

Arquitectura de Computadoras para Ingeniería

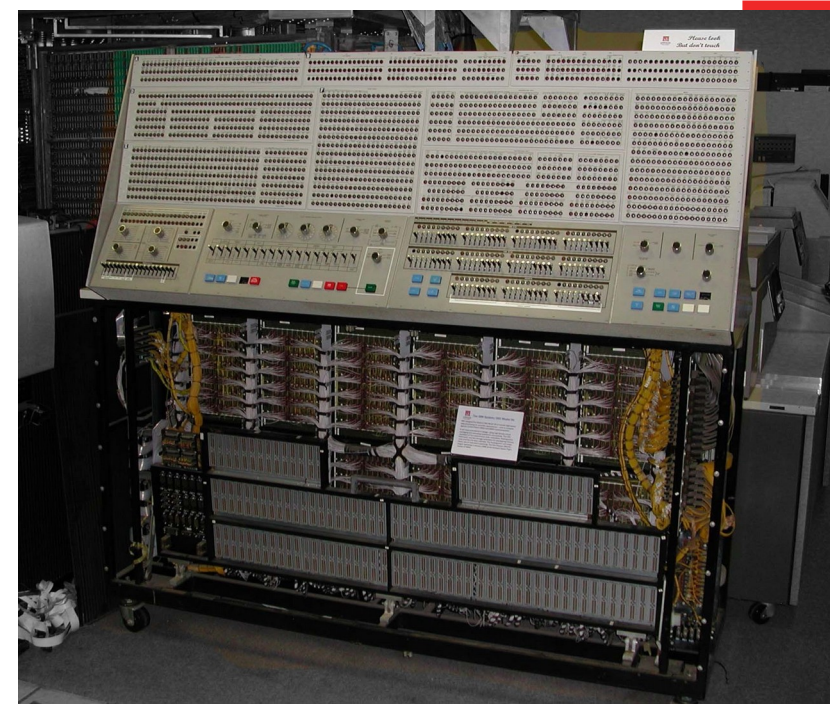
(Cód. 7526)
1° Cuatrimestre 2016

Dra. Dana K. Urribarri
DCIC - UNS

Algoritmo de Tomasulo

Algoritmo de Tomasulo

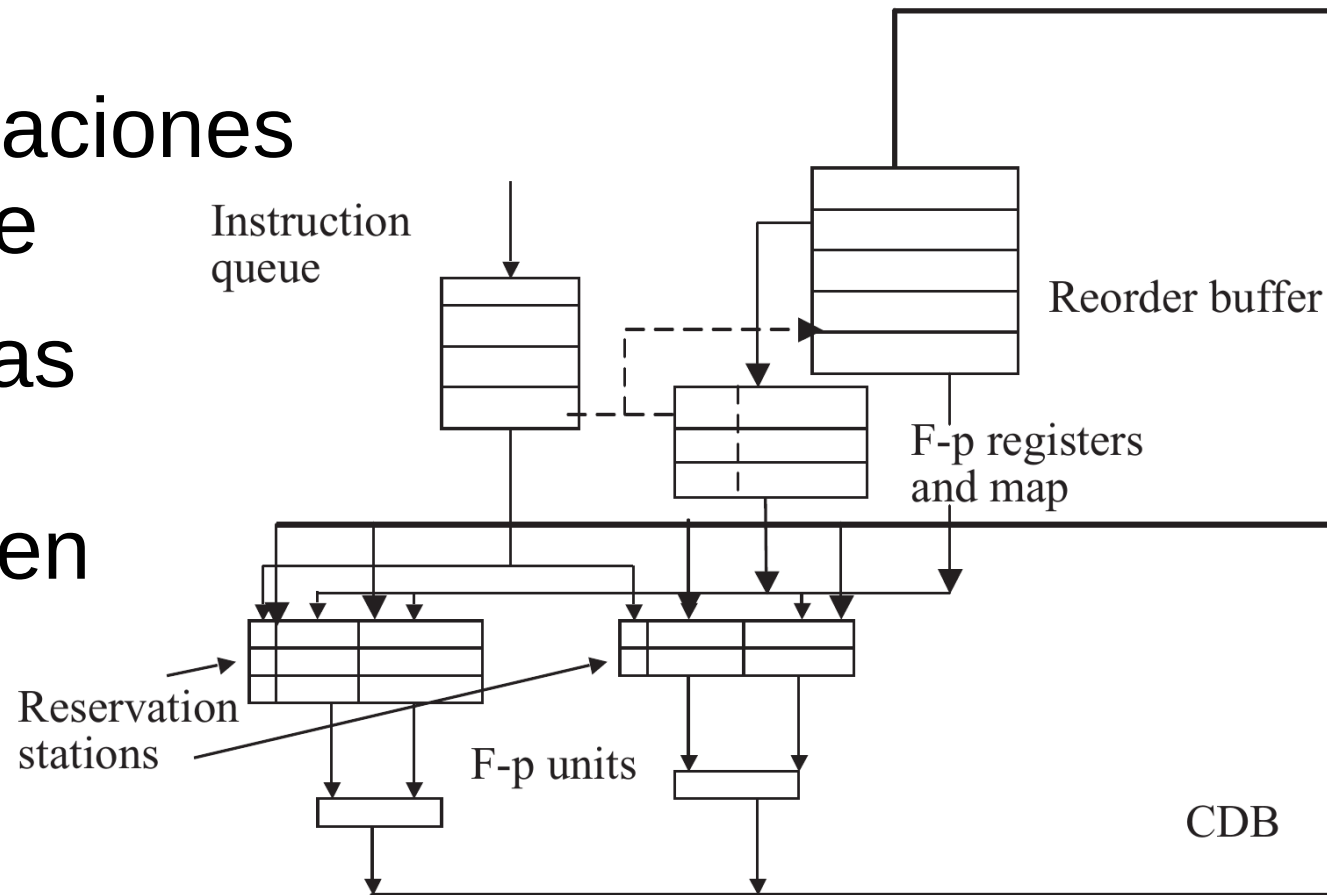
- 1967: IBM System 360/91
- Orientada a cómputo científico
→ diseño optimizado de la unidad de punto flotante.
- Un sumador en pipeline con 3 estaciones de reservación
- Un multiplicador-divisor con dos estaciones de reservación
- Una cola de instrucciones
- Conjunto de registros en punto flotante
- Buffers para los load y buffers para los store
- Un bus de datos común (CBD) para hacer *broadcast*: resultados del multiplicador, sumador y buffers de loads → estaciones de reservación, registros y buffers de store



