Instruction Level Parallelism (ILP)

- Pipelining supports a limited sense of ILP
  - e.g. overlapped instructions, out of order completion, hazard issues, bypass logic, etc.
- Remember

Pipeline CPI = Ideal Pipeline CPI + Struct. Stalls + RAW Stalls + WAR Stalls + WAW Stalls + Control Stalls

Issues involving crossing branches

- Now turn consideration to
  - methods which reduce the terms on the RHS of the equation
  - solution is of course a combo of HW and SW/Compiler methods

Methods

Technique	Reduces
Loop Unrolling	Control Stalls
Basic Pipeline Scheduling (we've seen some of this already)	RAW stalls
Dynamic Scheduling with Scoreboarding	RAW stalls
Dynamic Scheduling with register renaming	RAW stalls
Dynamic Branch Prediction	Control Stalls
Issuing multiple instructions per cycle a.k.a Superscalar	Ideal CPI
Compiler dependence analysis	Ideal CPI and data stalls (RAW, WAW, WAR)
Software pipelining and trace scheduling	Ideal CPI and data stalls
Speculation	data and control stalls
Dynamic memory disambiguation	RAW stalls involving memory

ILP within a Basic Block

- Basic Block definition
  - straight-line code - no branches out
  - single entry point at top
  - ==> real code is a bunch of basic blocks connected by branch statements
- Notice
  - branch frequency is approx. 15% of total mix
  - implies basic block size between 6 and 7 instructions
  - machine instructions don't do much
  - there's probably little in the way of parallelism
- Easiest target is the loop
  - already exploited by vector processors but using different mechanisms

Loop Level Parallelism

- Consider adding two 1000 element arrays

```
for (i=1, i<=1000, i=i+1
        x[i] = x[i] + y[i]
```

- Sure it's trivial but it illustrates the point
  - there is no dependence between data values produced in any iteration j and those needed in j+n for any j and n
  - truly independent - hence could be a 1000 way parallel program
  - independence means no stalls due to data hazards
  - problem is that we have to use that pesky branch instruction
    - prediction here is easy - BUT we still need to have a prediction scheme
- Vector Processor model
  - load vectors x and y (up to some machine dependent max)
  - then do *result-vec = xvec + yvec* in a single instruction

Some Assumptions

Isn't this course just full of them - this time they're stated

- Default DLX pipeline structure

Instruction Producing Value	Instruction Consuming Value	Intervening Clock Cycles to Avoid Stalls
FP ALU Op	FPU ALU Op	3
FP ALU Op	Store Double	2
Load Double	FP ALU Op	1
Load Double	Store Double	0
Integer Load	Integer ALU Op	1
Integer ALU Op	Integer ALU Op	0
Branch Delay Slots	Anything	1

Note: latencies are somewhat smaller than we saw last lecture
Reason: make the examples less cumbersome

DESENROLLAMIENTO DE BUCLES: COMPILACIÓN BÁSICA

- Suma de un escalar S a un vector X (valores en doble precisión):
  - **for** (i=1, i<=1000, i++)
    X[i] = X[i] +S
  - Compilación básica para DLX (se supone que: 1/ el escalar está contenido en F2 (F2#F3); 2/ inicialmente R1 apunta al primer componente del vector; 3/ por simplicidad, el vector termina en la dirección 8.

		Ciclo de emisión (relativo)
loop:	LD F0, 0(R1)	1
	ADDD F4, F0, F2	3
	SD 0(R1), F4	6
	SUBI R1, R1, #8	7
	BNEZ R1, loop	9
	NOP	10

PLANIFICACIÓN DEL BUCLE

		Ciclo de emisión (relativo)
loop:	LD F0, 0(R1)	1
	SUBI R1, R1, #8	2
	ADDD F4, F0, F2	3
	BNEZ R1, loop	4
	SD 8(R1), F4	6

- SD se ha llevado al hueco de retardo de la bifurcación.
  - Al quedar detrás de SUBI el desplazamiento pasa a ser 8.
- SUBI se coloca antes de ADDD para conseguir:
  - Eliminar la parada de ADDD por su dependencia con LD.
  - Eliminar la parada de BNEZ por su dependencia con SUBI.
- SD se emite un ciclo más tarde por su dependencia con ADDD.

DESENROLLAMIENTO DEL BUCLE

- Idea: Concatenar varios cuerpos del bucle en un bloque básico.
  - Ventajas:
    - Más instrucciones en el bloque => más posibilidades de planificarlo => menos paradas.
    - Menos pasadas por el nuevo bucle => ahorro de instrucciones de control del bucle (en nuestro ejemplo SUBI y BNEZ).
  - Inconvenientes:
    - Se necesitan más registros (no hay que utilizar los mismos en cada copia del bucle si no queremos perder las posibilidades de planificarlo).
    - Mayor tamaño de código.

BUCLE DESENROLLADO

Bucle desenrollado no planificado		Bucle desenrollado y planificado	
	Ciclo de emisión		Ciclo de emisión
loop: LD F0, 0(R1)	1	loop: LD F0, 0(R1)	1
ADDD F4, F0, F2	3	LD F6, -8(R1)	2
SD 0(R1), F4	6	LD F10, -16(R1)	3
LD F6, -8(R1)	7	LD F14, -24(R1)	4
ADDD F8, F6, F2	9	ADDD F4, F0, F2	5
SD -8(R1), F8	12	ADDD F8, F6, F2	6
LD F10, -16(R1)	13	ADDD F12, F10, F2	7
ADDD F12, F10, F2	15	ADDD F16, F14, F2	8
SD -16(R1), F12	18	SD 0(R1), F4	9
LD F14, -24(R1)	19	SD -8(R1), F8	10
ADDD F16, F14, F2	21	SUBI R1, R1, #32	11
SD -24(R1), F16	24	SD 16(R1), F12	12
SUBI R1, R1, #32	25	BNEZ R1, loop	13
BNEZ R1, loop	27	SD 8(R1), F16	14
NOP	28		

- Ciclos por elemento del vector procesado: 10 (básico), 6 (planificado), 7 (no planificado y desenrollado), 3,5 (desenrollado y planificado).

