uc3m | Universidad Carlos III de Madrid

Paralelismo a nivel de instrucción

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)

Departamento de Informática
Universidad Carlos III de Madrid

Licencia Creative Commons

Este trabajo se distribuye bajo licencia **Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional (CC BY-NC-ND 4.0)**.

Usted es libre de:

Compartir – copiar y redistribuir el material en cualquier medio o formato.
El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia.

Bajo las condiciones siguientes:

Reconocimiento – Debe **reconocer adecuadamente la autoría**, proporcionar un enlace a la licencia e **indicar si se han realizado cambios**. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.
NoComercial – No puede utilizar el material para una **finalidad comercial**.
SinObraDerivada – Si **remezcla, transforma o crea a partir** del material, no puede difundir el material modificado.

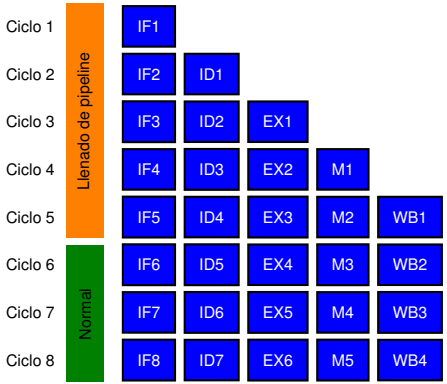
No hay restricciones adicionales – No puede aplicar **términos legales** o **medidas tecnológicas** que legalmente restrinjan realizar aquello que la licencia permite.

Texto completo de la licencia disponible en
https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode

Segmentación

- Técnica de implementación en la que una la ejecución de múltiples instrucciones se solapan en el tiempo.
  - Se divide una operación **costosa** en varias sub-operaciones simples.
  - Ejecución de las sub-operaciones por etapas.
- **Efectos**:
  - **Aumenta** el ***throughput***.
  - **No disminuye** la **latencia**.

Segmentación



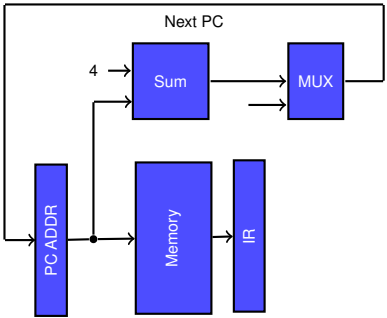
- **Latencia**:
  - Una instrucción requiere 5 etapas.
  - 5 ciclos.
- **Throughput**:
  - Termina una instrucción en cada ciclo (con pipeline lleno).
  - 1 instrucción por ciclo.

Etapas del pipeline

- Modelo simplificado:
  - 5 etapas.
  - **Modelos más realistas requieren muchas más etapas**.

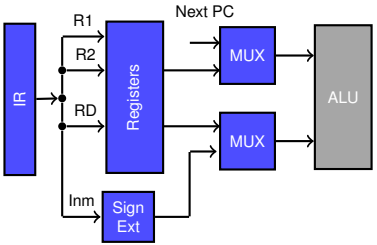
- **Etapas**:
  - **Captación** (*Instruction Fetch*).
  - **Decodificación** (*Instruction Decode*).
  - **Ejecución** (*Execution*).
  - **Memoria** (*Memory*).
  - **Post-escritura** (*Write-back*).

Captación



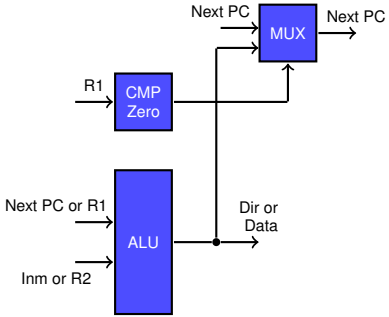
- Envío de PC a memoria.
- Lectura de instrucción.
- Actualización de PC.

Decodificación



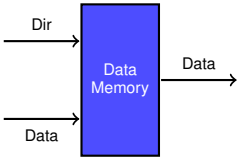
- Decodificación de instrucción.
- Lectura de registros.
- Extensión de signo de desplazamientos.
- Cálculo de posible dirección de salto.