Algoritmo de Tomasulo

Diferencias con el modelo original

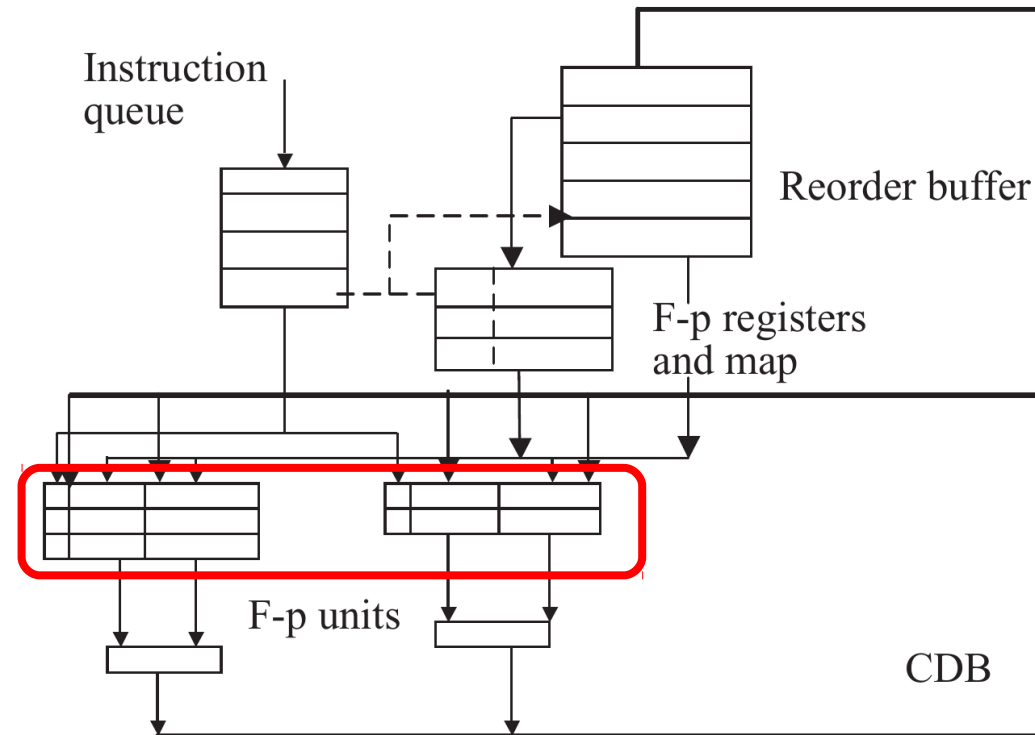
- No consideramos buffers de load y store.
Solamente operaciones de punto-flotante
- ROB para que las instrucciones terminen en orden
- Un mapeo de registros



Algoritmo de Tomasulo

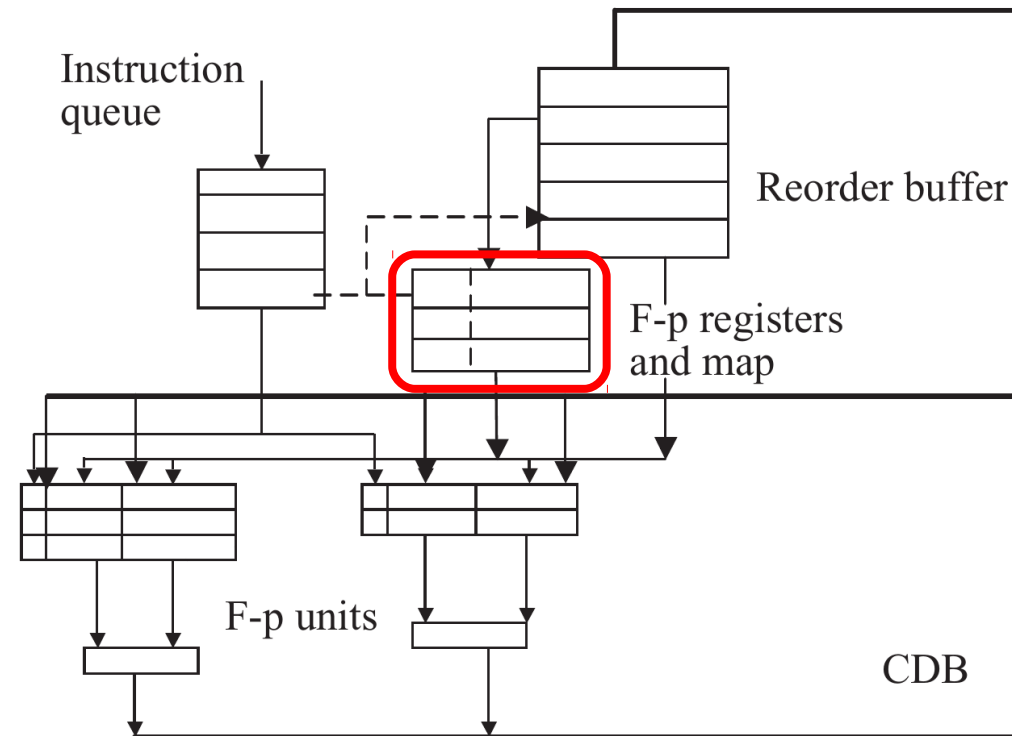
Las estaciones de reservación tienen 6 campos:

- Un bit indicando si la estación de reservación está libre o no
- Dos campos por operando: un flag y el dato. El flag indica si el dato es un valor o un nombre (tag).
- Un campo con un puntero a la entrada en el ROB (*tag*) donde se debe almacenar el resultado de la instrucción.



Algoritmo de Tomasulo

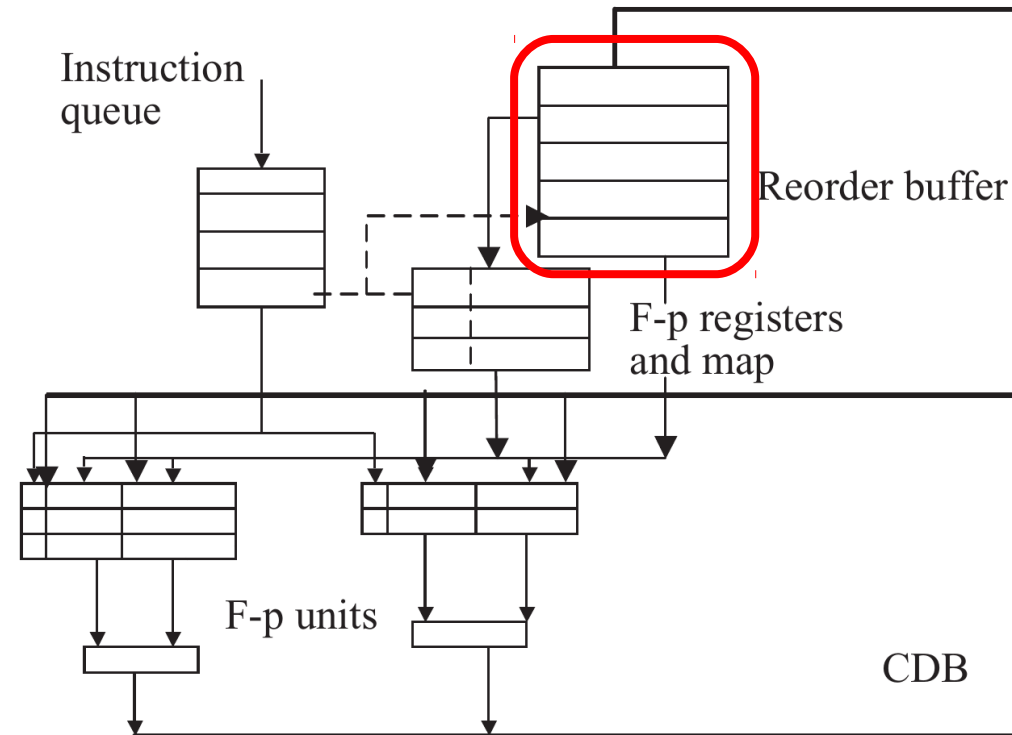
- El mapeo entre registros lógicos y físicos es con una tabla indexada por registro lógico.
- El nombre de un registro físico puede ser el índice a una entrada en el ROB (*tag*) o el registro lógico.