Things to Notice

- We had 8 more unused register pairs
  - hence we could have gone to an 8 block unroll without register conflict
  - no problem since the 1000 element array would still have broken cleanly (1000/8 = 125)
  - what if it had not - say with remainder R
  - so what just put R blocks (shuffled of course) in front of the Loop then start for real
  - you'll notice that you run out of registers but by cycling through the names you can still remove any stalls in this case
- Still (most compilers unroll early to expose code for later opt's)
  - there were many things the compiler had to notice to figure this one out (trickiest was the SD/SUBI swap - why?)
  - Key was the independent nature of each loop body
  - 3 types of dependence: *data*, *name*, and *control*

DEPENDENCIAS DE DATOS

- k tiene una **dependencia** de datos sobre i si:
  - k utiliza un resultado producido por i (dependencia RAW) o
  - k utiliza un resultado producido por j y j tiene una dependencia de datos sobre i.
- Las dependencias de datos son propiedades de los programas.
  - Determinan el orden en que deben calcularse los resultados.
  - Establecen un **límite al paralelismo** disponible (instrucciones dependientes no son paralelizables).
  - Producen o no paradas según las características concretas del cauce y la separación entre las instrucciones dependientes.
- El código se puede mejorar de dos modos:
  - Transformándolo para eliminar la dependencia.
  - Mantener la dependencia pero eliminar las paradas.

DEPENDENCIAS DE NOMBRE

- Sea i anterior a j: j tiene una **antidependencia de datos** sobre i si:
  - j escribe en un registro o zona de memoria (nombre) que es leído por i.
    - Es pues una dependencia WAR.
    - Para que no cree problema i debe leer antes de que escriba j.
- Sea i anterior a j: j tiene una **dependencia de salida** sobre i si:
  - Ambas instrucciones escriben en el mismo registro o zona de memoria.
    - Es pues una dependencia WAW.
    - Para que no cree problema i debe escribir antes que j.
- En estas **dependencias no hay** una **transmisión de** un **valor** entre las instrucciones (como en las de datos) => **se pueden eliminar cambiando**, en una de las instrucciones, **el nombre** del operando que crea conflicto.
  - El renombrado es más sencillo con los registros. Puede hacerse por hardware o mediante el compilador.

Control Dependence

- Since branches are conditional
  - some instructions will be executed and others will not
  - instructions before the branch don't matter
  - so only possibility is between a branch and instructions which follow it
- 2 obvious constraints to maintain control dep's
  - instructions controlled by the branch cannot be moved before the branch (since it would then be uncontrolled)
  - an instruction not controlled by the branch cannot be moved after the branch (since it would then be controlled)
- Note
  - transitive control dependence is also a factor
  - In simple pipelines - order is preserved anyway so no big deal
  - Of course the pipeline world could get hairier

DEPENDENCIAS DE CONTROL (cont.)

- Mantener las dependencias de control no es crítico.
  - Algunas se pueden eliminar, se pueden ejecutar instrucciones que no deberían ejecutarse (como se vio al estudiar las bifurcaciones retardadas)...
- Las dos propiedades críticas para que el programa se mantenga correcto son

  A. Preservar el comportamiento de las **interrupciones**:
     - Por ejemplo, reordenar la ejecución de las instrucciones no debe causar nuevas interrupciones.
       - Así dada la secuencia `[BEQZ R3, L1; DIV R1, R2, R3]` (considerando saltos no retardados), si colocamos la instrucción `DIV` antes de `BEQZ`, la instrucción `DIV` puede causar una excepción (división por cero) que no causaría si no la movemos.

  B. No cambiar el **flujo de datos** entre instrucciones que producen resultados e instrucciones que consumen (leen) esos resultados. [sigue]

DEPENDENCIAS DE CONTROL (cont.)

- Consideremos

```
        ADD   R1, R2, R3
        BEQZ  R4, L
        SUB   R1, R5, R6
L:      OR    R7, R1, R8
```

  - El valor de R1 usado por OR depende del camino seguido en la bifurcación.
  - La dependencia de datos no es suficiente para preservar el código correcto.
  - Hay que preservar la dependencia de control de SUB sobre BEQZ para evitar un cambio ilegal en el flujo de datos.

- La especulación y las instrucciones condicionales ayudan a preservar el comportamiento de las interrupciones y permiten cambiar dependencias de control manteniendo el flujo de datos. [Se estudiarán más adelante]

Now Consider

```
for (i=1; i<=1000; i++) {
        A[i+1] = A[i] + C[i];          /* S1*/
        B[i+1] = B[i] + A[i+1];}       /* S2*/
```

- S1 uses S1 value produced in an earlier iteration
- also S2 uses S2 value produced in an earlier iteration
- S2 uses an S1 value produced in the same iteration

Implication

- S1 depends *(loop carried)* on S1 hence their order must be preserved
- similar for the S2 carried dependence
- note if non-loop carried dependences were the only ones then the order would not matter

One More

```
for (i=1; i<=100; i++) {
        A[i] = A[i] + B[i];          /* S1*/
        B[i+1] = C[i] + D[i];}       /* S2*/
```

- S1 uses previous value of S2
- however the dependence is not circular since neither statement depends on itself
- and no S1 depends on S2 depends on S1 circularity either

Implications

- No cycle in dependencies ==> loop can be parallelized to:

```
A[1] = A[1] + B[1]
```

VIOLA - no loop carried dependence so UNROLL

```
for (i=1; i<=99; i++) {
            B[i+1] = C[i] + D[i];}       /* S1*/
            A[i+1] = A[i+1] + B[i+1];    /* S2*/

B[101]=C[100] + D[100]
```

Dynamic Pipeline Scheduling

Compiler does Static Scheduling

- Hardware rearranges instruction stream to reduce stalls
- Treats dependencies which are only known at run time - e.g. memory reference
- Simplifies the compiler
  - even permits compile for multiple machines to work
  - minimizes the need for speculative slot filling - NP hard problem anyway
- Common tactic with supercomputers and mainframes
- Now mainframe tactics fit on a chip

Dynamic Scheduling

a.k.a. dynamic instruction reordering

- Allow out of order issue
- Consider

```
DIVD    F0, F2, F4
ADDD    F10, F0, F8
SUBD    F12, F8, F14
```

hazards?