Ejecución



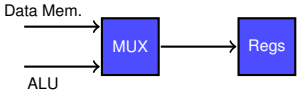
- Operación de ALU sobre operandos.
- Alternativas:
  - **Referencia a memoria**: Cálculo de dirección efectiva (suma registro base y desplazamiento).
  - **Instrucciones registro/registro**: Operación de ALU sobre registros obtenidos.
  - **Instrucciones Registro/dato inmediato**: Operación de ALU entre registro y dato inmediato.
  - **Bifurcación condicional**: Evaluación de la condición.

Memoria



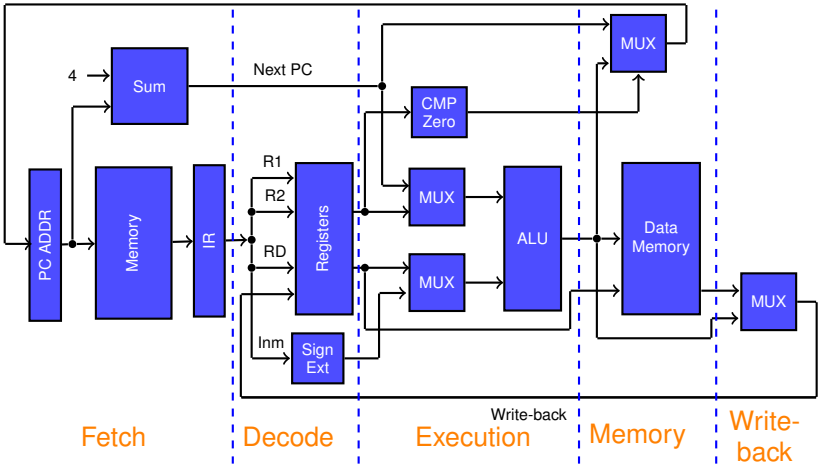
- Lectura o escritura en memoria.
  - **Lectura**: Acceso a memoria usando dirección efectiva calculada en EX.
  - **Escritura**: Escritura a memoria usando dirección efectiva calculada en EX y y valor de segundo registro leído.

Post-escritura



- Escritura de resultado en banco de registros.
  - **Operaciones de carga**: Valor leído de memoria.
  - **Instrucciones de ALU**: Valor calculado en etapa EX.