Algoritmo de Tomasulo

El ROB tiene dos propósitos

- Asegurar que las instrucciones terminen el orden
- Implementar el conjunto de registros físicos
- Tiene tres campos:
 - Dos campos para el registro físico: flag y dato. El flag (*ready-bit*) indica si el dato es el valor resultado de la instrucción o es un tag.
 - El nombre del registro lógico resultado



Algoritmo de Tomasulo

- Luego del fetch las instrucciones están en una cola de instrucciones
- Las siguientes etapas son:
 - Decode-rename
 - Dispatch
 - Issue
 - Execute
 - Commit

Algoritmo de Tomasulo

- Decode-rename:
 - Si la estación de reservación necesaria está llena:
⇒ Conflicto estructural
 - Si el ROB está lleno
⇒ Conflicto estructural
 - Si hay algún conflicto estructural se frena el flujo de instrucciones entrantes hasta que se resuelvan.
 - Una vez que se resuelvan los conflictos estructurales
 - Se reserva la estación de reservación
 - Se reserva la última entrada en el ROB

Algoritmo de Tomasulo

- Dispatch

Se completa la estación de reservación y la entrada del ROB:

- Para cada operando:
 - Si el mapeo indica un registro lógico, el registro contiene un valor válido.
 - Si el mapeo indica una entrada en el ROB, se chequea si esa entrada contiene un valor u otra entrada del ROB.
 - En cualquier caso, a la estación de reservación se le envía el contenido del registro lógico o de la entrada del ROB (indicado por el flag)
- Se mapea el registro resultado a la nueva entrada del ROB y se ingresa en la estación de reservación.
- Se encola la instrucción en el ROB indicando que lo asociado es un tag.

Algoritmo de Tomasulo

- Issue
 - Cuando ambos operandos en la estación de reservación están disponibles y la UF no está frenada esperando que el CDB haga *broadcast* del último resultado, se puede enviar la instrucción a la UF para que comience la ejecución.
 - Si varias estaciones de reservación están listas en el mismo ciclo, algún algoritmo de planificación decidirá cuál enviar primero.

Algoritmo de Tomasulo

- Execute
 - En el último ciclo de ejecución, la UF pide el control del CDB.
 - Si hay varias UF pidiendo el control del CDB en el mismo ciclo, se resuelve mediante algún esquema de prioridades (cableado).
 - Una vez que la UF toma el control del CDB
 - Hace un *broadcast* del resultado y del *tag* asociado a la instrucción.
 - El resultado se almacena en la entrada del ROB que indica el *tag* y se modifica el *ready bit*.
 - El resultado se almacena en todas las estaciones de reservación que tengan ese *tag* como operando y se modifica el *flag* de disponible.

Algoritmo de Tomasulo

- Commit
 - En cada ciclo, se controla el *ready bit* de la primera entrada del ROB.
 - Si está en verdadero, el valor se almacena en el registro lógico indicado en la entrada del ROB y se borra la entrada.

Inconsistencias en el uso de los nombres Issue y Dispatch

- CDC 6600 Scoreboard: issue → dispatch
- DEC Alpha 21164: Última etapa del front-end, S3, se corresponde con el issue del Scoreboard.
- Enviar una instrucción a una estación de reservación: Dispatch (tarea del *front-end*)
- Enviar una instrucción a una unidad funcional: Issue (tarea del *back-end*).
- Algoritmo de Tomasulo: dispatch → Issue

Algoritmo de Tomasulo: Ejemplo

i1: $R4 \leftarrow R0 * R2 \rightarrow$ Usa una estación de reservación del multiplicador.

i2: $R6 \leftarrow R4 * R8 \rightarrow$ Usa otra estación de reservación del multiplicador. RAW con *i1*

i3: $R8 \leftarrow R2 + R12 \rightarrow$ Usa una estación de reservación del sumador. WAR con *i2*

i4: $R4 \leftarrow R14 + R16 \rightarrow$ Usa otra estación de reservación del sumador. WAW con *i1*

Asumimos el multiplicador y el sumador en pipeline y con una latencia de 4 y 1 ciclos, respectivamente.

Ejemplo

Estado inicial

i1: $R4 \leftarrow R0 * R2$

i2: $R6 \leftarrow R4 * R8$

i3: $R8 \leftarrow R2 + R12$

i4: $R4 \leftarrow R14 + R16$

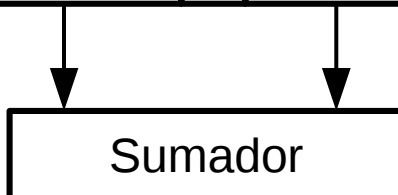
Register Map

Index	Map
...	
4	
5	
6	
7	
8	

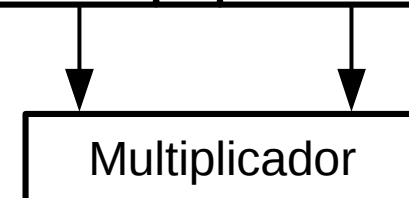
ROB

Ready bit	Data	Registro lógico

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 1

i1: $R4 \leftarrow R0 * R2$ *Decode-rename*

i2: $R6 \leftarrow R4 * R8$

i3: $R8 \leftarrow R2 + R12$

i4: $R4 \leftarrow R14 + R16$

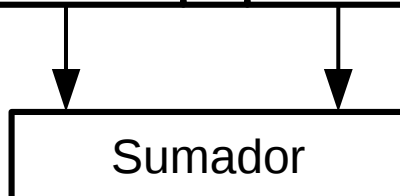
Register Map

Index	Map
...	
4	E1
5	
6	
7	
8	

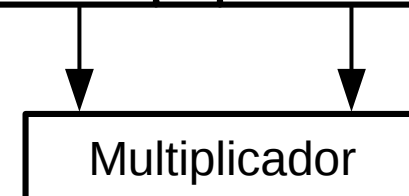
ROB

Ready bit	Data	Registro lógico
0	E1	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 2

$i1: R4 \leftarrow R0 * R2$ *Dispatch*
 $i2: R6 \leftarrow R4 * R8$ *Decode-rename*
 $i3: R8 \leftarrow R2 + R12$
 $i4: R4 \leftarrow R14 + R16$