  - DIVD has a long latency
  - ADDD has a data dependence on F0, SUBD does not
  - so swap them - compiler might have done this but so could the hardware
- Problems?
  - big time - so for now lets ignore precise interrupts

SCOREBOARD (marcador)

- **Hardware** que detecta los riesgos y **controla la emisión, ejecución y terminación de las instrucciones**.
  - Emisión en orden. Ejecución y terminación: fuera de orden.
    - Varias instrucciones en la etapa de ejecución (EX).
      - Varias unidades funcionales (posiblemente encauzadas) forman la etapa EX.
- En DLX
  - Supondremos dos multiplicadores FP (10 ciclos), un divisor FP (40 ciclos), un sumador FP (2 ciclos) y una unidad funcional entera (1 ciclo).
  - Consideraremos sólo las etapas relevantes para la técnica de *scoreboard*: *Issue*, *read operands*, *EX y write result*.

SCOREBOARD: ESTRUCTURA

- Tabla de **estados de instrucciones**: Indica en qué etapa está cada instrucción.
- Tabla de **estados de las unidades funcionales**. Por cada unidad funcional hay los siguientes campos:
  - *Busy*. Indica si la unidad funcional (U.F.) está ocupada o no.
  - Op. Operación que está siendo ejecutada, en su caso, en la U.F.
  - Fi. Registro destino del resultado de la U.F.
  - Fj, Fk. Registros fuente de los operandos que debe procesar la U.F.
  - Qj, Qk. Unidades funcionales que tienen como registros destino Fj y Fk.
  - Rj, Rk. Indicadores de que Fj, Fk están listos para ser procesados.
- Tabla **registros-resultados**. Por cada registro un campo.
  - Si una instrucción activa tiene el registro como su destino, el campo indicará qué **unidad funcional** escribirá en ese registro.
  - Si no, el campo queda en blanco.

SCOREBOARD: ESTADO INICIAL

Tabla de estados de las instrucciones.

Instrucción	Issue	Read operands	Execution complete	Write result

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional									

SCOREBOARD: COMPORTAMIENTO

- La etapa ID se divide en dos etapas:
  - ***Issue***.
    - Descodifica la instrucción. No se emite si hay **riesgo estructural o WAW**.
      - Si hay una unidad funcional libre para la instrucción y ninguna instrucción activa tiene el mismo registro destino, el *scoreboard* emite la instrucción y actualiza su estructura de datos interna.
    - Si una instrucción no se emite tampoco se emite ninguna instrucción posterior.
      - En consecuencia, el buffer de instrucciones FIFO que hay entre IF e ISSUE se llena con una instrucción más (la buscada por IF).

SCOREBOARD: COMPORTAMIENTO (cont.)

- ***Read operands***.
  - **Esperar** a que los **operandos fuente** estén **disponibles**.
    - Un operando fuente está disponible si ninguna instrucción activa emitida antes está pendiente de escribirlo o si el registro que contiene el operando está siendo escrito por una unidad funcional.
    - Cuando los operandos fuente están disponibles el *scoreboard* ordena que la unidad funcional lea los operandos de los registros e inicie la ejecución.
  - De esta forma **se resuelven las dependencias RAW dinámicamente**.
  - Las instrucciones pueden enviarse a ejecución **fuera de orden**.

❑ Etapa EX.

- Cuando una unidad funcional termina una ejecución lo notifica al *scoreboard.*

SCOREBOARD: COMPORTAMIENTO (cont.)

- Etapa ***write result*** (reemplaza a WB).
  - Una vez que la unidad funcional ha terminado el *scoreboard* verifica las posibles **dependencias WAR**.
    - Si hay, detiene la instrucción.
    - Si no hay escribe el resultado.
  - Ejemplo:

```
DIVD F0,F2,F4
ADDD F10,F0,F8
SUBD F8,F8,F14
```

  - SUBD se ejecutará antes que ADDD (porque esta tiene una dependencia con DIVD) pero será detenida en la etapa WB hasta que ADDD (que se emitió antes) haya leído sus operandos para evitar la dependencia WAR.

SCOREBOARD: COMPORTAMIENTO DETALLADO

Notaciones:

- FU: Unidad funcional usada por la instrucción.
- C(F): Contenido de la columna C, fila F de la tabla de estados de las unidades funcionales.
- Result(X): Contenido del componente X del vector registros-resultados, es decir, unidad funcional que escribirá en X.
- op: Operación que corresponde a la instrucción.
- 'D', 'S1', 'S2': Nombres (identificadores) de los registros destino, fuente1 y fuente2, respectivamente.

