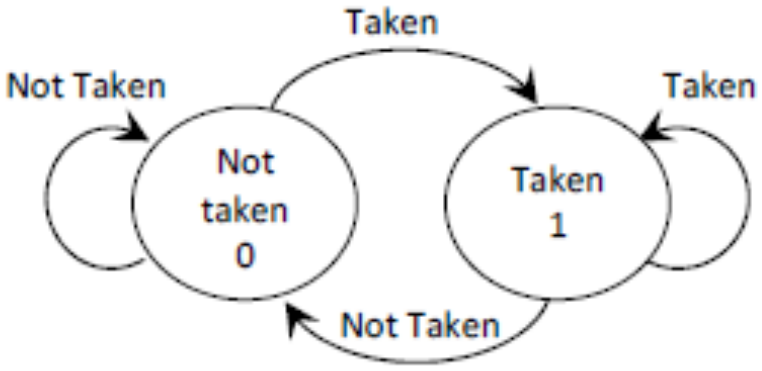
- Posibles soluciones:
  - Desechar pipeline (Branch penalty)
  - Duplicar el pipeline (uno que procesa el salto, otro que no lo procesa)
  - Predicción de saltos

Branch Prediction

- Un circuito específico (llamado “branch predictor”) evalúa las chances que el salto condicional sea tomado:
  - El pipeline se actualiza de acuerdo a la predicción (T/NT) del branch predictor
- En caso de ser una predicción errada, el pipelining es desechado (branch penalty)

Branch Predictor

- El “branch predictor” se actualiza en cada predicción
  - Si la predicción fue correcta
  - Si la predicción fue equivocada
- 1-bit Branch Predictor:



Pipeline - Pentium 4

- Pipeline de 20 etapas

- (TC,NI)= Trace Cache Next Instr.
- (TR,F)= Trace Cache Fetch
- D=Drive
- AR=Alloc and Rename
- Q=Queue
- S=Schedule
- D=Dispatch
- R=Register File
- E=Execute
- F=Flags
- BC=Branch Check
- D=Drive

Pipeline - Pentium 4

- TC Nxt IP (2 steps): Trace cache next instruction pointer. Busca en el branch target buffer (BTB) la próxima micro-instrucción a ser ejecutada.
- TC Fetch (2 steps): Trace cache fetch. Carga de la trace cache la micro-instrucción
- Drive (1 step): Envia la micro-instrucción a ser ejecutada al **resource allocator** y al circuito de **register renaming**.
- Alloc and Rename (3 steps):
  - Allocate (1 step). Chequea qué recursos del CPU son necesarios para ejecutar la micro-instrucción (por ejemplo, leer memoria and guardar en buffers)
  - Rename (2 steps): Si el programa usa uno de los 8 registros standard x86 estos son renombrados a uno de los 128 registros internos presentes en la Pentium 4.

Pipeline - Pentium 4

- Queue (1 step): Las micro-instrucción son encoladas de acuerdo a su tipo (por ejemplo, enteras o de punto flotante). Se mantienen encoladas hasta que la UF correspondiente está disponible.

- Schedule (3 steps). Las Microinstructions son planificadas para ser ejecutadas de acuerdo a su tipo (integer, floating point, etc). Antes de esta etapa, todas las instrucciones están en el mismo orden que en el programa. En esta etapa, el **scheduler** re-ordena las instrucciones de modo de mantener todas las unidades de ejecución ocupadas. Por ejemplo, si una FPU va a estar disponible, el scheduler busca una instrucción de punto flotante para ser ejecutada, incluso si está antes una instrucción distinta antes.