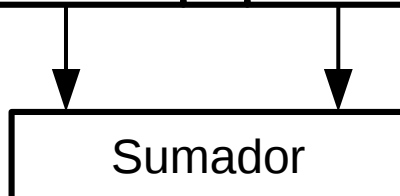
Register Map

Index	Map
...	
4	E1
5	
6	E2
7	
8	

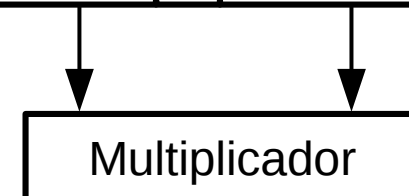
ROB

Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	1	Valor R0	1	Valor R2	E1



Ejemplo

Ciclo 3

i1: $R4 \leftarrow R0 * R2$ *Issue* → *Execute*
i2: $R6 \leftarrow R4 * R8$ *Dispatch*
i3: $R8 \leftarrow R2 + R12$ *Decode-rename*
i4: $R4 \leftarrow R14 + R16$

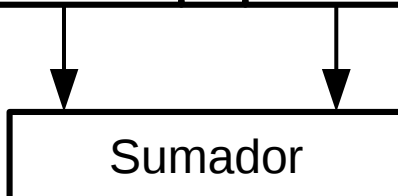
Register Map

Index	Map
...	
4	E1
5	
6	E2
7	
8	E3

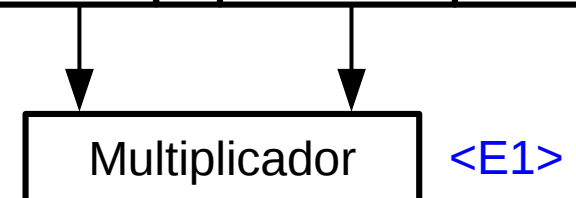
ROB

Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
0	E3	R8

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 4

i1: $R4 \leftarrow R0 * R2$ *Execute*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Dispatch*
i4: $R4 \leftarrow R14 + R16$ *Decode-rename*

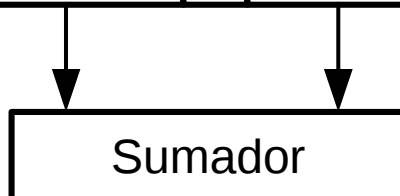
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

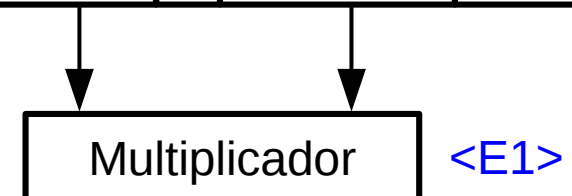
ROB

Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
0	E3	R8
0	E4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	1	Valor R2	1	Valor R12	E3



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 5

i1: $R4 \leftarrow R0 * R2$ *Execute*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Issue → Execute*
i4: $R4 \leftarrow R14 + R16$ *Dispatch*

Register Map

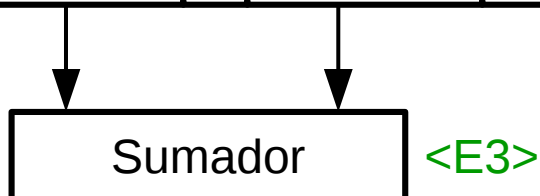
Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

ROB

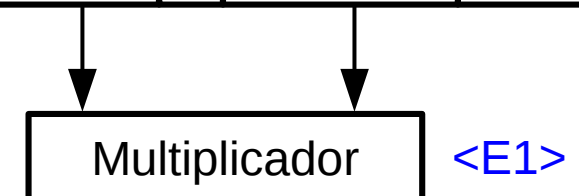
Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
0	E3	R8
0	E4	R4



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	1	Valor R14	1	Valor R16	E4



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 5 (al final)

i1: $R4 \leftarrow R0 * R2$ *Execute*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Exec (pide el CDB)*
i4: $R4 \leftarrow R14 + R16$ *Dispatch*

Register Map

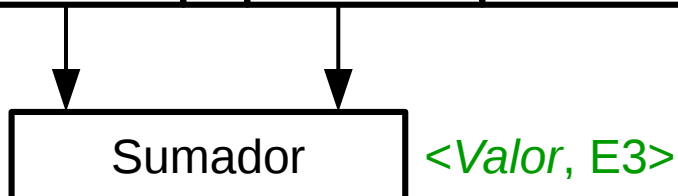
Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

ROB

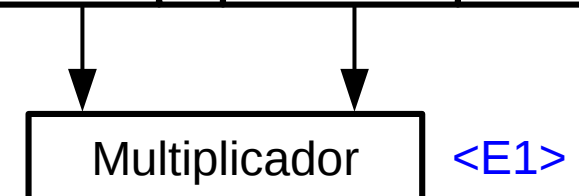
Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
0	E3	R8
0	E4	R4



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	1	Valor R14	1	Valor R16	E4



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 6

i1: $R4 \leftarrow R0 * R2$ *Execute*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Broadcast*
i4: $R4 \leftarrow R14 + R16$ *Issue → Execute*

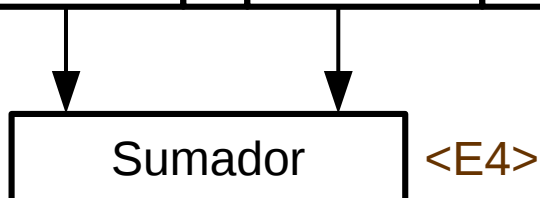
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

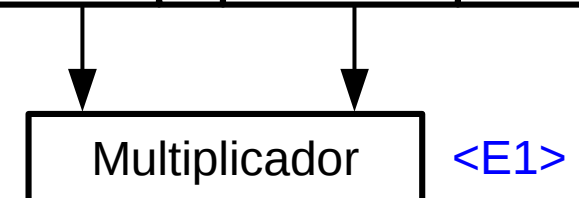
ROB

Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
1	Valor R8	R8
0	E4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 6 (al final)

i1: $R4 \leftarrow R0 * R2$ *Exec (pide el CDB)*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Broadcast*
i4: $R4 \leftarrow R14 + R16$ *Exec (pide el CDB)*

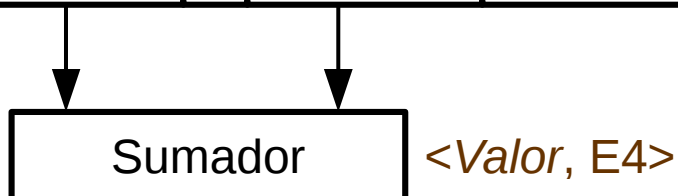
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

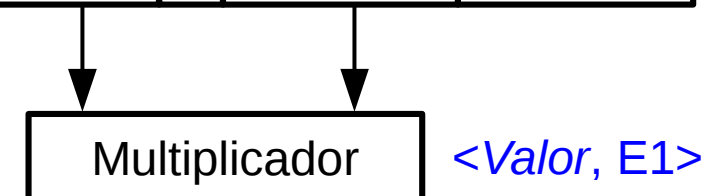
ROB

Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
1	Valor R8	R8
0	E4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 7

i1: $R4 \leftarrow R0 * R2$ *Exec-Waiting*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*
i4: $R4 \leftarrow R14 + R16$ *Broadcast*