Etapa	Esperar a	Registrar en tablas del *scoreboard*
Issue	Busy(FU) = No & Result('D') = Ø	Result('D') ← FU; Busy(FU) ← Yes; Op(FU) ← op; Fi(FU) ← 'D'; Fj(FU) ← 'S1'; Fk(FU) ← 'S2'; Qj(FU) ← Result('S1'); Rj(FU) ← (if Qj(FU) = Ø then Yes else No); Qk(FU) ← Result('S2'); Rk(FU) ← (if Qk(FU) = Ø then Yes else No);
Read operands	Rj(FU) ≠ No ≠ Rk(FU)	Rj(FU) ← No; Rk(FU) ← No
Execution complete	Que la unidad funcional termine	
Write result	∀f (Fj(f) ≠ Fi(FU) or Rj(f) = No) **&** (Fk(f) ≠ Fi(FU) or Rk(f) = No)	∀f (if Qj(f) = FU then {Rj(f) ← Yes; Qj(f) ← Ø} ∀f (if Qk(f) = FU then {Rk(f) ← Yes; Qk(f) ← Ø}; Result(Fi(FU)) ← Ø; Busy(FU) ← No; ∀c (if c ≠ Busy then c(FU) ← Ø)

Se ha supuesto una instrucción con dos registros fuente y un registro destino, para contemplar el caso más general. Si la instrucción no tuviese, por ejemplo, registro fuente S2, habría que ignorar Fk(FU), Qk(FU), Rk(FU).

CICLO 1

SCOREBOARD: EJEMPLO

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X			
LD	F2, 45(R3)				
MULTD	F0, F2, F4				
SUBD	F8, F6, F2				
DIVD	F10, F0, F6				
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	***Yes***	***Load***	***F6***	***R2***	Ø	Ø	Ø	***Yes***	Ø
Mult1	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Divide	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Ø	Ø	Ø	***Integer***	Ø	Ø	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	**X**		
LD	F2, 45(R3)				
MULTD	F0, F2, F4				
SUBD	F8, F6, F2				
DIVD	F10, F0, F6				
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F6	R2	Ø	Ø	Ø	***No***	Ø
Mult1	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Divide	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Ø	Ø	Ø	Integer	Ø	Ø	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	**X**	**X**
LD	F2, 45(R3)				
MULTD	F0, F2, F4				
SUBD	F8, F6, F2				
DIVD	F10, F0, F6				
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	***No***	**Ø**	**Ø**	**Ø**	Ø	Ø	Ø	**Ø**	Ø
Mult1	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Divide	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Ø	Ø	Ø	**Ø**	Ø	Ø	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	**X**			
MULTD	F0, F2, F4				
SUBD	F8, F6, F2				
DIVD	F10, F0, F6				
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	***Yes***	***Load***	***F2***	***R3***	Ø	Ø	Ø	***Yes***	Ø
Mult1	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Divide	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Ø	***Integer***	Ø	Ø	Ø	Ø	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	**X**		
MULTD	F0, F2, F4	**X**			
SUBD	F8, F6, F2				
DIVD	F10, F0, F6				
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3	Ø	Ø	Ø	***No***	Ø
Mult1	***Yes***	***Mult***	***F0***	***F2***	***F4***	***Integer***	Ø	***No***	***Yes***
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Divide	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	***Mult1***	Integer	Ø	Ø	Ø	Ø	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	**X**	
MULTD	F0, F2, F4	X			
SUBD	F8, F6, F2	**X**			
DIVD	F10, F0, F6				
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3	Ø	Ø	Ø	No	Ø
Mult1	Yes	Mult	F0	F2	F4	Integer	Ø	No	Yes
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	***Yes***	***Sub***	***F8***	***F6***	***F2***	Ø	***Integer***	***Yes***	***No***
Divide	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Mult1	Integer	Ø	Ø	***Add***	Ø	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	X	**X**
MULTD	F0, F2, F4	X			
SUBD	F8, F6, F2	X			
DIVD	F10, F0, F6	**X**			
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	***No***	***Ø***	***Ø***	***Ø***	Ø	Ø	Ø	***Ø***	Ø
Mult1	Yes	Mult	F0	F2	F4	***Ø***	Ø	***Yes***	Yes
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	Yes	Sub	F8	F6	F2	Ø	***Ø***	Yes	***Yes***
Divide	***Yes***	***Div***	***F10***	***F0***	***F6***	***Mult1***	Ø	No	Yes

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Mult1	***Ø***	Ø	Ø	Add	***Divide***	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	X	X
MULTD	F0, F2, F4	X	**X**		
SUBD	F8, F6, F2	X	**X**		
DIVD	F10, F0, F6	X			
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult1	Yes	Mult	F0	F2	F4	Ø	Ø	***No***	***No***
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	Yes	Sub	F8	F6	F2	Ø	Ø	***No***	***No***
Divide	Yes	Div	F10	F0	F6	Mult1	Ø	No	Yes

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Mult1	Ø	Ø	Ø	Add	Divide	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	X	X
MULTD	F0, F2, F4	X	X		
SUBD	F8, F6, F2	X	X	**X**	**X**
DIVD	F10, F0, F6	X			
ADDD	F6, F8, F2				

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult1	Yes	Mult	F0	F2	F4	Ø	Ø	No	No
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	***No***	***Ø***	***Ø***	***Ø***	***Ø***	Ø	Ø	***Ø***	***Ø***
Divide	Yes	Div	F10	F0	F6	Mult1	Ø	No	Yes

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Mult1	Ø	Ø	Ø	***Ø***	Divide	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	X	X
MULTD	F0, F2, F4	X	X		
SUBD	F8, F6, F2	X	X	X	X
DIVD	F10, F0, F6	X			
ADDD	F6, F8, F2	**X**	**X**	**X**	

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult1	Yes	Mult	F0	F2	F4	Ø	Ø	No	No
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	***Yes***	***Add***	***F6***	***F8***	***F2***	Ø	Ø	***No***	***No***
Divide	Yes	Div	F10	F0	F6	Mult1	Ø	No	Yes

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Mult1	Ø	Ø	***Add***	Ø	Divide	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	X	X
MULTD	F0, F2, F4	X	X	**X**	**X**
SUBD	F8, F6, F2	X	X	X	X
DIVD	F10, F0, F6	X			
ADDD	F6, F8, F2	X	X	X	

