ETSI Telecomunicación

ACSO I
(Arquitectura de Computadores y Sistemas Operativos)

Apuntes de Pak

Arquitectura de Computadores y Sistemas Operativos I

Apuntes de Pak (Francisco José Rodríguez Fortuño)
ETSI Telecomunicación. Universidad Politécnica de Valencia.
Primer cuatrimestre de 4º curso
Curso 2006/2007

Contenido

- Resumen de las prácticas, incluyendo algunos ejercicios de programar en bash
- Apuntes extensos de la asignatura (sólo hay referencia rápida del tema 1, ya que pienso que la asignatura es más de leer, entender y razonar, que de recordar detalles)
- Exámenes resueltos

Referencias y agradecimientos

Estos apuntes están basados en las clases de ACSO I que me impartió el profesor Dr. D. Miguel Ángel Mateo Pla así como en las transparencias elaboradas por él y los demás profesores de la asignatura, Doctores D. Julio Pons Terol y D. Pedro Pablo Cruz Alcázar.

Las figuras impresas que aparecen en estos apuntes están tomadas de dichas transparencias, así como el texto a máquina de los temas 3 y 4. Los exámenes que reproduzco en los apuntes son también de los ofrecidos por los profesores.

ACSO I - Prácticas que se incluyen en el examen

Práctica 1. Introducción al Sistema operativo Unix

1. Introducción

UNIX → multiproceso y multiusuario
cada proceso pertenece a un usuario
LINUX: versión libre de Unix que utilizamos en la práctica

2. Entrada al sistema

1. a través de un terminal conectado a la máquina
2. a través de la red: usuario en terminal cliente se conecta a un servidor

3. Notas generales

CTRL-C: Termina la ejecución de una orden
CTRL-D: Indica final de fichero

CTRL-S detiene] visualiz.
CTRL-Q reanuda] en pantalla

4. Formato de las órdenes

orden parametro parametro
espacios

- las opciones son parámetros que empiezan con '-' o con '--'
- las opciones pueden agruparse
  ls -l -a ≡ ls -la

5. Algunas órdenes básicas de Unix

- type nombre_comando : saber si el comando es interno o externo
- date : fecha y hora del sistema date [+format] ej + %a %d
- who : visualizar usuarios conectados al sistema
  who am i (detalles míos)
  whoami (devuelve Directamente el nombre)
- pwd : print working directory
- ls : listar el contenido de un directorio ls -l : detalles ls -a : incluir ocultos ls -r : recursivamente todos los subdirectorios

  propiet. grupo todos ← permisos
  - rwx rwx rwx
  - [- ficheros, d directorios, l enlaces]

- cd : change directory
- touch fichero : crea fichero vacio
- cat, file, more, less, head, tail
  file fichero : ver tipo de fichero
  cat fichero : mostrar fichero
  cat fich1 fich2 : mostrar ficheros concatenados
  more, less } paginadores ej: ls -l | more
  head fichero : inicio de fichero
  tail fichero : final de fichero

Caracteres comodines:
- * : cualquier cadena (incluyendo la vacía) → EXCEPTO las que comienzan por .
- ? : cualquier caracter

Para poder utilizar * y ? como carácteres, usar el caracter llamado Escape "\"
\\ se sustituye por \
* " " " *
\? " " " ?

[1-7] : sustituye a cualquier caracter simple dentro del rango 1 a 7 (ascii)
[1,3,4,7] ≡ [1347] = sustituye a cualquier caracter del conjunto

tail -5 : visualiza últimas 5 líneas
tail +5 : visualiza A PARTIR de la línea 5

- wc : devuelve número de líneas, palabras y caracteres de la entrada estándar recuerda CTRL-D Final Fichero
  wc --lines
  wc --words
- grep pattern : busca "pattern" en la entrada estándar, y por defecto o: grep pattern file ← lo busca en el fichero
  printea las líneas en la salida estándar
  ↑ línea entera en la que se ha encontrado pattern

6. Creación y borrado de directorios

mkdir directorio — - se pueden crear varios (separando con espacios)
-se pueden crear subdirectorios

rmdir directorio ← debe estar vacío

7. Copia, movimiento y borrado de ficheros

cp mv rm ej rm *.bak

cp -r dir1 dir2

8. Propiedad y protección

r: lectura
w: escritura
x: ejecución

	bin.	octal
- - -	000	0
- - x	001	1
- w -	010	2
- w x	011	3
r - -	100	4
r - x	101	5
r w -	110	6
r w x	111	7

rwx rwx rwx
propiet grupo todos

chown : cambiar prop.
chgrp : cambiar grupo

cambiar los permisos : **chmod** 644 hola ← fichero o directorio
644 → rw- r-- r--
x implica ENTRAR en el directorio

umask 644 ← especificar permisos por defecto

9. Redirección de E/S

ficheros estándar { stdin, stdout, stderr } por defecto se asignan esos ficheros a la entrada, salida y salida de error, se pueden cambiar por otro usando >, <, 2>

ls -l > lista.txt ← crea o sobreescribe
date >> lista.txt ← añade al final
} redireccionar **salida estándar** → pero no los errores

wc < fichero } redirecciona **entrada estándar**

cat fichero 2> error.txt } redirecciona la **salida de error**

ej: cat noexisto.dat > sal.dat 2> err.dat

en realidad
0>, 1>, 2>
0, 1 y 2 son los dispositivos de E/S estandar
> toma por defecto 1>

Tuberias : pipes ls /home | grep dis | sort -r (reverse) → ordena de 'z' a 'a' todos los archivos de /home que contienen "dis"

tar
c: crea o sobreescribe
v: modo detallado (verbose)
f: especificar nombre fichero contenedor
t: tabla contenido del archivo
x: extrae
z: comprimir o descomprimir ← equivalente a entubar gzip

tar czf hola ↔ tar c * | gzip > hola

10. Manejo de procesos

ejecución secuencial de procesos con punto y coma — ej: date ; ps ; who
ej: sleep 30 ; echo despierta!

ps
ps aux ↔ ps -a -x -u
top ↔ información mas completa ↳ h ayuda, q salir

kill : envía señal a un proceso
por defecto 15 terminacion,
9 terminación instantánea
ej kill -9 29437

11. Variables

Creacion y modificación con "="
yo=pepe

Referencia con "$"
echo $yo
echo -n $yo ← quita los retornos de la salida

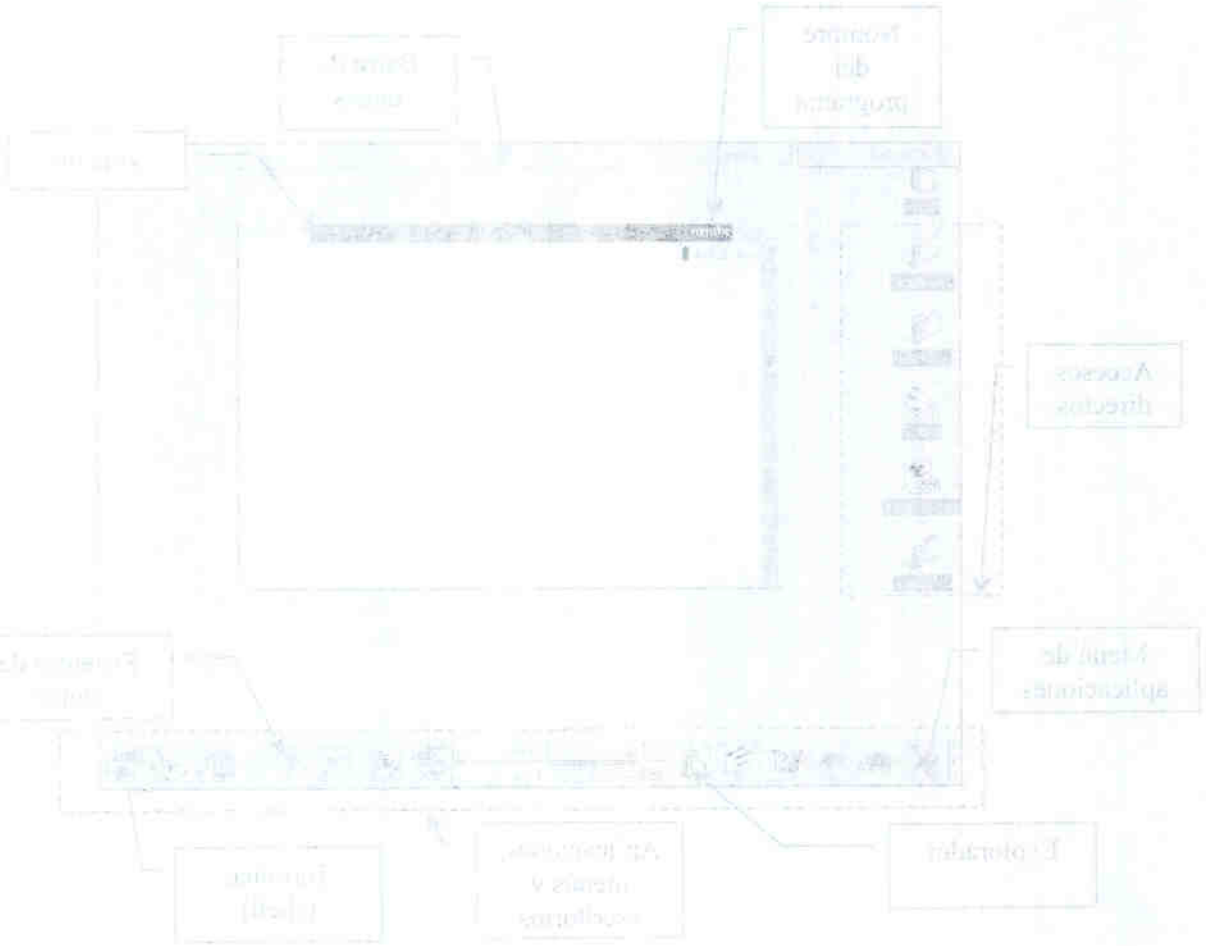
12 Comillas

- comillas simples: ' ' evitan que el shell interprete la cadena (igual que \ hace con un caracter)
- comillas dobles: " " evitar interpretación excepto variables
- comilla inversa: ` ` permiten que la salida de la orden que delimitan sea argumento de otra orden

```
Ejemplos: Variables y comillas:
------------------------------

~$ ruta1=/bin
~$ ruta2=/usr/bin
~$ yo=pak
~$ rutas=$ruta1:$ruta2/$yo
~$ echo $rutas
/bin:/usr/bin/pak

~$ echo 'rutas: - $rutas -'
rutas: - $rutas -
~$ echo "rutas: - $rutas -"
rutas: - /bin:/usr/bin/pak -
~$ echo "echo $rutas" > rutas2
~$ cat rutas2
echo /bin:/usr/bin/pak

~$ hoy=`date`
~$ echo $hoy
Sun Jan 28 00:42:38 CET 2007
```

12 Procesos en segundo plano

Añadir & al final del comando



Práctica 2: Programacion en shell

1. Introducción

"shell-script" es un fichero de texto cuyo contenido son órdenes que el shell debe interpretar como si nosotros las introdujésemos por teclado

editores simples en consola:
- pico
- nano

editor vi — lo básico → escribir comandos con ":" ej :help
- modo visualizar: pulsar V (sin ":", no es un comando)
- modo insertar: pulsar I (para parar usar ESC)
- guardar cambios: comando :write
- guarda como: comando :saveas filename
- deshacer: comando :undo

Cómo ejecutar shell script:

o bien	bash prueba
o bien	chmod +x prueba prueba
para que sea el propio shell	. prueba

Para que un fichero de texto lo interprete un determinado programa
primera línea `#! nombre_del_interprete`

ejemplo `#!/bin/bash`

2. Programación en shell

Variables: `A=hola` `echo $A` ← no dejar espacios, ya que creería que A es un comando

operaciones sobre variables:
- comando let: `let A=A+1` ⇕ `A=$[A+1]` — expresión tipo C (+, -, *, /, %) sin espacios y sin dólares
- comando expr → se le pasa como parámetros distintos los números y operaciones, por tanto usar espacios
  - `expr 2 + 1` ⇒ 3
  - `expr 2+1` ⇒ 2+1
  - `A=` `` `expr $A + 1` `` ← comillas invertidas

if, while, for

```
if orden
then
    ordenes
fi
```

```
if orden
then
    ordenes
elif
    ordenes
else
    ordenes
fi
```

```
while orden
do
    orden
done
```

```
for variable in lista-palabras
do
    ordenes
done
```

```
for i in 1 hola 3 no
do
    echo $i
done
```
⇒
1
hola
3
no

`for i in *` ← recorre los archivos del directorio

orden test (para poder hacer condiciones)

`test $A -gt 5`	para enteros, gt = "greater than"
`test $A = "hola"`	→ para cadenas
`test -e FILE`	→ FILE exists
`test -d FILE`	→ FILE is a directory
`test -f FILE`	→ FILE is a regular file

ge : greater or equal
lt : less than
le : less than or equal
eq : equal

también sirve (¡espacio!!)
`test $A -eq 5` ≡ `[ $A -gt 5 ]`

orden read asignar a una variable el contenido de una línea de la entrada estándar
↳ la 1ª línea

```
echo "¿Como te llamas?"
read nombre
echo "Hola $nombre"
```

ⓘ read espera como parámetro una variable por tanto no ponemos $, si lo pusiesemos, le estaríamos pasando como parámetro el contenido de la variable

Parámetros

$1 → parámetro 1
$2 → parámetro 2
${15} → parámetro 15

$0 → nombre del programa
$* → todos los parámetros
$# → número de parámetros

ⓘ si no hay parám $# no se sustituye por nada y queda como $#

```
for i in $*; do echo $i ; done
```
← muy util sabe que se puede poner así con ";"

→ **shift** : Desplaza los parámetros hacia la izquierda, eliminando el $1 y siendo $n = $n+1

→ **set** : cuidado: el comando set hace 2 cosas muy distintas

(1) ↳ set -v : que el shell imprima las líneas según las lee
set -x : que el shell imprima las órdenes y los argumentos
set +x : lo desactiva

(2) ↳ set lista_de_palabras : asigna la 1ª palabra a $1, la 2ª palabra a $2, etc, ...
muy práctico para descomponer una frase en palabras

cuidado: que la lista de palabras no empiece con guión o lo interpretará como en (1)
ej típico: set `ls -l`
solución: set a`ls -l` (solución un poco chapuza)

ejemplo orden read:

```
cat archivo.txt | while read aux ; do echo $aux ; sleep 0.2 ; done
```

```
#!/bin/bash
# enun1 numero fichero

# Enumera los numeros desde numero hasta 0 en la
# salida estándar o en el fichero si existe segundo
# parámetro


# Comprobar que me han pasado el parametro $1

if test a$1 = a
then
        echo 'Falta el parámetro numero'
        echo 'Formato comando: ./enun_1.sh numero [nombre_fichero]'
else

        # Comienza la enumeracion
        i=$1
        while test $i -ge 0
        do
                if test a$2 = a  # Si no existe parámetro  [fichero]
                then
                        echo $i
                else
                        #La salida estándar de echo (que será el entero i)
                        # la añade al final del archivo $2
                        echo $i >> $2
                fi
                let i=i-1
        done

fi
```

```
# Lista y cuenta los procesos del usuario que lanza el script

# Creamos una variable que almacene la cadena que se genera al ejecutar whoami
# (por eso usamos las comillas del tipo `)


NOMBREUSUARIO=`whoami`

echo
echo "Procesos del usuario $NOMBREUSUARIO"
echo "-----------------------------------"

set `ps aux`
#set lista_de_palabras asigna la primera palabra a $1, la segunda a $2
#y así sucesivamente

while test a$1 != a
do
        if test $1 = $NOMBREUSUARIO
        then
                shift 1
                echo $1 $9
                shift 9
                # Coger la 2ª y 10ª palabra de la
                #  linea de ps aux
        fi
        shift 1
done

echo
```

Dos problemas del examen

```
#!/bin/bash

# Listar en hexadecimal todos los numeros desde
# el 0x500 al 0x000


if test a$1 = a # Si no existe el parámetro [fichero]
then
        echo "500"
else
        echo "500" > $1
fi


for i in 4 3 2 1 0
do
        for j in f e d c b a 9 8 7 6 5 4 3 2 1 0
        do
                for k in f e d c b a 9 8 7 6 5 4 3 2 1 0
                do
                        if test a$1 = a # Si no existe el parámetro [fichero]
                        then
                                echo $i$j$k
                        else
                                echo $i$j$k >> $1
                        fi
                done
        done
done
```

```
#!/bin/bash

# Mostrar las lineas pares de un documento

FICHERO=$1

#Obtener numero de lineas
NUMLIN=`cat $1 | wc -l`

# Recorrer lineas pares
i=2
while test $i -le $NUMLIN
do
        echo `head --lines=$i $FICHERO | tail --lines=1`
        i=$[i+2]
done
```

Solución del profesor:

(i) entubar cat a un while

```
i=0
cat $1 |
        while read aux   # i.e. mientras queden lineas en la entrada estándar se asigna la primera linea a la variable aux
        do
                if test `expr $i % 2` -eq 0
                then
                        echo $aux
                fi
                let i=i+1
        done
```

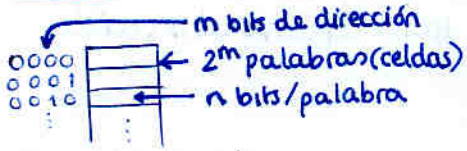
ejemplo: listar.sh para listar recursivamente en árbol:

```
# Primero, si tenemos algun parámetro $1, hacer cd $1
#(permitirá listar el directorio que elijamos)
if test a$1 != a
then
        cd $1
fi

# Segundo, recorremos recursivamente los subdirectorios
# (llamandonos a nosotros mismos)
for i in *              #Recorrer ficheros/directorios del directorio actual
do
        echo $2 $i      # Imprime el nombre del fichero o directorio $1
                        # precedido de $2, que ahora veremos lo que es (de
                        # momento, la primera vez es una cadena vacía)
        if test -d $i   # test -d FILE  == FILE exists and is a directory

        then
                $HOME/estudios/acso1/practica2/listar.sh $i --$2
                        # Se ejecuta a si mismo
                        # Como primer parámetro lo pasa el directorio a listar $i
                        # Como segundo parámetro le pasa --$2
                                # (es decir, la primera vez le pasará "--"
                                # ya que $2 estará vacio,
                                # la segunda vez le pasará "----",
                                # la tercera vez le pasará "------",
                                # y asi conseguimos el efecto de árbol tabulado)
        fi
done
```

Tema 1. Unidades Funcionales del Ordenador

Arquitectura Harvard: CPU accede a 2 memorias distintas (programa y datos)
Arquitectura vonNeumann: CPU accede a única mem

Memoria Central



- **acceso aleatorio**: dirección en un instante no depende de instantes anteriores
- comunicación CPU-memoria
  - síncrona: misma señal de reloj
  - asíncrona: protocolo de señales (ej: DTACK en 68000)

- Tipos de memoria:
  - **ROM**: hechas para ser leídas. Pueden tener límite de escrituras ej (fundidiodas) PROM → una vez EPROM, EEPROM
  - RAM dinámica: ⊣⊢ (barata, alta integración, lenta)
    - ≃ 2 trt/bit → alta integración → bajo coste
    - en reposo no disipan potencia
    - lentos en escritura y en lectura (al descargar hay que volver a cargar)
    - se descargan con el tiempo → ciclos de refresco
  - RAM estática: (cara, rápida) biestables
    - muchos trt/bit → menor integración → mayor coste
    - necesitan alimentación
    - conmutación mucho más rápida

Unidad aritmético lógica

- operador(es)
- registros específicos no visibles (acumuladores, ...)
- registros generales visibles
- indicadores de resultado: C: carry, N: negative, Z: zero, V: overflow

Unidad de control

- contador de programa
- registro de instrucción
- registros de acceso a memoria
  - MDR (mem data register)
  - MAR (mem address reg)
- circuito de control

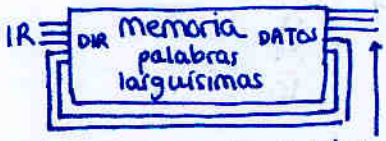
Tipos de circuito de control

→ Cableado

cientos de señales de control

complejidad enorme

→ Microprogramado



- reprogramable (cualquiera)
- más lento

las palabras son las propias señales de control

La ruta de datos

Explicación detallada de la ejecución de una instrucción

Interrupciones

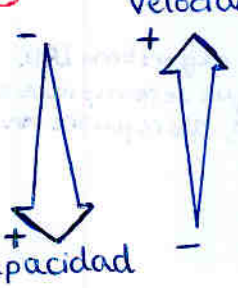
causas: fallo hardware (ej error bus) / software (ej: div por cero); línea externa (ej. tecla pulsada) / petición software (ej: trap)

Funcionamiento:
- salvo fallo hardware: primero se termina la instrucción en curso
- se guarda contexto actual (PC, registros, ...)
- se ejecuta interrupción
  - siempre la misma: que ella mire lo que pasa
  - o interrupción viene acompañada de un vector de interrupción que indica la rutina a ejecutar
- se recupera el contexto que se guardó

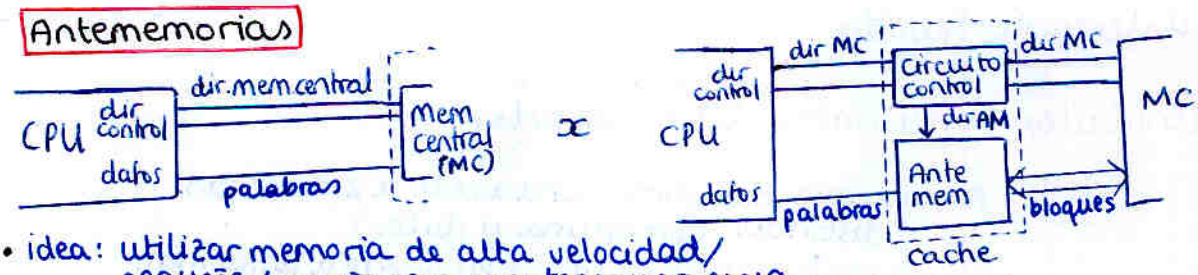
Otros aspectos:
- enmascaramiento (niveles de interrupción)
- localización de rutina mediante interrupciones vectorizadas
- niveles de privilegio (supervisor / usuario)

Estructura del sistema de memoria

Jerarquía de memoria:
- Registros internos
- Antememoria
- Memoria central
- Discos



Antememorias



- idea: utilizar memoria de alta velocidad/ pequeño tamaño para mantener una copia del bloque (bloque = unidad de transferencia entre antememoria y memoria)
- se aprovechan los casos de localidad temporal (volver a leer mismo dato)
  localidad espacial (leer datos cercanos al ya leído)
- circuito de control:
  dada una dirección de memoria:
  - se calcula en qué bloque de memoria está
  - método de asignación para averiguar en qué línea de cache debería estar
    - línea de cache ocupada por ese bloque: acierto
    - línea libre: FALLO - llenar con bloque de memoria
    - línea ocupada por otro bloque → FALLO CON REEMPLAZO
- políticas de actualización de memoria central:
  - write through: escritura inmediata a través de todos los niveles de memoria
  - write back: sólo se actualiza memoria central cuando haya que eliminar de la antememoria un bloque modificado → flag de modificado

· Métodos de asignación

→ Asignación directa

sólo puede haber como máximo un bloque de cada fila. Bloques consecutivos están en filas distintas ↯ para aprovechar localidad espacial

Antememoria

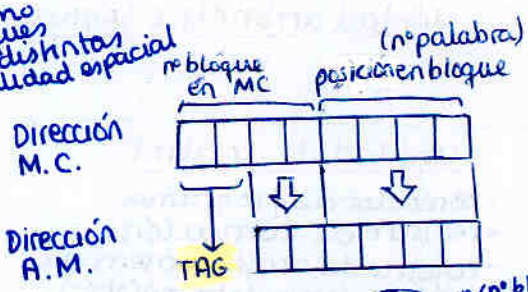
TAG	nº bloque
11	00
11	01
01	10
10	11

ejs de tag

Memoria

nº bloque = 4 primeros bits de la direcc.

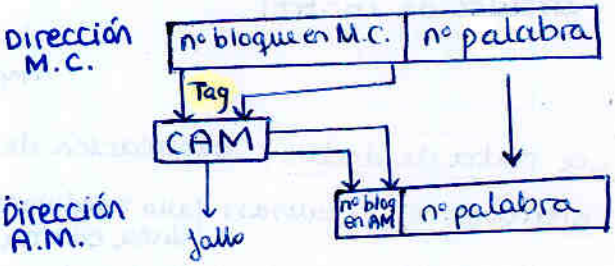
0000	0100	1000	1100
0001	0101	1001	1101
0010	0110	1010	1110
0011	0111	1011	1111



nº bloque AM = nº bloque MC (MOD) 2^n (nº bloques en AM) ↯ coger n bits LS

→ Asignación asociativa full-way

- cada bloque de M.C. a cualquier bloque de AM
- memoria asociativa (CAM) para conocer la asignación.



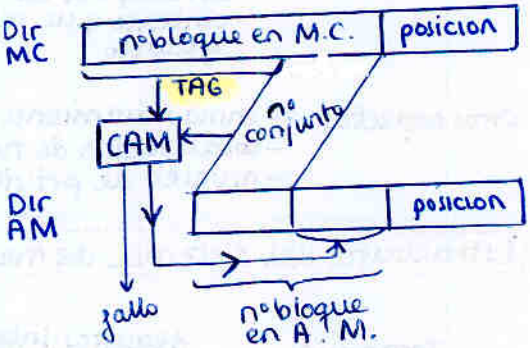
→ Asignación asociativa N-way

ej N=2

pueden haber como máximo dos bloques de cada fila

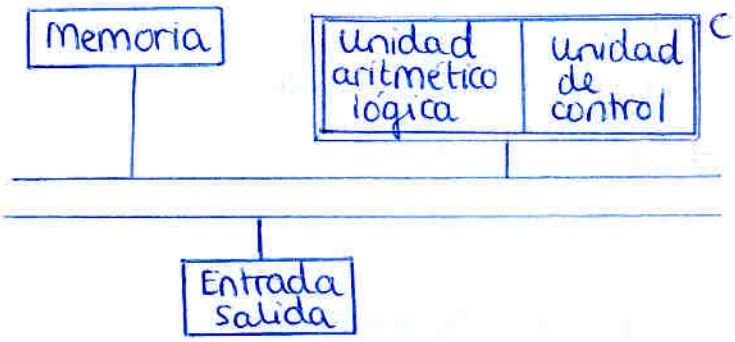
TAG	nº conjunto	nº bloque del conj.
01	00	0
10	00	1
00	01	0
10	01	1
11	10	0
–	10	1
00	11	0
01	11	1

0000	0100	1000	1100
0001	0101	1001	1101
0010	0110	1010	1110
0011	0111	1011	1111



en el fallo con reemplazo se usa el algoritmo LRU (Least Recently Used) el cual hace que se reemplace el bloque menos recientemente utilizado, ello requiere un flag adicional

Tema 1. Unidades Funcionales del Computador

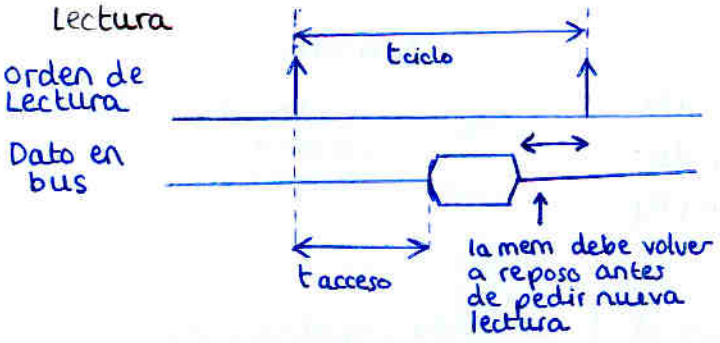


Tipo Harvard:
La CPU accede independientemente/diferencia 2 memorias: programa y datos

Tipo von Neumann:
La CPU accede a una única memoria, la cual tendrá programa/datos indistintamente.

1.1 Memoria Central

- Conjunto de celdas (palabras) de n bits (longitud de palabra) cada celda numerada (dirección) con m bits de dirección (max 2^m pal.)
- Acceso aleatorio (dirección en un instante no depende de anteriores instantes)



Comunicación CPU - memoria

Síncrona: señal de reloj común

Asíncrona: protocolo de señales (ej DTACK en 68000)

El funcionamiento de la CPU es independiente del tipo de memoria. Se consigue utilizando un controlador de memoria como intermediario.

Tipo de memoria

ROM: Hechas para ser leídas la gran mayoría de veces
Pueden tener límite de escrituras (PROM → fundir diodos → una vez)
ejs: PROM, EPROM, EEPROM

RAM dinámica: ⊣⊢ condensadores
- Ventajas:
  - ≥ 2 transistores/bit → alta integración → bajo coste
  - en reposo no disipan potencia
- Inconvenientes:
  - lentos en escritura
  - se descargan con el tiempo, requieren ciclos de refresco
  - en lectura hay que descargarlos y refrescarlos → lento

RAM estática: biestables
- Ventajas: conmutación mucho más rápida
- Inconvenientes:
  - necesitan alimentación continuamente
  - muchos transistores/bit
    - ↳ menor integración, mayor coste

1.2. Unidad aritmético lógica

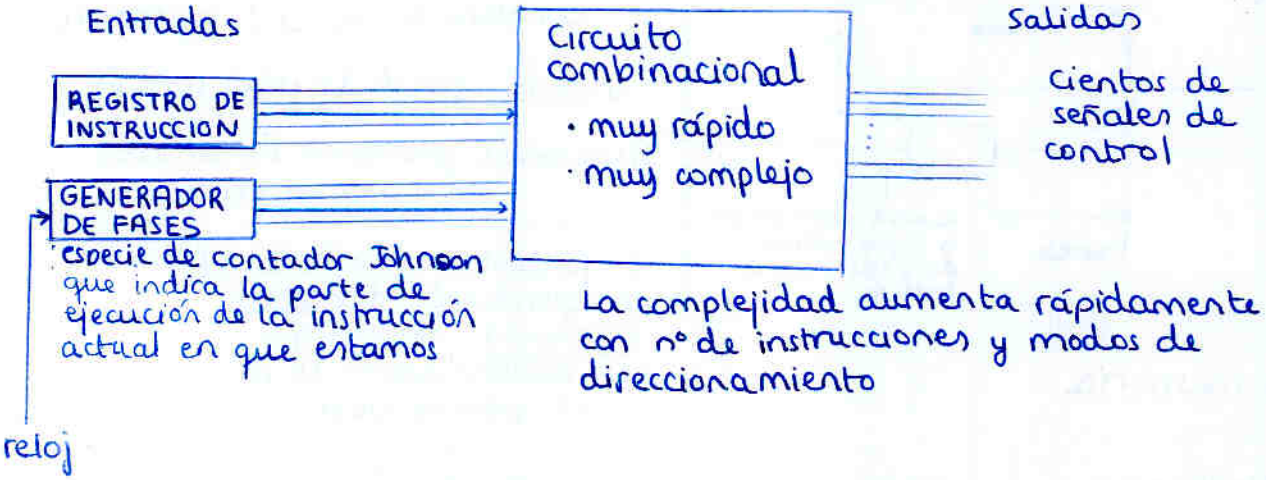
- Operador aritmético-lógico (1 o varios) (monofunción o multifunción)
- Registros específicos no visibles (acumuladores, tampones temporales, ...)
- Registros generales visibles
- Indicadores de resultados { C: carry (n bit + n bit = C + n bit)
  N: negative
  Z: zero
  V: overflow

1.3. Unidad de control

- Contador de programa: apunta a siguiente instrucción
- Registro de instrucción: almacena el código de la instrucción en curso
- Registro de acceso a la memoria central: ej MDR y MAR (data reg y address reg)
- Circuito de control: genera las señales de control para ejecutar la instrucción

Tipos de circuito de control

Cableado:



La complejidad aumenta rápidamente con nº de instrucciones y modos de direccionamiento

Microprogramado:

- Reprogramable (y por tanto ampliable y corregible) (versiones del firmware)
- más lento

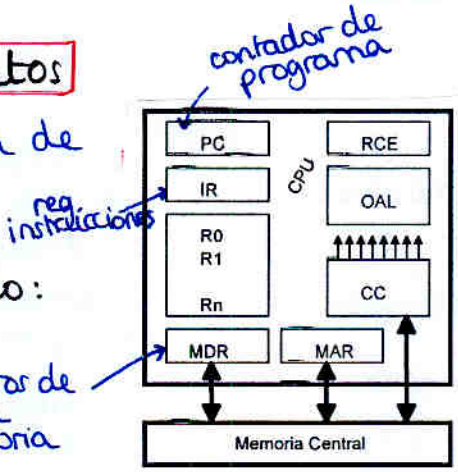
1.4. Entrada / Salida

- Comunicación con el exterior (máquina-máquina / hombre-máquina) ej: modem, pantalla, teclado
- Almacenamiento no volátil · ej: disco duro

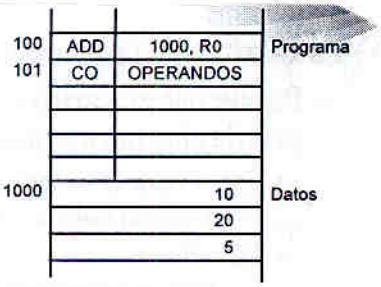
Cada periférico lleva asociado su controlador hardware con el que la CPU puede comunicarse a alto nivel.