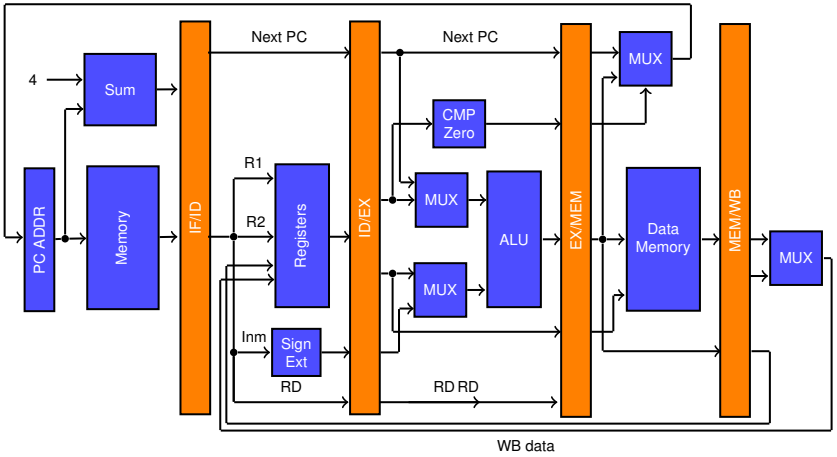
Arquitectura general



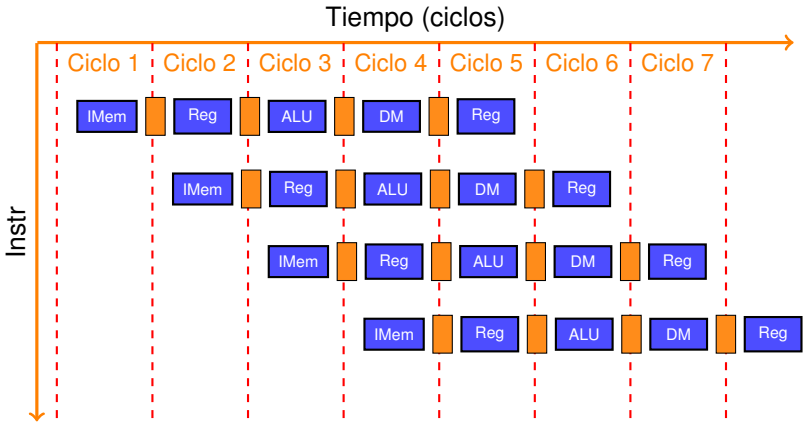
Efectos del pipeline

- Un pipeline de **profundidad n**, multiplica por **n** el **ancho de banda** necesario de la versión sin pipeline con la misma frecuencia de reloj.
  - Cachés, cachés, ...
- La separación de **caché** de **datos** e **instrucciones** elimina algunos **conflictos** de memoria.
- Las instrucciones que están en el pipeline **no deberían** intentar usar el mismo recurso en el mismo momento.
  - Introducción de registros de pipeline entre etapas sucesivas.

Comunicación entre etapas



El pipeline a lo largo del tiempo



- Lectura registros segunda mitad de ciclo.

Ejemplo

- Procesador no segmentado.
  - Frecuencia: 4 GHz.
  - 40% operaciones ALU $\rightarrow$ 4 ciclos.
  - 20% operaciones de bifurcación $\rightarrow$ 4 ciclos.
  - 40% operaciones de memoria $\rightarrow$ 5 ciclos.
  - Sobrecoste de segmentación $\rightarrow$ 0.1 ns.
- ¿Qué aceleración se obtiene con la segmentación?

$$t_{orig} = ciclo_{reloj} \times CPI = 0.25ns \times (0.6 \times 4 + 0.4 \times 5) = 1.1ns$$

$$t_{nuevo} = 0.25ns + 0.1ns = 0.35ns$$

$$S = \frac{1.1ns}{0.35ns} = 3.1$$

Riesgos

- Un **riesgo** es una situación que impide que la siguiente instrucción pueda comenzar en el ciclo de reloj previsto.
  - Los riesgos reducen el rendimiento de las arquitecturas segmentadas.
- Tipos de riesgos:
  - Riesgo estructural.
  - Riesgo de datos.
  - Riesgo de control.
- Aproximación simple ante riesgos:
  - Detener (*stall*) el flujo de instrucciones.
  - Las instrucciones que ya han iniciado continúan.

Riesgo estructural

- Se produce cuando el hardware **no puede** soportar todas las posibles secuencias de instrucciones.
  - En un mismo ciclo dos etapas de la segmentación necesitan hacer uso del **mismo recurso**.

- **Razones**:
  - Unidades funcionales no totalmente segmentadas.
  - Unidades funcionales no duplicadas.

- Este tipo de riesgos es evitable pero encarece el hardware.

Speedup de la segmentación

- **Speedup de la segmentación**:
  - t_{noseg}: Tiempo medio de instrucción en arquitectura no segmentada.
  - t_{seg}: Tiempo medio de instrucción en arquitectura segmentada.

$$S = \frac{t_{noseg}}{t_{seg}} = \frac{CPI_{noseg} \times ciclo_{noseg}}{CPI_{seg} \times ciclo_{seg}}$$

- En el caso ideal el **CPI** segmentado es 1.
  - Hay que añadir ciclos de detención por instrucción.

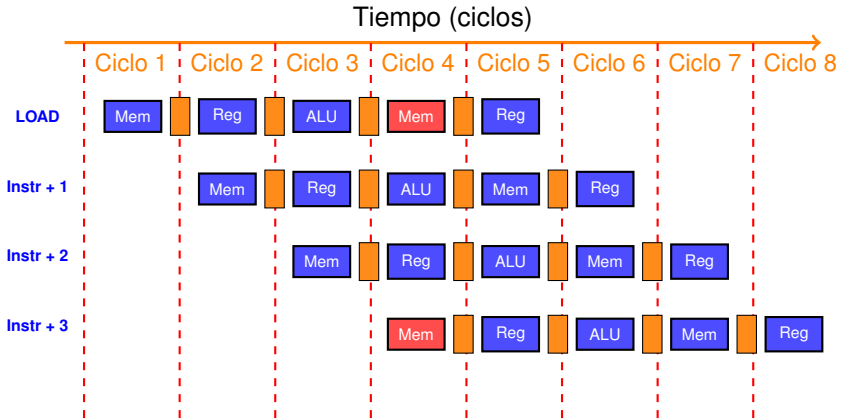
Speedup de la segmentación

- En el caso del procesador no segmentado.
  - $CPI = 1$, con $ciclo_{noseg} > ciclo_{seg}$.
  - $ciclo_{noseg} = N \times ciclo_{seg}$.
  - $N \rightarrow$ **Profundidad del pipeline**.