Register Map

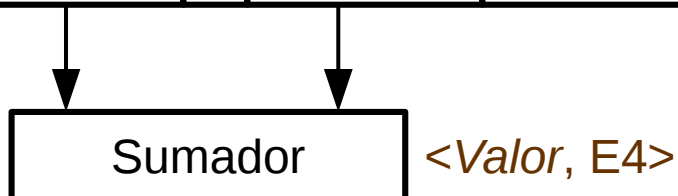
Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

ROB

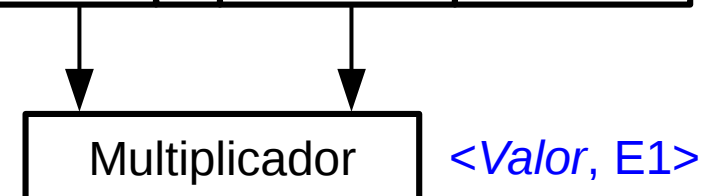
Ready bit	Data	Registro lógico
0	E1	R4
0	E2	R6
1	Valor R8	R8
1	Valor R4	R4



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	0	E1	1	Valor R8	E2



Ejemplo

Ciclo 8

- i1*: $R4 \leftarrow R0 * R2$ *Broadcast*
i2: $R6 \leftarrow R4 * R8$ *Issue-Waiting*
i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*
i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

Register Map

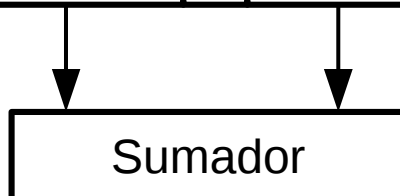
Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

ROB

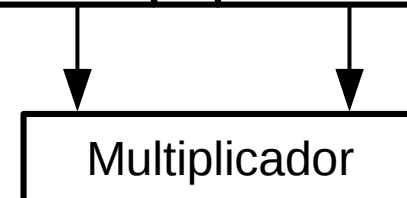
Ready bit	Data	Registro lógico
1	Valor R4	R4
0	E2	R6
1	Valor R8	R8
1	Valor R4	R4



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag
1	1	Valor R4	1	Valor R8	E2



<Valor, E1>

Ejemplo

Ciclo 9

i1: $R4 \leftarrow R0 * R2$ *Commit*
i2: $R6 \leftarrow R4 * R8$ *Issue* → *Execute*
i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*
i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

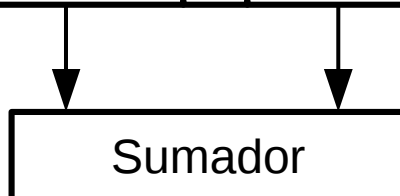
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

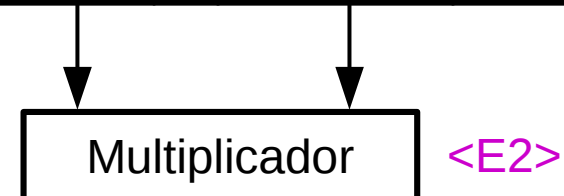
ROB

Ready bit	Data	Registro lógico
0	E2	R6
1	Valor R8	R8
1	Valor R4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 10, 11 y 12

i1: $R4 \leftarrow R0 * R2$ --

i2: $R6 \leftarrow R4 * R8$ *Execute*

i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*

i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

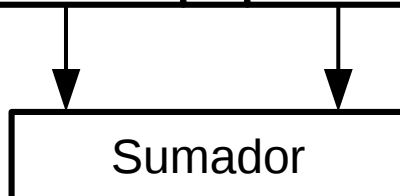
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

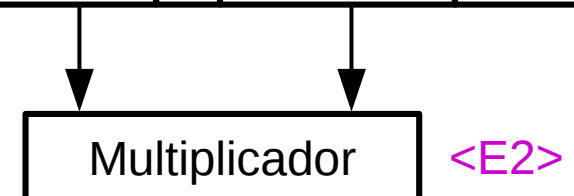
ROB

Ready bit	Data	Registro lógico
0	E2	R6
1	Valor R8	R8
1	Valor R4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 12 (al final)

i1: $R4 \leftarrow R0 * R2$ --

i2: $R6 \leftarrow R4 * R8$ *Exec (pide el CDB)*

i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*

i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

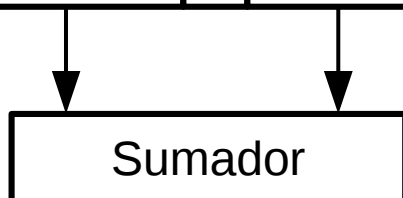
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

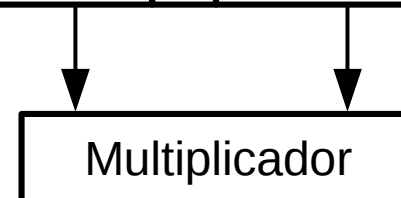
ROB

Ready bit	Data	Registro lógico
0	E2	R6
1	Valor R8	R8
1	Valor R4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



<Valor, E2>

Ejemplo

Ciclo 13

i1: $R4 \leftarrow R0 * R2$ --
i2: $R6 \leftarrow R4 * R8$ *Broadcast*
i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*
i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

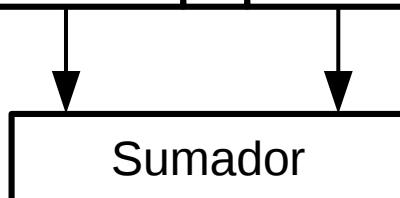
Register Map

Index	Map
...	
4	E4
5	
6	E2
7	
8	E3

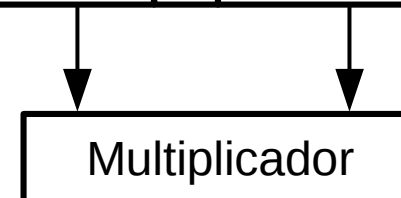
ROB

Ready bit	Data	Registro lógico
1	Valor R6	R6
1	Valor R8	R8
1	Valor R4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 14

i1: $R4 \leftarrow R0 * R2$ --
i2: $R6 \leftarrow R4 * R8$ *Commit*
i3: $R8 \leftarrow R2 + R12$ *Commit-Waiting*
i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

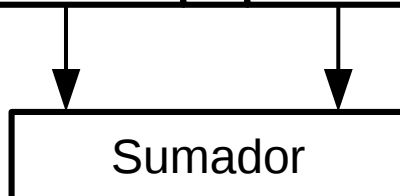
Register Map

Index	Map
...	
4	E4
5	
6	
7	
8	E3

ROB

Ready bit	Data	Registro lógico
1	Valor R8	R8
1	Valor R4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 15

i1: $R4 \leftarrow R0 * R2$ --
i2: $R6 \leftarrow R4 * R8$ --
i3: $R8 \leftarrow R2 + R12$ *Commit*
i4: $R4 \leftarrow R14 + R16$ *Commit-Waiting*