2. La ruta de datos

explicación detallada de la ejecución de una instrucción.



ejemplo:



Supongamos que el PC vale 100:
- MAR ← PC, leer, IncPC, esperar t_{acc}
- RI ← MDR, decodificar
- MAR ← 1000, leer, esperar t_{acc}
- Ejecución: R0 ← MDR + R0

Interrupciones

Causas:
- fallo hardware (ej: error de bus)
- fallo software (ej: division por cero)
- linea externa E/S (ej: tecla pulsada)
- petición software (ej: instrucción trap)

Funcionamiento:

Salvo fallos HW, primero se termina la ejecución en curso.

- Se guarda el estado actual (PC, registro estado, registros uso general, ...)
- Se ejecuta la rutina de atención a la interrupción
  - rutina siempre igual: que ella misma mire el origen de interrupción y decida que hacer
  - o bien la interrupción viene acompañada de vector de interrupción que indica la rutina a ejecutar.
- Se recupera el estado que se guardó (y por tanto continua el programa)

Otros aspectos:
- enmascaramiento: niveles de interrupción
- localización de rutina mediante interrupciones vectorizadas
- niveles de privilegio (supervisor / usuario) → algunas instrucciones prohibidas
  - típico en procesadores de propósito general.
  - sólo se para de un modo al otro mediante interrupciones (software o hardware)

3. Estructura del sistema de memoria

Jerarquía de memoria

Registros internos
Antememoria
memoria central
Discos

4. Antememorias

Idea: Utilizar memoria de **alta velocidad / pequeño tamaño** para mantener en ella **una copia del bloque** de memoria central que se halla utilizado más recientemente

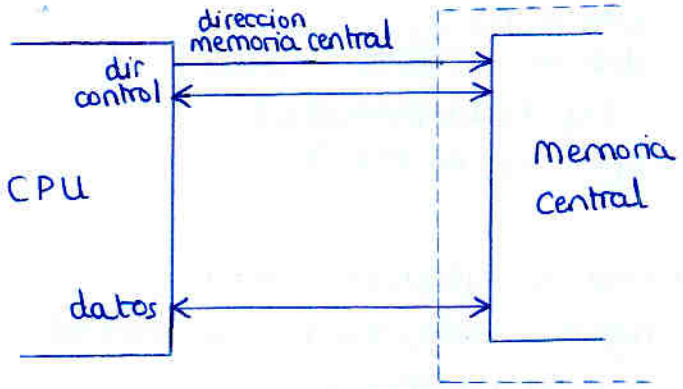
bloque: unidad de transferencia entre antememoria y memoria.

Ventajas:

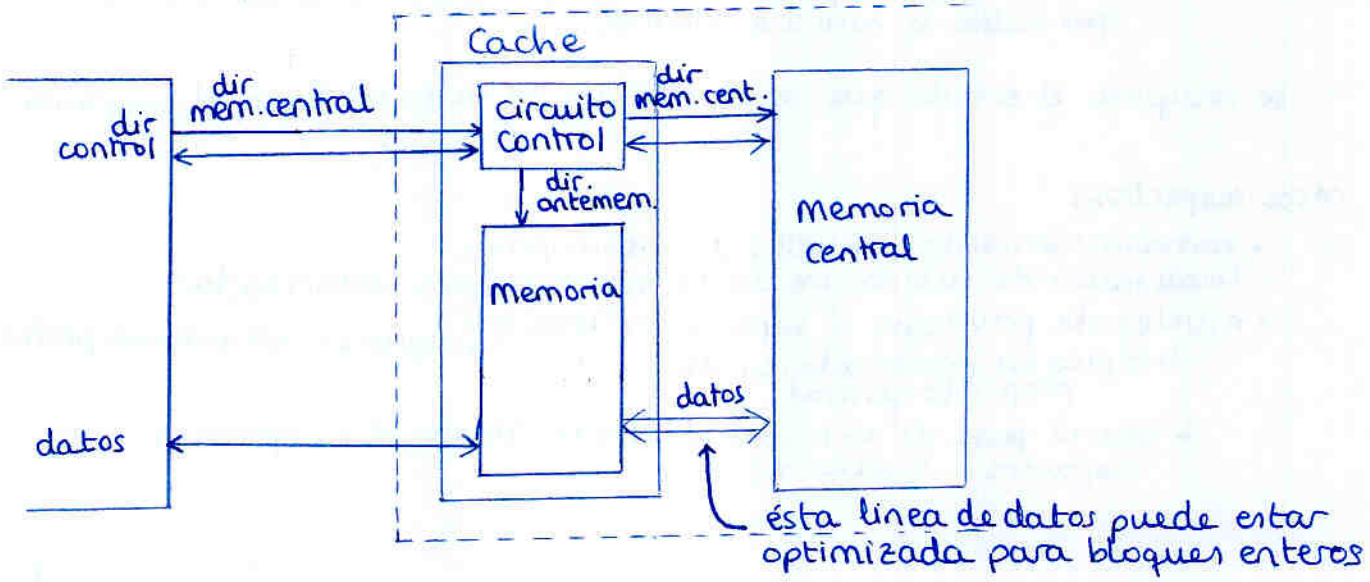
localidad temporal: me traigo el dato leído por si tengo que volverlo a leer

localidad espacial: me traigo todo el bloque cercano al dato leído por si lo voy a utilizar

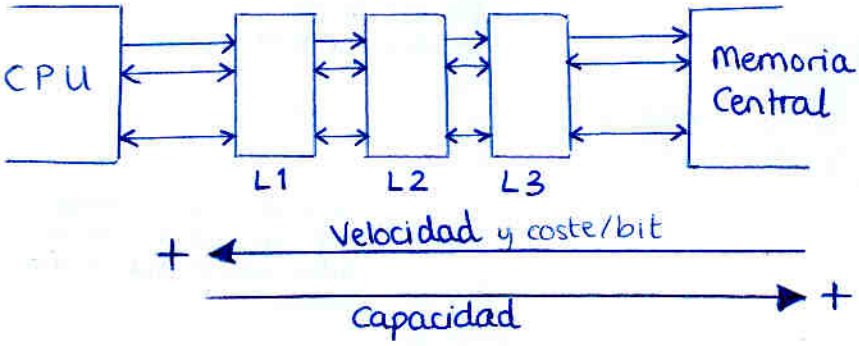
- Desde el punto de vista de la CPU:



- Podemos insertar una antememoria transparente a la CPU



se pueden poner tantos niveles como se quiera:



Funcionamiento:

- Se dividen memoria y cache en bloques de igual tamaño
- Dada una dirección de memoria:
  - Se calcula (sabiendo tamaño de bloque) en que bloque de memoria está (cogiendo los primeros bits)
  - Se utiliza un método de asignación para averiguar en que linea de cache (bloque de antememoria) debería encontrarse la copia
    - Linea ocupada por ese bloque → ACIERTO
    - Linea libre → FALLO → llenar con bloque de memoria
    - Linea ocupada por otro bloque → FALLO CON REEMPLAZO

- Si se escribe:
  Políticas de actualización de memoria central
  - Write through → escritura inmediata a través de todos los niveles de memoria
  - Write back → sólo se actualiza la memoria central cuando haya que eliminar de la antememoria un bloque modificado
    (el bloque conviene que tenga una "flag" de "modificado".

Métodos de asignación de bloques

Objetivo: Dirección de memoria central → convertir lo más rapidamente y sencillamente → Dirección de antememoria

Asignación directa

se intenta que los bloques que comparten linea de cache sean lo más lejanos posible

Cada bloque de memoria sólo puede ir a una linea de cache

Tag	bloque 0	←	b.0	b.128	b.256	...	b.3968
Tag	bloque 1	←	b.1	b.129	b.257	...	b.3969
	⋮		⋮	⋮	⋮		⋮
Tag	bloque i	←	b.i	b.i+128	b.i+256	...	b.i+3968
	⋮		⋮	⋮	⋮		⋮
Tag	bloque 127	←	b.127	b.255	b.383	...	b.4095

ejemplo sencillo:

- direcciones memoria: 16 bloques de 16 bytes
  en binario: x x x x | x x x x
  (nº bloque | nº byte en el bloque)
- direcciones antememoria: 4 bloques de 16 bytes
  en binario: x x | x x x x
  (nº bloque | nº byte)
- cada bloque en antemem tiene un tag: x x → tag que indica cual de los posibles bloques es:

sería:

antememoria

Tag	nº bloque		memoria: nº de bloque (i.e. 4 primeros bits de la dirección)			
x x	00	←	0000	0100	10 00	11 00
x x	01	←	0001	0101	10 01	11 01
x x	10	←	00 10	0110	10 10	11 10
x x	11	←	0011	01 11	10 11	11 11

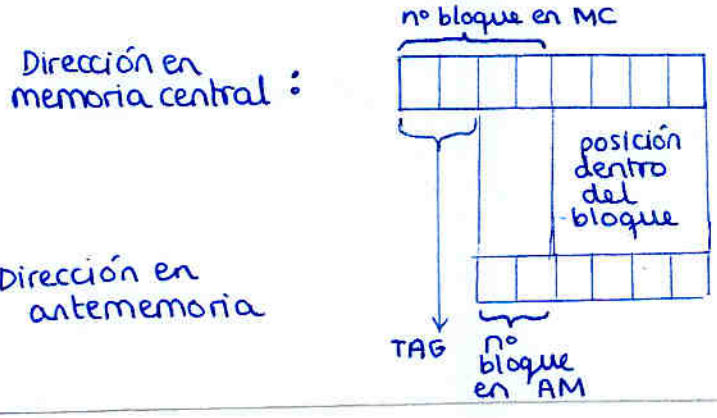
los bloques que comparten misma linea de AM sean lo más lejanos posibles (por eso se toman ultimos bits y no primeros)

En realidad se tiene:

nº bloque en antememoria = nº bloque en memoria MOD nº bloques en antememoria

si el nº de bloques en antememoria es 2^n, entonces la operación módulo es coger los n últimos bits

es decir, la conversión de una dirección a otra:



ejemplo: más realista

Memoria central: 64 kbytes → 16 bits
Antememoria: 2 kbytes → 11 bits
Tamaño bloque: 16 bytes → 4 bits

ver diagrama en pag. anterior

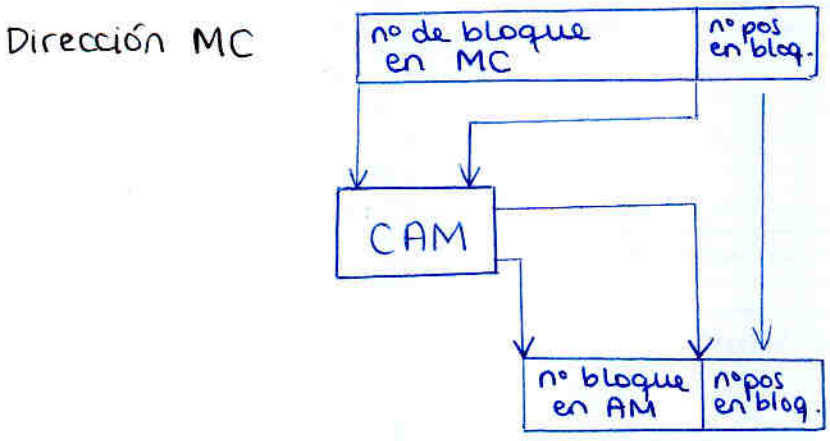
El circuito de control de la antememoria es MUY sencillo. simplemente copia directamente los 11 ultimos bits, y los 5 bits más significativos los compara (comparador 5 bits) con la TAG del bloque adecuado para saber si hay fallo (hay fallo si no coincide el tag)

Además el algoritmo para el fallo con reemplazo es obvio, simplemente sustituir el bloque.

El fallo obvio es la poca libertad. Cada bloque en MC tiene sólo un bloque en AM

Asignación asociativa (Full way associative)

- Cada bloque de MC puede ir a cualquier bloque de AM
- Se utiliza una memoria asociativa (CAM) para 'conocer' la asignación de bloques



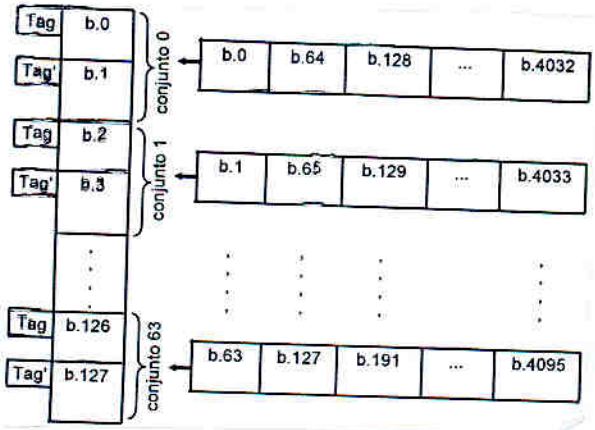
El hardware es muchisimo mas complejo y más lento

El algoritmo para el fallo con reemplazo es mucho mas complejo al requerir saber el bloque menos recientemente utilizado (para quitar ese) algoritmo LRU (Least Recently Used)

Asignación asociativa por conjunto de bloques
N-way associative

Es un buen compromiso entre las anteriores opciones

Cada bloque de la memoria central puede ir a cualquier bloque perteneciente a un conjunto de N bloques específico



- Asociación directa para decidir conjunto.
- Asociación asociativa dentro de cada conjunto.

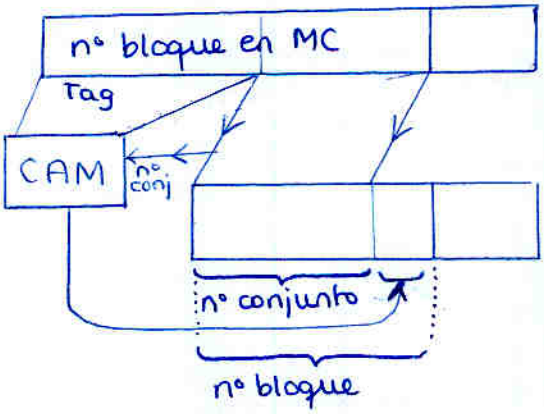
Esta vez deseamos que los bloques de MC asociados a un mismo conjunto sean lo más lejanos posibles, por lo que:

nº conjunto en AM = nº bloque en MC MOD nº de conjuntos en AM

↓

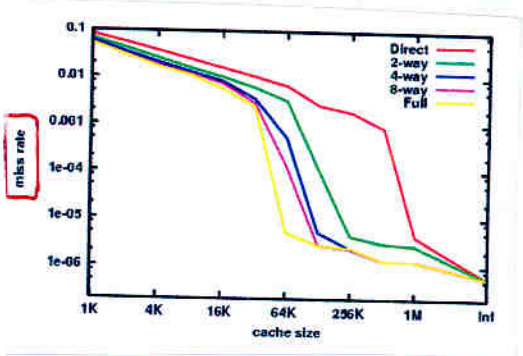
coger los n últimos bits (si nºconjuntos = 2^n)

Dirección MC



en el ejemplo de la figura N=2, la CAM le basta con recordar en cual de los 2 bloques (1 bit) del conjunto está

Comparación



Comparación de métodos de asignación para un programa determinado

Cada programa tiene un tamaño de cache determinado que le es suficiente, y aumentar el tamaño por encima no mejora notablemente el miss rate.

Problemas

1. Un computador: direcciones de memoria 20 bits
   memoria cache 1kB
   asociación asociativa por conjuntos
   ↳ conjuntos de 4 bloques
   ↳ 8 conjuntos

Si la CPU genera la dirección 7F21E (0111 1111 0010 0001 1110) cuando la cache se encuentra en la situación mostrada en la tabla 1:

CONJUNTO 0		CONJUNTO 1		CONJUNTO 2		CONJUNTO 3	
Bloque	Tag	Bloque	Tag	Bloque	Tag	Bloque	Tag
0	F05	0	100	0	021	0	101
1	F40	1	7A0	1	1A1	1	7F2
2	7E2	2	7F5	2	B27	2	041
3	803	3	6F2	3	1F5	3	7F3

CONJUNTO 5		CONJUNTO 6		CONJUNTO 7		CONJUNTO 8	
Bloque	Tag	Bloque	Tag	Bloque	Tag	Bloque	Tag
0	F50	0	000	0	022	0	011
1	7F2	1	0A0	1	3A1	1	7F2
2	012	2	7F2	2	1B7	2	014
3	103	3	101	3	F53	3	105

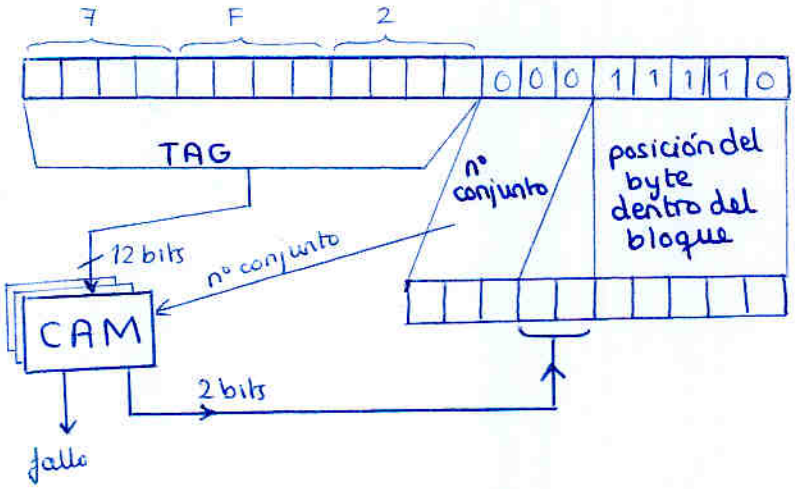
Tabla 1: Estado de las memorias CAM

Dibujemos el esquema de la conversión de direcciones

sabiendo $\frac{1\text{kB mem cache}}{8\text{ conjuntos}} = 128\text{ B/conjunto}$

$\frac{128\text{ b/conjunto}}{4\text{ bloques/conjunto}} = 32\text{ B/bloque} \Rightarrow$ 5 bits para distinguir el byte dentro de un bloque

En la dirección: 8 conjuntos ⇒ 3 bits para distinguir conjuntos
4 bloques/conjunto ⇒ 2 bits para distinguir bloque dentro del conjunto



Vemos que en la CAM, en el conjunto 0, no aparece el tag 7F2

Por tanto hay fallo con reemplazo

5. Sea una matriz (**Matriz**) de bytes de 256x256 elementos, almacenada en memoria por filas a partir de la posición de memoria 0x00010000. En esta situación, la posición (i,j) de la matriz se encontraría en la posición de memoria: 0x00010000+(i*256)+j → matriz almacenada por filas

Se quiere ejecutar sobre esta matriz un algoritmo que suma el valor de las tres primeras columnas de la matriz, para lo que se plantean dos algoritmos:

ALG_1	ALG_2

`for(i=0;i<256;i++)`
`  for(j=0;j<3;j++)`
`  Rb=Rb+Matriz[i,j]`
	`for(j=0;j<3;j++)`
	`  for(i=0;i<256;i++)`
	`  Rb=Rb+Matriz[i,j]`

Sabiendo que el código se ejecutara en un procesador que genera direcciones de 32 bits, con una memoria cache de datos de 32 KB con asignación asociativa por conjuntos de 4 bloques y tamaño de bloque de 256 bytes.

¿Cuántos fallos y aciertos se producirán durante la ejecución de cada algoritmo?

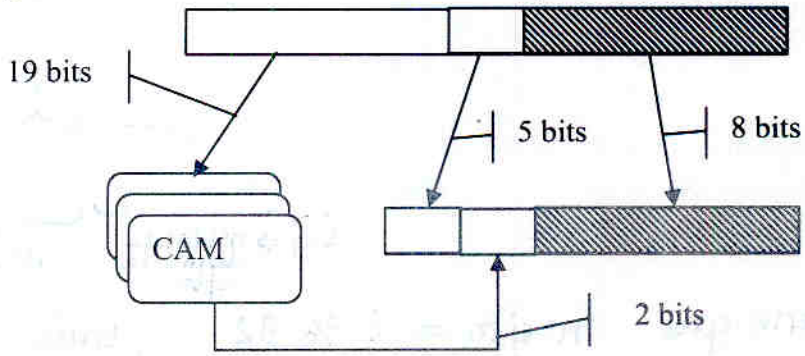
En la respuesta deberá incluir el esquema de asignación de direcciones de memoria a dirección de cache y la justificación de la respuesta a la pregunta anterior. Suponemos que la matriz nunca se había accedido hasta este momento y que las variables i, j y Rb están mapeadas en registros del procesador

↳ i.e. no hay accesos a memoria para i, j, Rb

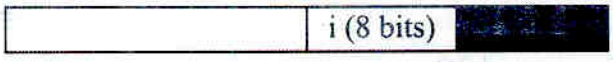
2 ptos

SOLUCIÓN:

Esquema de asignación



En esta situación podemos ver que para acceder a la posición (i,j) de la matriz se genera una dirección de 32 bits de la siguiente forma:



Cómo se puede apreciar, esto significa que una línea cabe en un bloque de cache, y que la variable i indica que bloque de memoria accedemos. En la memoria cache podemos tener simultáneamente hasta 127 bloques de memoria.

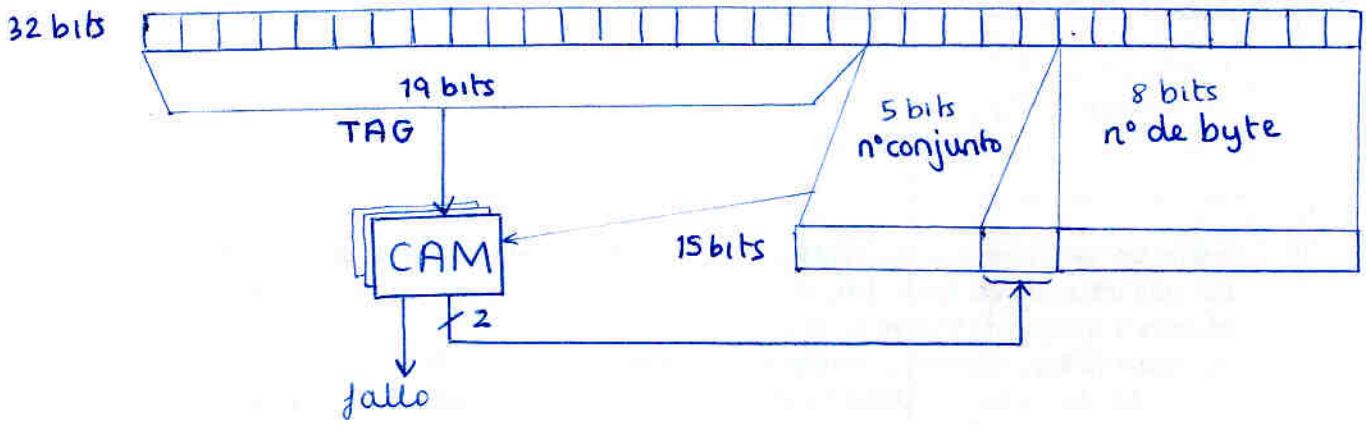
El ALG_1 accede a 256 bloques y en cada uno de ellos hace 3 accesos consecutivos, esto significa: 256 fallos (primer acceso a cada bloque) y 512 aciertos (los dos accesos siguientes).

El ALG_2 accede a 256 bloques, pero sólo hace 1 accesos consecutivos en cada uno de ellos, y luego repite tres veces. Tras 128 accesos se ha llenado la cache, los siguientes 128 la llenan con otros bloques (256 fallos). Si se vuelve a empezar vuelven a fallar los bloques iniciales: en total 768 fallos.

NOMBRE:

- Direcciones de 32 bits
- Memoria cache 32 kB → 15 bits
  - → asignación asociativa por conjuntos
  - → 4 bloques por conjunto → 2 bits para el bloque
  - → 256 bytes/bloque → 8 bits para el número de byte

$$\frac{32\ \text{KB}}{256\ \text{bytes/bloque}} = 128\ \text{bloques} \rightarrow 32\ \text{conjuntos} \rightarrow 5\ \text{bits para el conjunto}$$



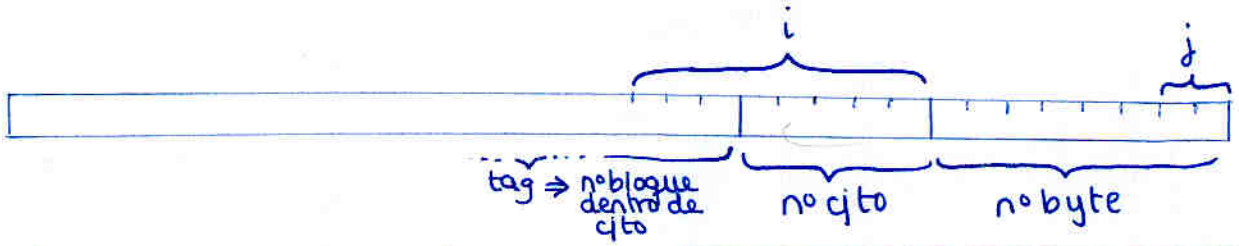
La dirección de cada acceso según i, j será:

0x00010000
+ i * 0x100
+ j

= 0x 0 0 0 1 [i] [0] [j]

i ← i requiere 8 bits (i.e. 2 dígitos) al multiplicar x 256 es desplazar 8 a la izquierda (i.e. 2 dígitos)

j ← j requiere 2 bits



se tiene que $\text{nº cjto} = i \ \% \ 32$

truco: % 32 da como resultado un número entre 0 y 31 (i.e. resultado 5 bits)
≡ 5 LSB

Para ALG 1

	Nº conjunto	Tag	
i=0, j=0	0	0	→ Fallo (esta vacio)
j=1	0	0	} Acierto
j=2	0	0	
i=1, j=0	1	0	→ Fallo (a pesar de ser mismo tag, es distinto conjunto ↓ mismos 2 bits de bloque)
j=1	1	0	} Acierto
j=2	1	0	
⋮			
i=31 j=0			FAA
1			
2			

Ya hemos usado un bloque en cada conjunto
Podemos hacer 4 paradas de este tipo hasta llenar los 4 bloques de cada conjunto ($i < 128$)

Es decir, en 4 primeras pasadas que i hace por todos los ejtos

$4 \times 32 \times 1 = 128$ fallos
$4 \times 32 \times 2 = 256$ aciertos

Al hacer la quinta pasada, los conjuntos tienen llenos sus 4 bloques
De nuevo, el primer acceso es un fallo con reemplazo y los dos siguientes aciertos

Y así hasta acabar (i.e. 3 pasadas más)

Total finalmente:
128 fallos + 128 fallos con reemplazo → 256 fallos
512 aciertos

Para ALG 2

			nº conjunto	tag	
j=0	i=0	(*)	0	0	→ Fallo (está vacío)
	i=1		1	0	→ Fallo (está vacío)
	i=2		2	0	→ Fallo (está vacío)
	⋮		⋮	⋮	
	i=31		31	0	→ Fallo " "
	i=32		0	1	→ Fallo " " (es un nuevo bloque dentro de mismo ejto)
	i=33		1	1	→ Fallo " "
	⋮		⋮	⋮	
	i=63		31	1	→ Fallo " "
	⋮		⋮		
	i=127		31	3	→ Fallo " " ← ya ha llenado los 4 bloques de cada conjunto
(2*)	i=128		0	0	→ Fallo con reemplazo ← se vuelven a rellenar
	⋮		⋮	⋮	
	i=255		31	3	→ " " "
j=1	i=0		0	0	→ Fallo con reemplazo
	⋮		⋮	⋮	Es una lástima; sería un acierto si aún estuviera en cache el bloque que se escribió en (*), pero por desgracia éste se reemplazó en (2*) por otros bloques
	i=255		31	3	→ Fallo con reemplazo
j=2	⋮				

Acabar siendo todo fallos: total $256 \times 3 = 768$ fallos

Tercer algoritmo ALG 3

```
for (j=0; j<3 ; j++)
    for ( i= 0; i<128 ; i++)
        Rb = Rb + Matriz [i,j]

for (j= 0; j< 3 ; j++)
    for ( i= 128; i< 256; i++)
        Rb= Rb + Matriz [i,j]
```

Claramente este algoritmo está pensado para no caer en el fallo de ALG 2 (i.e. sobreescribir bloques que luego íbamos a aprovechar)

		nº ciclo	nº bloque	
j = 0	i = 0	0	0	→ Fallo
	⋮			
	i = 127	31	3	→ Fallo

} 128 Fallos

---

en este punto hemos llenado toda la cache, afortunadamente ahora cambia j y accedemos al byte consecutivo al que se accedió en mismo bloque

j = 1	i = 0	0	0	} 128 aciertos
	⋮	⋮	⋮	
	i = 127	31	3	
j = 2	i = 0	0	0	} 128 aciertos
	⋮	⋮	⋮	
	i = 127	31	3	

---

j = 0	i = 128	0	0	→ Distinto tag pero mismo bloque: Fallo con reemplazo
	⋮	⋮	⋮	
	i = 255	31	3	

} 128 Fallos con reemplazo

asignación asociativa sustituye el nuevo tag que se le pide en el lugar del bloque que llevaba más sin ser usado

---

en este momento se han reemplazo todos los bloques de la cache, al cambiar ahora j se reutilizan estos nuevos bloques ya que se accede al byte consecutivo al que se accedió tras reemplazar antiguo bloque

j = 1	i = 128	0	0	} 128 aciertos
	⋮	⋮	⋮	
	i = 255	31	3	
j = 2	i = 128	0	0	} 128 aciertos
	⋮	⋮	⋮	
	i = 255	31	3	

Total: 256 fallos, 512 aciertos] igual que ALG1

Tema 2. Introducción a los Sistemas Operativos

1. Funciones del SO

- Máquina extendida
  - Permite una abstracción del HW
  - Las aplicaciones ven un interfaz independiente del HW. Las aplicaciones hacen "llamadas al sistema"

- Gestión de recursos
  - El SO da acceso y reparte los recursos del sistema entre las aplicaciones y usuarios (ej: CPU, memoria, impresoras, ...)

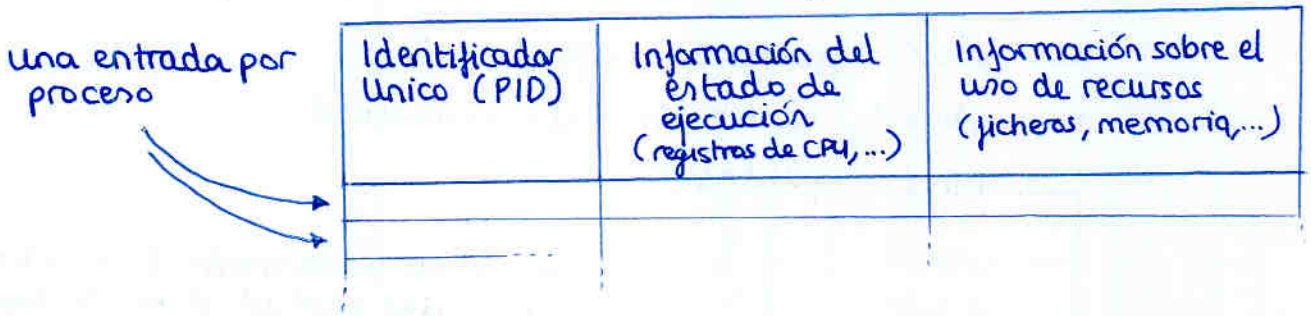
2. Visiones del SO

El usuario: no conoce los servicios del SO
conoce las órdenes de un intérprete de ordenes (ya sea consola o gráfico)

El programador: conoce los servicios del SO
crea los programas que usan los usuarios
sus programas acceden a los servicios del SO mediante llamadas al sistema

3. Procesos

- Un proceso es un programa en ejecución
  (no confundir programa con proceso)
- Un SO multiproceso tiene una tabla de procesos



4. Ficheros

- Son una abstracción de almacenamiento
- Los datos del fichero tienen una relación conocida por el SO o por la aplicación o usuario que lo creó.
- El SO guarda información adicional del archivo (permisos, fecha creación, ...)
- El SO ve los ficheros como un vector de bloques
- Permite las llamadas de creación (creat), eliminación (ulink) o modificación (read, write, lseek)

5. Procesos en UNIX

- UNIX: multiproceso y multiusuario
- relación jerárquica (padre/hijo) entre procesos

Proceso padre de todos: INIT PID = 1	- el primero que se ejecuta - inicializa la máquina cuando se enciende - monitoriza y "limpia" - cierra todo lo que inicializó cuando se apaga

Un proceso siempre debe saber porqué sus hijos mueren:

- Cuando un proceso muere, no libera todos los recursos, sino que mantiene una información para el padre.
  El proceso no morirá del todo liberando todos sus recursos hasta que el padre no lea dicha información y conozca las causas.
  Un proceso en ese estado se llama **zombi**

Si un padre con hijos muere, se dice que los procesos hijos se quedan **huérfanos**.

ps f ps -e f	Ver procesos con la jerarquía padre-hijo -e: ver TODOS los procesos del sistema

6. Ficheros en UNIX

Se tiene una organización jerárquica gracias a los directorios
↓
en realidad son también ficheros

Directorio raíz
El único no contenido en otro directorio
El SO sabe como encontrarlo

- etc/ ficheros de configuración
- sbin/ ejecutables del administrador
- bin/ ejecutables
- usr/
- root/
- home/

Al existir únicamente éste árbol, cualquier unidad debe "colgar" de éste (montar unidad)

archivo /etc/fstab : dónde montar cada unidad
file system table

ruta: secuencia de nombres de directorio que indica la posición de un fichero

Todos los procesos saben encontrar únicamente 2 directorios

→ Directorio raiz [/]

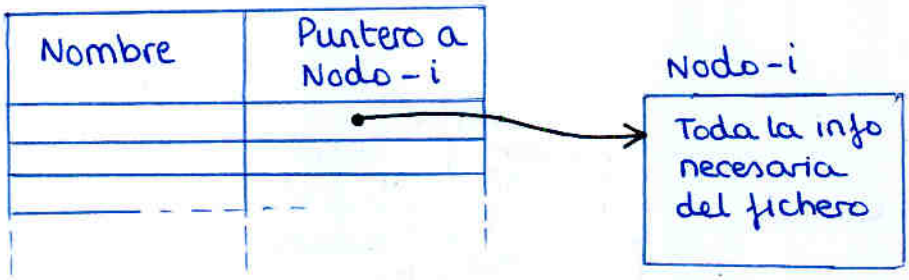
- Directorio actual

A partir de éstos pueden encontrar rutas que empiecen en uno de esos 2 directorios

- ruta absoluta desde directorio raiz ej /home/pak/
- ruta relativa desde directorio actual ej ./programas/java/
  ej ./../toni/] equiv.
  ej programas/java/

¿Qué son los directorios?