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult1	***No***	***Ø***	***Ø***	***Ø***	***Ø***	Ø	Ø	***Ø***	***Ø***
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	Yes	Add	F6	F8	F2	Ø	Ø	No	No
Divide	Yes	Div	F10	F0	F6	***Ø***	Ø	***Yes***	Yes

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	***Ø***	Ø	Ø	Add	Ø	Divide	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción		Issue	Read operands	Execution complete	Write result
LD	F6, 34(R2)	X	X	X	X
LD	F2, 45(R3)	X	X	X	X
MULTD	F0, F2, F4	X	X	X	X
SUBD	F8, F6, F2	X	X	X	X
DIVD	F10, F0, F6	X	**X**		
ADDD	F6, F8, F2	X	X	X	**X**

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult1	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	***No***	***Ø***	***Ø***	***Ø***	***Ø***	Ø	Ø	***Ø***	***Ø***
Divide	Yes	Div	F10	F0	F6	Ø	Ø	***No***	***No***

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Ø	Ø	Ø	***Ø***	Ø	Divide	Ø	Ø	Ø

SCOREBOARD: EJEMPLO (cont.)

Tabla de estados de las instrucciones.

Instrucción	**Issue**	Read operands	**Execution complete**	**Write result**
LD F6, 34(R2)	X	X	X	X
LD F2, 45(R3)	X	X	X	X
MULTD F0, F2, F4	X	X	X	X
SUBD F8, F6, F2	X	X	X	X
DIVD F10, F0, F6	X	X	**X**	**X**
ADDD F6, F8, F2	X	X	X	X

Tabla de estados de las unidades funcionales.

Unidad funcional	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult1	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Mult2	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Add	No	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Divide	***No***	***Ø***	***Ø***	***Ø***	***Ø***	Ø	Ø	***Ø***	***Ø***

Tabla registros-resultados.

	F0	F2	F4	F6	F8	F10	F12	…	F30
Unidad funcional	Ø	Ø	Ø	Ø	Ø	***Ø***	Ø	Ø	Ø

SCOREBOARD: COSTES, BENEFICIOS Y LIMITACIONES

- Mejora del rendimiento en el CDC 6600:
  - 70% en los programas FORTRAN.
  - 150% en los programas escritos directamente en ensamblador.
- Coste:
  - En el CDC la lógica del *scoreboard* resultó de una complejidad similar a la de una unidad funcional. ¡Sorprendentemente bajo!
  - El principal componente del coste es el gran número de buses entre los registros y las unidades funcionales.
    - Hay un número limitado de buses lo que supone una limitación al número de unidades funcionales que pueden, a la vez, continuar las etapas *read operands* y *write result*.

SCOREBOARD: COSTES, BENEFICIOS Y LIMITACIONES (cont.)

- **Interés actual** en técnicas de planificación dinámica como esta.
  - Motivado por el objetivo de emitir varias instrucciones por ciclo (el coste de tener más buses es, por tanto, inevitable) e ideas como la especulación.

- Limitaciones:
  - Número de entradas del *scoreboard*. Determina el tamaño de la ventana.
    - Ventana es el conjunto de instrucciones examinadas como candidatas para ser ejecutadas.
  - Número y tipos de unidades funcionales.
    - Esto determina que haya más o menos riesgos estructurales.
  - La aparición de dependencias WAR y WAW.
    - Consecuencia de la ejecución y terminación fuera de orden.

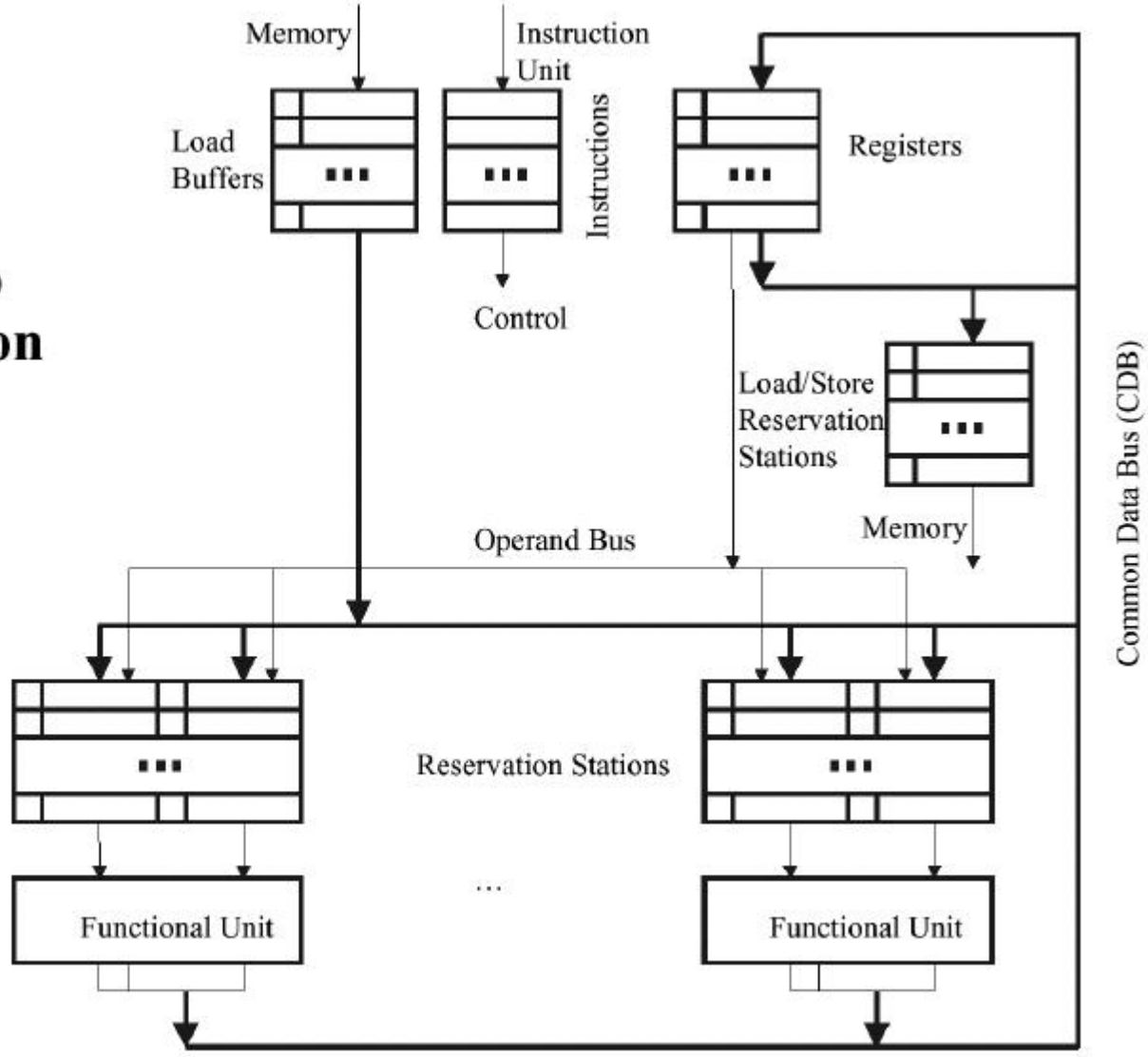
PLANIFICACIÓN DINÁMICA MEDIANTE EL ALGORITMO DE TOMASULO