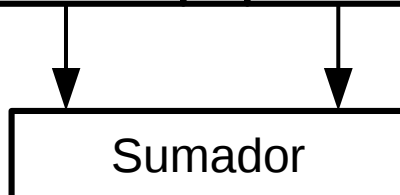
Register Map

Index	Map
...	
4	E4
5	
6	
7	
8	

ROB

Ready bit	Data	Registro lógico
1	Valor R4	R4

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Ejemplo

Ciclo 16

$i1: R4 \leftarrow R0 * R2$ --
 $i2: R6 \leftarrow R4 * R8$ --
 $i3: R8 \leftarrow R2 + R12$ --
 $i4: R4 \leftarrow R14 + R16$ *Commit*

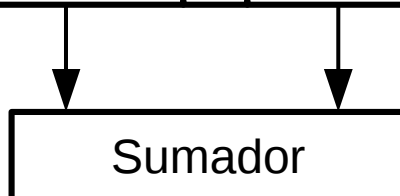
Register Map

Index	Map
...	
4	
5	
6	
7	
8	

ROB

Ready bit	Data	Registro lógico

Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



Free	Val/ tag	Oper1	Val/ tag	Oper2	Tag



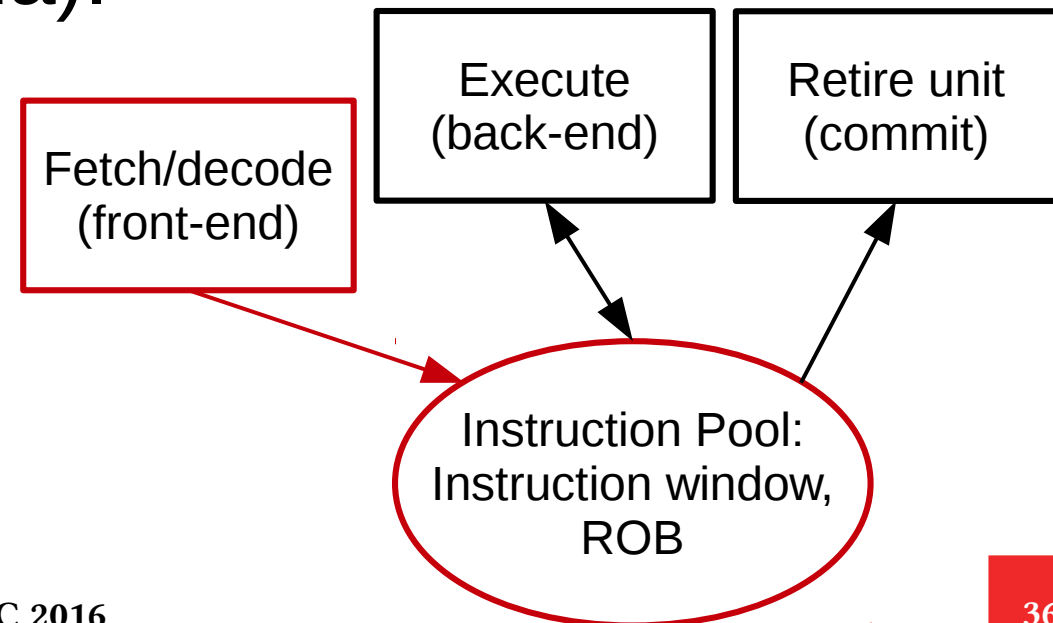
Microarquitectura Pentium P6 (i686)

Microarquitectura Pentium P6 (i686)

- Microarquitectura implementada en el procesador Pentium Pro (1995)
 - Pentium II (1997) y Pentium III (1999)
- Pentium 4 fue diferente
- Pentium M y las microarquitecturas Core están basadas en Pentium P6.

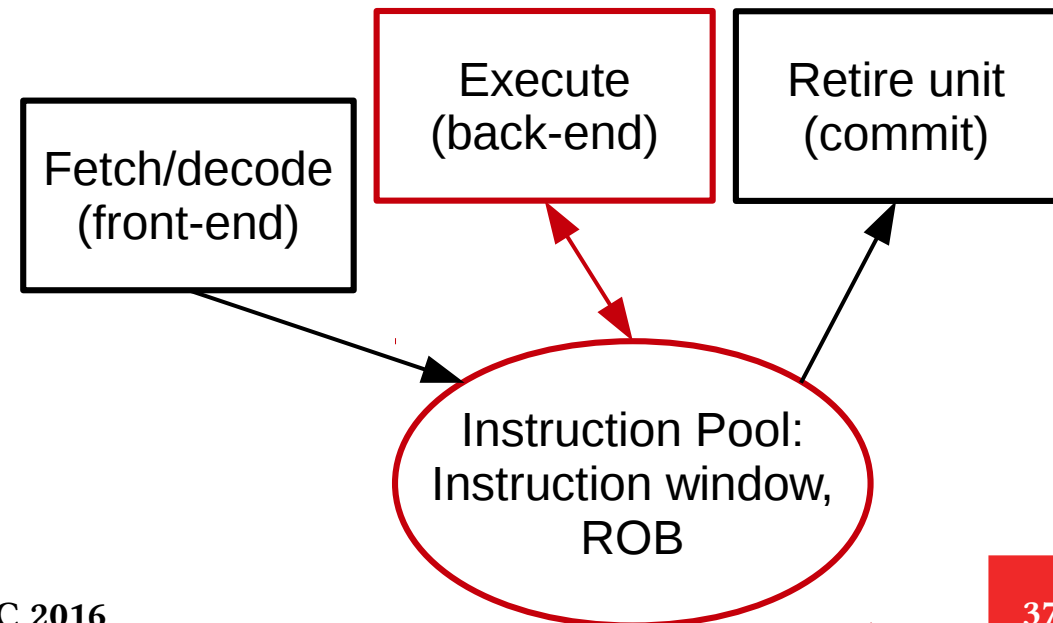
Esquema

- La unidad de *fetch/decode* implementa el *front-end* del pipeline (en orden).
- Hace el *fetch* de la instrucción, decodifica, renombra los registros y despacha la instrucción a la ventana de instrucción (estación de reservación centralizada).