- Los directorios son ficheros con un formato conocido por el SO
- Cada directorio es una tabla con todos los ficheros (incluyendo otros directorios) que contienen.



siempre existen las entradas . y ..

- . : directorio actual → contiene puntero al nodo-i del propio directorio
- .. : directorio padre → contiene puntero al nodo-i del directorio padre

Hard-Links:

Todos los nodo-i de un fichero (incluyendo si éste es un directorio) poseen un contador de referencias.

Cuando se borra un fichero de la tabla de un directorio, lo que en realidad se borra no es el fichero, sino el puntero al nodo-i del fichero, y además se reduce el contador de referencias del nodo-i en uno.

Como puede haber más de una entrada de directorio apuntando al nodo-i de un fichero (HARD-LINK), el fichero no quedará inaccesible hasta que las referencias a su nodo-i (junto al contador de referencias) lleguen a cero.

El nodo-i de un fichero que es un directorio tiene como mínimos dos referencias:

- La que existe en la entrada correspondiente de la tabla de su directorio padre.
- La que existe en la entrada '.' de su propia tabla

Además, tendrá una referencia por cada entrada '..' en las tablas de sus subdirectorios.

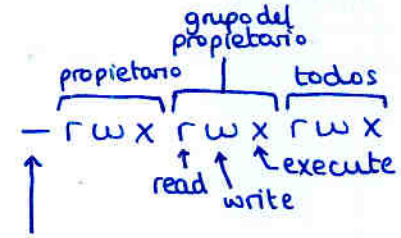
Por tanto un directorio tiene muchos hard-link y por eso para borrarlos se usa el comando especial rmdir

ls -l	La segunda columna muestra el número de referencias al nodo-i del correspondiente fichero (contador de ref.) · si no es un directorio suele ser 1 · si es un directorio suele ser 2 + nº subdirectorios

Orden ls

ls -l

La primera columna



- : fichero
d : directorio
c : dispositivo
l : enlace simbólico (soft link)
es un archivo que almacena una ruta, que se sustituye en lugar de sí mismo (como un shortcut)

Usuarios:

UID : user identifier
GID : group identifier

un usuario puede pertenecer a muchos grupos

7. El intérprete de órdenes

- Es una aplicación
- Proporciona una interfaz entre usuario y SO
- Operaciones básicas [navegación por el sistema de archivos, ejecución de programas]
- Existen ordenes internas y externas
  - internas: dentro del propio intérprete
  - externas: requiere la creación de un proceso hijo

Tema 3:
Concepto de Arquitectura

1 Definición de arquitectura.
2 Taxonomía de Flynn
3 Tipos de paralelismo
4 Factores a considerar en el diseño
5 Análisis de prestaciones

1 Definición de arquitectura:

Concepto clásico

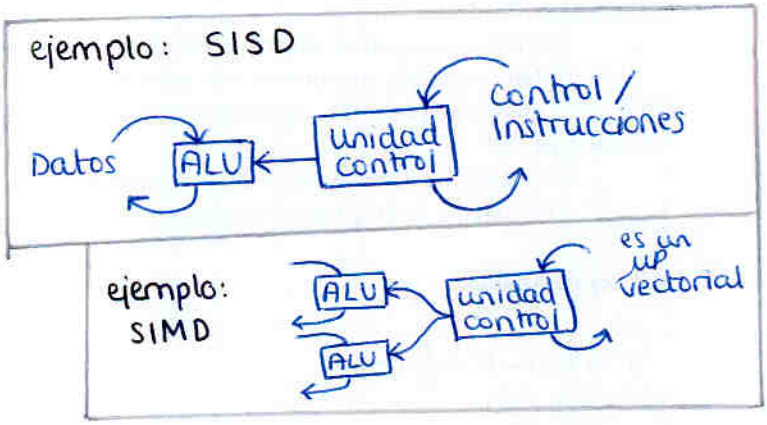
- Myers " En los sistemas de computación, la arquitectura puede definirse como la distribución de funciones a un nivel o frontera dentro del sistema, y la definición precisa de esa frontera"
- Arquitectura de computadores = Arquitectura del juego de instrucciones
- Visión del programador en ensamblador

Concepto moderno

- La arquitectura de un computador debe tener en cuenta además la organización y la implementación
  - Arquitectura del juego de instrucciones. ¿Qué puede hacer el computador?
  - Organización. ¿Con qué elementos y cómo están interconectados?
  - Implementación. ¿Con qué tecnología?

2 Taxonomía de Flynn

- Flujo de instrucciones o de control.
- Flujo de datos.
- Cuantos flujos simultáneos existen. Categorías:
  - SISD (Single Instruction Single Data)
  - SIMD (Single Instruction Multiple Data)
  - MISD (Multiple Instruction Single Data)
  - MIMD (Multiple Instruction Multiple Data)



3 Tipos de paralelismo

- Paralelismo espacial: replicar entidades, normalmente iguales, de forma que trabajen de forma simultánea (Flynn).
- Paralelismo temporal: dividir una tarea compleja en subtareas, cada una realizada en elementos diferentes. En un mismo instante se realizan varias subtareas procesando datos distintos. El paralelismo temporal también se denomina: segmentación, superposición, pipeline.

4 Factores a considerar en el diseño

- Requisitos funcionales: qué características de funcionamiento se desean.
- Tendencias en tecnología y software
- Equilibrio entre hardware y software.
  - Software: coste bajo de rectificación de errores, sencillez de diseño y actualización.
  - Hardware: mayor rapidez
- Tendencias en el coste
  - Contribución de cada componente al coste final
  - Coste y precio muy diferentes.

4.1 Requisitos funcionales

- Aplicaciones
  - Propósito general Potencia equilibrada
  - Propósito especifico Características particulares
  - Científica Soporte coma flotante
  - Comercial Soporte base de datos, transacciones...
- Compatibilidad
  - A nivel lenguaje Diseño flexible (compilador)
  - Binaria Menos flexible
- Estandares
  - Coma flotante IEEE, IBM
  - E/S SCSI, VME, IDE...
  - Redes Soporte para Ethernet, Token Ring...
  - Lenguajes de alto nivel C/ C++, COBOL, Java, FORTRAN
- Sistemas operativos
  - Direccionamiento Limita tamaño aplicaciones y datos
  - Gestión de memoria Paginación, segmentación
  - Protección
  - Gestión de procesos Soporte a multitarea
  - Interrupciones
  - Niveles de ejecución

4.2 Tendencias en tecnología y software

- El diseño de una arquitectura debería sobrevivir a los cambios en la tecnología y el software
- Tendencias Hardware
  - Circuitos integrados: cada tres años x2 el nº de transistores y su velocidad (un poco menos)
  - Memorias: Cada tres años x4 el tamaño y cada 10 años se reduce en un 1/3 el tiempo de acceso.
  - Discos: Cada año x2 la capacidad y cada 10 años 1/3 el tiempo de acceso
- Tendencias SW
  - Necesidades de memoria se duplican cada año
  - Utilización de lenguajes de alto nivel

4.3 Ley de Amdhal

$$T' = T \cdot (1 - F) + \frac{T}{S} \cdot (F)$$

- T = tiempo antes de mejora
- F = parte del total debido a la parte a mejorar
- S = en cuanto se mejora esa parte
- T'= Tiempo tras mejora

- Conclusiones:

– Hay que mejorar el caso más común
– La parte que no se mejora impone un límite superior a la mejora obtenida

5 Análisis de prestaciones

- Definición de prestaciones según punto de vista:
  - Usuario: acabar pronto.
  - Administrador: terminar muchos trabajos.
- Necesario medir prestaciones para:
  elegir alternativas de diseño (DISEÑADORES) y ayudar en la compra (USUARIOS)
- Necesarias unidades de medida
  - Medidas microscópicas: Miden las prestaciones atendiendo a conceptos de bajo nivel
  - Medidas macroscópicas: basadas en el tiempo de ejecución de programas y la definición de los mismos

5.1 Medidas microscópicas

- MIPS: Millones de Instrucciones Por Segundo.
  - "Mide la velocidad" de ejecución de instrucciones.
  - Las instrucciones pueden ser diferentes entre máquinas
- MFLOPS: Millones de operaciones en coma flotante por segundo.
  - "Mide la velocidad" de cálculo en coma flotante
  - Útil en aplicaciones científicas
  - Un FLOP se define en alto nivel:
    a[i] = b[i] + c[i] * d [i]

CPI: Ciclos por instrucción

- Parámetros S y M:

Se define un programa que se utilizará para hacer la comparación, el tiempo de ejecución no es importante

- Parámetros S y M:
  - S: Tamaño del programa en bits
  - M: Cantidad de información en bits transferida con memoria (código y datos)
- El objetivo es minimizar estas cantidades
  - Anterior al concepto de cache, no se tiene en cuenta la presencia de este tipo de memoria

5.2 Medidas macroscópicas

- Utilizar como medida el tiempo tardado en ejecutar un programa, especificando dicho programa.
- Tipos de programas:
  - Propios
  - Programa reales: *Compresor JPEG.*
  - Núcleos (kernels). Fragmentos de código de programas reales: *Linpack.*
  - Benchmarks. Programas de coste conocido o diseñados para medir prestaciones (sintéticos): *Drhystone.*
    - Benchmark: cualquier programa que se utiliza para medir prestaciones.

Problemas

- Programas Reales: Sólo cuando existe la máquina y el programa se puede encontrar para ella.
- Programas basados en bucles: Los compiladores pueden detectarlos y generar optimizaciones especiales para esos bucles/instrucciones.
- Benchmarks sintéticos: pueden estar diseñados o implementados para obtener mejores resultados en una máquina concreta.

– SPEC: Standard Performance Evaluation Corporation

- Comparación de máquinas: Se suele realizar sobre una máquina conocida (Athlon XP 2800+)
- Para "creer" unos resultados se debe conocer:
  - El benchmark concreto que se ha utilizado
  - La configuración HW exacta de la máquina de prueba
  - La configuración SW exacta de la máquina de prueba
  - El COMPILADOR y las opciones de compilación que se utilizaron para generar los ejecutables.

Los benchmarks no son válidos para siempre o para cualquier tipo de máquina

Ejemplo: Ley de Amdhal

Tiempo de CPU { 50% coma flotante
20% es sqrt (que se incluye en coma flotante)

¿Qué es mejor ?

a) mejorar HW que ejecuta sqrt 10 veces más rápido
b) mejorar toda la unidad de coma flotante, factor de mejora 1'6

$$T' = T(1-F) + \frac{T}{S}\cdot F \quad \Rightarrow \quad \frac{T'}{T} = (1-F) + \frac{F}{S}$$

a) $S = 10$, $F = 0'2$ $\quad \frac{T'}{T} = 0'82 = 41/50 \quad \longrightarrow \quad \frac{T}{T'} = 1'220 = 50/41$

b) $S = 1'6$, $F = 0'5$ $\quad \frac{T'}{T} = 0'8125 = 13/16 \quad \longrightarrow \quad \frac{T}{T'} = 1'231 = 16/13$

} mejora de la CPU

Podíamos calcular la mejora de coma flotante usando
a) $F = 0'4$
b) $F = 1$

además $M_{CPU} = \dfrac{1}{0'5 + \frac{0'5}{M_{CF}}}$

2. La empresa DIGIPRINT dispone de una imprenta de altas prestaciones con un sistema de impresión monitorizado por un procesador en tiempo real. El sistema corrige el color de las copias conforme las imprime. Cada copia impresa supone tres operaciones:
   - Un proceso P de generación de una copia impresa que incorpora el escaneado de la hoja generada. La duración de este proceso es invariable, pues depende de un sistema mecánico y óptico fijo.
   - Un proceso A de análisis de la imagen obtenida en la fase P. Este proceso es realizado por un sistema de n procesadores digitales de señal (*DSP*) que operan en paralelo. La productividad del proceso es proporcional al número de procesadores *DSP* instalados.
   - Un proceso C de ajuste de parámetros de impresión que debe de realizarse a partir de los datos suministrados por A y que debe realizarse antes de imprimir una nueva hoja. Este proceso lo realiza un procesador que funciona a 500 MHz.

Inicialmente, se ajusta el sistema para que imprima N hojas por segundo. Para conseguirlo, se observa que son necesarios un mínimo de 20 procesadores *DSP* para que el trabajo se realice en el plazo disponible. Con dicho número de DSP, los procesos P, A y C ocupan (respectivamente) el 30%, el 40% y el 30% del tiempo de operación total del sistema y no sobra tiempo para comenzar a imprimir una nueva hoja.
El equipo completo cuesta 10,000 euros. Se pretende ahora multiplicar por 2 la productividad global (que se mide en copias por segundo). Para ello, se puede reemplazar el procesador actual por otro con un reloj a 1 GHz, que ejecuta el mismo programa que el anterior (con el mismo CPI), por un coste de 1000 euros, y si es necesario, la adquisición de procesadores DSP a razón de 100 euros la unidad.
Calcula utilizando la ley de Amhdal:

a) Si no se sustituye el procesador, ¿cuál es el mínimo número de procesadores *DSP* necesarios para conseguir la nueva productividad?
b) Si se sustituye el procesador por el modelo más rápido, ¿cuál será el mínimo número de procesadores *DSP* necesarios para conseguir dicha productividad?
c) Qué configuración consigue la productividad propuesta con el coste mínimo, teniendo en cuenta que cada hoja a imprimir es un trabajo independiente. Considera que eliminar procesadores DSP no afecta al coste.

Justifica cualitativa y cuantitativamente todas las respuestas.1,5 ptos

a) Si sólo mejoramos la parte de DSP

Con 20 DSP, el proceso dura un 40% del total

Añadiendo m DSP's, el factor de mejora es $1+\frac{m}{20}$

$$\frac{T'}{T} = 1-F+\frac{F}{S}$$

siendo $\boxed{S = 1+\frac{m}{20}}$ Factor de mejora si inicialmente habían 20 DSP's y añado m más

$F = 0'4$ · truco: F es la cantidad ANTES de la mejora

$$= 0'6 + \frac{0'4}{S}$$

No podemos alcanza $\frac{T'}{T} = 0'5$ mejorando sólo el proceso B que es un 0'4 del total!

b) A — 30% → sin mejora
C — 40% → mejora S_c desconocida
P — 30% → mejora $S_p = 2$

$$T' = T\left(0'3 + \frac{0'4}{S_c} + \frac{0'3}{S_p}\right)$$

Ley de Ahmdal cuando se mejoran varias partes cada una con distinto factor de mejora

conociendo $S_{global} = \frac{T}{T'} = 2$

podemos despejar S_c

Otro método:
Primero calcular la mejora de P

$$T_2 = T\left(1-F+\frac{F}{S}\right)$$
$$= T\left(0'7 + \frac{0'3}{2}\right)$$
$$= 0'85\,T$$

Y ahora aplico la mejora de C

$$T' = T_2\left(1-F+\frac{F}{S}\right)$$

$$\frac{T'}{T} = \left(1-\frac{0'4}{0'85}+\frac{0'4/0'85}{S_c}\right)\cdot 0'85$$

Cuidado: ya no es el 40% ahora es $\frac{0'4}{0'85}$

$$\boxed{F = \frac{0'4}{(0'4+0'3+\frac{0'3}{2})}} \text{ duración total tras mejora}$$

Se obtiene $S_c = 8$

Si hay 20 DSP's, necesitaremos $8 \cdot 20 = 160$ DSP's además del nuevo µProcesador ($S_p = 2$) para reducir el tiempo a la mitad.

Tema 4: Diseño del juego de instrucciones

1 Clasificación de los juegos de instrucciones
2 Direccionamiento de memoria
3 Operaciones
4 Evolución de la arquitectura del juego de instrucciones
5 Ejemplo de juego de instrucciones: el DLX

1 Clasificación de los juegos de instrucciones

- Según el almacenamiento de los operandos:
  - Pila (Ej.: HP 3000). (típico en calculadoras)
- Los datos y resultados se almacenan en una pila. Operandos implícitos.
  - Acumulador (Ej.: M6809, DSP).
- Hay un registro *acumulador* que almacena implícitamente uno de los operandos y sobre el cual se deposita el resultado.
  - Registros de propósito general (Ej.: PowerPC).
- Los datos y resultados se almacenan en un banco de registros o memoria. Todos los operandos son explícitos.

Ejemplo: C = A+B

Pila

```
PUSH A
PUSH B
ADD
POP C
```

Acumulador

```
LOAD A
ADD B
STORE C
```

Registros de propósito general

```
LOAD R1,A      LOAD R1,A      MOVE C,A
ADD R1,B       LOAD R2,B      ADD C,B
STORE C,R1     ADD R3,R1,R2
               STORE C,R3
```

- Otros criterios:
  - Número de operandos explícitos por instrucción
  - Ubicación de los operandos
  - Operaciones
  - Tipos y tamaños de los operandos

1.1 Clasificación de las máquinas
de registros de propósito general

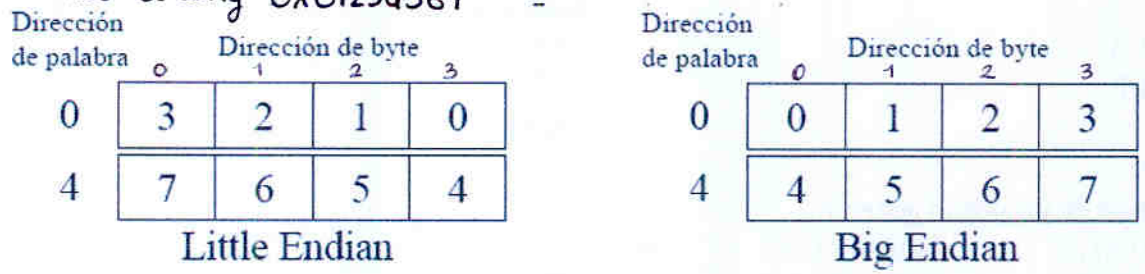
Número de direcciones mem/instruc. típica ALU	Máximo número de operandos permitidos	Ejemplos	
0	2	IBM RT-PC	registro-registro ó load/store
	3	SPARC, MIPS, HP PA, ARM	
1	2	M68000, 80x86	registro-memoria
	3	IBM 360	
2	2	PDP-11, National 32x32, IBM 360	
	3		memoria-memoria
3	3	VAX	memoria-memoria

2. Direccionamiento de memoria

· Interpretación de las direcciones
– Memorias direccionables por byte, acceso por palabras.
– Convenios para numerar los bytes de una palabra (para saber en que orden se serializa un elemento multibyte)

1. *Little Endian:* En menor dirección (little) se almacena el byte de menor peso (endian). (Intel)
2. *Big Endian:* En mayor dirección (big) se almacena el byte de menor peso (endian). (Motorola, SPARC, ARM)

ej: almacenar el long 0x01234567



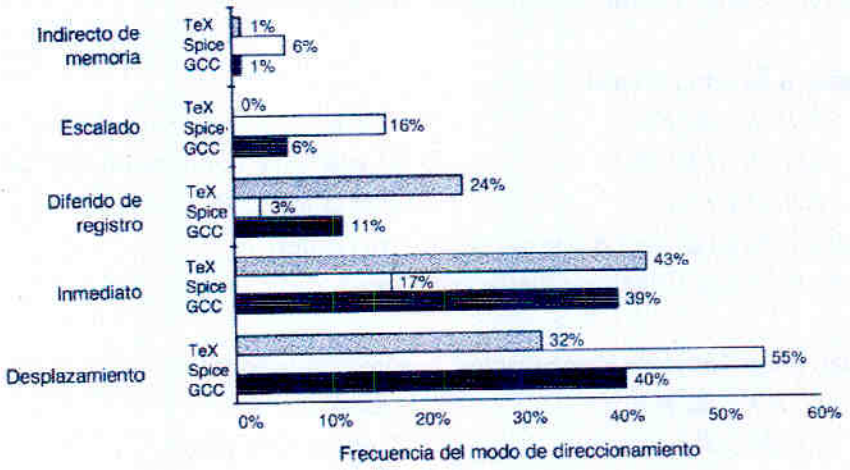

```
/* ENDIAN.C */
#include <stdlib.h>
void main() {
      unsigned long a1=0x12345678;
      unsigned short *s1,i;
      unsigned char *b1;
      printf("%0x\n",a1);
      s1=(unsigned short*)&a1;
      for(i=0;i<2;i++) {
            printf("%0x ",*s1++);
      }
      printf("\n");
      b1=(unsigned char*)&a1;
      for(i=0;i<4;i++) {
            printf("%0x ",*b1++);
      }
      printf("\n");
}
```

2.1 Modos de direccionamiento

Modo de direcciona-miento	Ejemplo instrucción	Significado	Cuándo se usa
Registro	`Add R4, R3`	R4←R4+R3	Cuando un valor está en un registro.
Inmediato o literal	`Add R4,#3`	R4←R4+3	Para constantes. En algunas máquinas, literal e inmediato son dos modos diferentes de direccionamiento.
Desplaza-miento	`Add R4, 100(R1)`	R4←R4+M[100+R1]	Acceso a variables locales.
Registro diferido o indirecto	`Add R4, (R1)`	R4←R4+M[R1]	Acceso utilizando un puntero o una dirección calculada.
Indexado	`Add R3, (R1+R2)`	R3←R3+M[R1+R2]	A veces útil en direccionamiento de arrays—R1=base del array; R2=cantidad de índices
Directo o absoluto	`Add R1, (1001)`	R1←R1+M[1001]	A veces útil para acceder a datos estáticos; la constante que especifica la dirección puede necesitar ser grande

Modo de direcciona-miento	Ejemplo instrucción	Significado	Cuándo se usa
Indirecto o diferido de memoria	`Add R1,@(R3)`	R1←R1+M[M[R3]]	Si R3 es la dirección de un puntero *p*, entonces el modo obtiene **p*
Auto-incremento	`Add R1,(R2)+`	R1←R1+M[R2] R2←R2+*d*	Util para recorridos de arrays en un bucle. R2 apunta al principio del array; cada referencia incrementa R2 en el tamaño de un elemento, *d*.
Auto-decremento	`Add R1,-(R2)`	R2←R2-*d* R1←R1+M[R2]	El mismo uso que autoincremento. Autoincremento/decremento también puede utilizarse para realizar una pila mediante introducir y sacar (push y pop)
Escalado o Indice	`Add R1,100(R2)[R3]`	R1← R1+M[100+R2+R3*d]	Usado para acceder arrays por índice. Puede aplicarse a cualquier modo de direccionamiento básico en algunas máquinas

- Modos más utilizados



Nota útil: Para ver el código máquina de un ejecutable en Linux

objdump -s ejecutable | less

Además si compilamos con gcc con la opción -g aparecerán las lineas en C al mirar el código máquina.

3 Operaciones

• Tipos
  - Aritmética entera, lógica binaria.
  - Transferencia: Load/store o move.
  - Control: Bifurcaciones y saltos, llamada/retorno de subprograma.
  - Sistema: Llamada al S.O., gestión de niveles de ejecución
  - Coma flotante: aritmética y conversiones real-entero.
  - Decimal: aritmética y conversiones BCD
  - Tiras: transferencia, comparación, búsqueda.
  - Gráficas: Operaciones con pixels, o grupos de bits

3.1. Instrucciones de control *(i.e. saltos)*

• Tipos de cambios del control de flujo:
  - Saltos condicionales (branch) {Tex 66, Spice 75, GCC 78} (% respecto al total de saltos)
  - Saltos incondicionales (jump) {18,12,12%}
  - Llamadas/retorno a procedimiento (call/return) {16,13,10%}

• Modos de direccionamiento:
  - Relativo al PC (permite al programa ser relocalizable (ejecución independiente de la posición de memoria en que esté el programa) y además al ser tipicamente posicion cercana, requiere pocos bits para la dirección de salto)
  - Indirecto a registro o desplazamiento (salta a la dirección almacenada en un registro)

• Especificar la **condición** de salto:

a. Códigos de condición

```
SUB R1,R2,R3        ; R1=R2-R3
CMP R1,#0           ; Si R1=0, el indicador Z vale 1
BEQ eti             ; Bifurca si Z = 1
```

- Ventaja: Las comparaciones pueden eliminarse en algún caso:

```
SUB R1,R2,R3        ; R1=R2-R3, Si R1=0, el indicador Z vale 1
BEQ eti             ; Bifurca si Z=1
```

- Inconvenientes: Problemas en máquinas segmentadas.

b. Registro de uso general.

```
SUB R1,R2,R3        ; R1=R2-R3
SEQ R10,R1,#0       ; Si R1 vale 0, se deposita en R10 un 1
BNEZ R10,eti        ; Bifurca si R10<>0
```

- Ventajas: Regularidad del juego de instrucciones.
- Inconvenientes: Consumo de un registro.

c. Instrucción única de comparación y salto condicional.

```
SUB R1,R2,R3        ; R1=R2-R3
C&BEQ R1,#0,eti     ; Si R1=0, se salta a eti
```

- Ventajas: Reducción del tamaño del código.
- Inconvenientes: Puede ser demasiado trabajo para una sola instrucción aumentando los **CPI** o el ciclo de reloj.

4 Evolución de la arquitectura del juego de instrucciones

• Los primeros computadores eran simples: Pocas instrucciones y sólo uno o dos modos de direccionamiento que se ejecutaban directamente sobre el *hardware*.

• 1964: IBM lanza la familia 360, con microprogramación (i.e. La unidad de control era microprogramada en lugar de cableada)

- *hardware* relativamente simple.
- juego de instrucciones potente gracias a los microprogramas.

• Casi todos los fabricantes incorporan la microprogramación en sus diseños.

¿Por qué se introduce la microprogramación?

1. Diferencia de velocidad entre CPU y memoria es tal que enlentecer un poco la unidad de control casi no afecta nada.
2. Reducción del desnivel semántico entre ensamblador y lenguajes de alto nivel.

• El cambio (años 70)

1. Tecnología: se inventa la memoria *cache* o antememoria: la velocidad de la CPU es ahora similar a la de la memoria (que ahora ademas es RAM de semiconductores)
2. Problemática de la microprogramación:
   - Difícil depuración y mantenimiento de los programas (bajo nivel de las instrucciones, espacio de la memoria de control limitado,...).
3. Medidas sobre la ejecución de programas reales.
   - Los compiladores utilizan un subconjunto del juego de instrucciones.
   - Las instrucciones complejas o demasiado específicas raramente se utilizan.

• Maquinas RISC.

- RISC: (Reduced Instruction Set Computers) frente a las CISC (Complex ...)
  1. Instrucciones simples (que son las más empleadas) ejecutadas directamente en hardware.
  2. Compilador optimizante para cubrir el desnivel semántico.

• Principios de diseño RISC

- Analizar las operaciones reales y delimitar las operaciones clave.
- Diseñar la ruta de datos óptima para estas operaciones clave.
- Diseñar el juego de instrucciones incluyendo sólo las que implementan las operaciones clave en la ruta de datos.
- Añadir nuevas instrucciones, sólo si no ralentizan la máquina.
- Repetir el proceso para los demás recursos del computador.
- Al haber simplificado la unidad de control, se puede hacer cableada.

5 Ejemplo de juego de instrucciones: el DLX

- DLX:"DeLuXe". Es un procesador docente.
- **Instrucciones sencillas y rápidas, con las funciones sofisticadas en el *software*.**
- Características
  - Carga/almacenamiento.
  - Segmentación eficiente.
  - Formato fácilmente decodificable.
  - Buen soporte al compilador.
- Registros y organización de la memoria
  - **32 registros de uso general de 32 bits (R0 a R31).**
  - R0 es siempre 0 (sólo se puede leer)
  - 32/16 registros de simple (F0 a F31)/doble precisión (F0 a F30, sólo los pares).
  - Memoria:
    - Direccionable al *byte*
    - Convenio *Big Endian*.
    - Accesible únicamente por medio de instrucciones *load/store*.
    - Direcciones de 32 bits.
    - Acceso a *byte* y *halfword (2 bytes)*.
    - Accesos siempre alineados. (i.e. La dirección de acceso debe ser múltiplo del tamaño accedido)

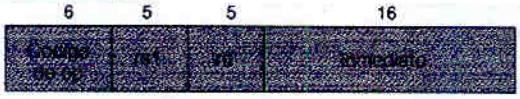
5.1 Tipos de instrucciones del DLX

- **Cargas/almacenamientos.** *(ejs: lb R3 "load byte" ; lbu "load byte unsigned")*
  - Sobre cualquier registro de uso general (excepto R0) o coma flotante.
  - Sólo modo de direccionamiento desplazamiento, con 16 bits. (usar R0 para 0)
  - Acceso a *byte* y *halfword*, **con y sin extensión de signo**.

    ejemplo: lb R3, a(R0) [byte en dir: a+0 -> R3] si a=-1=0xFF, entonces R3=0xFF FF FF FF
- **Aritméticas.** *(ejs: sub R1,R2,R3 "R2-R3->R1")*
  - Instrucciones registro-registro, 3 direcciones.
  - Operaciones aritméticas sencillas (incluye MUL y DIV), lógicas, desplazamientos y de comparación.
  - También permite el modo inmediato para todas, 16 bits (instrucción LHI para cargas de constantes mayores de 16 bits).
  - Las instrucciones de comparación depositan su resultado en un registro de uso general.
- **Control.** *(ejs: bz R2,#valor "salto a (PC + valor) si R2=0")*
  - Salto incondicional: Relativo al PC (26 bits) e indirecto a registro.
  - Salto incondicional con enlace (guarda PC en un registro): Relativo al PC e indirecto a registro.
  - Salto condicional: Sólo = 0 y ≠ 0 : Relativo al PC (16 bits)
- **Coma flotante.**
  - Suma, resta, multiplicación y división, en simple y doble precisión.
  - Transferencia entre registros de coma flotante.
  - Transferencia entre registros de coma flotante y de uso general.
  - Conversión entero⇔flotante.
  - Comparación

5.2 DLX: Formato de instrucciones

Instrucción tipo-I



Codifica: Carga y almacenamiento de bytes, palabras, medias palabras
Todos inmediatos (rd ← rs1 op inmediato)

Instrucciones de salto condicional (rs1 es registro, rd no usado)
Bifurcación a registro, Bifurcación y enlace a registro
(rd 0 0, rs = destino, inmediato = 0)

Instrucción tipo-R

Operaciones ALU registro-registro: rd ← rs1 func rs2
Función codifica la operación del camino de datos: Add, Sub...
Registros especiales de lectura/escritura y transferencias

Instrucción tipo-J

Bifurcación y bifurcación y enlace
Trap y RFE

DISCA-23

5.3 Instrucciones acceso a memoria

Instrucción ejemplo	Nombre de la instrucción	Significado
LW R1,30(R2)	Cargar palabra	$R1 \leftarrow_{32} M[30+R2]$
LW R1,1000(R0)	Cargar palabra	$R1 \leftarrow_{32} M[1000+0]$
LB R1,40(R3)	Cargar byte	$R1 \leftarrow_{32} (M[40+R3]_0)^{24} \# \# M[40+R3]$
LBU R1,40(R3)	Cargar byte sin signo	$R1 \leftarrow_{32} 0^{24} \# \# M[40+R3]$
LH R1,40(R3)	Cargar media palabra	$R1 \leftarrow_{32} (M[40+R3]_0)^{16} \# \# M[40+R3] \# \# M[41+R3]$
LF F0,50(R3)	Cargar flotante	$F0 \leftarrow_{32} M[50+R3]$
LD F0,50(R2)	Cargar doble	$F0 \# \# F1 \leftarrow_{64} M[50+R2]$
SW 500(R4),R3	Almacenar palabra	$M[500+R4] \leftarrow_{32} R3$
SF 40(R3),F0	Almacenar flotante	$M[40+R3] \leftarrow_{32} F0$
SD 40(R3),F0	Almacenar doble	$M[40+R3] \leftarrow_{32} F0$; $M[44+R3] \leftarrow_{32} F1$
SH 502(R2),R3	Almacenar media	$M[502+R2] \leftarrow_{16} R3_{16..31}$
SB 41(R3),R2	Almacenar byte	$M[41+R3] \leftarrow_{8} R2_{24..31}$

DISCA-24

5.3 Instrucciones de salto

Instrucción ejemplo	Nombre de la instrucción	Significado
J nombre	Bifurcación	PC←nombre; ((PC+4)−2^{25}) ≤ nombre < ((PC+4)+2^{25})
JAL nombre	Bifurcación y enlace	R31←PC+4; PC←nombre; ((PC+4)−2^{25}) ≤ nombre < ((PC+4)+2^{25})
JALR R2	Bifurcación y enlaza registro	R31←PC+4; PC←R2
JR R3	Bifurcación a registro	PC←R3
BEQZ R4,nombre	Salta igual a cero	if (R4==0) PC←nombre; ((PC+4)−2^{15}) ≤ nombre < ((PC+4)+2^{15})
BNEZ R4,nombre	Salta no igual a cero	if (R4!=0) PC←nombre; ((PC+4)−2^{15}) ≤ nombre < ((PC+4)+2^{15})

5.4 Lista Instrucciones (1)

Tipo de instrucción/Código de op.	Significado de la instrucción
Transferencias de datos	**Transfiere datos entre registros y memoria, o entre registros enteros y FP o registros especiales; sólo el modo de direccionamiento de memoria es un desplazamiento de 16 bits + contenido de un GPR**
LB,LBU,SB	Carga byte, carga byte sin signo, almacena byte
LH,LHU,SH	Carga media palabra, carga media palabra sin signo, almacena media palabra
LW,SW	Carga palabra, almacena palabra (a/desde registros enteros)
LF,LD,SF,SD	Carga punto flotante SP, carga punto flotante DP, almacena punto flotante SP, almacena punto flotante DP
MOVI2S, MOVS2I	Transfiere desde/a GPR a/desde un registro especial
MOVF, MOVD	Copia un registro de punto flotante o un par en DP en otro registro o par
MOVFP2I,MOVI2FP	Transfiere 32 bits desde/a registros FP a/desde registros enteros

Aritmética/Lógica	Operaciones sobre datos enteros o lógicos en GPR/s; la aritmética con signo causa un trap en caso de desbordamiento
ADD,ADDI,ADDU,ADDUI	Suma, suma inmediato (todos los inmediatos son de 16 bits); con signo y sin signo
SUB,SUBI,SUBU,SUBUI	Resta, resta inmediato; con signo y sin signo
MULT,MULTU,DIV,DIVU	Multiplica y divide, con signo y sin signo; los operandos deben estar en registros de punto flotante; todas las operaciones tienen valores de 32 bits
AND,ANDI	And, and inmediato
OR,ORI,XOR,XORI	Or, or inmediato, or exclusiva, or exclusiva inmediata
LHI	Carga inmediato superior: carga la mitad superior de registro con inmediato
SLL, SRL, SRA, SLLI, SRLI, SRAI	Desplazamientos: ambos inmediatos (S__I) y forma variable (S__); los desplazamientos son desplazamientos lógicos a la izquierda, lógicos a la derecha, aritméticos a la derecha (aritmético = con extensión signo) (con registro)
S___,S___I	Inicialización condicional: «__» puede ser LT, GT, LE, GE, EQ, NE

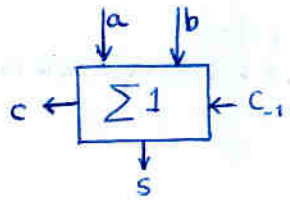
Control	Saltos y bifurcaciones condicionales; relativos al PC o mediante registros
BEQZ,BNEZ	Salto GPR igual/no igual a cero; desplazamiento de 16 bits desde PC+4
BFPT,BFPF	Test de bit de comparación en el registro de estado FP y salto; desplazamiento de 16 bits desde PC+4
J, JR	Bifurcaciones: desplazamiento de 26 bits desde PC (J) o destino en registro (JR)
JAL, JALR	Bifurcación y enlace: guarda PC+4 en R31, el destino es relativo al PC (JAL) o un registro (JALR)
TRAP	Transfiere a sistema operativo a una dirección vectorizada; ver Capítulo 5
RFE	Volver al código del usuario desde una excepción; restaurar modo de usuario; ver Capítulo 5

Punto flotante	Operaciones en punto flotante en formatos DP y SP
ADDD,ADDF	Suma números DP, SP
SUBD,SUBF	Resta números DP, SP
MULTD,MULTF	Multiplica punto flotante DP, SP
DIVD,DIVF	Divide punto flotante DP, SP
CVTF2D, CVTF2I, CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convierte instrucciones: CVTx2y convierte de tipo x a y, donde x e y pueden ser uno de: I (Entero), D (Doble precisión), o F (Simple precisión). Ambos operandos están en los registros FP
___D,___F	Compara DP y SP: «__» puede ser LT, GT, LE, EQ, NE; pone bit de comparación en registro de estado FP

Tema 5. Unidades Segmentadas

ejemplo: sumador de K bits

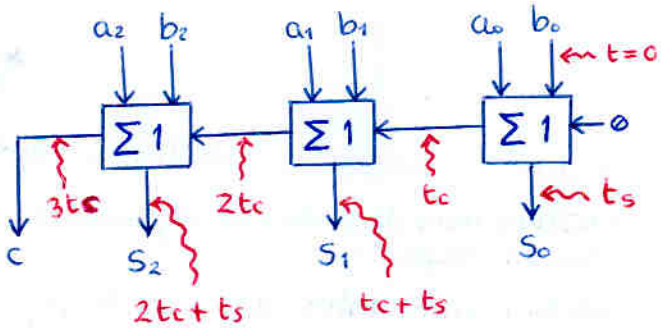
Sumador de 1 bit



$s = a \oplus b \oplus c$ — tiempo de suma $t_s = 3t_p$

$c = (ab) + (aC_{-1}) + (bC_{-1})$ — tiempo de carry $t_c = 2t_p$

sumador de K bits sin segmentar (ej K=3)



En hacer n sumas se tarda

$n \cdot T$

Unidad sin segmentar tarda un tiempo T en realizar la operación.