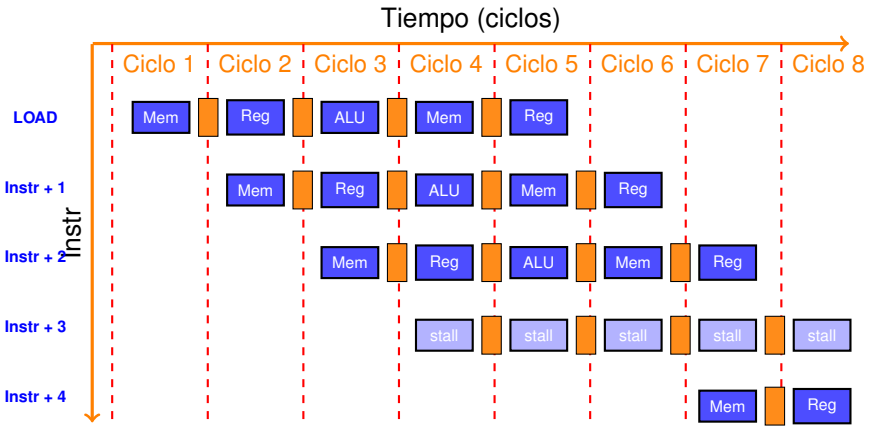
Speedup

$$S = \frac{N}{1 + \text{ciclos detención por instrucción}}$$

Riesgos estructurales: ejemplo



Riesgos estructurales: ejemplo



Ejemplo

- Se consideran dos diseño alternativos:
  - **A**: Sin riesgos estructurales.
    - Ciclo de reloj $\rightarrow 1ns$
  - **B**: Con riesgos estructurales.
    - Ciclo de reloj $\rightarrow 0.9ns$
    - Instrucciones de acceso a datos con riesgo $\rightarrow 30\%$.
    - Penalización: 1 ciclo.
- ¿Alternativa más rápida?

$$t_{inst}(A) = CPI \times ciclo = 1 \times 1ns = 1ns$$

$$t_{inst}(B) = CPI \times ciclo = (0.7 \times 1 + 0.3 \times (1 + 1)) \times 0.9ns = 1.17ns$$

Riesgo de datos

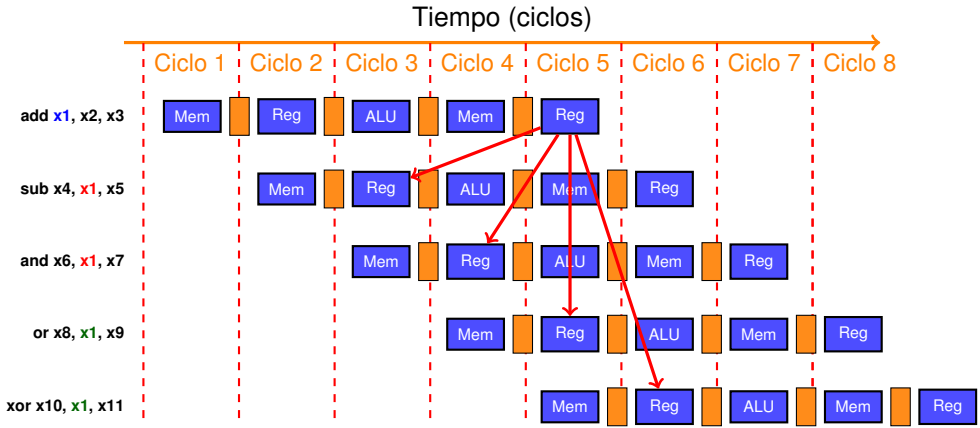
- Se produce un **riesgo de datos** cuando la segmentación modifica el orden de accesos de lectura/escritura a los operandos.