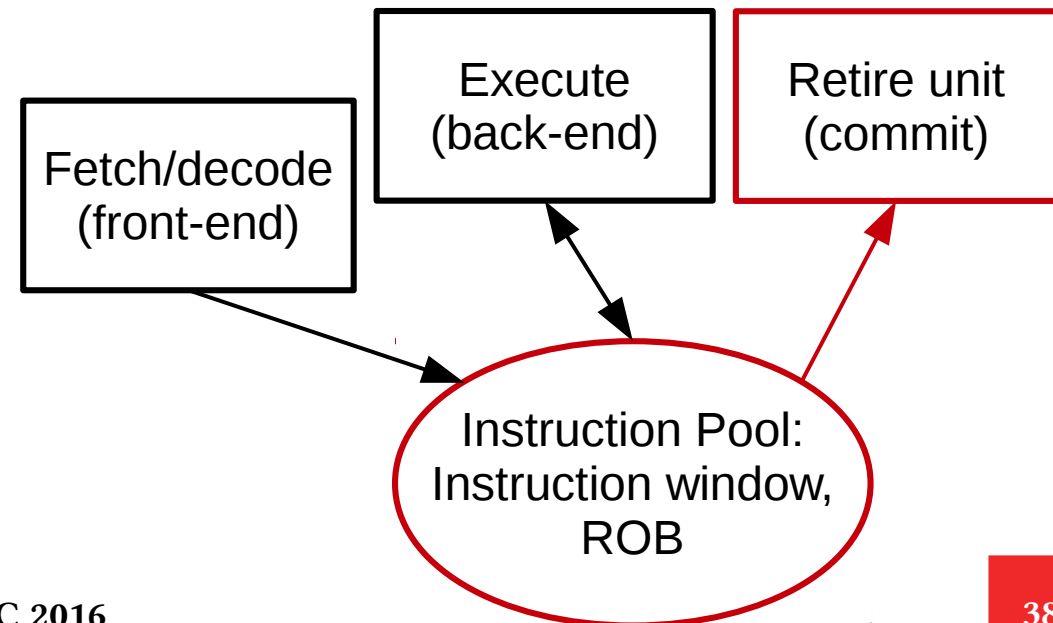
Esquema

- Las instrucciones del *pool* se envían a la unidad de ejecución (fuera de orden).
- El resultado retorna a la estación de reservación y al ROB.



Esquema

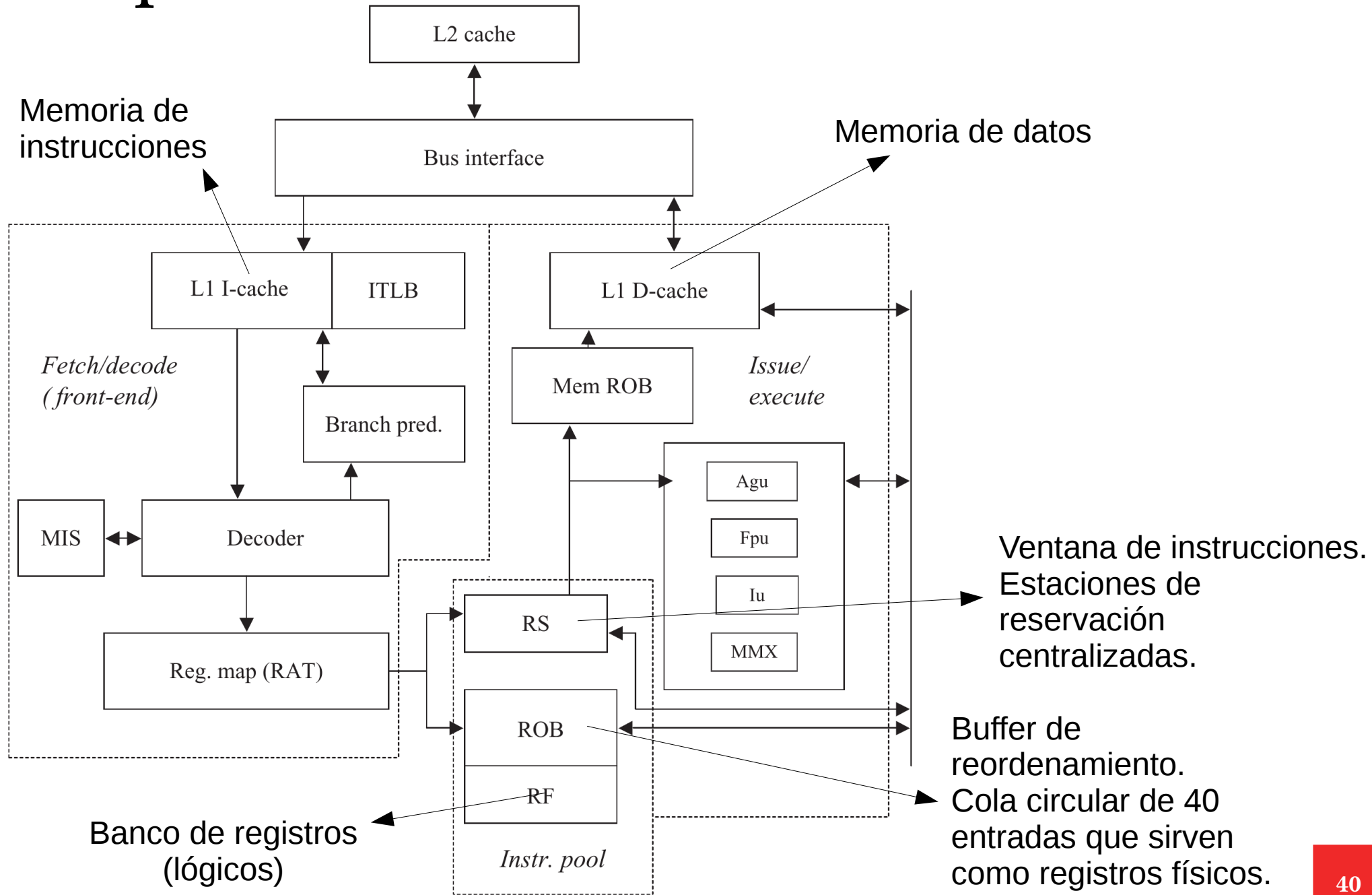
- Finalmente, la unidad de retiro hace el *commit* de la instrucción (escribiendo en registros y en memoria, en orden).



Esquema

- Pipeline de 14 etapas
- Arquitectura CISC
 - Instrucciones de tamaño variable.
 - Decodificar el set de instrucciones es más complejo que en el caso de una arquitectura RISC.
 - Se transforman las instrucciones en una secuencia de operaciones estilo RISC, llamadas μ ops
 - Del *front-end* al *Instruction pool* pueden pasar entre 3 y 6 μ ops.
 - Del *Instruction pool* al *back-end* pueden pasar 5 μ ops.
 - Todas las μ ops que componen una instrucción deben hacer el *commit* en el mismo ciclo (la ejecución debe ser atómica).

Esquema



Front-end

- Front-end: 8 etapas
 - Fetch de la instrucción y predicción de branch: 4 etapas
 - Decode: 2 etapas
 - Renombramiento de registros: 2 etapas
 - Dispatch: 1 etapa

Front-end

4 etapas del Fetch

1) Seleccionar la fuente del próximo PC:

La instrucción inmediata siguiente, la predicción del *branch* o la dirección correcta del *branch* si la predicción fue errada.

La predicción se hace en esta etapa usando un *Branch target buffer*. Demanda dos ciclos acceder el buffer y enviar la predicción de vuelta a la primera etapa.

2) 3) y 4) Si no hay branch o se predijo no tomado, se hace el fetch de las instrucciones al *Instruction Buffer* (si está vacío). Si el IB no está vacío, *Stall* del pipeline.