Fijarse en la salida más lenta

$$T = (K-1)t_c + t_s$$

En cada instante se están utilizando un sumador y el anterior acabando la suma.

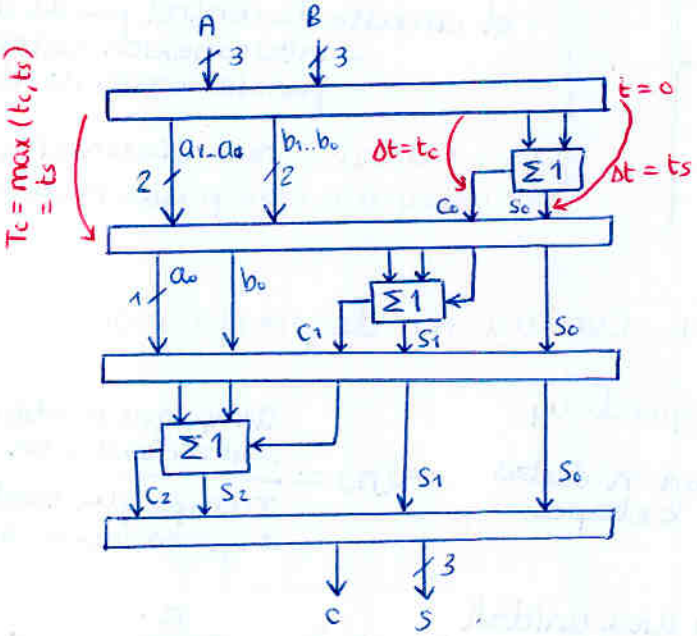
Se desaprovecha el resto

sumador segmentado de K bits (ej K=3)

- dividimos en etapas (K etapas)
- usamos registros entre etapas
- Podemos ir empezando una nueva suma en la 1ª etapa según las siguientes van avanzando
- se avanza una etapa cada tiempo de ciclo (T_c) (que será el load de los registros)
- El tiempo de ciclo vendrá dado por la etapa más lenta

$$T_c = t_s \quad (+ t_r)$$

↑ tiempo de registro



- En hacer una suma se tarda en total $K \cdot T_c$ (más que el no segmentado)
- En hacer n sumas se tarda

$$T_{n\,sumas} = (n + (k-1))\,T_c$$

menos que el no segmentado si n↑

↑ 1ª etapa n veces

↑ que la ultima suma acabe las K-1 etapas que le quedan

2. Conceptos

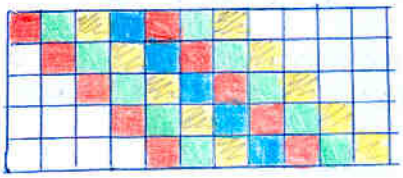
- Tiempo de ciclo: periodo de reloj y registros
  será igual al tiempo de etapa máximo
- Registros: necesitamos almacenar resultados entre etapas
- Tabla de reserva

se ve de forma gráfica la utilización de cada etapa

si una etapa se usa durante más de un ciclo, limitaremos la cadencia de entrada

3. Clasificación

- Unidad segmentada lineal
  todas las etapas duran un ciclo



- Unidad segmentada no lineal ej
  - utiliza mas de una vez alguna(s) de sus etapas
  - no es posible introducir un dato en cada ciclo de reloj (iniciación) ya que de lo contrario habría colisión

- Unidad monofunción: sólo es capaz de realizar una función
- Unidad multifunción:
  - el circuito de control puede alterar:
    - interconexión entre etapas
    - funcionamiento de las mismas
  - → estática: reconfiguración sólo es posible con unidad vacía
  - → dinámica: se puede reconfigurar para cada iniciación

4. Evaluación de prestaciones

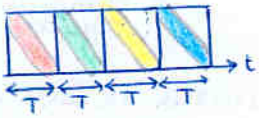
- Speed-up

con n datos y k etapas

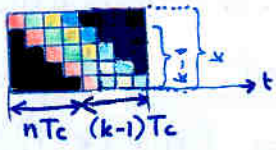
$$S(n) = \frac{\text{Tiempo que tardaría la unidad no segmentada en hacer n operaciones}}{\text{Tiempo que tarda la unidad segmentada en hacer n operaciones}}$$

en una unidad lineal:

$$S(n) = \frac{n \cdot T}{(n+(k-1))\,T_c}$$



$n \cdot T$



$(n+(k-1))\,T_c$

$$S(\infty) = \frac{T}{T_c} \simeq K$$

$$S \leq K$$

máximo speedup al que podemos aspirar en unidad segmentada es igual al nº de etapas

· Eficiencia

Para n operaciones $\quad \eta(n) = \dfrac{\text{nº huecos marcados en rejilla} \cdot T_c}{\text{nº huecos totales en rejilla} \cdot T_c}$

en una unidad lineal: $\quad \eta(n) = \dfrac{n \cdot (T_c \cdot k)}{k \cdot (n + (k-1)) T_c} \xrightarrow{n \to \infty}$ T ← tiempo que tardaba la unidad no segmentada

$$\eta(n) \overset{n \uparrow}{\simeq} \frac{S(n)}{k}$$

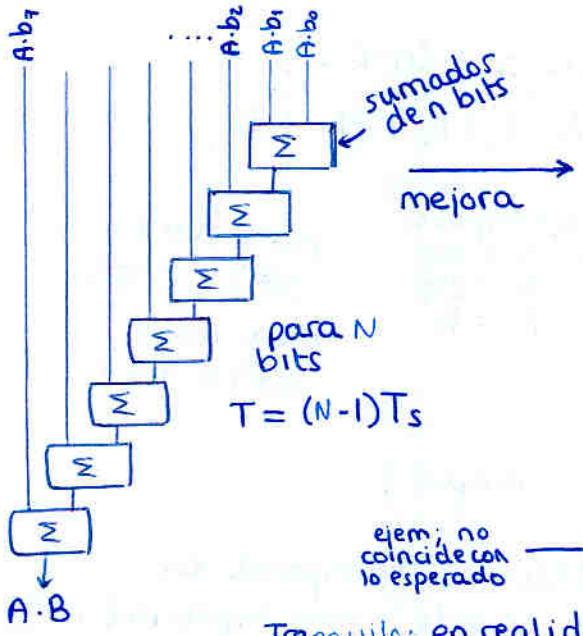
· Productividad

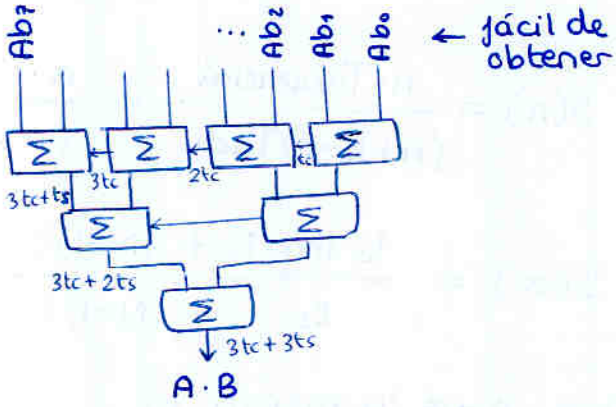
Para n operaciones $\quad W(n) = \dfrac{\text{nº operaciones}}{T \text{ total}}$

en una unidad lineal $\quad W(n) = \dfrac{n}{(n+k-1)T_c} \rightarrow$

$$W(n) = \frac{S(n)}{T}$$

ejemplo: multiplicador CPA





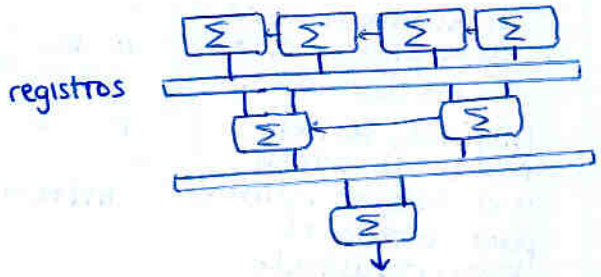
· Usamos el mismo nº de sumadores

ejem; no coincide con lo esperado → $T = \log_2 N \, t_s + (\frac{N}{2} - 1) t_c$ sin segmentar.

Tranquilo: en realidad las conexiones son más complejas y los sumadores van siendo cada vez de más bits, pero nos centraremos en la segmentación suponiendo que el multiplicador es así de sencillo

i.e. en realidad esto no es un multiplicador, pero se le parece y supondremos que lo es.

- Segmentamos en $k = \log_2 N$ etapas



Nuestro objetivo es calcular el factor de mejora para n operaciones de multiplicación $S(n)$

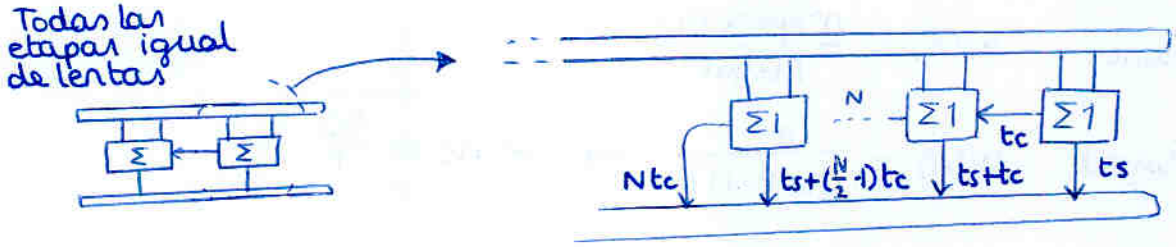
$$S(n) = \frac{n \cdot T_{secuencial}}{(n+k-1)\,T_{ciclo}}$$

$T_{secuencial} = T = t_s \log_2 N + (\frac{N}{2}-1)t_c$

$k = n^o \text{ etapas} = \log_2 N$

$T_{ciclo} = t_s + (N-1)t_c$

Hay que hallar T_{ciclo}; duración de la etapa más larga
i.e. duración de la señal más lenta en la etapa más larga



$$T_{ciclo} = t_s + (N-1)t_c$$

$$S(n) = \frac{n \cdot T_{secuencial}}{(n+k-1)\,T_{ciclo}} = \frac{n \cdot (t_s \log_2 N + (\frac{N}{2}-1)t_c)}{(n+\log_2 N - 1)(t_s + (\frac{N}{2}-1)t_c)}$$

$$S(\infty) = \frac{t_s \log_2 N + (\frac{N}{2}-1)t_c}{t_s + (\frac{N}{2}-1)t_c}$$

ejemplo:
$t_c = 3tp$
$t_s = 2tp$
$N = 16$

para $n=5$
$S(5) = 0'80$

para $n=\infty$
$S(\infty) = 1'27$

¡Para $n \to \infty$ la mejora es paquísima!

¡Encima para n's bajos se empeora! ¿Porqué?

Hemos tratado de introducir paralelismo temporal en un diseño que tenía un elevado paralelismo espacial.
Eso no suele dar buen resultado.

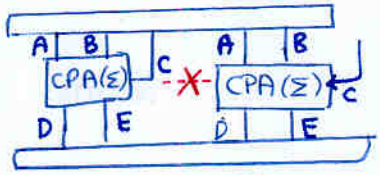
Otra posible segmentación habría sido segmentar a su vez cada sumador en N etapas

$K_{total} = N \cdot \log_2 N$

$T_{ciclo} = t_s$

ejemplo: multiplicador CPA

En lugar de ir pasando los acarreos al sumador contiguo, los va sacando a la salida.
El último no tiene que esperar a los de su derecha.
Al final, con el resultado 'extraño' y el resto, se suman para obtener el buen resultado



A+B+C = 2D+E

con CPA se logra

$$S(n) = \frac{n(t_s \log_2 N + (N-1)t_c)}{(n+\log_2 N - 1)\,t_s}$$

Tema 6. Unidades de Instrucción Segmentadas

1. La ruta de datos del DLX

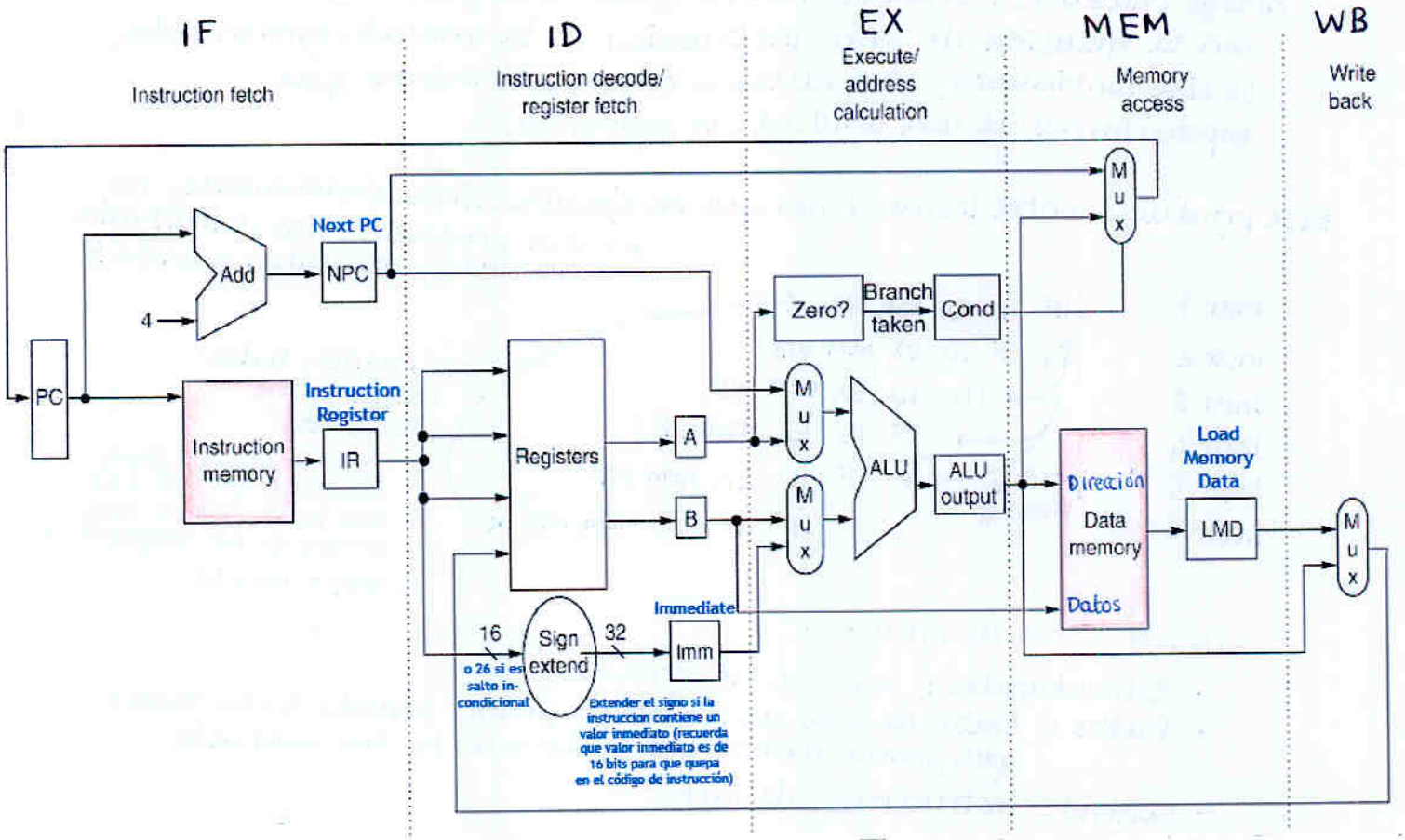
Ruta de datos sin segmentar; hay hasta 5 ciclos por instrucción

- Instrucción NOP: 2 ciclos
- Instrucción LOAD: 5 ciclos
- TODAS las demás: 4 ciclos

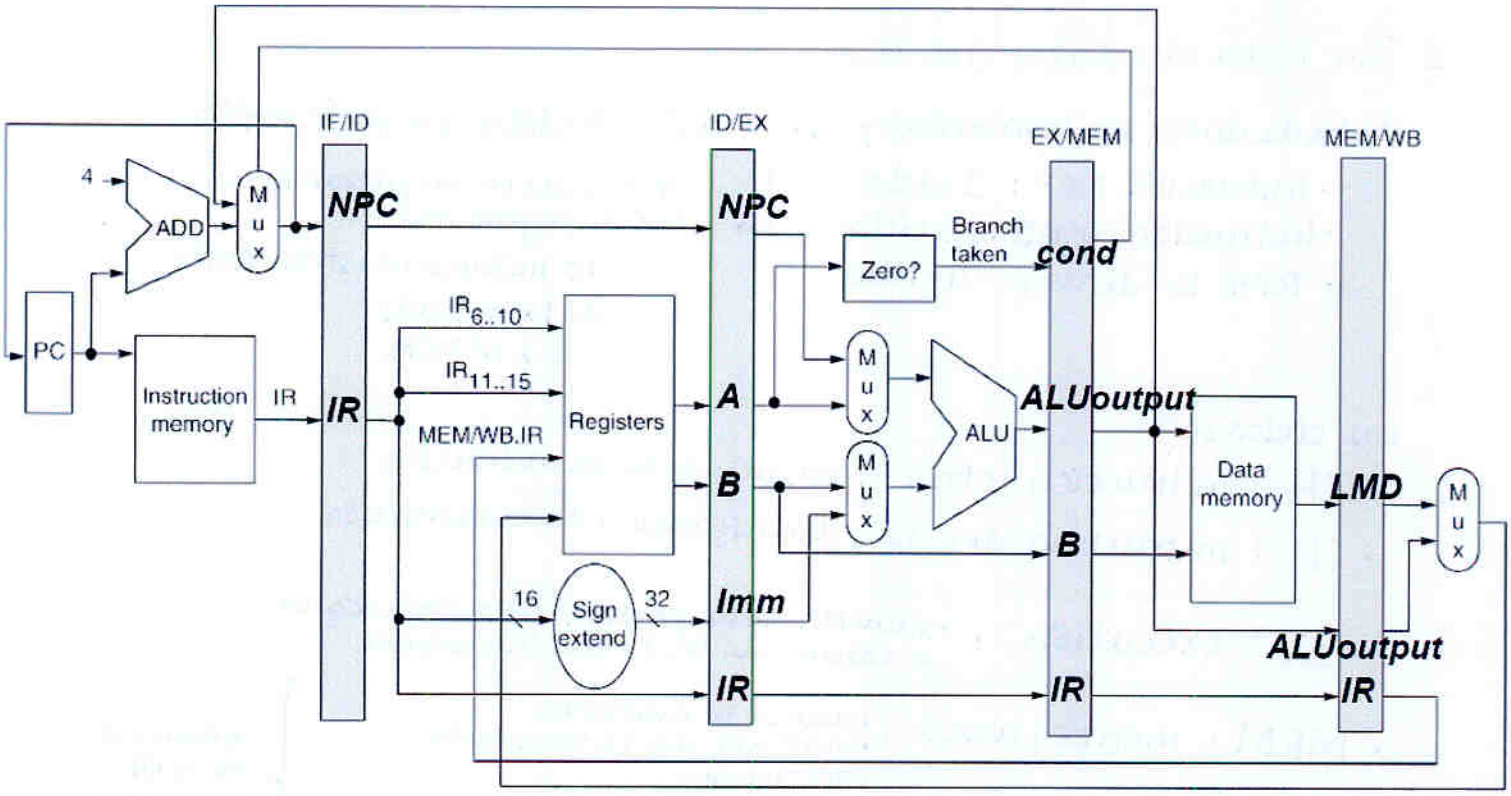
⇒ nº de ciclos en un programa con DLX sin segmentar
= 4 × nº total instrucciones
+ 1 × nº loads
− 2 × nº NOPs

Los ciclos son:

• IF : instruction fetch : búsqueda de la instrucción	
• ID : instruction decode : decodificación de la instrucción lectura de los registros	
• EX : execution : • ejecución de la operación aritmético lógica o cálculo de la dirección efectiva	
• MEM : memory access : • acceso a memoria • escritura del PC tras salto (operaciones aritmético lógicas no pasan por este ciclo)	} únicamente los LOAD pasan por ambos ciclos
• WB : write-back : • escritura en registros (store y saltos no pasan por este ciclo)	



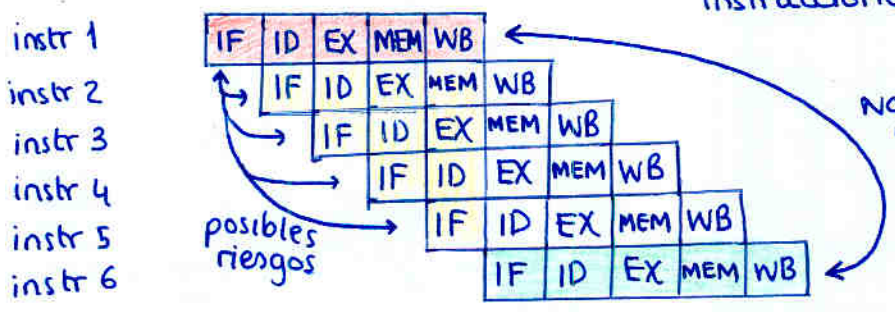
2. Segmentación del ciclo de instrucción



- Riesgos

· Riesgo (hazard): situación en la cual no se puede continuar con la ejecución de una instrucción en la unidad segmentada (o si se continuara, el resultado no sería el mismo que esperaríamos de una unidad sin segmentar)

Pueden producirse entre instrucciones que se ejecutan simultaneamente, no pueden producirse (en el DLX) entre instrucciones separadas mas de 5



NO PUEDEN haber riesgos entre estas dos

en realidad luego veremos que en DLX sólo hay riesgos entre instrucciones separadas cuatro o menos

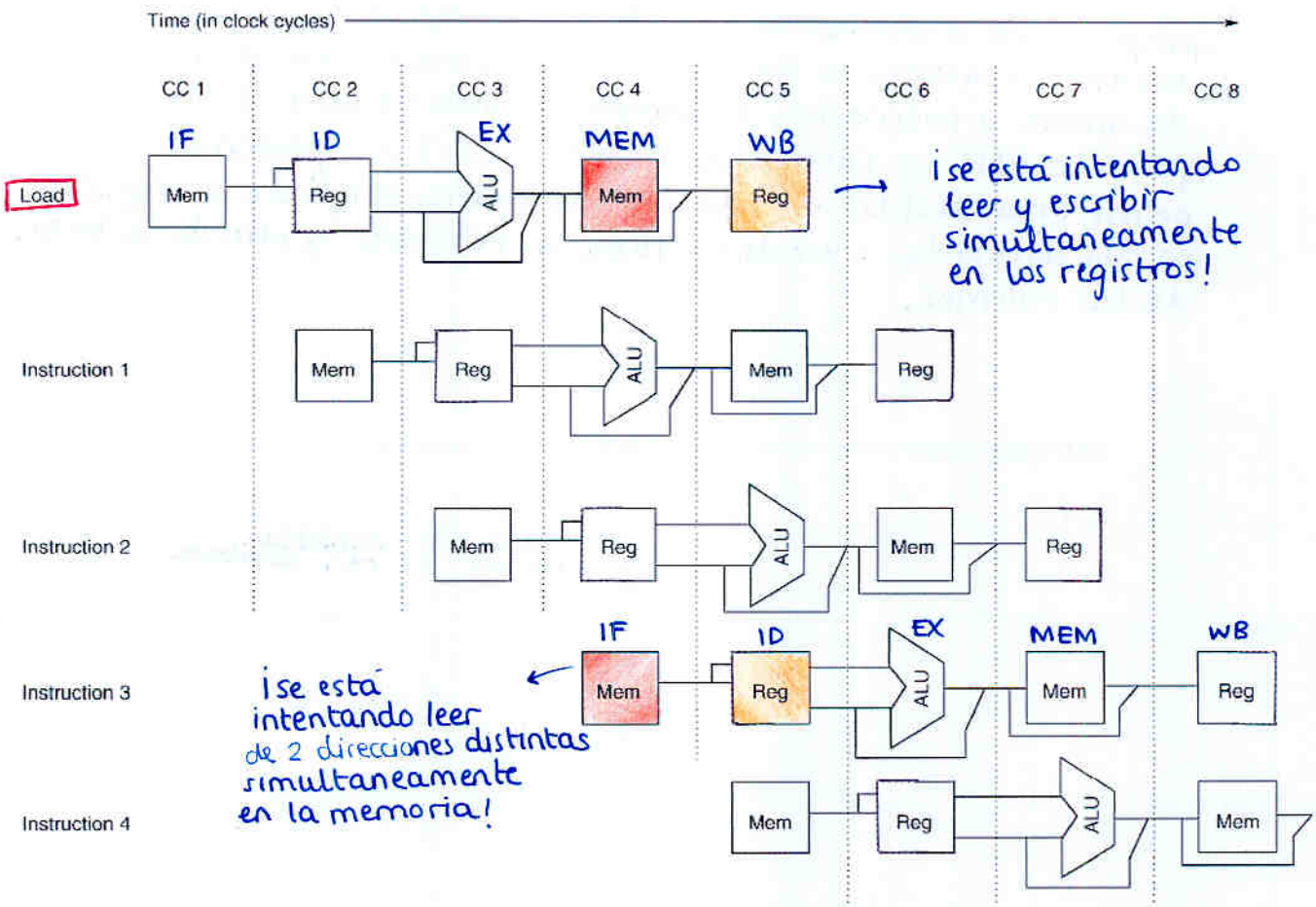
Clasificación de riesgos:

- Estructurales: uso del hardware
- Datos: tratar de usar un resultado previo cuando la instruccion que genera dicho resultado aún no ha acabado
- Control: instrucciones de salto

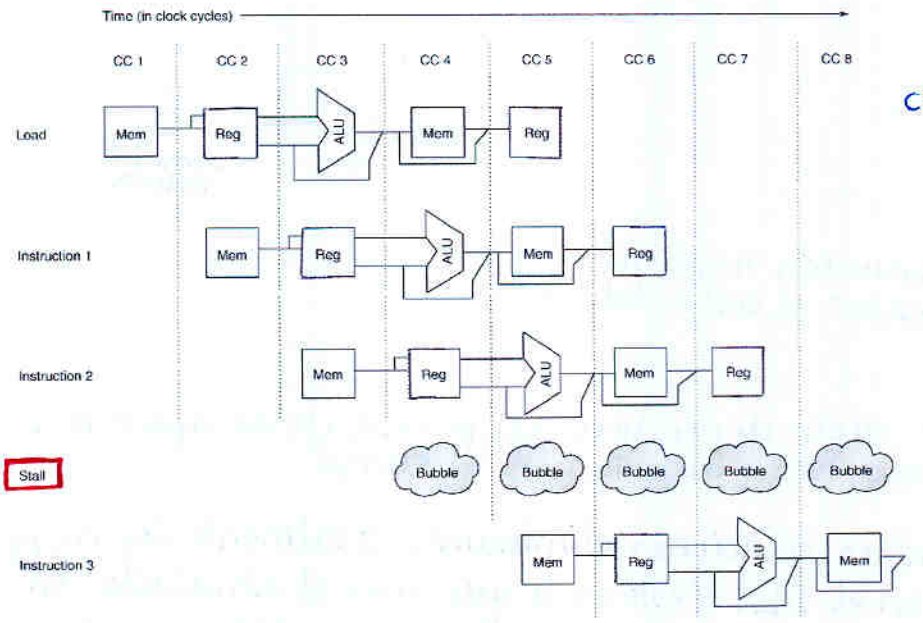
3. Riesgos estructurales

El hardware no permite todas las combinaciones posibles de las instrucciones presentes en la unidad

ejemplo: 2 problemas estructurales:



- Posible solución: Retrasar las operaciones que causen conflicto
  stall ⇒ pérdida de prestaciones (aumentarán los CPI)



$$CPI = \frac{n^o \text{ ciclos}}{n^o \text{ instrucciones}} = \frac{n + 4 + n^o \text{loads}}{n}$$

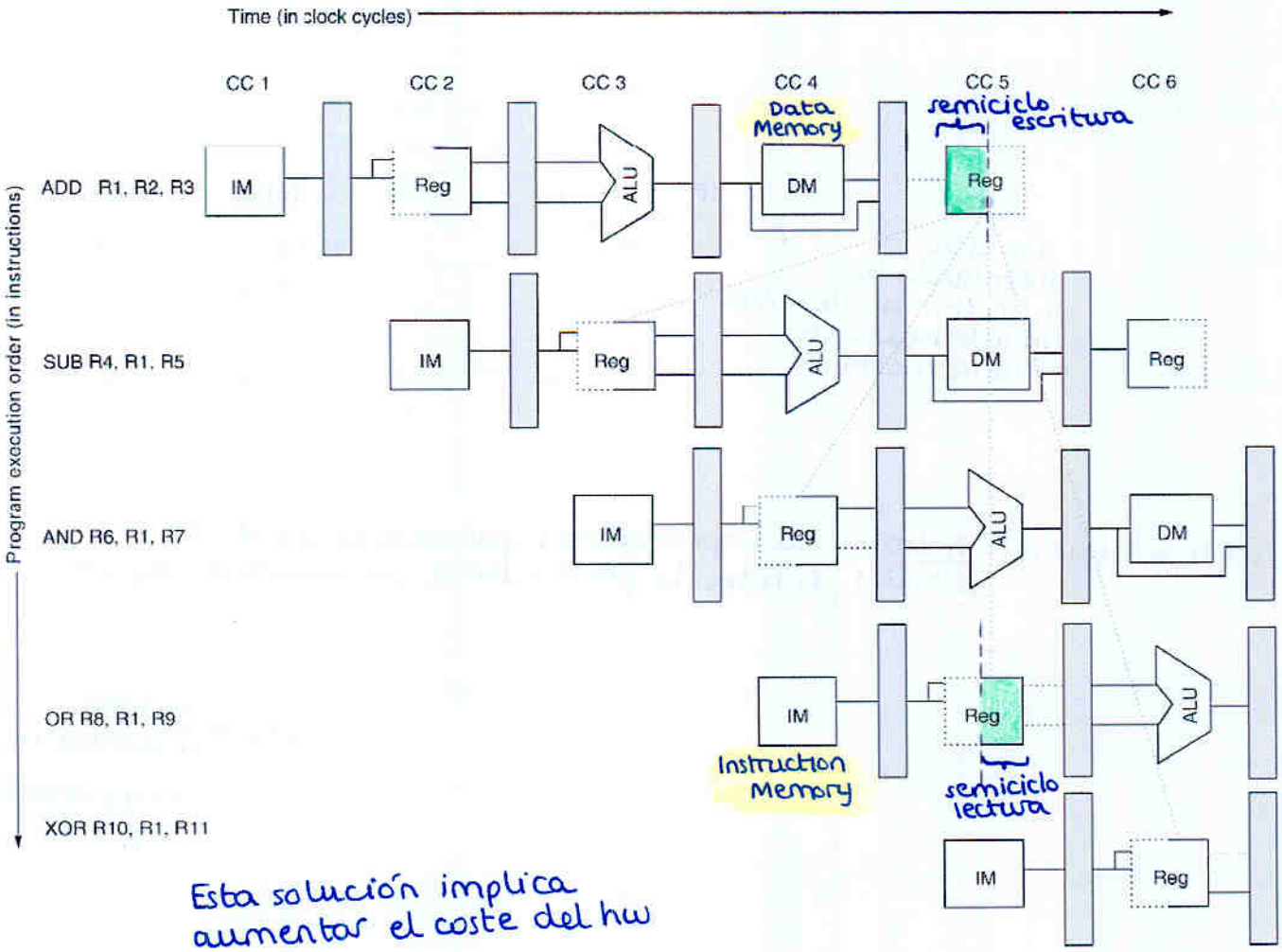
Esta solución presenta bajo coste de implementación

Posible solución:

Replicar el recurso hardware para que sea posible esa combinación

- ejemplo: arquitectura Hardvard (en lugar de von Neumann) que tiene antememorias de instrucciones y datos separadas

- ejemplo: para solucionar el conflicto con los registros; ya que el acceso a registros es muy rápido comparado con el tiempo de acceso a memoria, el tiempo de ciclo vendrá fijado por este último (fase más lenta), y por tanto podemos permitimos dividir el ciclo donde había conflicto con registros en dos semiciclos: uno de escritura en registros y otro de lectura de los mismos.