Ejemplo

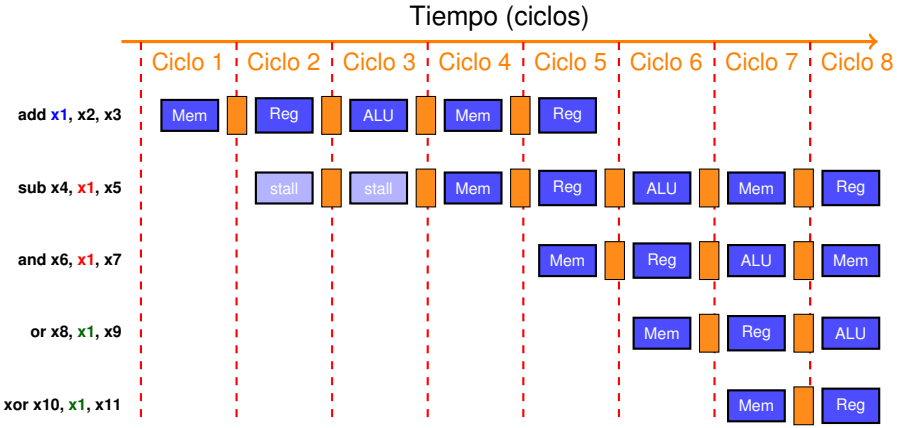
```
I1: add  x1, x2, x3
I2: sub  x4, x1, x5
I3: and  x6, x1, x7
I4: or   x8, x1, x9
I5: xor  x10, x1, x11
```

- **I2** lee **x1** antes que **I1** la modifique.
- **I3** lee **x1** antes de que **I1** la modifique.
- **I4** obtiene valor correcto.
  - Banco de registros leído en segunda mitad de ciclo.
- **I5** obtiene valor correcto.

Riesgos de datos



Detenciones en riesgos de datos



Riesgos de datos: RAW

- Lectura después de escritura (**Read After Write**).
  - La instrucción **i+1** trata de leer un dato antes de que la instrucción **i** lo escriba.

Ejemplo

```
i :    add x1, x2, x3
i+1:   sub x4, x1, x3
```

- Si hay dependencia de datos, las instrucciones:
  - No pueden ejecutarse en paralelo ni solaparse completamente.
  - La instrucción **sub** necesita el valor de **x1** producido por la instrucción **add**.

- **Soluciones**:
  - **Detección hardware**.
  - **Control por el compilador**.

Riesgos de datos: WAR

- Escritura después de lectura (**Write After Read**).
  - La instrucción **i+1** modifica operando antes de que la instrucción **i** lo lea.

Ejemplo

```
i :    sub x4, x1, x3
i+1:   add x1, x2, x3
i+2:   mul x6, x1, x7
```

- Conocido como **anti-dependencia** en compiladores.
  - Reutilización de nombre
  - La instrucción **add** modifica **x1** antes de **sub** la lea.

- **No puede** ocurrir en un **pipeline de 5 etapas**.
  - Todas las instrucciones de 5 etapas.
  - Lecturas siempre en etapa 2.
  - Escrituras siempre en etapa 5.

Riesgos de datos: WAW

- Escritura después de escritura (**Write After Write**).
  - La instrucción **i+1** modifica el operando antes de que la instrucción **i** lo modifica.

Ejemplo

```
i :    sub x1, x4, x3
i+1:   add x1, x2, x3
i+2:   mul x6, x1, x7
```

- Conocido como **dependencia de salida** en compiladores.
  - Reutilización de nombre
  - La instrucción **add** modifica **x1** antes de **sub** la modifique.

- **No puede** ocurrir en un **pipeline de 5 etapas**.
  - Todas las instrucciones de 5 etapas.
  - Escrituras siempre en etapa 5.