- Esquema, inventado por Robert Tomasulo e implementado por primera vez en el IBM 360/91 (año 1967).
- Tras un largo periodo en que decayó su uso, desde los años 90 es de empleo generalizado, con múltiples variantes.
  - Procesadores recientes que usan variantes de este esquema: Alpha 21264, HP 8000, MIPS 10000,Pentium II, PowerPC 604, …
- Combina la redenominación de registros (para eliminar dinámicamente las dependencias WAR y WAW) con elementos clave del *scoreboarding*.
- Emisión en orden. Ejecución y terminación: fuera de orden.
- Control distribuido (en el scoreboard es centralizado).

ALGORITMO DE TOMASULO

- Cuando una instrucción descodificada no tiene disponibles sus operandos, en lugar de esperar en la etapa *issue* impidiendo que avancen otras instrucciones...
  - Es enviada a una **estación de reserva**, o **RS** (*Reservation Station*), asociada con la unidad funcional que utilizará.
  - La instrucción **esperará** en la estación de reserva hasta que sus **operandos** estén **disponibles**.
    - Para saber si están disponibles observará el **bus** de resultados o **CDB** (*Common Data Bus*).
- Cuando todos los operandos de la instrucción estén disponibles, es enviada a la correspondiente unidad funcional para su ejecución.
- El resultado se envía al CDB y, desde el CDB, al registro destino y a todas las estaciones de reserva que estuvieran esperando ese resultado.

Tomasulo Organization



TOMASULO: ESTRUCTURA DETALLADA

- Cada **estación de reserva** es identificable mediante una marca única y se compone de los siguientes **campos**:
  - **Busy**. La estación de reserva está ocupada.
  - **Op**. La operación a ejecutar.
  - **Q_j (Q_k)** Identificador de la **estación de reserva que producirá el valor del operando** fuente S1 (S2).
    - Un valor cero indica que el operando fuente ya está disponible (o es innecesario).
  - **V_j (V_k)**. Valor del operando fuente S1 (S2), si Q_j (Q_k) vale cero.
- **Estaciones de reserva del buffer** de **carga**. Campos:
  - **Address**. Dirección de la que debe leer la instrucción *load.*
  - **Busy**. Ocupado o no. Si *busy=no*, se ignora *address*.

TOMASULO: ESTRUCTURA DETALLADA (cont.)

- **Estaciones de reserva del buffer** de **almacenamiento**. Campos:
  - **Address**. Dirección en la que debe escribir la instrucción *store.*
  - **Q_i**. Identificador de la **RS** cuya instrucción **producirá el resultado** que debe almacenarse en memoria.
  - **V**. Valor a escribir.
  - **Busy**. Ocupado o no. Si *busy=no*, los otros campos se ignoran.
- La **tabla registros-resultados** contiene, por cada registro**,** un **campo Q_i** que indica:
  - El identificador de la **RS** cuya instrucción **producirá el resultado** que debe almacenarse en ese registro.
  - Valor cero: ninguna instrucción activa tiene como destino el registro.

TOMASULO: COMPORTAMIENTO DETALLADO

- Notaciones:
  - **oper**: operación que corresponde a la instrucción.
  - **‘D’, ‘S1’, ‘S2’**: nombres (identificadores) de los registros destino, fuente1 y fuente2, respectivamente.
  - **result**: valor producido por la instrucción de carga o aritmético-lógica.
  - **trr**: tabla registros-resultados.
  - **rsstore**: tabla de estaciones de reserva del buffer de almacenamiento.

TOMASULO: COMPORTAMIENTO DETALLADO (cont.)

- Etapa ***issue*** (**supondremos** que la instrucción es **aritmético-lógica**).
  - **Esperar** a que al menos una **estación de reserva**, **r**, asociada a la FU necesitada por la instrucción esté **desocupada**. Entonces...
    - **Enviar** la instrucción a la estación de reserva r, es decir:
      - Actualizar campos de la estación de reserva **r** (RS[r]):

      RS[r].busy ← yes; RS[r].op ← oper;

      if (trr['S1'].Q_i = 0) **then** { RS[r].V_j ← S1; RS[r].Q_j ←0 }
      else { RS[r].Q_j ← trr['S1'].Q_i } /* campos Q_j/V_j */

      if (trr['S2'].Q_i = 0) **then** { RS[r].V_k ← S2; RS[r].Q_k ←0 }
      else { RS[r].Q_k ← trr['S2'].Q_i } /* campos Q_k/V_k */

      - Hacer la siguiente anotación: trr['D'] ← r.