Esta solución implica aumentar el coste del hw

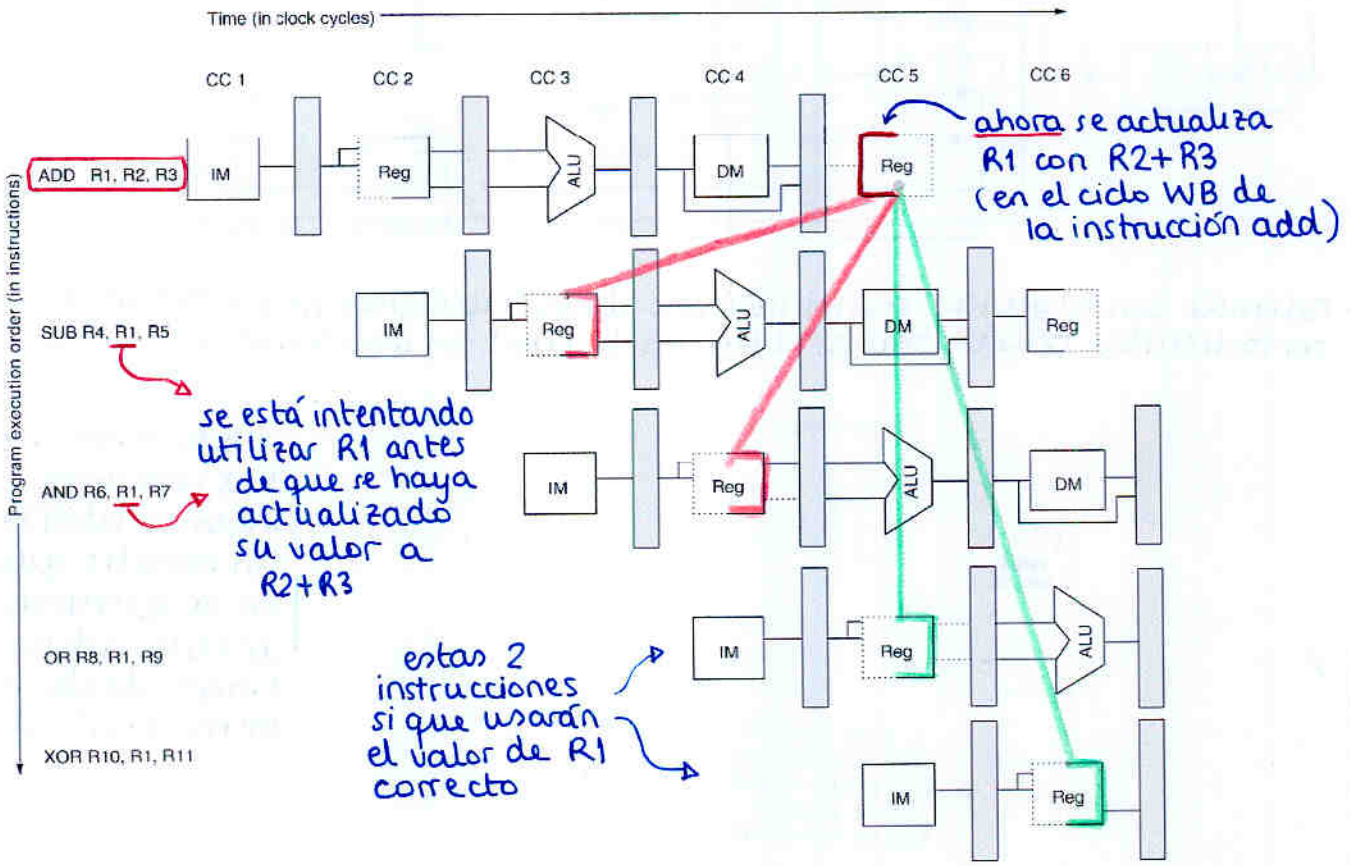
La solución a elegir dependerá del porcentaje de aparición de las combinaciones que originan el riesgo

Si aplicamos ambas soluciones eliminamos totalmente los riesgos estructurales en el DLX (esto es lo que hace el simulador en prácticas, salvo con un caso especial usando DS1 que como veremos puede introducir riesgos estructurales en casos puntuales)

4. Riesgos de datos

Se intenta utilizar un resultado de una instrucción anterior que aún no ha acabado

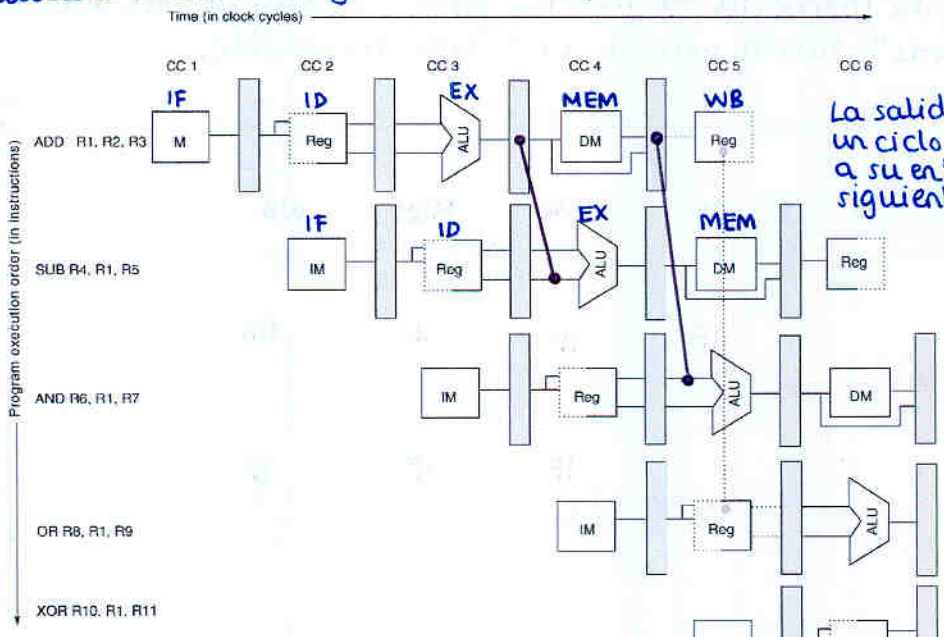
ejemplo:



1ª solución: insertar CICLOS DE ESPERA

2ª solución: cortocircuitos ⇒ realimentación

Hacemos uso de que, a pesar de que un registro se actualiza en fase WB, el resultado no obstante ya estaba disponible desde la fase EXE

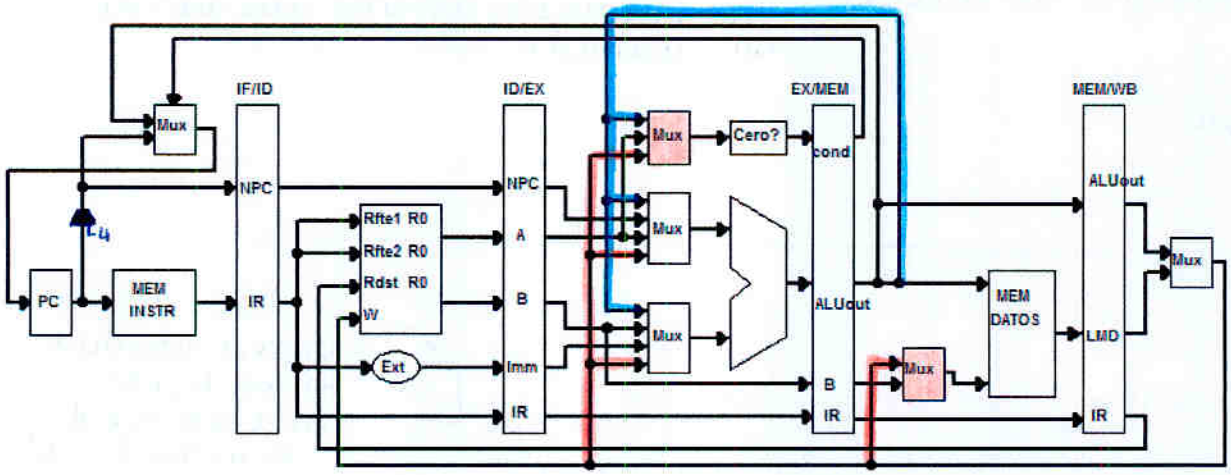


La salida de la ALU en un ciclo se realimenta a su entrada en el siguiente ciclo

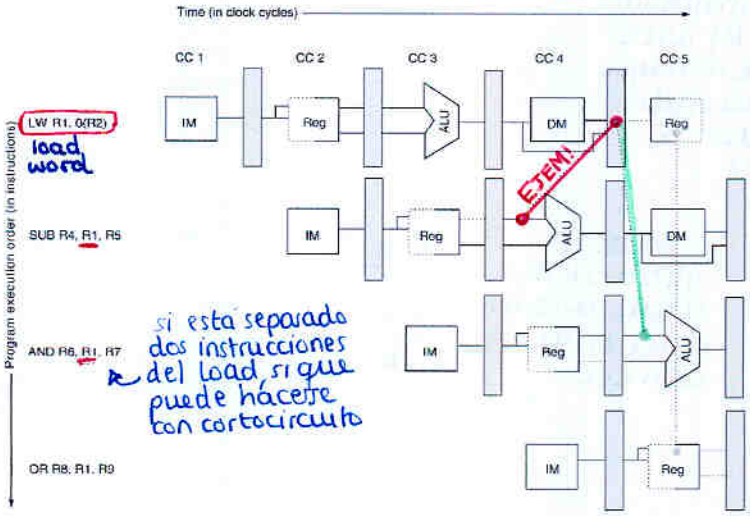
No hace falta que tenga el dato en ID, sabe que cuando empiece EX ya se lo pasarán

Así solucionamos CASI todos los riesgos de datos (excepción: load, ver detrás)

Problemas :- el hardware se complica mucho:



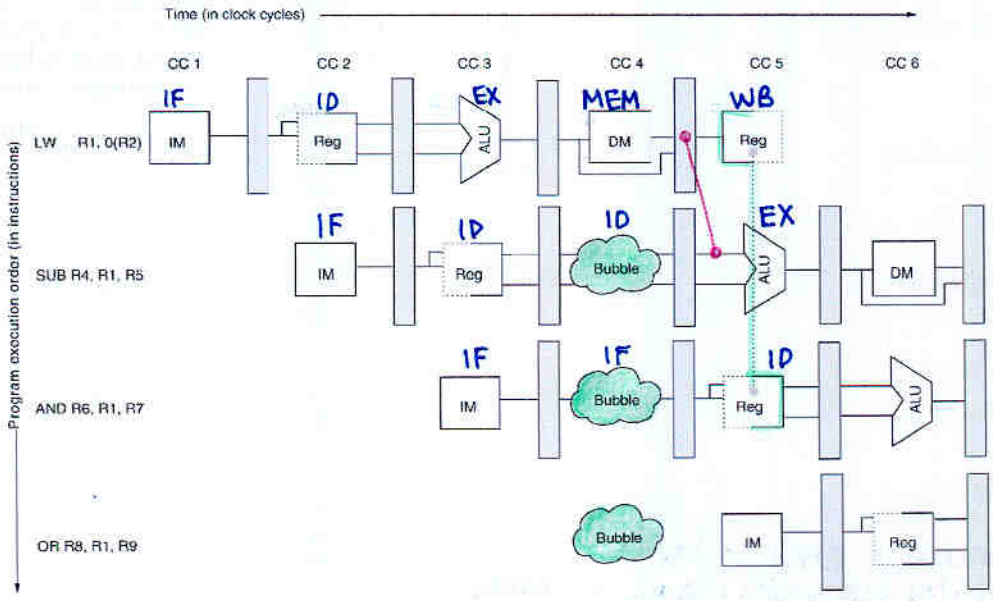
- Además con algunas combinaciones de instrucciones necesitaríamos cortocircuitos hacia atrás en el tiempo, lo cual es imposible:



si está separado dos instrucciones del load, sí que puede hacerse con cortocircuito

Esto ocurre en el DLX cuando se intenta utilizar un registro que en la instrucción anterior debía leerse desde la memoria. (load)

La solución a este ultimo problema es introducir CICLOS DE ESPERA: parar la ejecución de una instrucción la cual necesita que un dato aún sea leído de memoria → (implica parar todas las instrucciones que van "detrás") hasta que el dato esté disponible.



se aumentan los CPI

por tanto:

Con cortocircuitos:

Hay un ciclo de espera siempre que haya una operación aritmética sobre un registro al cual se le hace un LOAD en la instrucción anterior

- **Tecnicas de compilación**

La inserción de ciclos de espera puede evitarse o reducirse el número:
- Insertando instrucciones NOP en el ensamblador
- Que el programador (o el compilador, previo a la compilación estandar) reorganice las instrucciones para garantizar que no se lea el registro cargado despues de una instrucción load.

ejercicio $a = b + c$, $d = e - f$ — Analizar los siguientes códigos si se utiliza cortocircuito + ciclos de espera

Puesto que se utiliza la tecnica de cortocircuitos + ciclos de espera, sabemos que está garantizada la compatibilidad binaria con el caso sin segmentar (i.e. el resultado final de un código será el mismo que si la unidad fuera no segmentada). Por lo tanto para analizar el resultado final lo haremos como si fuera el caso no segmentado.

a, b, c, d, e, f : direcciones de memoria
ra, rb, rc, rd, re, rf : registros

código 1

```
lw   rb, b
lw   rc, c
add  ra, rb, rc
sw   a, ra
lw   re, e
lw   rf, f
sub  rd, re, rf
sw   d, rd
```

- `lw rc, c` / `add ra, rb, rc`: se intenta usar rc, que es cargado en la instrucción anterior ⇒ ciclo de espera
- `sw a, ra`: cortocircuito (luego puedo hacer cortocircuito)
- `lw rf, f` / `sub rd, re, rf`: ciclo de espera + cortocircuito
- `sw d, rd`: cortocircuito

2 ciclos de espera (4 cortocircuitos)

resultado final:
- $rb \leftarrow b$
- $rc \leftarrow c$
- $ra \leftarrow rb + rc = b + c$
- $a \leftarrow ra = b + c$
- $re \leftarrow e$
- $rf \leftarrow f$
- $rd \leftarrow re - rf = e - f$
- $d \leftarrow rd = e - f$

resultado: $a = b + c$, $d = e - f$

↳ nota: una vez implementado el hw de cortocircuitos, tener más o menos de ellos no afecta para nada, pero lo cuento simplemente para que se entienda lo que ocurre

código 2

```
lw   rb, b
lw   rc, c
add  ra, rb, rc
lw   re, e
lw   rf, f
sub  rd, re, rf
sw   a, ra
sw   d, rd
```

- `lw rb, b` / `lw rc, c` / `add ra, rb, rc`: ciclo de espera + cortocircuito
- `lw re, e` / `lw rf, f` / `sub rd, re, rf`: ciclo de espera + cortocircuito
- `sw a, ra`
- `sw d, rd`: cortocircuito

2 ciclos de espera (3 cortocircuitos)

resultado
- $a = b + c$
- $d = e - f$

código reorganizado

```
lw rb, b
lw rc, c
lw re, e
add ra, rb, rc
lw rf, f
sw a, ra
sub rd, re, rf
sw d, rd
```

- add ra, rb, rc ← ya no hace falta ciclo de espera! (aunque sí c.c. porque hace 2 ciclos se modificó rc)
- sw a, ra → cortocircuito (entre MEM/WB.ALUout → ALUin)
- sub rd, re, rf ← no hace falta ciclo de espera
- sw d, rd ← sí c.c. por rf
- sw d, rd ← sí c.c

rb ← b
rc ← c
re ← e
ra ← rb+rc = b+c
rf ← f
a ← ra = b+c
rd ← re − rf = e − f
d ← rd = e − f

a = b+c
d = e − f

0 ciclos de espera ! ~ Este es el mejor código
(4 cortocircuitos)

¡ El resultado es el mismo !

Ciclos de espera en el ultimo código si no hubieran cortocircuitos

lw rb, b	IF	ID	EX	MEM	WB										
lw rc, c		IF	ID	EX	MEM	WB									
lw re, e			IF	ID	EX	MEM	WB								
add ra, rb, rc				IF	ID	(ID)	EX	MEM	WB						
lw rf, f					IF	(IF)	ID	EX	MEM	WB					
sw a, ra							IF	ID	(ID)	EX	MEM	WB			
sub rd, re, rf								IF	(IF)	ID	EX	MEM	WB		
sw d, rd										IF	ID	(ID)	(ID)		...

- Primer semiciclo: el WB guarda valor en RC
- Segundo semiciclo: la operación Add puede leer el registro RC

↑ ciclo de espera

↑ ciclo de espera

↑ 2 ciclos de espera

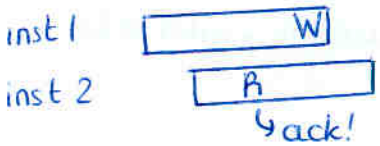
En total 4 ciclos de espera, por no haber usado cortocircuitos

nota: Los últimos dos ciclos de espera se hubieran evitado con un único cortocircuito, pero el segundo ciclo de espera hubiera requerido 2 cortocircuitos para no existir

curiosamente este ciclo de espera ha servido también para que sub rd, re, rf se haya esperado a que lw rf, f acabe

Otros riesgos de datos (que no ocurren en el DLX)

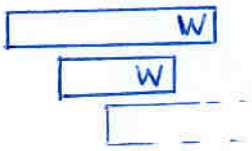
- **Read - After - Write**



Para que el resultado sea correcto, hay que "read after write"
"leer después de escribir"

→ Éste es el único que ocurre en el DLX

- **Write - After - Write**



se escribe en orden distinto al correcto (peligroso si se estaba escribiendo en el mismo registro)

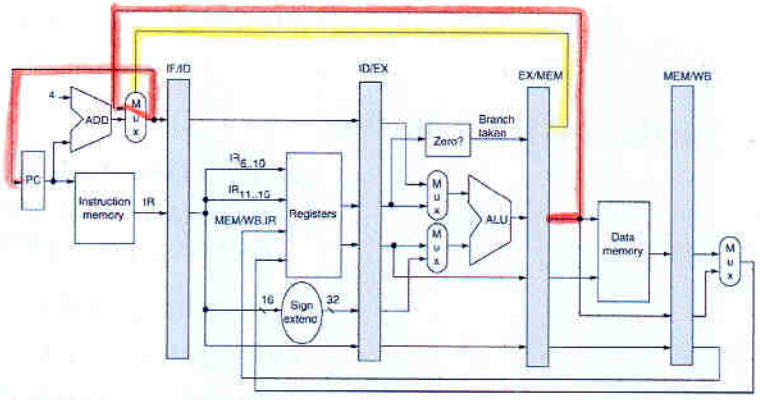
Esto sólo puede ocurrir en circunstancias extrañas: ej: unidad de ejecución no lineal y multifunción, en la cual cada instrucción puede seguir un camino muy diferente

- **Write - After - Read**

ejemplo:
- para cada operando de la instrucción hay una itapa ID → el ultimo operando de una instrucción podría tener su etapa ID dónde se lee después de que la instrucción siguiente ya haya escrito su primer operando (predecremento)
- Es usual en los procesadores CISC con predecremento, postdecremento, preincremento, postincremento,... de operandos.

5. Riesgos de control

Recordemos que la actualización del PC en una instrucción de salto se hace durante la fase MEM (i.e. estará disponible AL FINAL de esa etapa, y por tanto sólo podremos leer la instrucción siguiente en la siguiente etapa)
Por tanto en total vemos que es necesario esperar 3 ciclos.



Jump / Branch	IF	ID	EX	MEM	WB
					IF

↑ el PC nuevo se escribe tras la fase MEM

3 ciclos de espera por cada JUMP es una gran pérdida de prestaciones

Para un programa donde la proporción de instrucciones saltan es igual a A

$$CPI(n) = \frac{n^{o}ciclos}{n^{o}instr.} = \frac{(n+4)+3\times A\cdot n}{n}$$ (k-1)

Para un valor típico $A = 15\% \Rightarrow CPI(\infty) \to 1'45$
Estudiaremos las técnicas mas sencillas para ahorrar CPI

- **Técnicas que mantienen compatibilidad binaria**

– Técnicas de predicción (apuesta)

- Predict Taken: apuesto a que se va a saltar, por tanto tras el IF del salto, hago las instrucciones del destino. Si resulta que sí se saltaba: perfecto, me he ahorrado 3 ciclos de espera.
  Si resulta que no se saltaba, deshago las instrucciones ya hechas y me habré quedado como si hubiera insertado ciclos espera.

Para deshacer las siguientes 3 instrucciones en el DLX resulta ser muy fácil, ya que en 3 ciclos no les ha dado tiempo a cambiar nada, y no tengo más que cambiarlas por NOP.

Jump o Branch	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	X	X		
			IF	ID	X	X	X	
				IF	X	X	X	X

Para poder usar Predict Taken, necesitamos saber el destino del salto nada más acabar etapa IF, lo cual en el DLX es imposible.
Los procesadores modernos se van guardando una tabla con el PC donde hay saltos y su PC de destino, además de si la última vez se falló la apuesta.

- Predict Not-Taken

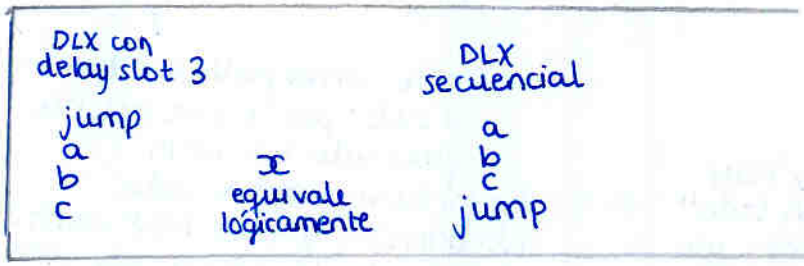
  Es más sencillo, se apuesta que no se va a saltar. Nos podemos ahorrar 3 ciclos de espera si acertamos la apuesta, y si no, se deshacen las instrucciones y nos quedamos igual.

• Técnicas sin compatibilidad binaria

- Delay Slot 3 (o salto retardado)

Consiste en que la unidad de control no haga nada especial en los saltos para evitar riesgos de control; simplemente sigue ejecutando las 3 instrucciones siguientes, y si salta; salta sin deshacerlas.

Es como si las 3 instrucciones siguientes estuvieran antes del salto



Debe ser el compilador, o el programador en ensamblador, el que solucione los posibles riesgos de control.

Una forma obvia de hacer esto es poner siempre tres NOP detrás de cada salto.
Cuidado: esto dará el mismo resultado que el caso secuencial, pero sigue sin existir compatibilidad binaria ya que las instrucciones no son las mismas (¡hay 3 NOP de más!!)

Ventaja: La unidad de control es mucho más sencilla

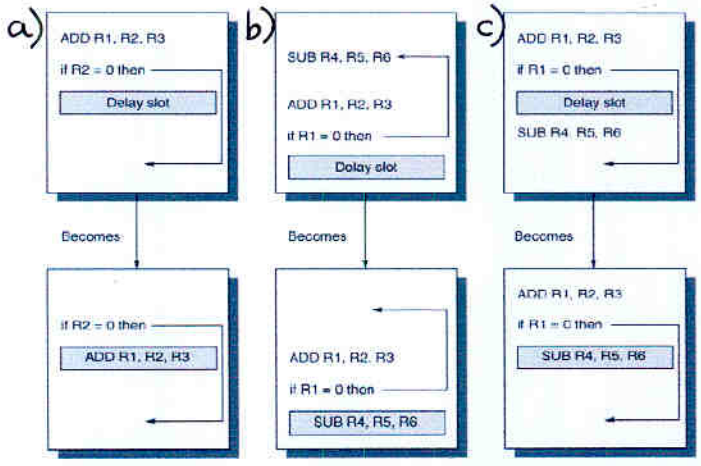
Reordenación: Puedo intentar reordenar el código poniendo instrucciones adecuadas en lugar de los 3 NOPs. (Es como si tuviéramos 3 slots para rellenar con instrucciones, de ahí el nombre). Deben ser instrucciones válidas tanto si se salta como si no.

Estrategias de compilación: ¿De dónde saco las instrucciones que sustituirán a los NOP? Hay 3 posibles lugares dónde buscar:

a) de antes del salto

b) del destino del salto

c) de después del salto



ejemplo: contar números impares a partir de vector de words que acaba en pos a + r4

```
loop: sub r4, r4, #4
      lw  r5, a(r4)
      lw  r6, b(r4)
      add r7, r5, r6
      andi r8, r5, #0x01
      beqz r8, par
```

nop
nop
nop

Inicialmente ponemos 3 NOPs tras cada salto

⟹ Intentamos sustituirlos por instrucciones que sean válidas tanto si se salta como si no.

```
loop: sub r4, r4, #4
      lw  r5, a(r4)
      andi r8, r5, #0x01
      beqz r8, par
```

lw r6, b(r4)
nop
add r7, r5, r6

Ésta DEBE estar antes del salto ya que r8 se usa como condición de salto

y ésta también ya que R8 depende de R5

sólo hemos podido "rellenar 2 slots" por lo que nos ha quedado un NOP.

Por fortuna hemos sido inteligentes y éste NOP sustituye a un ciclo de espera por estar usando r6 tras una operación de lectura de memoria

Como vemos, hay que "estar en todo" y ser ingeniosos; ya que hay muchos grados de libertad y hay que tener en cuenta riesgos de datos, estructurales y de control, todos a la vez, y en todas sus variantes.

el programa continúa con:

```
      add r1, r1, #1
par   sw  c(r4), r7
      bnez r4, loop
```

nop
nop
nop

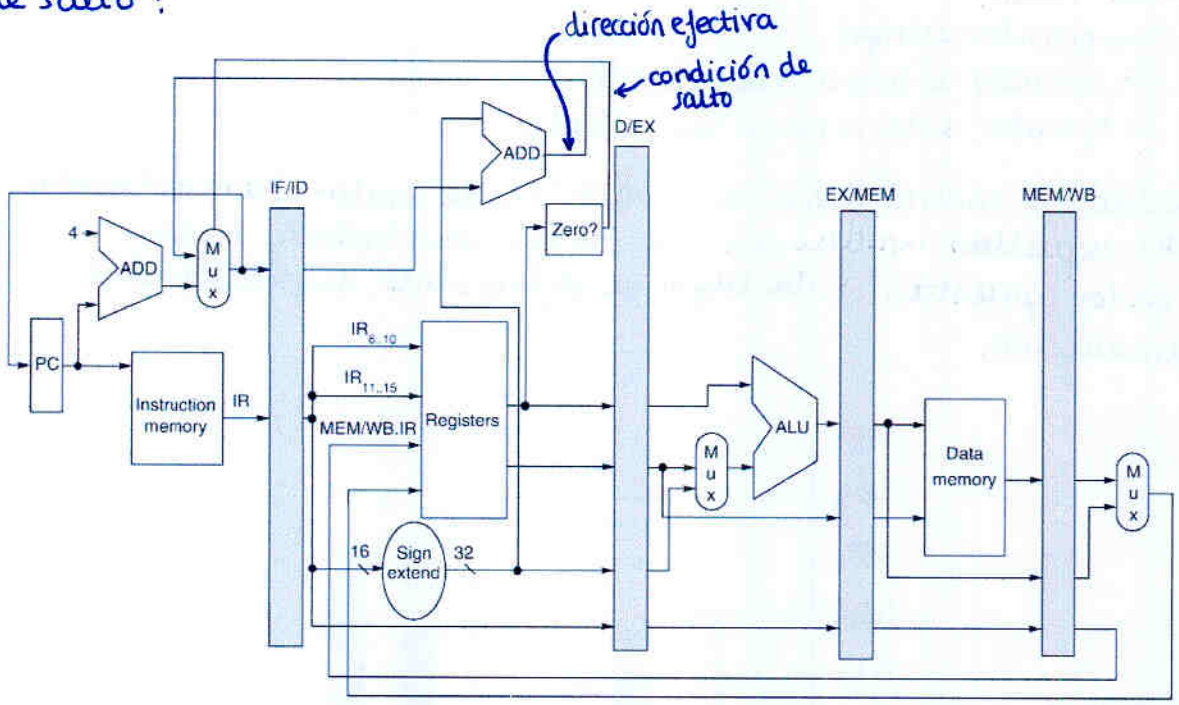
```
      add r1, r1, #1
par
      bnez sw c(r4), r7
```

sw c(r4), r7
nop
nop

· DLX modificado: Reducir el coste de los riesgos de control

Interesaría modificar el hardware para que no haya que esperar 3 ciclos antes de que se modifique el PC.

¿Y si intentamos meter en ID todo lo necesario para las instrucciones de salto?



Salto	IF	ID	EX	MEM	WB	
Instrucc. correcta tras salto		X	IF	ID	EX	MEM	WB

PC actualizado

Ventajas:
- Sólo se necesita un único ciclo de espera
- Podremos utilizar **Delay slot 1** (más sencillo de aprovechar) (sólo hay que sustituir un NOP)

Desventajas:
- Ahorá una instrucción de salto está leyendo los registros en el primer semiciclo de ID (recuerda que WB escribe en el primer semiciclo de ID!!)