Soluciones a los riesgos de datos

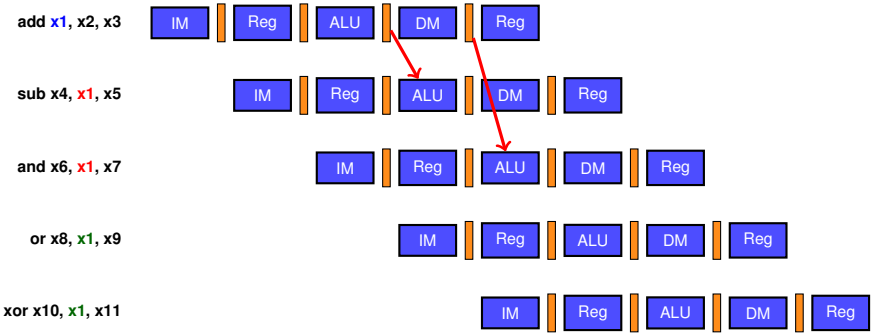
- **Dependencias RAW**:
  - Envío adelantado (**forwarding**).

- **Dependencias WAR y WAW**:
  - **Renombrado de registros**.
    - Estáticamente por el compilador.
    - Dinámicamente por el hardware.

Envío adelantado (*forwarding*)

- Técnica para evitar algunas detenciones de datos.
- **Idea básica**:
  - No hace falta esperar a que el resultado se escriba en el banco de registros.
  - Ya está en los registros de segmentación.
  - Se puede usar ese valor en vez del que hay en el banco de registros.
- **Implantación**:
  - Los resultados de las fases EX y MEM se escriben en registros de entrada a ALU.
  - La lógica de *forwarding* selecciona entre entradas reales y registros de *forwarding*.

Forwarding

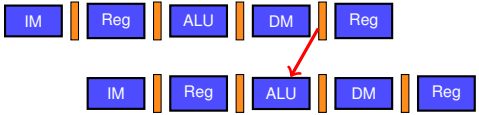

add x1, x2, x3
IM
Reg
ALU
DM
Reg
sub x4, x1, x5
IM
Reg
ALU
DM
Reg
and x6, x1, x7
IM
Reg
ALU
DM
Reg
or x8, x1, x9
IM
Reg
ALU
DM
Reg
xor x10, x1, x11
IM
Reg
ALU
DM
Reg

Limitaciones del *forwarding*

- No todos los riesgos se pueden evitar con forwarding.
  - ¡No se puede viajar hacia atrás en el tiempo!

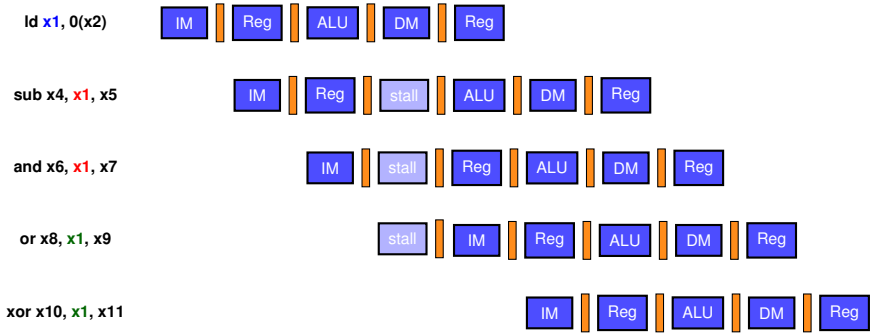
Ejemplo

```
I1: ld   x1, (0)x2
I2: sub  x4, x1, x5
I3: and  x6, x1, x7
I4: or   x8, x1, x9
I5: xor  x10, x1, x11
```



- Si el riesgo no se puede evitar se debe introducir una detención.

Detenciones en acceso a memoria



Resumen

- La segmentación aumenta el *throughput* pero no reduce la latencia.
- Una arquitectura segmentada requiere mayor ancho de banda de memoria.
- Tres tipos de riesgos.
  - Riesgos estructurales.
  - Riesgos de datos.
  - Riesgos de control.
- Los riesgos en el pipeline provocan detenciones.
  - Degradación del rendimiento.
- Tres tipos de riesgos de datos: RAW, WAR, WAW.
  - Mitigación: Envío adelantado, compilador, planificación dinámica.

Referencias

- **Computer Architecture: A Quantitative Approach**. 6th Ed. Hennessy and Patterson. Morgan Kaufmann. 2017.
  - Sección C.1 – *Introduction*.
  - Sección C.2 – *The Major Hurdle of Pipelining – Pipeline Hazards*

uc3m | Universidad Carlos III de Madrid

Paralelismo a nivel de instrucción

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)

Departamento de Informática
Universidad Carlos III de Madrid