TOMASULO: COMPORTAMIENTO DETALLADO (cont.)

- Etapa ***execute***.
  - Esperar operandos y FU disponibles, es decir, esperar hasta que

    (RS[r].Q_j = 0) and (RS[r].Q_k = 0) and (FU disponible)
  - Entonces ejecutar la instrucción (los operandos están en V_j y V_k).
- Etapa ***write result***.
  - Esperar a que la ejecución de la instrucción haya terminado y el CDB esté disponible. Entonces...

    $\forall$x (**if** (RS[x].Q_j = r) **then** {RS[x].V_j $\leftarrow$ result; RS[x].Q_j $\leftarrow$ 0};
    $\forall$x (**if** (RS[x].Q_k = r) **then** {RS[x].V_k $\leftarrow$ result; RS[x].Q_k $\leftarrow$ 0};
    $\forall$x (**if** (rsstore[x].Q_i = r) **then** {rsstore[x].V $\leftarrow$ result; rsstore[x].Q_i $\leftarrow$ 0};
    $\forall$x (**if** (trr[x].Q_i = r) **then** {Reg[x] $\leftarrow$ result; trr[x].Q_i $\leftarrow$ 0};
    RS[r].busy $\leftarrow$ no

Tomasulo Example Cycle 0

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2					Load1	No			
LD	F2	45+	R3					Load2	No			
MULT	F0	F2	F4					Load3	No			
SUBD	F8	F6	F2									
DIVD	F10	F0	F6									
ADDD	F6	F8	F2									

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
	0	Add3	No									
	0	Mult1	No									
	0	Mult2	No									

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0			*FU*									

Tomasulo Example Cycle 1

Instruction status *

Instruction		j	k	Issue	Execution complete	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	No					
0	Add2	No					
	Add3	No					
0	Mult1	No					
0	Mult2	No					

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU				Load1					

* No hay una tabla "instruction status" pero la información equivalente existe (distribuida por el hardware)

Tomasulo Example Cycle 2

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result
LD	F6	34+	R2	1	*	
LD	F2	45+	R3	2		
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	No					
0	Add2	No					
	Add3	No					
0	Mult1	No					
0	Mult2	No					

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU		Load2		Load1					

Note: Unlike 6600, can have multiple loads outstanding

*** Se supone que LOAD consume 2 ciclos en la etapa execution**

Tomasulo Example Cycle 3

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	No					
0	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD		R(F4)	Load2	
0	Mult2	No					

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- **Note: registers names are removed (“renamed”) in Reservation Stations; MULT issued vs. scoreboard**
- **Load1 completing; what is waiting for Load1?**

Tomasulo Example Cycle 4

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4			Load2	Yes	45+R3		
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4								
DIVD	F10	F0	F6									
ADDD	F6	F8	F2									

Reservation Stations					*S1*	*S2*	*RS for j*	*RS for k*				
	Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*				
	0	Add1	Yes	SUBD	M(34+R2)			Load2				
	0	Add2	No									
		Add3	No									
	0	Mult1	Yes	MULTD		R(F4)	Load2					
	0	Mult2	No									

Register result status											
Clock			*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4		*FU*	Mult1	Load2		M(34+R2)	Add1				

- **Load2 completing; what is waiting for it?**

Tomasulo Example Cycle 5

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4								
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2									

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	2	Add1	Yes	SUBD	M(34+R2)	M(45+R3)						
	0	Add2	No									
		Add3	No									
	10	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
5			*FU*	Mult1	M(45+R3)		M(34+R2)	Add1	Mult2			

Suponemos: Nº ciclos consumidos en la etapa execution: suma, 2; multiplicación, 10; división, 40.

Tomasulo Example Cycle 6

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4								
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6								

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	1	Add1	Yes	SUBD	M(34+R2)	M(45+R3)						
	0	Add2	Yes	ADDD		M(45+R3)	Add1					
		Add3	No									
	9	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6			*FU*	Mult1	M(45+R3)		Add2	Add1	Mult2			

- **Issue ADDD here vs. scoreboard?**

Tomasulo Example Cycle 7

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4	7							
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6								

Reservation Stations					*S1*	*S2*	*RS for j*	*RS for k*				
	Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*				
	0	Add1	Yes	SUBD	M(34+R2)	M(45+R3)						
	0	Add2	Yes	ADDD		M(45+R3)	Add1					
		Add3	No									
	8	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7			*FU*	Mult1	M(45+R3)		Add2	Add1	Mult2			

- **Add1 completing; what is waiting for it?**

Tomasulo Example Cycle 8

Instruction status					Execution	Write					
Instruction		*j*	*k*	*Issue*	*complete*	*Result*			Busy	Address	
LD	F6	34+	R2	1	3	4		Load1	No		
LD	F2	45+	R3	2	4	5		Load2	No		
MULT	F0	F2	F4	3				Load3	No		
SUBD	F8	F6	F2	4	7	8					
DIVD	F10	F0	F6	5							
ADDD	F6	F8	F2	6							

Reservation Stations					*S1*	*S2*	*RS for j*	*RS for k*				
	Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*				
	0	Add1	No									
	2	Add2	Yes	ADDD	M()-M()	M(45+R3)						
	0	Add3	No									
	7	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
8			*FU*	Mult1	M(45+R3)		Add2	M()-M()	Mult2			

Tomasulo Example Cycle 9

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6								

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	1	Add2	Yes	ADDD	M()–M()	M(45+R3)						
	0	Add3	No									
	6	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
9			*FU*	Mult1	M(45+R3)		Add2	M()–M()	Mult2			