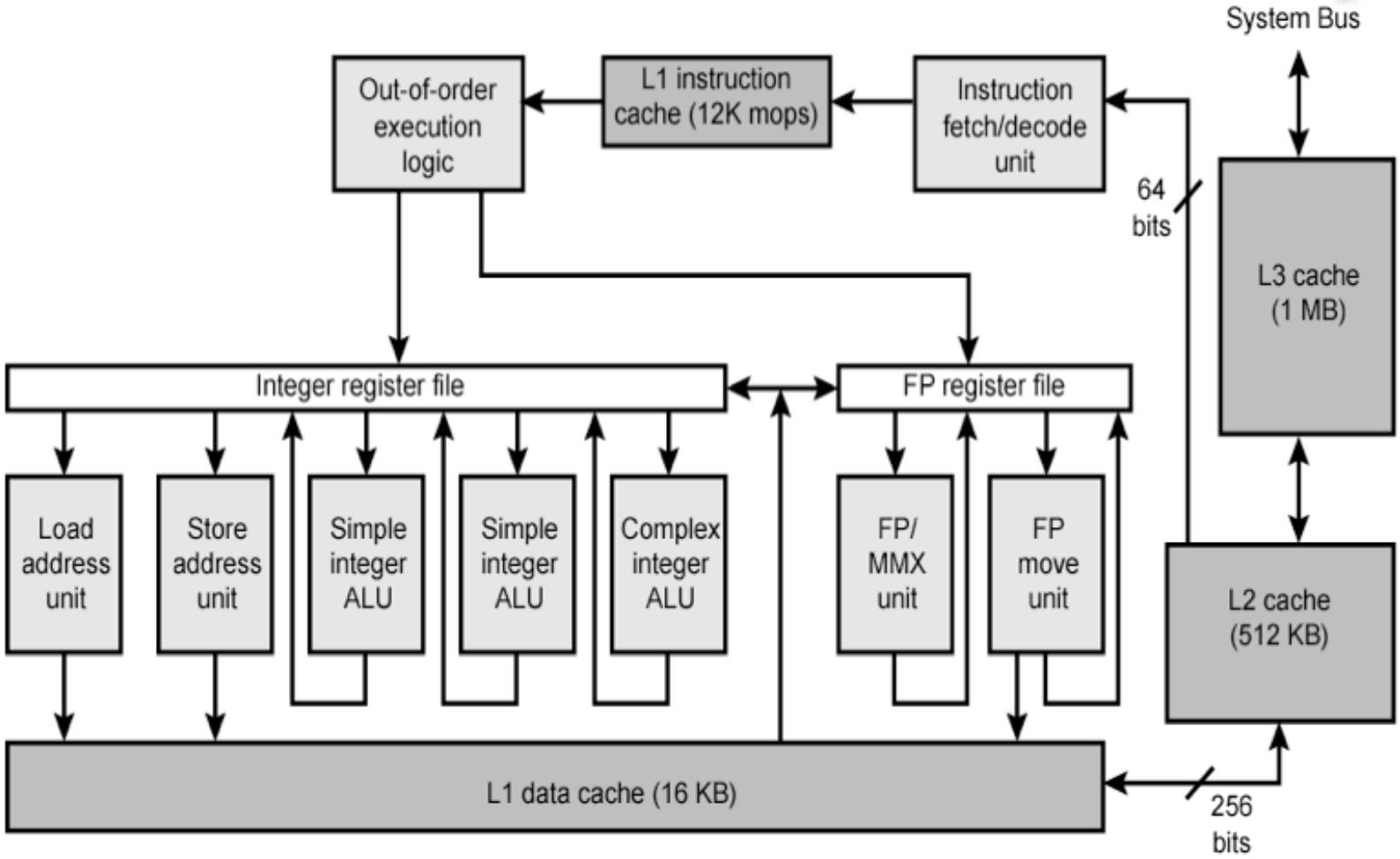
- Dispatch (2 steps). Envía las micro-instrucciones a su correspondiente unidad de ejecución.

- Register file (2 steps). Los registros internos son leídos.

Pipeline - Pentium 4

- Execute. Ejecuta las micro-instrucciones
- Flags (1 step). Los flags son actualizados.
- Branch check (1 step). Chequea si el salto tomado por el programa es el mismo predicho por el Branch Predictor.
- Drive (1 step): Envía los resultados del chequeo de branch al branch target buffer (BTB) presente a la entrada del procesador.

Pentium 4 - Micro-arquitectura



Pipeline

- El pipeline no debe afectar el comportamiento funcional de la computadora
  - Nunca provee resultados incorrectos o “ejecuta" una instrucción errónea
  - Adquiere mayor performance cuando se maximiza su ocupación
  - La lógica digital (circuitería) es más compleja que un diseño sin pipeline.

Resumen

- Performance
  - Formas de Medir Performance
  - Performance Relativa vs. Performance Absoluta
  - Benchmarking
- Pipelining
  - Conflictos
  - Branch Prediction
  - Ejemplo: Pentium 4

Bibliografía

- Performance
  - Null - Cap. 10
- Pipelining
  - Null – Cap. 5.5 (Instruction Level Pipelining)