  ⇒ Implica empeorar riesgos de datos, ya que tenemos que contemplar un nuevo tipo de riesgo que puede suponer ciclos de espera (no se puede cortocircuitar para solucionarlo)
- mayor coste hw (nuevo sumador)

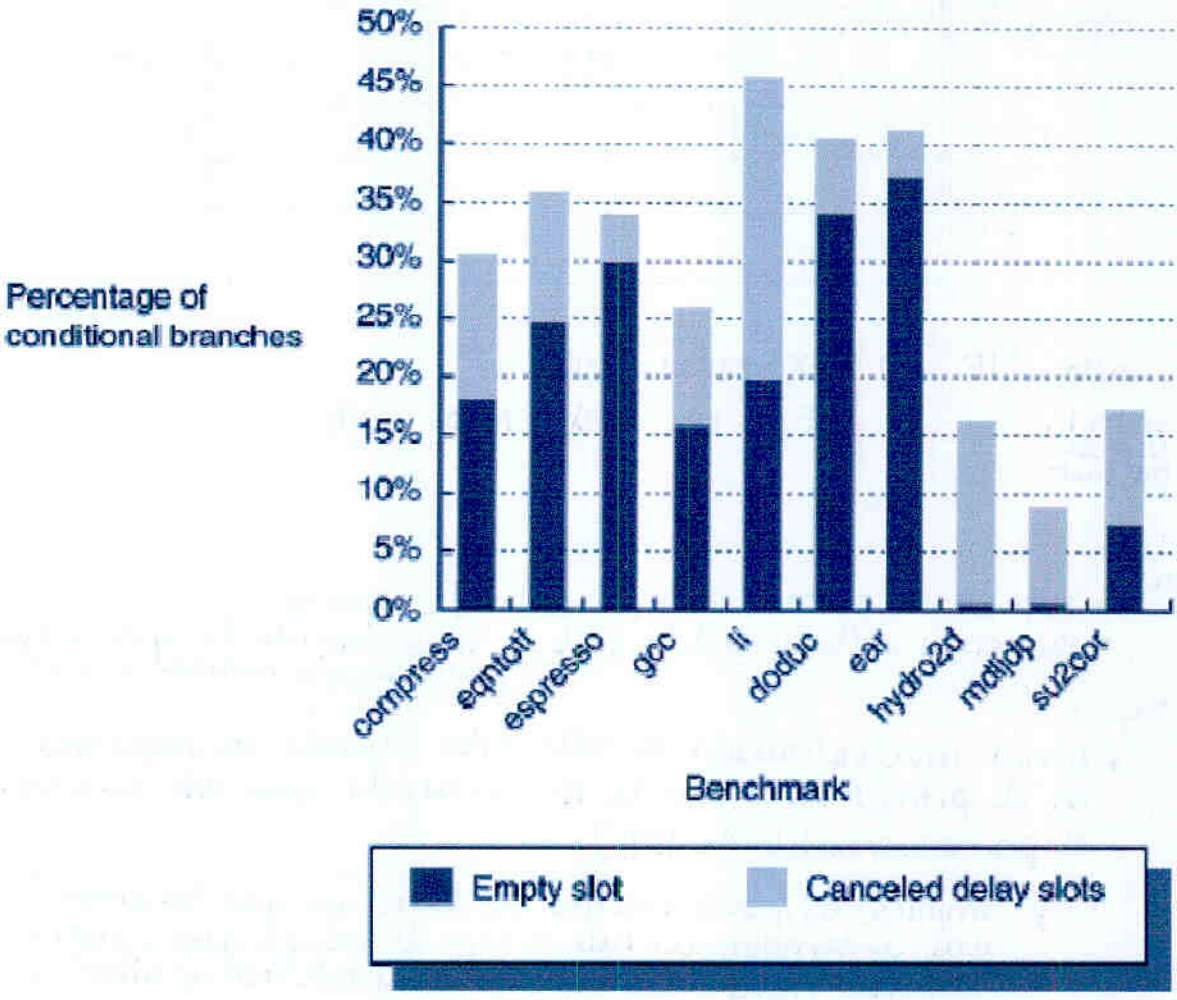
Éste DLX modificado puede utilizar:
- Ciclos de espera
- Ciclos de espera + cortocircuitos
- Predict Not Taken
- Delay Slot 1 ← en el simulador sólo hay esta opción para este DLX

- Salto retardado con cancelación

El programador colabora con la unidad de control.
Es como un delay slot, pero se pueden poner casos en la instrucción de salto (que tendrá un campo en el código de instrucción)

- ejecutar siempre
- ejecutar sólo si se ha saltado
- ejecutar sólo si no se ha saltado

La unidad de control tiene la capacidad de anular (convirtiendo en NOP) aquellas instrucciones que no debieron haberse hecho (fallo en la apuesta) → da lugar a delay slots desperdiciados por cancelación.



Explicación detallada del incremento de riesgos en Delay Slot 1:

** SIN Delay Slot 1

(1) IF | ID | EX | M | WB |
(2) IF | ID | EX | M | WB |
(3) IF | ID | EX | M | WB |
(4) IF | ID | EX | M | WB |

Si no hay DS1, y no hay cortocircuitos, los riesgos de datos se pueden evitar si la dependencia tiene dos instrucciones entre medias, es decir, entre (1) y (4) no hay problemas porque el WB del (1) y el ID (4) coinciden. Da igual que el (4) sea un salto, necesita los datos al final de ID.
Si hubiese cortocircuitos, no habría ciclos de parada por riesgos de datos salvo cuando hay un load seguido de una operación que utilice el dato cargado, en cuyo caso habría un ciclo de espera

** CON Delay slot 1

(1) IF | ID | EX | M | WB |
(2) IF | ID | EX | M | WB |
(3) IF | ID | EX | M | WB |
(4) IF | ID | EX | M | WB |

Sin cortocircuitos. Si lo que tenemos es un salto en (4), no basta con que haya dos slots entre los riesgos de datos porque cuando el (1) está en WB y el (4–salto) está en ID no es correcto, ya que el DS1 lo que hace es calcular la dirección del próximo PC en ID y por tanto necesita los datos al principio de ID.
Si hay cortocircuitos sucede lo mismo pero esta vez entre (3) y (4). Deberían tener una separación de al menos dos para que no hubiesen riesgos. (Cortocircuitando el resultado de EX al inicio de ID)
Si los riesgos son entre instrucciones que no son saltos, no necesitan los datos hasta el final de ID, así que , en ese sentido no empeoran los riesgos de datos con DS1 (comprobado en el DLX).

Conclusión:
DS1 sólo empeora los riesgos de datos si estos tienen lugar entre una instrucción cualquiera con un salto que venga después. Si es con cortocircuitos, tendremos que dejar 1 slot entre ellas y sin cortocircuitos tendremos que dejar 3 slots.

6. Excepciones

Posibles clasificaciones:

- Síncrona/Asíncrona ej: TRAP ó E/S
- Petición de usuario/Forzada
- Enmascarable por usuario o no
- A mitad instrucción o entre instrucciones
- Continuar o terminar el programa

Excepciones posibles en el DLX

IF, MEM } El acceso a memoria puede resultar en:
- acceso desalineado
- violación de protección (dir. memoria sin permiso o no existe)
- fallo de página

ID } – código de operación ilegal ej: escribir en R0

EX } – excepciones aritméticas ej: div por cero

WB } – ninguna aparte de E/S

En todas las anteriores puede haber una excepción de E/S (interrupción)

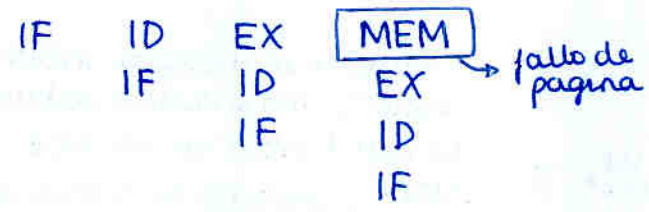
Actuación en DLX secuencial

lw r3, a(r0), 1000 IF ID EX [MEM] WB

MEM → fallo de página

- guardo el PC y registro de estado
- salto a rutina de atención a la interrupción
- al acabar, vuelvo a cargar PC y reg. de estado

Actuación en el segmentado

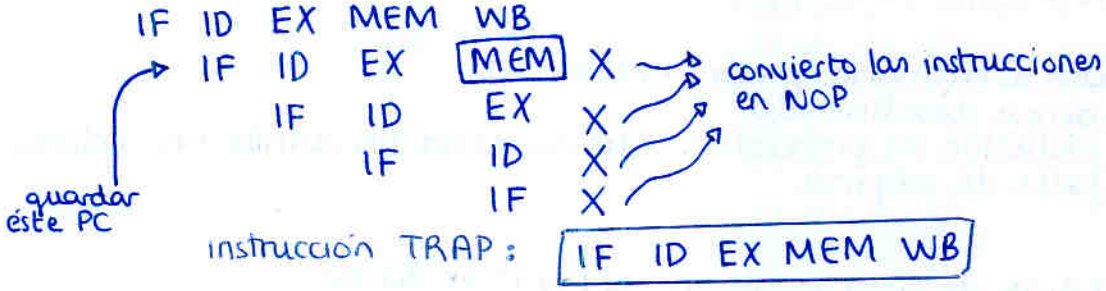


¿¿Qué hacemos ?!
Hay que llegar a un acuerdo
Hay que lograr que el resultado sea el mismo que el del secuencial

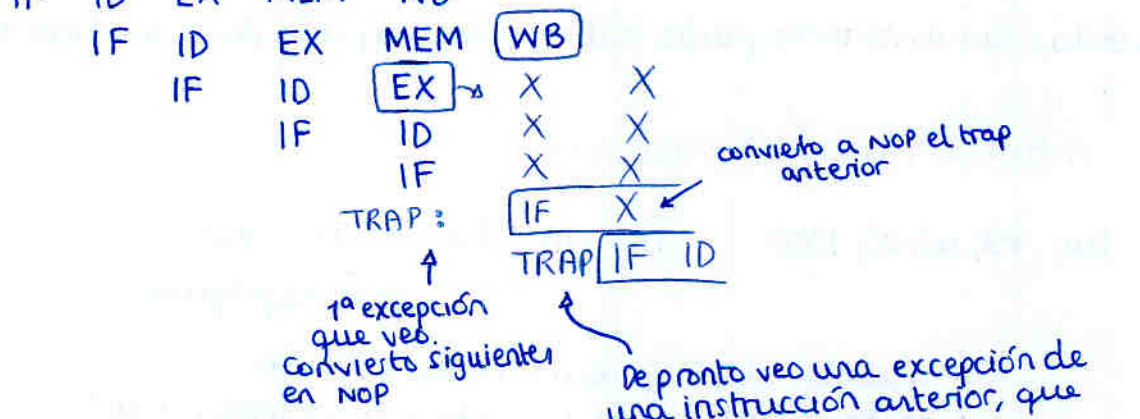
· Excepciones precisas

comportamiento idéntico a no segmentado: excepciones precisas
- se terminan correctamente las instrucciones anteriores
- se abortan la instrucción que generó la excion y todas las siguientes
- tras completar la rutina de servicio se relanza el programa comenzando por la instrucción que lanzó la excepción.

ej en el DLX



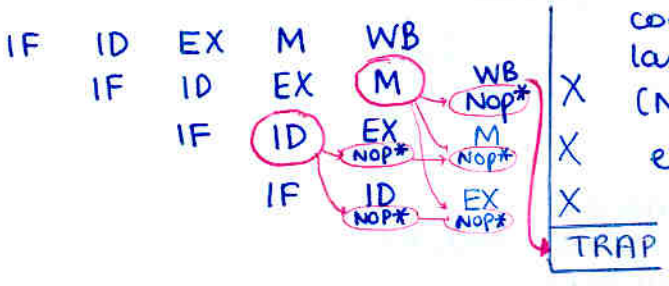
complicación: ¿y si hay dos interrupciones?



La unidad de control se complicaría mucho si debe tener todo esto en cuenta. Hay una solución mejor.

¿Convertimos en NOP el TRAP anterior y hago nuevo TRAP?

solución



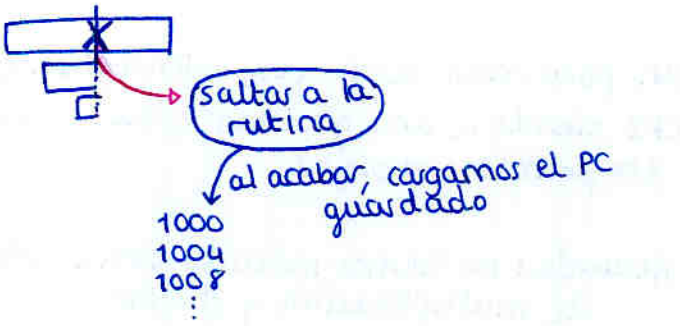
Cuando se detecta interrupción, se convierte instrucción actual y todas las siguientes en un NOP especial (NOP*), pero no se comienza la ejecución del TRAP hasta que esa instrucción NOP* no llegue a la fase WB. (todas las instruc. anteriores hayan acabado)

De esta forma nos aseguramos de que sólo se ejecutará el TRAP de la instrucción más antigua con una gran sencillez en la unidad de control. A cambio, en los casos simples se perderán ciclos comparado con atender la excepción inmediatamente

· Problema al usar delay slot (salto retardado)

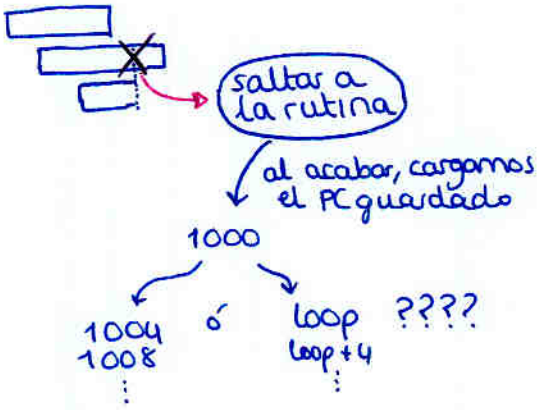
normalmente:

```
1000  lw
1004  add
1008  sub
 ⋮     ⋮
```



con delay slot 1:

```
996   bnez r1, loop
1000  lw
1004  add      (delay slot)
```



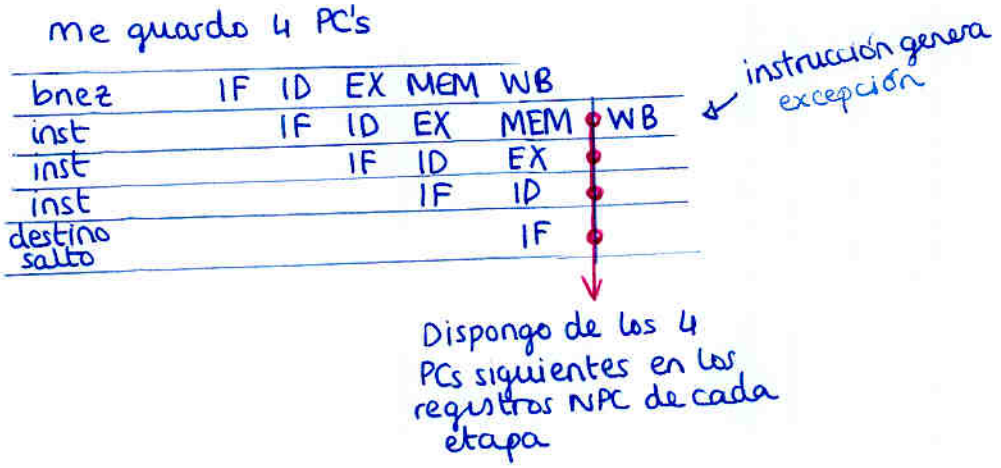
¿cómo sabemos por dónde hay que continuar?

solución: En lugar de guardar el PC actual, guardamos el PC actual y el siguiente (en el ejemplo 1004 o loop)

Para delay slot N: me guardo PC actual y los N siguientes

ejemplo: delay slot 3

me guardo 4 PC's



¿Y si la interrupción no llega en el primer delay slot (por ej llega en el tercero)? En ese caso bastaría con guardar 2 PC's, pero ¿para qué vamos a complicar a la unidad de control haciendo que distinga casos si guardar 4 PC's es una solución genérica que funciona siempre? Mejor que guarde siempre 4, a pesar de que en algunos casos no haga falta.

Conclusiones DLX segmentado

- Un poco más caro que sin segmentar
- CPI tiende a uno gracias a haber resuelto o minimizado los posibles riesgos

- Recuerda: no hemos incluido coma flotante ni las operaciones de multiplicación y división.

Tema 7. Procesadores superescalares

1. Operaciones multiciclo

Problema:
- Instrucciones enteras complejas (mult y div) → Necesitan fases de ejecución mucho más largas
- Instrucciones en coma flotante → Necesitan fases de ejecución mucho más largas

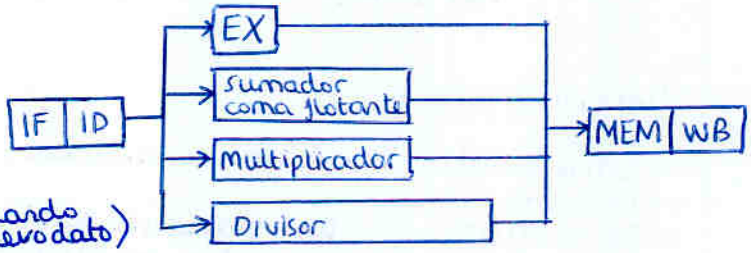
Soluciones:
- Aumentar el periodo de reloj (tiempo de ciclo) penaliza a las otras fases no tan largas
- Mas HW para mejorar fase EX no siempre es posible

Solución: Permitir fase EX con muchos ciclos

- operaciones multiciclo
- Distinto nº de ciclos según la instrucción

⇓

Segmentación no lineal
↳ Política de iniciaciones (saber cuando meter nuevo dato)



Tiempo de evaluación (Tev) y tasa de iniciación (IR)

Se dan los valores Tev e IR para cada operador.

ejemplo: sumador/restador de coma flotante { Tev = 4, IR = 1 cada ciclo

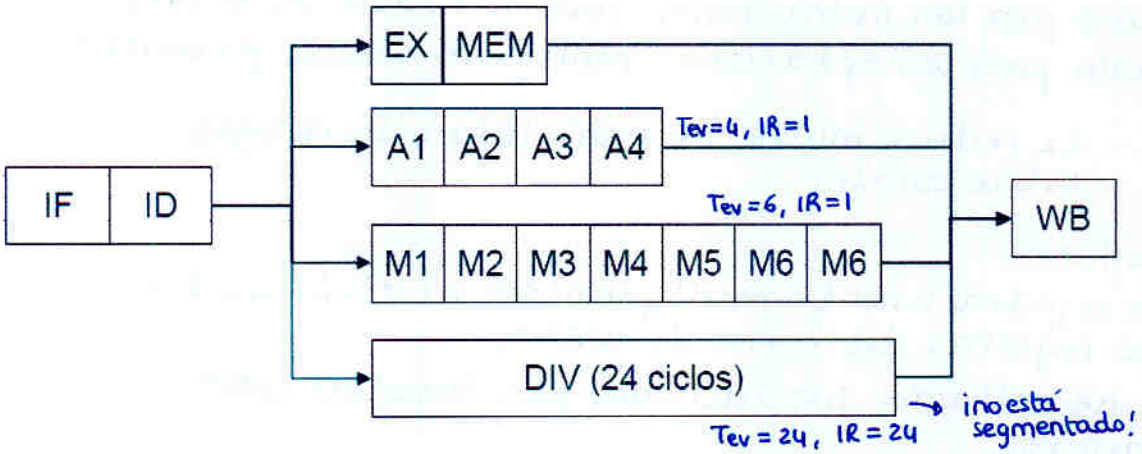
Divisor { Tev = 24, IR = 1 cada 24 ciclos → parece que se utiliza todo el hardware durante los 24 ciclos

otros ejemplos:

Tev = 6, IR = 3 → sería un operador segmentado en 2 etapas de 3 ciclos cada una

Tev = 6, IR = 1 cada 7 ciclos → puede que se necesite un tiempo extra después de sacar el resultado antes de poder recibir dato nuevo (ej: vaciar memoria)

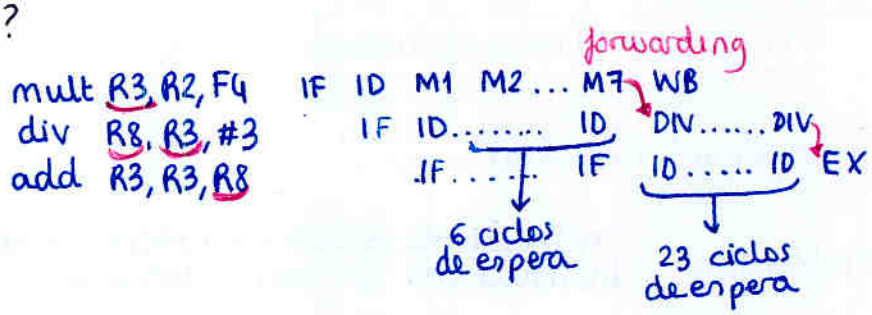
Operaciones multiciclo en el DLX



¿Introduce nuevos riesgos?

Sí, y requiere añadir ciclos de espera

Ocurre cuando una instrucción requiere del resultado de una instrucción anterior



Paralelismo a nivel de instrucción
ILP

: concepto cuantitativo de cómo de independientes son ciertas instrucciones entre sí i.e. si necesitan esperarse unas a otras o si pueden ejecutarse a la vez o en paralelo

ILP ↑ ⇒ ciclos de espera ↓ ⇒ CPI ↓

2. Dependencias

- Estructurales ↔ Pueden considerarse o no como dependencias
- De Datos
- De Nombre
- De Control

Dos instrucciones son independientes si se pueden reordenar ↔ pueden ejecutarse simultaneamente

2.1. Dependencias Estructurales

Cuando dos instrucciones utilizan un mismo elemento en el mismo ciclo

ejemplo: El divisor son 24 ciclos sin segmentar; no puede haber mas de una división procesándose simultaneamente

ejemplo: una instrucción simple con enteros trata de escribir en el banco de registros a la vez que una instrucción multiciclo que empezó unos ciclos antes

Es muy caro hacer registros que permitan 2 escrituras simultaneas.

Se opta por utilizar dos bancos de registros

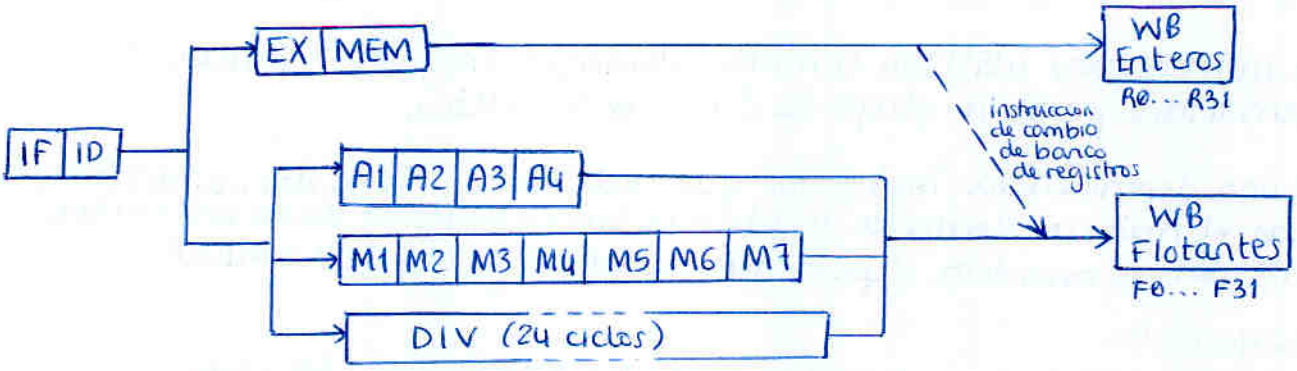
- Banco para las instrucciones simples (banco de enteros)
- Banco para las operaciones multiciclo (coma flotante)

Ventaja: Se reduce mucho la posibilidad de riesgos estructurales

Inconvenientes:

- No se podrá usar la multiplicación ni división sobre los registros del banco de enteros
- se necesitarán instrucciones para transferir entre registros.

Con 2 bancos de registros se tiene:



Se eliminan las dependencias estructurales entre instrucciones simples y operaciones multiciclo, pero aún pueden existir entre las propias operaciones multiciclo

ejemplo:

mult F1, F2, F3	IF	ID	M1	M2	M3	M4	M5	M6	M7	WBF		
inst		IF	ID	EX	M	WB						
inst			IF	ID	EX	M	WB					
addf F4, F5, F6				IF	ID	A1	A2	A3	A4	A4	WBF	
addf					IF	ID	A1	A2	A3	A3	A4	WBF

No pueden escribir en el banco de registros de coma flotante de forma simultánea. La suma debe esperarse en fase A4 (1 ciclo de espera) y la suma que viene detrás no puede pasar a A4 y debe esperar en A3.

2.2 Dependencias de Datos

sean i, j dos instrucciones, donde j va detrás lógicamente de i

CUIDADO: El orden LÓGICO de 2 instrucciones puede no coincidir con el orden en ensamblador: ejemplo → una instrucción que actúa como delay slot va ANTES LÓGICAMENTE que la instrucción destino del salto anterior.

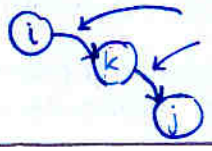
Existe dependencia de datos entre ellas sí:

- $i \Rightarrow j$ i produce un resultado usado por j
- $i \Rightarrow k \Rightarrow j$ existe instrucción k entre i y j que tiene dependencia de datos con ambas.
  Esta cadena puede ser tan larga como el propio programa

De forma equivalente:

i y j tienen dependencia de datos $\longleftrightarrow$ Deben tener una ordenación relativa entre ellas determinada para el correcto funcionamiento

Además:



En medio de la cadena de instrucciones i, k, j puedo poner cualquier instrucción independiente de k y j (puede ser dependiente de i)

2.3 Dependencias de nombre

Dos instrucciones utilizan el mismo elemento (registro o posición de memoria) pero sin flujo de datos entre ellas.

Es una dependencia "más floja" que la de datos. i.e. si dos instrucciones usan el mismo elemento y además tienen flujo de datos entre ellas, entonces se considera dependencia de DATOS y NO DE NOMBRE.

clasificación:
- Antidependencia: j escribe sobre un elemento que i lee (leyó)
- Dependencia de salida: j e i escriben sobre mismo elemento

las dependencias de nombre se deben únicamente a la elección que el programador ha hecho de registros y memoria; no es algo inherente al programa.

ejemplo típico: int aux; se usa repetidas veces a lo largo de una función (buclas, contadores, ...) sin tener nada que ver unos contenidos con otros, simplemente porque el programador no quiso hacer int aux2; ...

ejemplo:
```
mult F3, F2, F4
add  R2, R2, R2
add  R8, R8, R8
addF F2, F4, F10
```
no hay dependencia de datos, pero NO se pueden reordenar porque usan mismo registro F2
i.e. hay dependencia de nombre (en este caso antidependencia)

2.4 Dependencias de control

La ordenación de las instrucciones respecto a una instrucción de salto previa.
TODA instrucción tiene dependencia de control con algun salto (el que llamó a la función)

ejemplo: problema de examen

```
1.          BNEZ R2, salto
2.          ADD  R3,R3,#8
3.          LDD  F3, 0(R3)
4.    salto: SUBD F2, F1, F3
5.          ADDD F5, F2, F1
6.          ADDD F7, F7, F5
7.          DIVD F7, F4, F6
```

Dependencias

Instrucción	Estructural	de Datos	Antidependencia	de Salida	de Control
3	NO	2,4,5,6	NO	NO	1
4	5,6	2,3,5,6	NO	6	1
5	4,6	2,3,4,6	6	NO	1

2.5. Resumen: Dependencias

Para que 2 instrucciones puedan presentar un riesgo hay 2 condiciones:
- Deben estar ambas simultaneamente dentro de la unidad segmentada
- Deben tener alguna dependencia entre ellas

ILP ↓ ⇐⇒ Muchas dependencias entre instrucciones de un bloque básico ⇒ Aumenta posibilidad de riesgos (y por tanto de ciclos de espera) ⇒ Disminuye CPI

Bloque básico: secuencia de instrucciones entre dos instrucciones de salto (tamaño típico medio 6~7 instrucciones)
- → Instrucciones dentro de un bloque tienen dependencia de control con los dos saltos
- → Instrucciones dentro de un bloque pueden tener dependencias entre ellas

Objetivo: reducir dependencias ⇐⇒ aumentar ILP ⇒ reducir CPI

métodos:
- gestión estática: el compilador reordena el código
- gestión dinámica: el procesador cambia el orden de ejecución de instrucciones

→ Gestión estática

Técnica: loop unrolling (desenrollado de bucles)

```
int i;
for (i=0; i<64; i++)
  c[i] = a[i] + b[i]
```

⇒

```
for (i=0,i<64;) {
  c[i] = a[i] + b[i];
  i++
  c[i] = a[i] + b[i];
  i++
  c[i] = a[i] + b[i];
  i++
  c[i) = a[i) + b[i);
  i++;
}
```

⇒

```
for (i=0; i<64; i+=4)
  c[i] = a[i] + b[i]
  c[i+1] = a[i+1] + b[i+1]
  c[i+2] = a[i+2] + b[i+2]
  c[i+3] = a[i+3] + b[i+3]
}
```

⇩

Ahora hemos eliminado la dependencia de datos en el bloque básico (se pueden reordenar)

⇩

Ventajas: reducido el número de saltos y de comparaciones (menos riesgos de control)

Coste: - Código de programa mayor
- Podría implicar traer otro bloque de memoria de programa a la cache

Gestión estática que puede hacer el compilador:

- Desenrollado de bucles para reducir dependencias de control
- Técnicas para reducir dependencias de datos
- Elegir los registros para disminuir riesgos de nombre

3. Gestión dinámica de instrucciones

El circuito de control (en HW) aumenta el ILP reordenando la ejecución de las instrucciones.

- Las instrucciones independientes se ejecutan lo más simultaneamente posible
- Las instrucciones dependientes se ejecutan en order.

Para ello, una instrucción dependiente puede estar esperando a que finalice alguna de las instrucciones de las cuales depende, y mientras, en lugar de parar también todas las que vienen detrás, dejar que vayan pasando las instrucciones que sean independientes de las que se encuentran en la unidad

i.e. el HW detecta cuando una instruccion no tiene dependencias y la ejecuta aunque delante haya otra parada

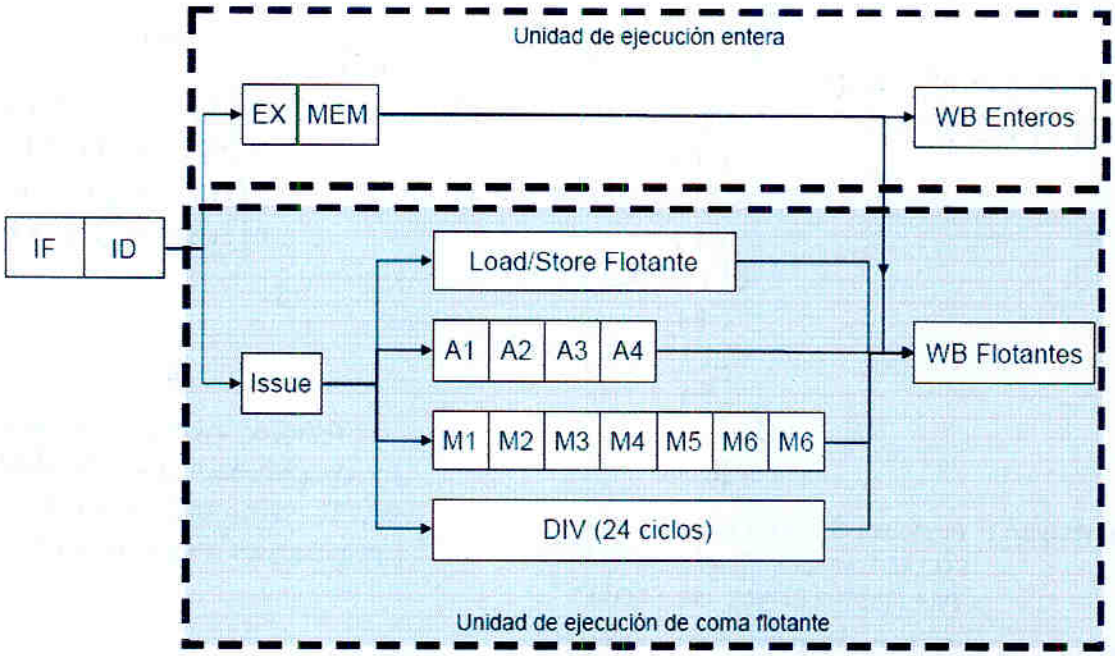
Ventajas:

- Compilador más sencillo
- soluciona dependencias desconocidas en compilación
- No hay que optimizar código
  ↳ compatibilidad binaria

Desventajas:

- HW más complicado

3.1 Algoritmo de Tomasulo



La gestión dinámica se aplica sólo a instrucciones multiciclo

Étapa issue: clasifica las instrucciones
- si los operandos aún no están disponibles → esperar
- si el operador aún no está disponible → esperar
- si está disponible operador y operandos → ejecutar

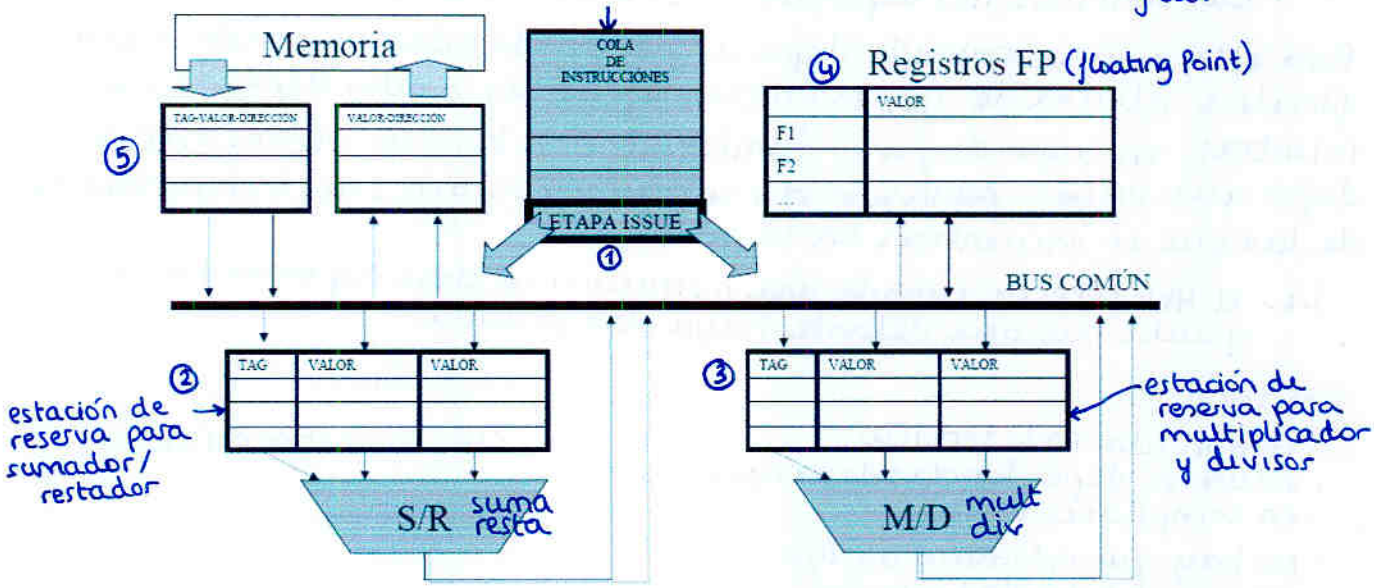
Las instrucciones que esperan no deben impedir el avance de nuevas instrucciones que vengan detrás y estén en condiciones de ser ejecutadas

¿Dónde esperan las instrucciones?