Front-end

2 etapas de Decode:

- 1) Se toman tres instrucciones del IB y se envían a 3 decoders que operan en paralelo.
Uno solo puede decodificar *branches*.
- 2) En caso de un *branch* (o un salto incondicional) se calcula la dirección de salto y se verifica con la dirección de la predicción. En caso de diferencia, se descarta lo que esté en el *front-end* y se reinicia en la etapa 1 con la dirección correcta.
Además, se actualiza el BTB.
Si no hubo error en el flujo de instrucciones, se almacenan hasta 6 μ ops en una cola de μ ops.

Front-end

2 etapas de Renombramiento de registros

- 1) Toma 3 μ ops a la vez de la cola de μ ops y renombra los registros.

El mapeo de registros se llama *Register Alias Table* (RAT), indica con un bit si el registro está mapeado a un registro lógico (EAX, EBX, etc) o a una de las 40 entradas del ROB.

El RAT debe tener 6 puertos de lectura.

Se reservan tres entradas disponibles en el ROB (cola circular), si no hay disponibles (conflicto estructural) se frena la unidad de renombramiento hasta que poder reservarlos.

- 2) Actualiza el ROB (para cada μ op se marca si es la primera, la última o ninguna $\rightarrow$ atomicidad de la instrucción) y si es necesarios los buffers de load y store.

Las entradas del ROB son usadas como registros físicos.

Front-end

1 etapa de Dispatch

1) Cuando haya 3 estaciones de reservación disponibles para reservar, se despachan las instrucciones a las estaciones de reservación.

Como las estaciones de reservación no se liberan en orden, se utiliza un mapa de bits para saber cuáles están libres.

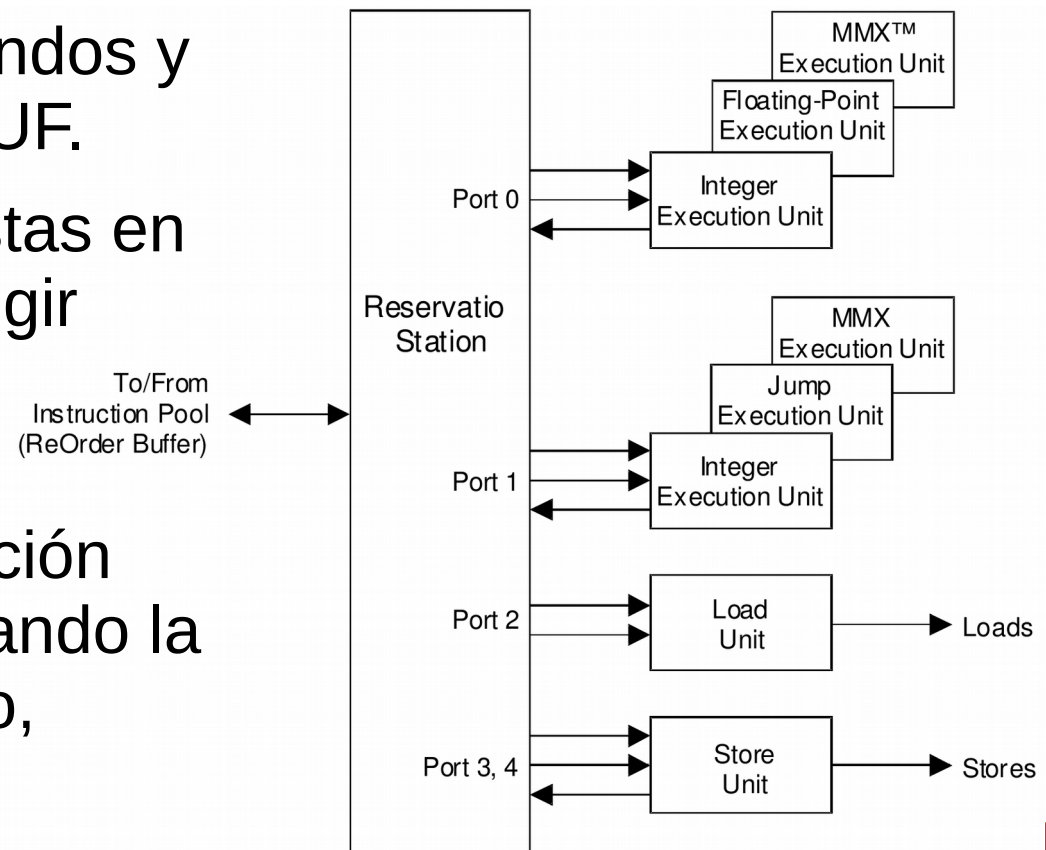
Back-end

- Back-end: 3 etapas
 - Planificación
 - *Issue*
 - Ejecución

Back-end

1 etapa de Planificación:

- Por ciclo podrían comenzar a ejecutar hasta 5 μ ops por vez. Esto va a depender de la disponibilidad de los operandos y de la disponibilidad de las UF.
- Cuando hay varias μ ops listas en el mismo ciclo, se debe elegir cual puede continuar.
- La planificación asigna la prioridad a las μ ops en función del tiempo que lleva esperando la instrucción. A mayor tiempo, mayor prioridad.



Back-end

1 etapa de *Issue*

- Se envían las μ ops a las UF.
- Una vez que se envía μ op se libera la estación de reservación.

Etapas de Execute

- La latencia de la ejecución de las UF depende de la operación que realicen.
- Una vez que termina la ejecución (si el CDB está disponible) se hace el *broadcast* del resultado y se guarda en el ROB.

Commit

3 etapas para *Commit*

- 1) Se resuelven los conflictos de control (error en la predicción de *branches*, excepciones, ...)

Estamos dejando de lado lo que concierne a loads y stores.

Todas las μ ops que constituyen una misma instrucción se retiran juntas y demanda dos ciclos:

- 2) Leer el contenido del ROB
- 3) Escribir en el banco de registros.

Predicción de los *branches*

- Si se erró la predicción del *branch*:
 - 1) Se vacía le *front-end* y se comienza con las instrucciones correspondientes al nuevo PC hasta el renombramiento de registros.
 - 2) El *back-end* continúa hasta que terminen todas las μ ops. Las μ ops delante del *branch* en el ROB pueden hacer el *commit*.
 - 3) Se descartan las μ ops que siguen al *branch* y se borran del RAT todos los mapeos al ROB. Se prosigue con la normal operación, a partir del renombramiento de registros.
- Es necesario un predictor sofisticado.
 - En el mejor de los casos, luego de errar la predicción, pasaron 12 ciclos (9 de *front-end* y 3 de ejecución)

Bibliografía



- Capítulo 3. Multiprocessor Architecture. From simple pipelines to chip multiprocessor. Jean-Loup Baer. Cambridge University Press. 2010.

Suplementaria

- Capítulo 3 y Apéndice C. John L. Hennessy & David A. Patterson. *Computer Architecture. A Quantitative Approach*. Elsevier Inc. 2012, 5ta Ed.
- Capítulo 4. David A. Patterson & John L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface*. Elsevier Inc. 2014, 5ta Ed.