Tomasulo Example Cycle 10

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10							

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	Yes	ADDD	M()–M()	M(45+R3)						
	0	Add3	No									
	5	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
10			*FU*	Mult1	M(45+R3)		Add2	M()–M()	Mult2			

- **Add2 completing; what is waiting for it?**

Tomasulo Example Cycle 11

Instruction status					Execution	Write					
Instruction		*j*	*k*	*Issue*	*complete*	*Result*			Busy	Address	
LD	F6	34+	R2	1	3	4		Load1	No		
LD	F2	45+	R3	2	4	5		Load2	No		
MULT	F0	F2	F4	3				Load3	No		
SUBD	F8	F6	F2	4	7	8					
DIVD	F10	F0	F6	5							
ADDD	F6	F8	F2	6	10	11					

Reservation Stations					*S1*	*S2*	*RS for j*	*RS for k*				
	Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*				
	0	Add1	No									
	0	Add2	No									
	0	Add3	No									
	4	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
11			*FU*	Mult1	M(45+R3)		(M-M)+M()	M()–M()	Mult2			

- **Write result of ADDD here vs. scoreboard?**

Tomasulo Example Cycle 12

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10	11						

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
	0	Add3	No									
	3	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
12			*FU*	Mult1	M(45+R3)		(M-M)+M()	M()–M()	Mult2			

- **Note: all quick instructions complete already**

Tomasulo Example Cycle 13

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10	11						

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
		Add3	No									
	2	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
13			*FU*	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

Tomasulo Example Cycle 14

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3				Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10	11						

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
	0	Add3	No									
	1	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
14			*FU*	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

Tomasulo Example Cycle 15

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3	15			Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10	11						

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
		Add3	No									
	0	Mult1	Yes	MULTD	M(45+R3)	R(F4)						
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
15			*FU*	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

- **Mult1 completing; what is waiting for it?**

Tomasulo Example Cycle 16

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3	15	16		Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10	11						

Reservation Stations					*S1*	*S2*	*RS for j*	*RS for k*				
	Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*				
	0	Add1	No									
	0	Add2	No									
		Add3	No									
	0	Mult1	No									
	40	Mult2	Yes	DIVD	M*F4	M(34+R2)						

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
16			*FU*	M*F4	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

- **Note: Just waiting for divide**

Tomasulo Example Cycle 55

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3	15	16		Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5								
ADDD	F6	F8	F2	6	10	11						

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
		Add3	No									
	0	Mult1	No									
	1	Mult2	Yes	DIVD	M*F4	M(34+R2)						

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
55			*FU*	M*F4	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

Tomasulo Example Cycle 56

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3	15	16		Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5	56							
ADDD	F6	F8	F2	6	10	11						

Reservation Stations												
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*				
	0	Add1	No									
	0	Add2	No									
		Add3	No									
	0	Mult1	No									
	0	Mult2	Yes	DIVD	M*F4	M(34+R2)						

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
56			*FU*	M*F4	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

- **Mult 2 completing; what is waiting for it?**

Tomasulo Example Cycle 57

Instruction status												
Instruction		*j*	*k*	*Issue*	*Execution complete*	*Write Result*			Busy	Address		
LD	F6	34+	R2	1	3	4		Load1	No			
LD	F2	45+	R3	2	4	5		Load2	No			
MULT	F0	F2	F4	3	15	16		Load3	No			
SUBD	F8	F6	F2	4	7	8						
DIVD	F10	F0	F6	5	56	57						
ADDD	F6	F8	F2	6	10	11						

Reservation Stations								
	Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS for j Qj*	*RS for k Qk*
	0	Add1	No					
	0	Add2	No					
		Add3	No					
	0	Mult1	No					
	0	Mult2	No					

Register result status												
Clock				*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
57			*FU*	M*F4	M(45+R3)		(M–M)+M()	M()–M()	M*F4/M			

- **Again, in-oder issue, out-of-order execution, completion**

Compare to Scoreboard Cycle 62

Instruction status

Instruction		j	k	Issue	Read operand	Execution complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU								

- **Why takes longer on Scoreboard/6600?**

TOMASULO vs. SCOREBOARD

- Principal diferencia: tratamiento de las **dependencias WAR y WAW**.
  - Con *scoreboard* las dependencias WAW y WAR producen paradas. Con el algoritmo de Tomasulo, no, gracias a su **redenominación de registros**.
    - Al enviar la instrucción a la estación de reserva **se cambian** los **identificadores** de los **registros por los de las estaciones de reserva** de las instrucciones que calcularán sus valores =>
    - **WAR**(x,J,K) (x: operando; J,K: instrucciones). Si al emitir J, x está disponible, se guardará en la RS de J, antes de que K lo pueda modificar. Si x no está disponible se redenomina a la RS de la instrucción I que lo producirá. No importará que K lo escriba antes, ya que **J no tomará el valor de x sino del CDB** cuando lo produzca I.
    - **WAW**(D,J,K). Al emitir J se anota trr[‘D’] ← r_J y al emitir K, trr[‘D’] ← r_K. Puesto que la emisión se hace en orden, **sólo la última instrucción actualizará** realmente **el destino** D.

TOMASULO: OTRAS CARACTERÍSTICAS

- Eficiente en la ejecución solapada de bucles.
  - Ignorando el tiempo de control del bucle se consigue una eficiencia **equivalente** a la obtenida mediante desenrollamiento y planificación **estáticos**.
    - Las estaciones de reserva permiten la ejecución solapada de varias copias del bucle **sin** necesidad de **tantos registros** como en el desenrollamiento y planificación estáticos.
- Inconvenientes:
  - **Complejidad** => gran cantidad de hardware.
  - Rendimiento limitado por la existencia de un **único CDB**.
    - **Varios** CDB => **más hardware** (en particular, el hardware asociativo de comparación de marcas en las RS crece con cada CDB adicional).