¿En etapa ISSUE? No

¿En primer ciclo del operador? Sí, pero falla cuando otra instrucción pide el mismo operador

¿En estructuras de datos (salas de espera)? (estaciones de reserva) Perfecto → una para suma y resta; → una para mult y div



Explicación:

- Llega una instrucción a la etapa Issue; la etapa issue hace 3 cosas:
  (1). Asigna un TAG (etiqueta) a la instrucción
  (2). Busca los operandos de la instrucción, pudiendo obtener, como luego se explica, 0, 1 o 2 valores y 0, 1 o 2 tags.
  (3). Escribimos, en el elemento (registro/memoria) destino de la instrucción, el TAG de la instrucción (sustituyendo lo que hubiera)

  3 PASOS NUNCA OLVIDAR NINGUNO

  Ahora se envía la instrucción a la estación de reserva del operador adecuado; almacenándose el TAG junto a los operandos (valores y/o tags)
- Desde el punto de vista de la instrucción, está dentro de un operador virtual (aunque en realidad esté en cola)

- Los registros en coma flotante, campos "valor" en las estaciones de reserva, y en general cualquier campo VALOR de la figura anterior, pueden contener dos cosas:
  - Valor en coma flotante: representa el valor en coma flotante e indica que no hay actualmente ninguna instrucción en la unidad segmentada que lo vaya a cambiar.
  - TAG de instrucción: indica que ese valor aún no está disponible ya que está pendiente de modificación por una instrucción que está actualmente dentro de la unidad segmentada y es la que lleva asociado ese tag.

  Esto es un método muy ingenioso para conocer la existencia de dependencias:

- Cuando la primera etapa de un operador queda libre y por tanto puede entrar a operarse una nueva instrucción, se mira a la primera instrucción de su estación de reserva (asociada a ese operador) y se comprueba si están disponibles ambos operandos; para ello no hay más que comprobar que contengan un valor y no un TAG, ya que si tuvieran un TAG significaría que necesitan el resultado de una instrucción anterior con dicho TAG que aún no ha acabado. i.e. dependencia

  Si la instrucción tiene disponibles ambos valores, puede entrar al operador, si por el contrario tiene alguna dependencia, tendrá que seguir esperando y se comprobará la siguiente instrucción.

  Así sucesivamente, de forma que nos aseguramos de que las instrucciones independientes pueden pasar a ejecutarse sin tener que esperar a otras instrucciones anteriores que aún no dispongan de sus operandos.

- Al acabar un operador con su instrucción (i.e. la fase Writeback) se escribe el tag y el resultado en el bus común a toda la unidad de coma flotante; esto puede implicar tener que insertar ciclos de espera para evitar dos writeback simultáneos

Cuando en la fase WB se escribe el TAG y el resultado de la instrucción en el bus común, TODOS los elementos que contengan dicho TAG lo sustituirán por el valor resultado; por ejemplo uno de los operandos de una instrucción que está esperando en la estación de reserva.

(i) En el mismo ciclo en el que se escribe un tag y un valor en el bus común, puede entrar al operador una instrucción que estaba esperando ese resultado como uno de sus operandos

ejemplo:

Registros Coma Flotante: F1=1, F2=2, F3=3, F4=4

ESCRITURA DEL TAG EN DESTINO

se ha reescrito F1 antes de acabar op1 ⇒ no para nada (i) ya que el tag que la instrucción Div tiene para F1 sigue siendo op1

F1=op1 F2=op2 F1=op3

MULT F1, F2, F3	ISSUE TAG LECTURA OPERANDOS op1 2,3	M1	M2	M3	...	WB Escribo en Bus común op1 6
DIV F2, F1, F4	ID	ISSUE op2 op1,4	dv_1 op2 op1,4	dv_1	...	D1 op2 6,4
add F1, F2, F3	F1	ID	ISSUE op3 op2,3	av-1 op3 op2,3	...	av-1 op3 op2,3

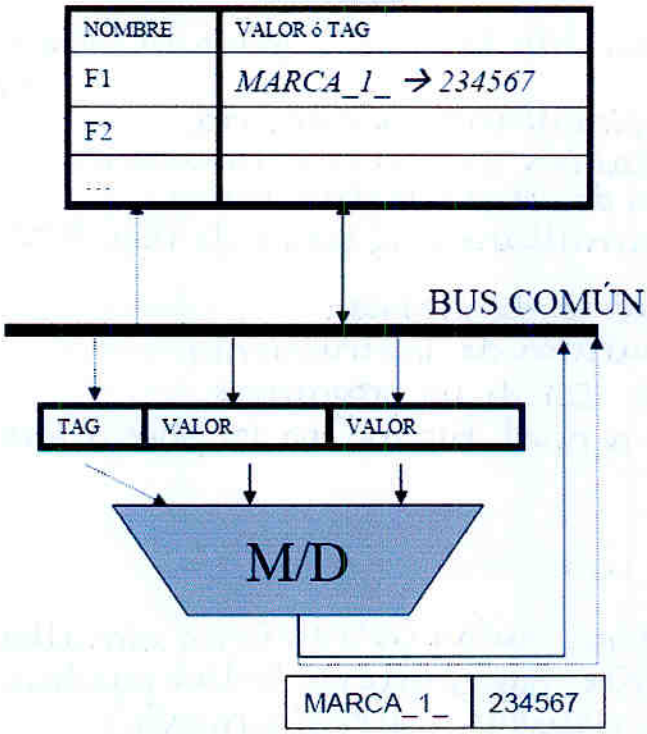
En la lectura de operandos ha leido el TAG

divisor virtual 1. En realidad está en la "cola de espera" No puede entrar a M1 porque le falta un operando

se escribe en el bus común op1 6

↳ por tanto op2 actualiza uno de sus operandos cambiando el TAG op1 por el valor 6; y en el MISMO CICLO puede ya entrar al operador por tener ya los operandos disponibles

↳ El registro F1 no se pone valor 6 porque se reescribió el tag op3 sobre el de op1



4. Aumento de prestaciones

El tiempo de ejecución de un programa viene dado por 3 factores:

$$Te = I \times CPI \times Tclk$$

- I: nº instrucc
- CPI: $\frac{\text{ciclos}}{\text{instrucción}}$
- $Tclk$: tiempo ciclo

Con la segmentación se intenta $CPI = 1$
Intentar reducir aún más Te da lugar a 3 familias de computadores:

- Reducir CPI a $1/n$ → Superescalares (grado n)
- Reducir nº instrucciones → VLIW (very long instruction word)
- Reducir Tclk → Supersegmentados

ejemplo:
Pentium 4 { superescalar grado 2 / supersegmentado

Athlon: superescalar grado 6

4.1 Superescalares

- m instrucciones en un ciclo de reloj (grado del computador m)
  $CPI = \frac{1}{m}$
- Implica:
  - accesos simultáneos a memoria
  - decodificación de varias instrucciones
  - ejecución de varias instrucciones
  - accesos simultáneos al banco de registros

  replicar hardware
- Aumenta la posibilidad de riesgos:
  ↳ necesaria Gestión dinámica de instrucciones
  ↳ extraer máximo ILP de los programas
  ↳ compatibilizar a nivel binario con los procesadores escalares

Superescalares no uniforme

Si IF está libre, se cogen varias instrucciones simultaneamente, pero en el ciclo siguiente puede que no todas puedan entrar a la vez (y tenga que ejecutar secuencialmente)

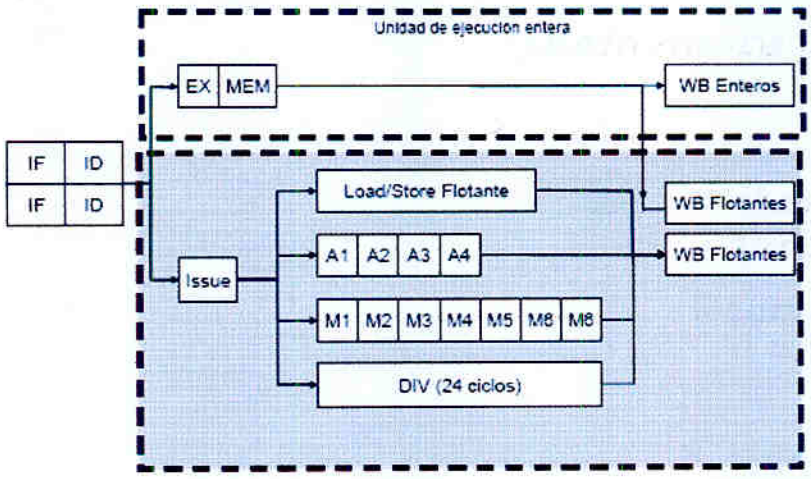
Se imponen limitaciones en el tipo de instrucciones ejecutables a la vez

ejemplo (grado 2)
se pueden hacer simultaneamente:
- 2 instrucciones enteras
- 1 entera + 1 coma flotante
- 1 load/store + 1 aritmética

así sólo hay que replicar parte del hw

- Aparecen riesgos estructurales debido a que no siempre se pueden lanzar m instrucciones
- Si el compilador conoce las limitaciones del procesador puede intentar agrupar instrucciones que se puedan ejecutar simultáneamente

El DLX podría ser superescalar no uniforme si duplicásemos IF e ID (grado m=2)

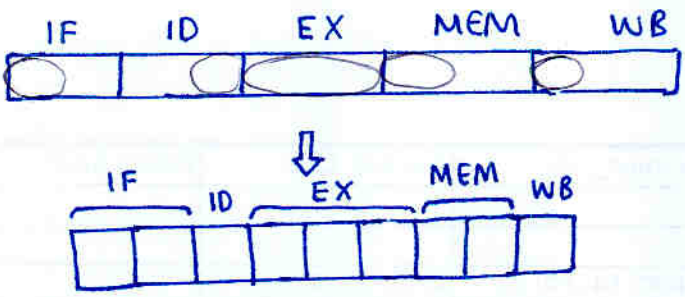


La limitación sería no poder hacer simultaneamente 2 operaciones enteras ni 2 en coma flotante; sí que podríamos hacer simultaneamente una de cada

4.2 Procesadores supersegmentados

Se divide cada etapa en t subetapas (t = grado de supersegmentación)

ejemplo: En el DLX algunas etapas van muy holgadas por culpa de EX que va ajustada



Se supersegmentan las etapas que sean cuello de botella.

- Logro reducir el tiempo de ciclo
- Necesita más hw (aunque no tanto más como el superescalar)

Problemas:

- más etapas → mas riesgos
- más frecuencia → más disipación : más potencia
  - clock skew
  - tiempo de los registros deja de ser despreciable frente al tiempo de la etapa

Existen versiones superescalares - supersegmentadas

4.3 Procesadores VLIW

Very Long Instruction Word

idea: "no tener unidad de control" (en realidad sí, pero muy pequeña)

El compilador cumple esa función al escribir las palabras de instrucción

- Una instrucción puede decirle a <u>cada</u> operador de la CPU lo que tiene que hacer (un campo para cada operador) en este ciclo
- El compilador debe resolver todo (riesgos, tiempos, ...)

No hay NINGUNA compatibilidad binaria

Ventajas:
- menor nº instrucciones
- menor complejidad hardware

ejemplo: Instrucción con los siguientes campos

Mem 1	Mem 2	Aritmet_CF1	Aritmet_CF2	Entera/salto

ejemplo:

```
loop
  ld   F0, 0(R1)
  addd F4, F0, F2
  sd   0(R1), F4
  sub  R1, R1, #8
  bnez R1, loop
```

Mem1	Mem2	Aritmet_CF1	Aritmet_CF2	Entera/Salto
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2	
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2	ADDD F24,F22,F2	
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2		
SD -16(R1),F12	SD -24(R1),F16			
SD -32(R1),F20	SD -40(R1),F24			SUB R1,R1,#56
SD -48(R1),F4				BNEZ R1,loop

ejemplos de máquinas reales:

Tipo	Siguiente Instrucción	Detección riesgos	Reordenación	Comentarios adicionales	Ejemplos
Super segmentado	Estática	HW	Estática	8 etapas	MIPS R4000 Pentium 4
Superescalar (estático)	Dinámica	HW	Estática	Ejecución en orden	Sun UltraSparcII Sun UltraSparcIII
Superescalar (Dinámico)	Dinámica	HW	Dinámica	Fuera de orden en algunas op.	IBM Power2, PowerPC
Superescalar (Especulación)	Dinámica	HW	Dinámica y especulación	Reordenación y especulación	Pentium III y 4 MIPS R10K Athlon, Opteron PowerPC
VLIW	Estática	SW	Estática	Sin riesgos entre inst.	Trimedia, i860, Transmeta Crusoe
EPIC	Estática (Excepciones)	SW/HW	Estática (Excepciones)	Compilador marca riesgos	Itanium Itanium2

→ reduce complejidad para reducir el consumo

Ejercicio Enero 2006

Dado el siguiente programa para el DLX con operadores de coma flotante:

```
buc: LDD      F2,a(R2)
     LDD      F4,b(R2)
     SUB      R1,R1,#1
     MULTD    F8,F2,F6
     ADDD     F8,F8,F10
     SDD      c(R2),F8
     BNEZ     R1,buc
     ADD      R2,R2,#8
```

Sabiendo que se emplea una técnica de salto retardado con tamaño de Slot1, y que los operadores disponibles son:

MULTIPLICACIÓN	: Tev=4	IR=1 por ciclo
SUMA	: Tev=2	IR=1 por ciclo
LOAD/STORE	: Tev=3	IR=1 cada 2 ciclo

Suponiendo:

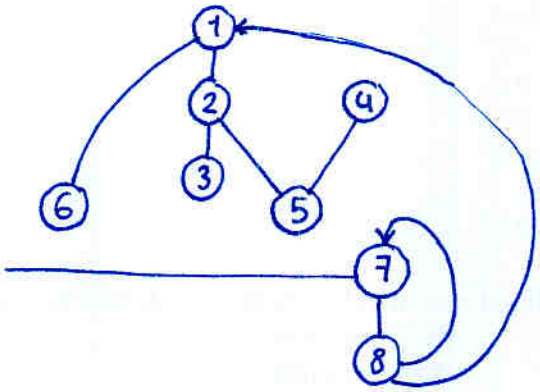
CASO 1 : CON TOMASULO: (Resuelto en hoja 2)

CASO 2: SIN TOMASULO (Resuelto en hoja 3)

CASO 1

- con cortocircuitos → salida de cualquier operador puede ir a entrada de cualquier otro
- sin tomasulo

Dependencias en el bucle



La última instrucción de cada parada se empieza al cabo de $12N$, y al final faltarán 4 más

$$CPI = \frac{12N + 4}{8N} \longrightarrow 1'5$$

CASO 2

- con tomasulo
- cortocircuitos solo en lo entero

seguimiento de los registros

F0 : l1 X l10
F4 : a1 X
F8 : a2 X
F3 m1

8 instrucciones para que entre la ultima instruccion
7 instrucciones hasta que acaba

$$CPI = \frac{8N + 7}{8N} \longrightarrow 1$$ wow

CASO 1

NOMBRE:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	IF	ID	L1	L1	L2	WB																	
2		IF	ID	ID	ID	A1	A2	WBF															
3			IF	IF	IF	ID	ID	L1	L1	L2	WBF												
4						IF	IF	ID	A1	A2	A2	WB											
5								IF	ID	ID	M1	M2	M3	M4	WBF								
6									IF	IF	ID	L1	L1	L2	L2	WBF							
7											IF	ID	EX	M	WBE								
8												IF	ID	EX	M	WBF							
1													IF	ID	L1	L1	L2	WB					

cortocircuito

CASO 2

NOMBRE:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	ISS ℓ1,x	L1	L1	L2	WBF ℓ1																		
2	ID	ISS a1,x,ℓ1	Av1 a1,x,ℓ1	Av1	A1	A2	WBF a1																
3	IF	ID	ISS st1,x,a1	LSv1	LSv1	LSv1	L1	L1	L2	WBF st1													
4		IF	ID	ISS a2,x,x	Av2	A1	A2	WBF a2															
5			IF	ID	ISS m1,a2,a1	Mv1	Mv1 m1,x,a2	M1	M2	M3													
6				IF	ID	ISS st2,x,x	LSv2	LSv2	L1	L1													
7					IF	ID	EX	MEM	WBE														
8						IF	ID	EX	MEM	WBE													
1,2ª							IF	ID	ISS ℓ10,x	LSv1													

Tema 8. Multiprocesadores

1. Conceptos y clasificación

- Definición: · Conjunto de procesadores interconectados diseñados para la ejecución conjunta de aplicaciones
- Justificación: · Lograr mayor potencia de cálculo
  · se pueden utilizar los µP estándar como elemento constructivo
- Dos técnicas para mejorar prestaciones:
  - Multiprogramación: Los µP se reparten los procesos/aplicaciones (no comparten un único proceso, aunque pueden haber varias aplicaciones que se ejecuten para un bien común)
  - Aplicaciones distribuidas: Los µP trabajan para disminuir el tiempo de ejecución de una aplicación

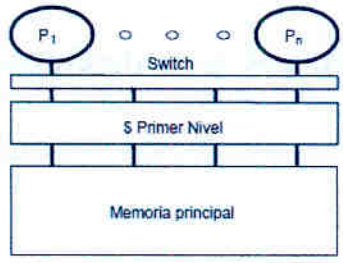
Clasificación

- Según modelo de programación:
  · Variables compartidas
  · Paso de mensajes
- Según la arquitectura del sistema de memoria:
  · Memoria compartida ⇒ variables compartidas
  · Memoria distribuida ⇒ paso de mensajes

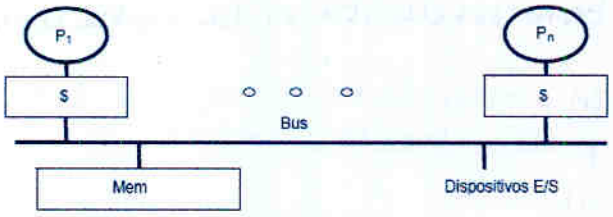
  ↔ con uno se puede simular el otro

- Alternativas de interconexión:

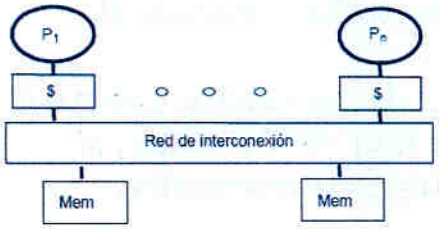
Nota: el símbolo $ representa la cache



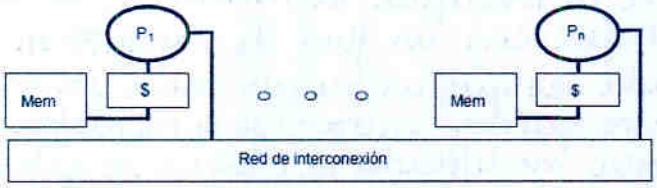
(a) Cache compartida



(b) Memoria compartida con bus



(c) Dancehall



(d) Memoria distribuida

2. Multiprocesadores simétricos SMP

- Comparten misma memoria y espacio de direccionamiento, cada uno con su cache.

  → Todos tienen el mismo tiempo de acceso a la memoria (son UMA)

> UMA ⟺ En la fórmula del tiempo de acceso a memoria no aparece ni el identificador del procesador ni la dirección de memoria

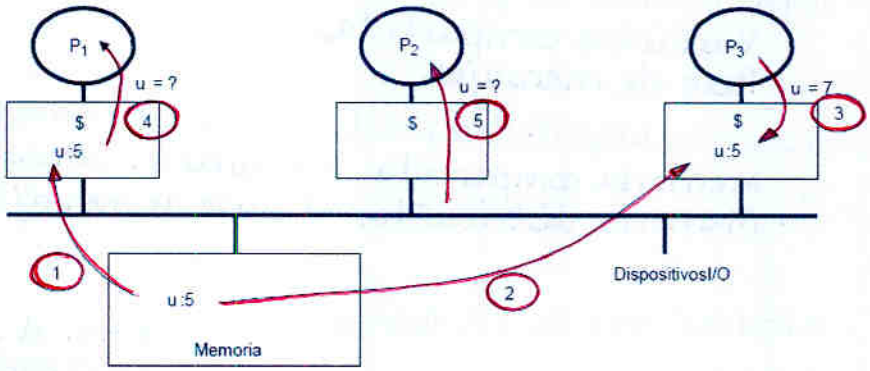
  → Lo más comun es memoria compartida con bus, y modelo de programación de variables compartidas.

Tienen un serio problema: la coherencia de cache

2.1 Coherencia de cache

si cada uP tiene su cache ¿cómo podemos estar seguros de que la que tenemos nosotros es 'la buena'?

ejemplo:
si la secuencia es ①,②,③,④,⑤
¿qué 'u' leeran P1 y P2? ¿5o7?



Habrá que diseñar algoritmos para asegurar la coherencia de las variables compartidas.
Habrá que tener en cuenta si la cache usa writeback o writethrough

Recuerda: en procesadores trabajando individualmente el controlador de cache podía hacer 2 cosas

- Writeback:
  - se traen a la cache los bloques de memoria que el uP utilice
  - si el uP modifica la cache, el controlador marca el bloque con un flag de 'MODIFICADO'
  - cada vez que un bloque deba quitarse de la cache (porque otro quiera entrar) se comprueba el flag MODIFICADO, si está modificado, el bloque se actualiza en memoria.
- Write through:
  - se traen a la cache los bloques de memoria que el uP utilice
  - si el uP modifica la cache, el controlador hace el mismo cambio en la memoria, garantizando que siempre estará actualizada.
    Es el método más sencillo pero mas lento.

Nota: diagramas de estado de un bloque determinado en cache:

→ Nos dicen la forma de actuar del controlador de cache

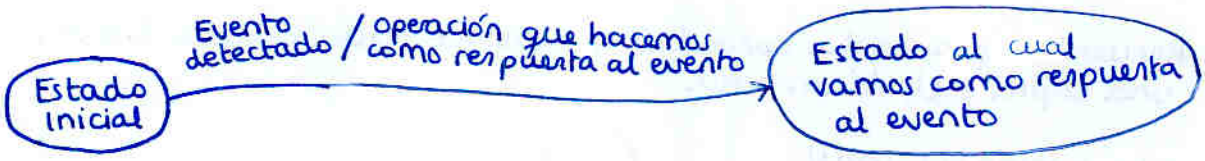
- Cada bloque de memoria tendrá asociado una instancia distinta del diagrama, y a su vez cada controlador de cache tendrá sus propias instancias para cada bloque.

ejemplo: diagrama de estado del bloque N en un sistema con un único procesador usando writethrough.

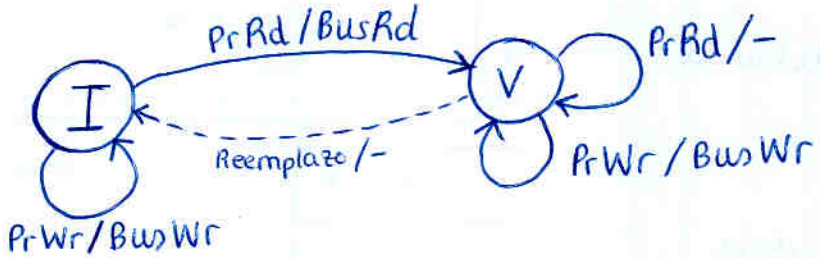
- El controlador de cache puede recibir los siguientes eventos:
  - PrRd : el procesador quiere leer el bloque (específicamente el bloque al que pertenece el diagrama)
  - PrWr : el procesador quiere escribir alguna posición del bloque
- El controlador de cache puede hacer las siguientes operaciones en el bus.
  - — : no hacer nada en el bus de memoria
  - BusRd : leer del bus de memoria el bloque
  - BusWr : escribir en el bus de memoria el bloque
- El bloque tiene dos estados posibles para el controlador
  - I : inválido; no está el bloque en la cache
  - V : válido; está el bloque en la cache

(NOTA: Si se usara writeback existiría un tercer estado M (modificado) para los bloques con ese flag activado.)

- Nomenclatura:



- El diagrama queda:



- Con estos diagramas podemos definir el algoritmo de coherencia de cache que usan los controladores de cache
- En los algoritmos que veremos nunca incluimos el evento reemplazo porque complicaría el diagrama y además resulta que no aporta información del algoritmo

Algoritmo de espionaje (snoopy) ≡ (fisgón)

- Algoritmo para mantener la coherencia de cache que se basa en que el controlador de cache, además de atender a su uP asociado, monitoriza el bus común para saber 'que hacen los demás' y poder actuar en consecuencia

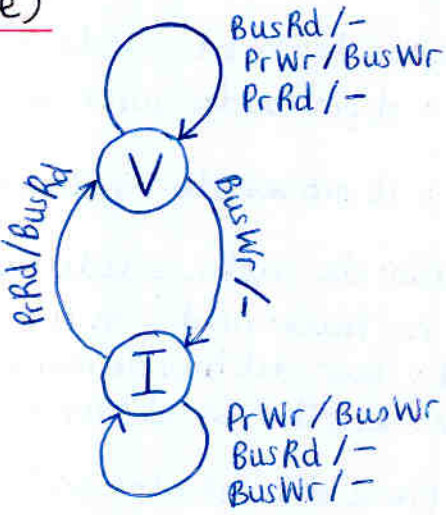
- Con write-through (caso simple)

Estados:
- V: válido
- I: no válido

Eventos generados por el procesador:
- PrRd
- PrWr

operaciones detectadas en el bus:
- BusRd
- BusWr

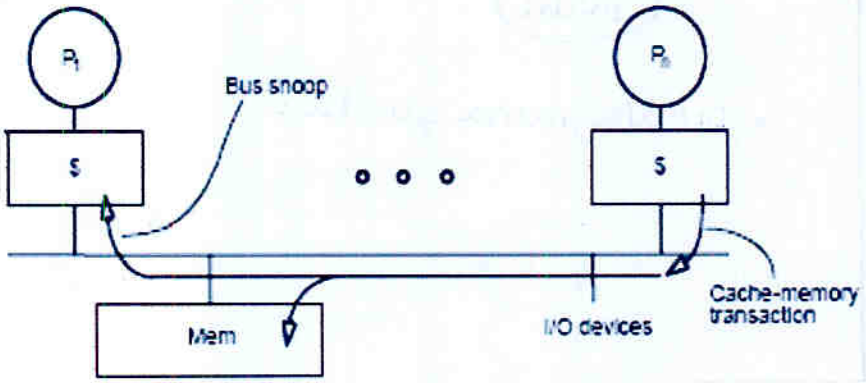
operaciones generadas en el bus:
- BusRd
- BusWr



Notas:
- Es un algoritmo de invalidación, si alguien escribe en un bloque que tengo en cache, lo marcaré como inválido

- Recuerda que cada operador y a su vez cada bloque tienen una copia del diagrama

- Es una opción muy mala porque hay demasiadas operaciones en el bus (se malgasta ancho de banda)



ejemplo:
- Procesador 200MHz
  CPI = 1
  15% instrucciones son store de 8 bytes

⇓

cada procesador genera 30 Mstores o 240 MB datos por segundo

⇓

Un bus de 1 GB/s sólo puede soportar 4 uP sin saturarse (y eso sin contar las lecturas)

solución:
- Usar writeback ya que las escrituras acertadas no usan el bus
- Requiere protocolo mas complejo (ejemplo MSI)

Protocolo writeback con invalidacion → Protocolo MSI

Tiene 3 estados

I : inválido : bloque no presente en cache

S : shared (compartido) : el contenido del bloque que tengo en cache coincide con el de memoria central

M : modificado : He modificado mi bloque; la copia del bloque que tengo en mi cache es la única copia correcta (actualizada) del bloque en todo el sistema. (sólo puede haber como máximo un controlador de cache del sistema que tenga el bloque en este estado.)

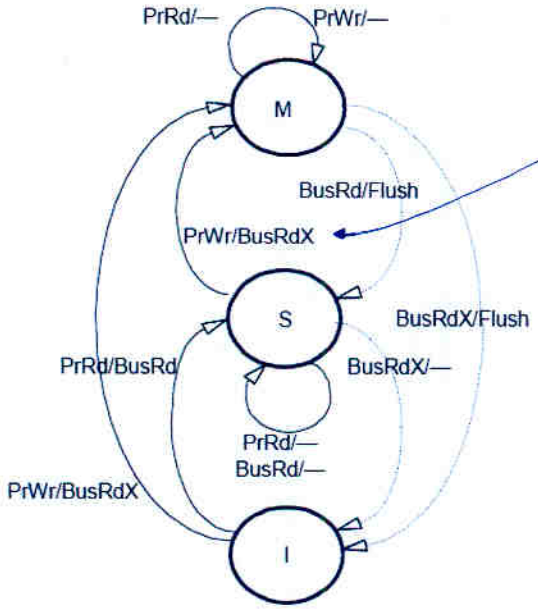
Posibles eventos en el bus :

BusRd : quiero leer el bloque como consecuencia de una operación de lectura

BusRdX : quiero leer el bloque ya que mi procesador ha pedido una instrucción de escribir en él.

BusWB : actualizar el bloque en memoria

Flush : actualizo el bloque en memoria y además se lo doy DIRECTAMENTE a otro controlador que pide la versión actualizada que sólo yo tengo. (es mucho más rápido)



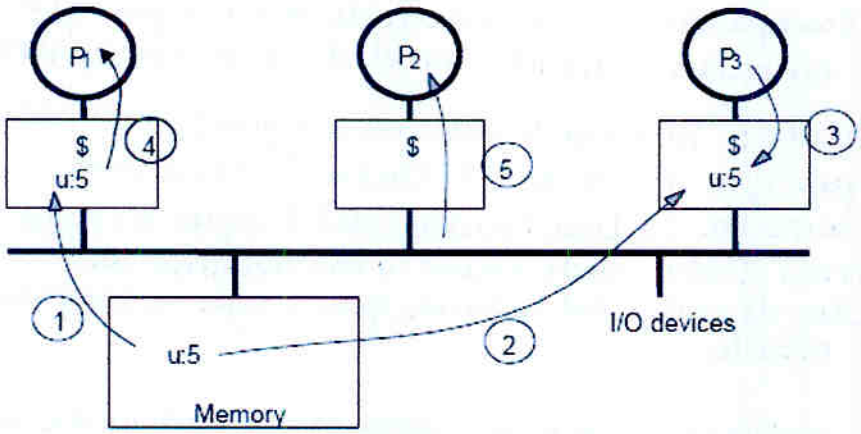
Para cada bloque hay como máximo un controlador en estado M

Éste BusRdX es un poco tonto (malgasto ancho de banda en bus) pero es necesario para que los demás sepan que quiero escribir en el bloque

↓

Posible mejora: incluir evento en el bus para indicar a los demás que quiero modificar. (Bus Upgr)
Habría que estudiar si merece la pena complicar el control (y tal vez necesitaremos incluir una línea de control más en el bus, lo cual no es trivial) a cambio de esa pequeña mejora.

ejercicio: usar MSI

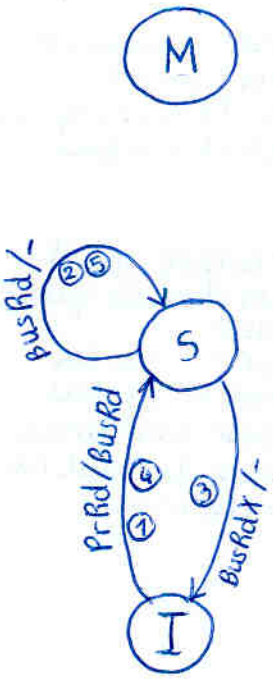


①②...⑤ representa el orden de acontecimientos

Dibujamos el diagrama de estados del bloque que contiene la variable u para cada controlador

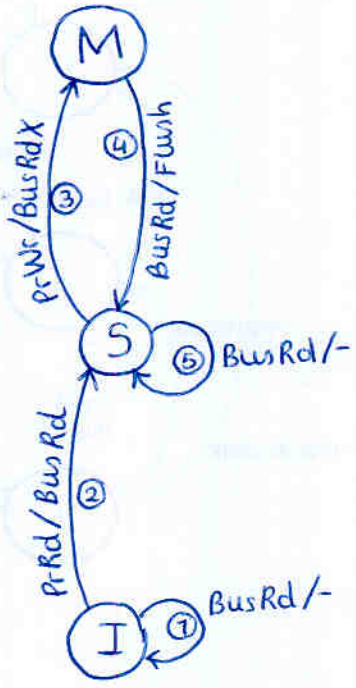
- (cada controlador obviamente puede tener el bloque en un estado distinto de los otros)
- (entre los eventos ①②...⑤ pueden ocurrir mil cosas con otras variables en bloques que no sean el de u, cada bloque sería un ejercicio con sus 3 diagramas de estados)

P1



P2

P3



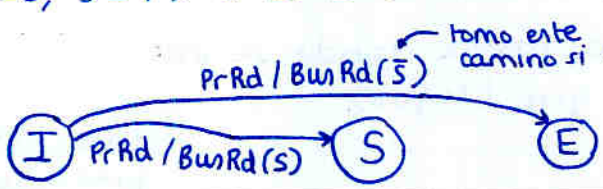
Notas: en ③, Pr1 llega antes a I que Pr3 a M
en ④, Pr1 hace BusRd leyendo los datos que Pr3 está poniendo en el bus con su Flush

Protocolo de invalidación muy utilizado → MESI

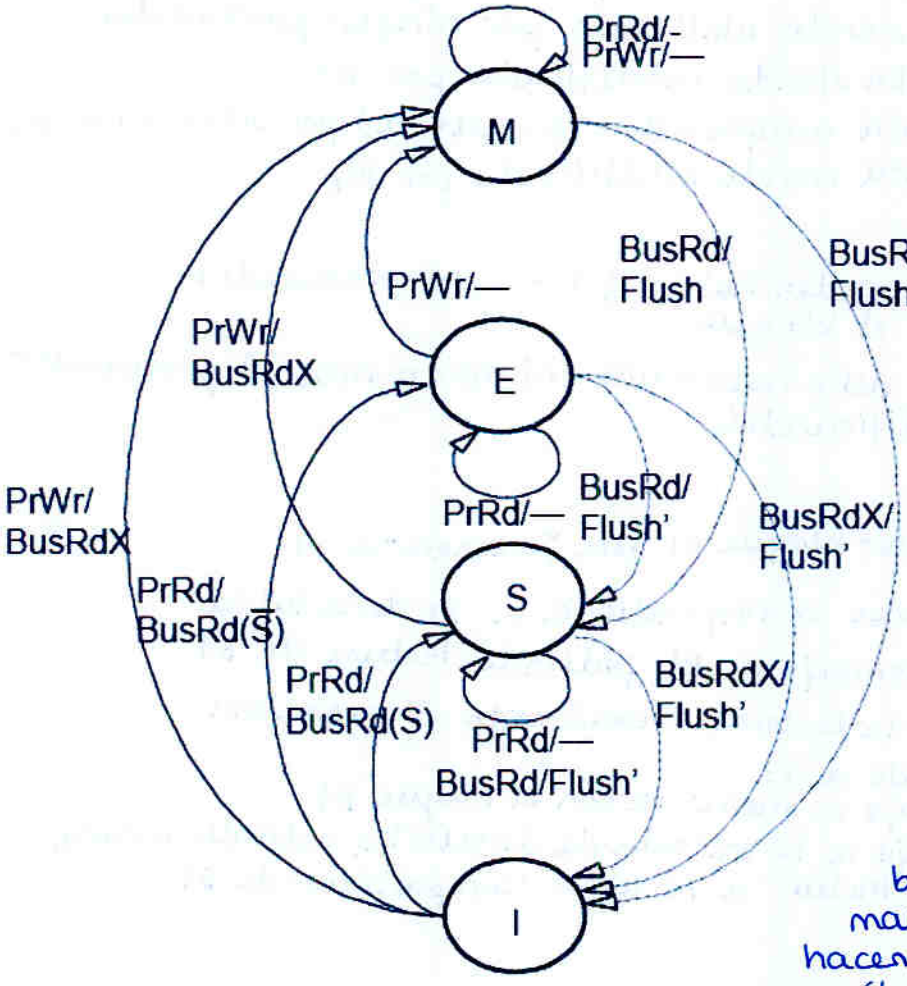
Estado nuevo: E : estoy leyendo este bloque, y además estoy seguro de que soy el único controlador que tengo ese bloque en cache.

Nueva línea del bus (S) : línea NOR cableada, donde los controladores ponen, como respuesta a un BusRd, un 1 si tienen el bloque o un 0 si no lo tienen

se utiliza ésta línea para saber si, estando en I, he de pasar a E (cuando S=0, $\overline{S}=1$) o a S (cuando S=1) al llegar un PrRd.



Nueva acción: Flush' (flush prima) : envío cache a cache sin enviar a memoria (típicamente será mucho más rápido)



Nota: al profesor le escamar los flush' que salen del estado S porque son poco útiles

Ventaja del estado E:
si mi procesador lee, puedo pasar a M sin avisar a nadie

⇓

Es un efecto muy beneficioso si la mayoría de lecturas se hacen sobre bloques que sólo uso yo.

Caso de que la red de interconexión no sea bus

- No cabe la posibilidad de técnicas de espionaje
- ejemplo: memoria distribuida, (ver VII-1), cada µP tiene su memoria, y para acceder a la memoria de otro tengo que pedir permiso.

Soluciones:

- Intercambio de mensajes
- Información común accesible por todos (directorios)

Consiste en una tabla donde se indica qué procesadores están utilizando qué bloques.

	P1	P2	P3	P4	M
B1	X		X	X	
B2					
B3		X			X
B4			X		
B5				X	X

- Cada columna representa un procesador, salvo la última que indica si el bloque está siendo modificado
- Cada fila representa un bloque

B1: bloque compartido (en lectura) por varios procesadores (P1, P3, P4)
B2: bloque no está siendo utilizado por ningún procesador
B3: bloque que está siendo modificado por P2
B4: bloque que está compartido (en lectura) por procesador P3
B5: bloque que está siendo modificado por P4

Nota:

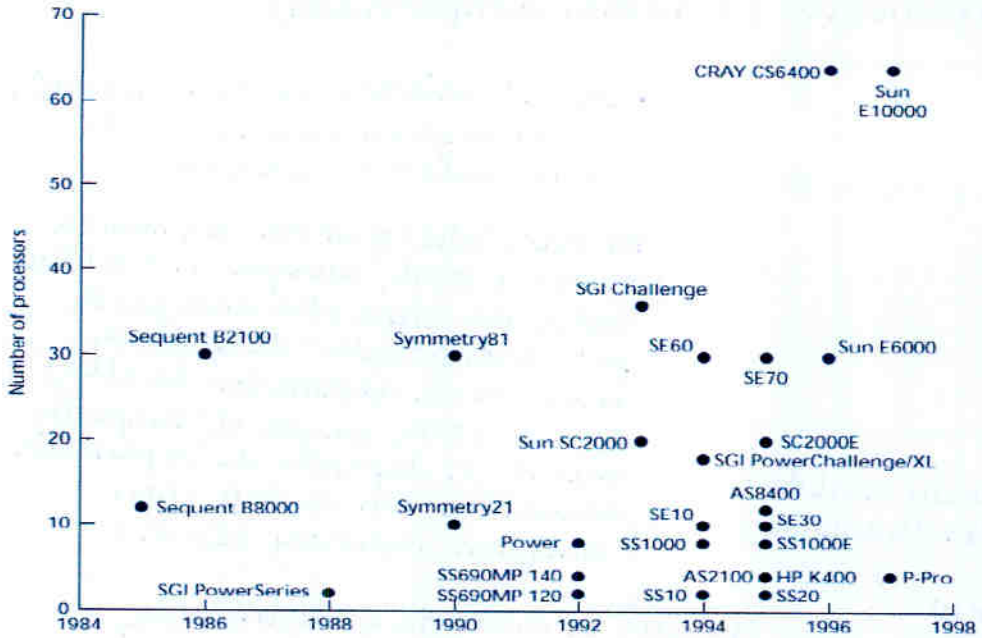
- Si M está a 0 pueden haber 0, 1 o varios procesadores compartiendo el bloque
- Si M está a 1, debe haber uno (ni mas ni menos) procesador que este modificándolo
  → serían entradas incorrectas en la tabla

ejemplo: P2 quiere leer bloque B1 que "pertenece" a P1

1º: P2 lee la entrada correspondiente a B1 de la tabla
2º: P2 manda mensaje a P1 pidiendo lectura de B1
3º: P1 le concede la lectura, mandando 3 mensajes:

1- Mensaje a P2
2- Mensaje a memoria con el bloque B1
3- Mensaje a la entrada de directorios, actualizándola para incluir a P2 como 'compartidor' de B1

ejemplos de SMP comerciales basados en bus



ejemplo de precios:

Intel Xeon DP (2 proc.) ≃ 2500 €
Intel Xeon MP (4 proc.) ≃ 6000 €
AMD Athlon MP (2 proc.) ≃ 2000 €
Power PC Apple G5 (2 proc.) ≃ 3000 €

Procesadores multicore

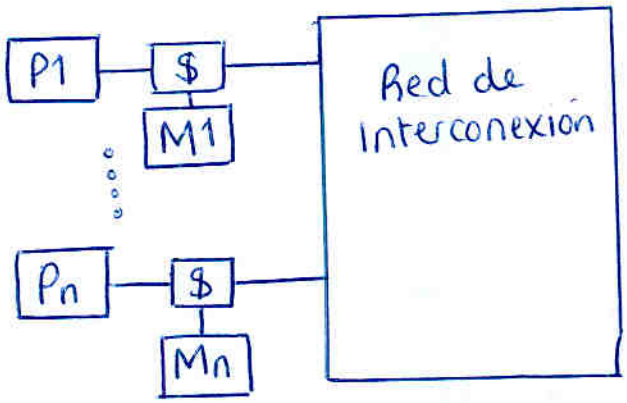
- Han tenido un boom en los últimos dos años (estamos a 2006 ☺)
- Consiste en varias CPU completas en un mismo encapsulado
- Dentro del chip se comportan como un multiprocesador de memoria compartida
  - ↳ Problemas de coherencia de caché
- Están pensados para repartirse la ejecución de múltiples tareas (i.e. útil en SO multitarea)

ejemplos:

Athlon 64 x 2	Cell Processor
Pentium D	Sun Niagara
Opteron D	Intel Itanium
G5 Dual	

3. SSMP : scalable shared memory multiprocessors

- Memoria distribuida, pero todos los µP comparten el mismo espacio de direccionamiento (variables compartidas)



- Los mecanismos de comunicación están implementados en los controladores de cache

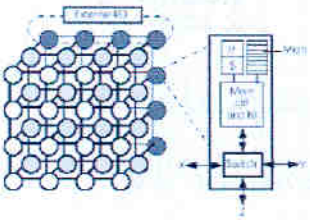
Por tanto, todos los µP ven una misma memoria total, aunque en realidad cada uno tiene sólo una parte, pero el controlador de cache se encarga de abstraerlos de ello;

No son UMA, ya que el tiempo de acceso SI depende de la posición accedida. Son NUMA (non uniform memory access)

Principal ventaja → son muy escalables (fáciles de ampliar) → mayor nº de µP

Aunque la memoria es distribuida, el modelo de programación es memoria compartida, y por tanto funcionan los programas pensados para SMP's.

3. SSMP

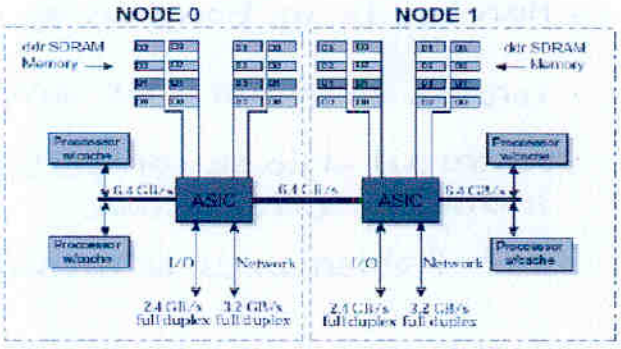


- Cray T3E
  - Hasta 1000 procesadores
  - DEC-Alpha
  - Memoria local
  - Interfaz de red integrado en el sistema de memoria
  - Links 650 MB/s
  - E/S externa desde nodos especiales

http://www.cray.com

3. ccNUMA: aldebaran.upv.es

- SGI Altix 3700: http://www.sgi.com/servers/altix
- Nodos: hasta 4 Itanium 2 y 32GB RAM

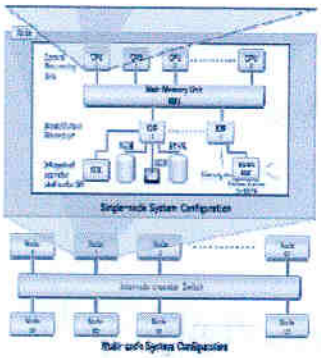


4. DSMP Distributed Memory Scalable Multiprocessor

- Cada procesador tiene su memoria y su E/S
- No hay problemas de coherencia de cache
- modelo de programación: paso de mensajes mediante operaciones de E/S explícitas

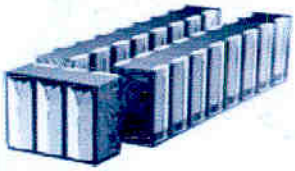
se intenta que la red de interconexión tenga un ancho de banda grande y una latencia muy pequeña de forma que la trx de un mensaje entre 2 nodos cueste parecido a un acceso a memoria.

- NEC SX6 Vector http://www.sw.nec.co.jp/hpc/sx-e/sx6/index.html



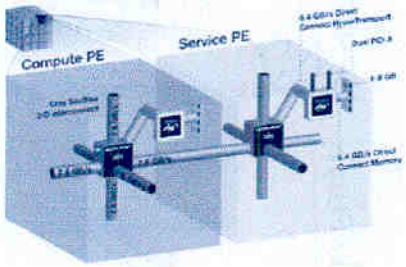
Nodos: SMP

Entre nodos: DSMP



- CRAY XT3 http://www.cray.com/products/xt3/
- Más de 30000 procesadores (Opteron)

Cray XT3 Scalable Architecture



5. Redes de estaciones de trabajo (cluster) (NOW)

- Multiprocesadores de bajo coste
- Aprovechar máquinas conectadas en red de area local para que agrupen y compartan sus recursos
  - → La RI no será tan rápida (ej: ethernet (100/1000 Mb)
  - → El HW no consigue identificar a la red como un único computador, debe ser el SW quien se encargue de ello

ejemplos:

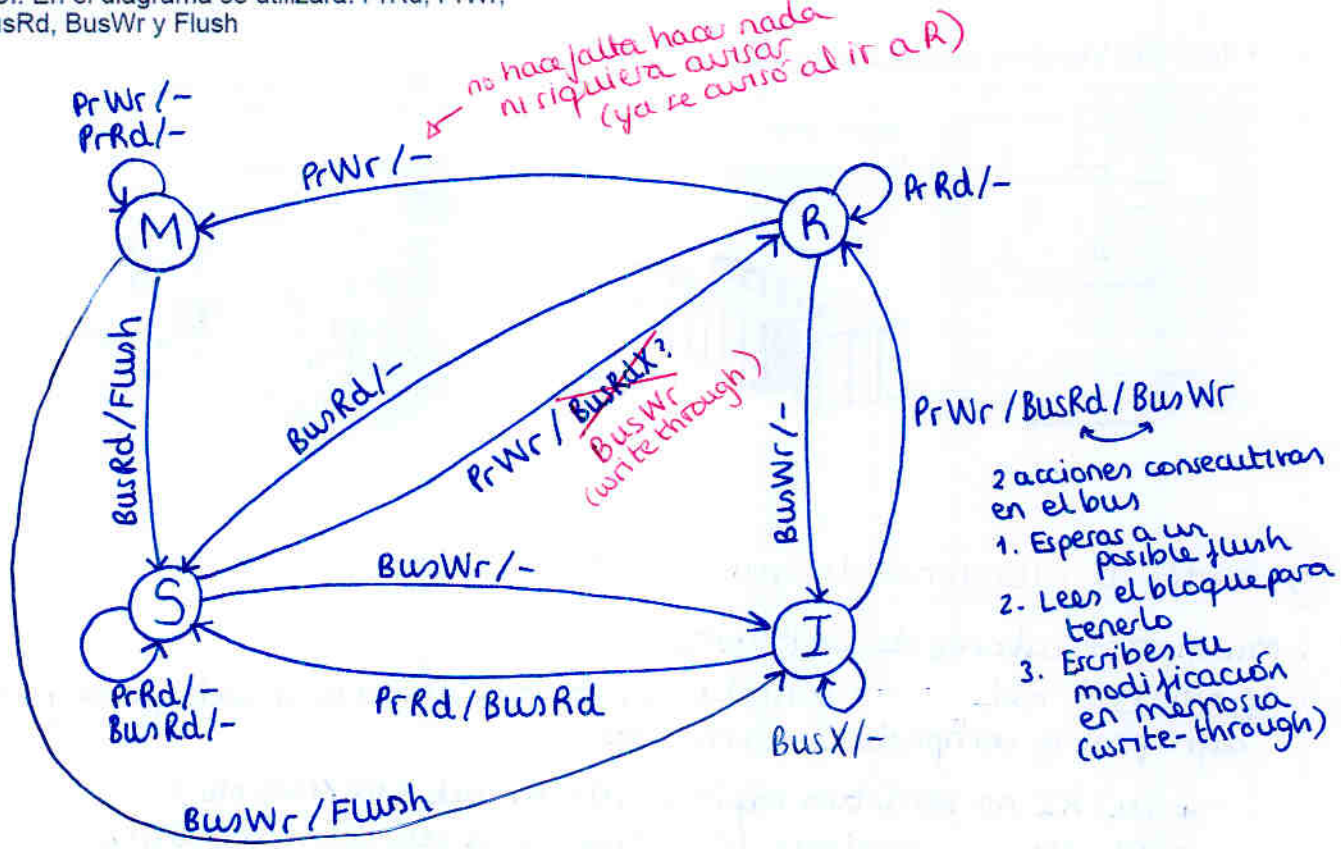
- PVM: parallel virtual machine es un software que crea la abstracción de un único computador
- CONDOR: tiene un pool de tareas que va repartiendo por todas las máquinas para que estén ocupadas siempre

problema:

Se ha observado que los procesadores a veces escriben una única palabra en un bloque de la cache. Para optimizar este caso, en vez de usar siempre write-back, se propone el siguiente protocolo:

1. La primera escritura sobre un bloque el procesador realiza un write-through y pone el bloque en un estado R (reserved).
2. Una escritura en un bloque en estado R realiza una transición al estado M que utiliza write-back.

Dibujar el diagrama de estados modificando el MSI. En el diagrama se utilizará: PrRd, PrWr, BusRd, BusWr y Flush



Examen Julio 2003

ARQUITECTURA DE COMPUTADORES Y SISTEMAS OPERATIVOS I
Julio 2003

1. Comenta, si existen, cuales son las diferencias en la ejecución del siguiente bucle de un guión shell si se cambia el * por un $*:

```
cd /etc
for i in *
do
        echo $i
done
```

(1 pto)

2. Justifica si las siguientes sentencias son ciertas o falsas:
   a. Una matriz de 2 dimensiones de (m,n) elementos se almacena en memoria como un vector de (m * n) elementos consecutivos. El tiempo medio de acceso a los elementos de esa matriz depende en gran medida de la secuencia en que se haga dicho acceso (por filas o por columnas).
   b. Para construir una versión super segmentado de un procesador, es necesario replicar todas las unidades de ejecución presentes en la versión normal.
   c. Se dispone de dos procesadores, uno con una frecuencia de reloj de 2GHz y otro con una frecuencia de reloj de 3GHZ. Si los dos ejecutan el mismo juego de instrucciones, el procesador con una frecuencia mayor tendrá un valor de MIPS más alto.
   d. Segmentar cualquier circuito o unidad funcional implica una mejora del tiempo de ejecución. Esta mejora será como máximo igual al número de etapas de la segmentación.
   e. El esfuerzo (tiempo y dinero) de desarrollo de los componentes de un computador, debe ser igual para cada uno de ellos.

(0,5 ptos por sentencia)

3. Haz una traza de la ejecución del siguiente código si se utiliza Tomasulo para la gestión de las operaciones en coma flotante. Se utiliza DS1 para los saltos y se utilizan cortocircuitos para las operaciones enteras. Inicialmente **R2=2**.

   Operadores disponibles:
   - Sumador/Restador: Tev=2; IR=1 cada ciclo; 2 estaciones reserva
   - Multiplicador: Tev=3; IR=1 cada ciclo; 2 estaciones reserva
   - Memoria: Tev=2; IR1 cada 2 ciclos; 4 estaciones reserva
   - Divisor: Tev=4; IR=1 cada 2 ciclos; 1 estaciones reserva

```
        ADD   R5,R0,R1
        LDF   F0,a(R1)
        LDF   F1,b(R1)
buc:    ADDF  F2,F4,F6
        ADD   R1,R1,#1
        DIVF  F8,F0,F4
        SUB   R2,R2,#1
        ADDF  F4,F0,F1
        MULTF F6,F1,F8
        BNEZ  R2,buc
        SDF   c(R1),F6
        SUB   R5,R1,R5
        SD    q(R1),R5
```

(3 ptos)

Nombre y apellidos:

4. En muchos procesadores modernos se utilizan caches de datos y de instrucciones separadas. Sabiendo que muchos de ellos están segmentados, ¿por qué se utilizan estas caches separadas en vez de una cache mayor unificada?

(1 pto)

5. Un computador genera direcciones de memoria de 16 bits, está provisto de una memoria cache de 2KB, asociativa por conjuntos.
Si la CPU genera las siguientes direcciones de memoria cuando la cache se encuentra en la situación mostrada en la tabla 1:

N°Acceso	Dir Hexadecimal	Dir. Binario
1	0x1E70	0001111001110000
2	0x2050	0010000001010000
3	0x20D0	0010000011010000
4	0x1E77	0001111001110111
5	0x2064	0010000001100100
6	0x20D0	0010000011010000
7	0x21ED	0010000111101101
8	0x2078	0010000001111000
9	0x20D0	0010000011010000

Justifica todas las respuestas, se debe incluir el diagramas de correspondencia de las direcciones de cache y CPU.

a) ¿Cuántos fallos y aciertos se producen?
b) Para cada acceso a memoria, ¿a qué dirección de memoria cache se accede? Si se reemplaza un bloque de caché justifica la elección del mismo.
c) Suponiendo que cuando se rellenó la tabla 1 inicialmente, se hizo un único acceso a la dirección 0 de cada uno de los bloques, indicar las 16 direcciones de memoria (en orden) para que la cache se encuentre en el estado de la Tabla 1. Suponer que los accesos se realizaron por conjuntos (primero los del conjunto cero y al final los del conjunto 3)

CONJUNTO 0		CONJUNTO 1		CONJUNTO 2		CONJUNTO 3	
Bloque	Tag	Bloque	Tag	Bloque	Tag	Bloque	Tag
0	39	0	29	0	02	0	01
1	0F	1	77	1	10	1	64
2	1E	2	20	2	7F	2	03
3	10	3	1E	3	4D	3	5E
LRU: 1-0-3-2		LRU: 2-1-0-3		LRU: 1-2-3-0		LRU: 3-2-1-0	

Tabla 1: Estado de las memorias CAM

0x0	**0000**
0x1	**0001**
0x2	**0010**
0x3	**0011**
0x4	**0100**
0x5	**0101**
0x6	**0110**
0x7	**0111**

0x8	**1000**
0x9	**1001**
0xA	**1010**
0xB	**1011**
0xC	**1100**
0xD	**1101**
0xE	**1110**
0xF	**1111**

Tabla 2: Conversión hexadecimal a binario

(2,5 puntos)

Nombre y apellidos:

ARQUITECTURA DE COMPUTADORES Y SISTEMAS OPERATIVOS I
Julio 2003

1. Comenta, si existen, cuales son las diferencias en la ejecución del siguiente bucle de un guión shell si se cambia el * por un $*:

```
cd /etc
for i in *
do
        echo $i
done
```

(1 pto)

SOLUCIÓN;
Con * se muestran los ficheros del directorio /etc. Este caracter lo cambia el shell por todos los ficheros.
Con $* se hace referencia a todos los parámetros que se le pasaron al guión shell, salvo $0. En este caso saldrian por pantalla $1, $2…

2. Justifica si las siguientes sentencias son ciertas o falsas:
   a. Una matriz de 2 dimensiones de (m,n) elementos se almacena en memoria como un vector de (m * n) elementos consecutivos. El tiempo medio de acceso a los elementos de esa matriz depende en gran medida de la secuencia en que se haga dicho acceso (por filas o por columnas).

Solución: CIERTO
Al utilizar caches, la forma de acceso va a determinar en gran medida el número de aciertos y fallos, lo que implicará un tiempo de acceso efectivo diferente. Si el acceso a los elementos de la matriz coincide con el orden de almacenamiento, habrá un fallo por cada nuevo bloque al que accedamos. Si el acceso lo hacemos en otro orden es más que posible que tengamos muchos más fallos.

   b. Para construir una versión super segmentado de un procesador, es necesario replicar todas las unidades de ejecución presentes en la versión normal.

Solución: FALSO
Los procesadores super segmentados no replican HW. Son procesadores con un gran número de etapas, entre las que se añaden registros. El número y tipo de las unidades son los mismos que en una versión no super segmentada.

   c. Se dispone de dos procesadores, uno con una frecuencia de reloj de 2GHz y otro con una frecuencia de reloj de 3GHZ. Si los dos ejecutan el mismo juego de instrucciones, el procesador con una frecuencia mayor tendrá un valor de MIPS más alto.

Solución: FALSO
El valor de MIPS depende del CPI del procesador y del periodo de reloj. Si la versión a 2 GHZ tiene un CPI menor que la versión de 3GHz, no sólo es posible que no tenga un MIPS más pequeño, lo puede tener mucho mayor. Como ejemplo podemos pensar en el AMD Athlon y el Pentium 4, que a una velocidad de reloj completamente diferente pueden tener el mismo valor de MIPS.

Nombre y apellidos:

d. Segmentar cualquier circuito o unidad funcional implica una mejora del tiempo de ejecución. Esta mejora será como máximo igual al número de etapas de la segmentación.

Solución: FALSO
Segmentar no tiene porque aumentar las prestaciones (tiempo de ejecución) de una unidad. Además la mejora, si existe, dependerá de tener suficientes datos de entrada a procesar. La única parte cierta de la sentencia, es que la mejora siempre será inferior al número de etapas en que se haya segmentado.

e. El esfuerzo (tiempo y dinero) de desarrollo de los componentes de un computador, debe ser igual para cada uno de ellos.

Solución: FALSO
Sólo hay que aplicar la ley de Amdhal. Lo normal es aplicar el mayor esfuerzo a aquel componente que sea el que más tiempo se utilice.

(0,5 ptos por sentencia)

3. Haz una traza de la ejecución del siguiente código si se utiliza Tomasulo para la gestión de las operaciones en coma flotante. Se utiliza DS1 para los saltos y se utilizan cortocircuitos para las operaciones enteras. Inicialmente **R2=2**.

   Operadores disponibles:

   - Sumador/Restador: Tev=2; IR=1 cada ciclo; 2 estaciones reserva
   - Multiplicador: Tev=3; IR=1 cada ciclo; 2 estaciones reserva
   - Memoria: Tev=2; IR1 cada 2 ciclos; 4 estaciones reserva
   - Divisor: Tev=4; IR=1 cada 2 ciclos; 1 estaciones reserva

```
        ADD   R5,R0,R1
        LDF   F0,a(R1)
        LDF   F1,b(R1)
buc:    ADDF  F2,F4,F6
        ADD   R1,R1,#1
        DIVF  F8,F0,F4
        SUB   R2,R2,#1
        ADDF  F4,F0,F1
        MULTF F6,F1,F8
        BNEZ  R2,buc
        SDF   c(R1),F6
        SUB   R5,R1,R5
        SD    q(R1),R5
```

(3 ptos)
(SOLUCIÓN: ver hoja adjunta)

Nombre y apellidos:

NOMBRE: Problema 3. Julio 2003. Tomasulo, DS1, cortocircuitos para enteros

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	ADD R5,R0,R1	IF	ID	EX	MEM	WBE																		
2	LDF F0,a(R1)		IF	ID	ISS op1 a(R1)	L	L'	WBF op1 val																
3	LDF F1,b(R1)			IF	ID	ISS op2 b(R1)	Lv1	L	L'	WBF op2 val														
4 buc	ADDF F2,F4,F6				IF	ID	ISS op3 val,val	S1	S2	S2	WBF op3 val													
5	ADD R1,R1,#1					IF	ID	EX	MEM	WBE														
6	DIVF F8,F0,F4						IF	ID	ISS op4 val,val	D1	D1'	D2	D2'	WBF op4 val										
7	SUB R2,R2,#1							IF	ID	EX	MEM	WBE												
8	ADDF F4,F0,F1								IF	ID	ISS op5 val,val	S1	S2	S2	WBF op5 val									
9	MULTF F6,F1,F8									IF	ID	ISS op6 val op4	Mv1	M1	M2	M3	WBF op6 val							
10	BNEZ R2,buc										IF	ID PC←buc	EX	M	WBE									
11	SDF c(R1),F6										DS1	IF	ID	ISS op7 op6	Lv1	Lv1	L	L'	WBF op7 val					
4 buc	ADDF F2,F4,F6												IF	ID	ISS op8 val op6	Sv1	S1	S2	S2	WBF op8 val				
5 P-3	ADD R1,R1,#1													IF	ID	EX	MEM	WB						

R1 : val ⑨val
R2 : 2 ⑪1 ⑥0
R5 : val

F0 : val ④op1 ⑦valor
F1 : val ⑤op2 ⑨valor
F2 : val ⑥op3 ⑩valor ⑭op8 ⑲val
F4 : val ⑩op5 ⑭val ⑱op10 ㉓val
F6 : val ⑰op6 ⑲op11 ㉕val
F8 : val ⑧op4 ⑬val ⑯op9 ㉑val

NOMBRE:

		14	15	16	17 / 4	18 / 5	19 / 6	20 / 7	21 / 8	22 / 9	23 / 10	24 / 11	25 / 12	26 / 13	27 / 14	28 / 15	29 / 16	30 / 17	31 / 18	32 / 19	33 / 20	34 / 21	35 / 22	36 / 23
6	DIVF F8,F0,F4	IF	ID	ISS op9 val,val	D1	D1'	D2	D2'	WBF op9 val															
7	SUB R2,R2,#1		IF	ID	EX	MEM	WBE																	
8	ADDF F4,F0,F1			IF	ID	ISS op10 val,val	S1	S2	S2	WBF op10,val														
9	MULTF F6,F1,F8				IF	ID	ISS op11 val,op9	Mv1	M1	M2	M3	WBF op11 val													
10	BNEZ R2,buc					IF	ID PC←11	EX	MEM	WBE														
11	SDF C(R1),F6						IF	ID	ISS op12 op11	Lv1	Lv1	L	L'	WBF op12,val										
12	SUB R5,R1,R5					DS1		IF	ID	EX	MEM	WBE												
13	SD 9(R1),R5								IF	ID	EX	MEM	WBE											

p-4

1	IF	ID	EX	M	WBE																						
2		IF	ID	ISS	L	L	WBF																				
3			IF	ID	ISS	lv1	L	L	WBF																		
4				IF	ID	ISS	A1	A2	A2	WBF																	
5					IF	ID	EX	M	WBE																		
6						IF	ID	ISS	D1	D1	D2	D2	WBF														
7							IF	ID	EX	M	WBE																
8								IF	ID	ISS	A1	A2	A2	WBF													
9									IF	ID	ISS	mv1	M1	M2	M3	WBF											
10										IF	ID	EX	M	WBE													
11											IF	ID	ISS	lv1	lv1	L	L	WBF									
4												IF	ID	ISS	av1	A1	A2	A2	WBF								
5													IF	ID	EX	M	WBE										
6														IF	ID	ISS	D1	D1	D2	D2	WBF						
7															IF	ID	EX	M	WBE								
8																IF	ID	ISS	A1	A2	A2	WBF					
9																	IF	ID	ISS	mv1	M1	M2	M3	WBF			
10																		IF	ID	EX	M	WBE					
11																			IF	ID	ISS	lv1	lv1	L	L	WBF	
12																				IF	ID	EX	M	WBE			
13																					IF	ID	EX	M	WBE		

4. En muchos procesadores modernos se utilizan caches de datos y de instrucciones separadas. Sabiendo que muchos de ellos están segmentados, ¿por qué se utilizan estas caches separadas en vez de una cache mayor unificada?

(1 pto)

Solución: La separación de caches en datos + instrucciones se debe principalmente a dos motivos. Uno de ellos es la resolución de riesgos estructurales en los procesadores segmentados y super escalares. Por otra parte, la forma de acceso a instucciones y a datos es diferente.
Las instrucciones se acceden típicamente de forma secuencial, forma de acceso que optimiza el uso de la cache y que dará como resultados pocos fallos (salvo saltos). Por otra parte los datos se acceden de formas más diversas y en varias zonas de memoria simultáneamente.
Separar las dos caches permite que el diseño del controlador de cache sea diferente para instrucciones y para datos.

5. Un computador genera direcciones de memoria de 16 bits, está provisto de una memoria cache de 2KB, asociativa por conjuntos.
Si la CPU genera las siguientes direcciones de memoria cuando la cache se encuentra en la situación mostrada en la tabla 1:

N°Acceso	Dir Hexadecimal	Dir. Binario
1	0x1E70	0001111001110000
2	0x2050	0010000001010000
3	0x20D0	0010000011010000
4	0x1E77	0001111001110111
5	0x2064	0010000001100100
6	0x20D0	0010000011010000
7	0x21ED	0010000111101101
8	0x2078	0010000001111000
9	0x20D0	0010000011010000

Justifica todas las respuestas, se debe incluir el diagramas de correspondencia de las direcciones de cache y CPU.

a) ¿Cuántos fallos y aciertos se producen?
b) Para cada acceso a memoria, ¿a qué dirección de memoria cache se accede? Si se reemplaza un bloque de caché justifica la elección del mismo.
c) Suponiendo que cuando se rellenó la tabla 1 inicialmente, se hizo un único acceso a la dirección 0 de cada uno de los bloques, indicar las 16 direcciones de memoria (en orden) para que la cache se encuentre en el estado de la Tabla 1. Suponer que los accesos se realizaron por conjuntos (primero los del conjunto cero y al final los del conjunto 3)

CONJUNTO 0		CONJUNTO 1		CONJUNTO 2		CONJUNTO 3	
Bloque	Tag	Bloque	Tag	Bloque	Tag	Bloque	Tag
0	39	0	29	0	02	0	01
1	0F	1	77	1	10	1	64
2	1E	2	20	2	7F	2	03
3	10	3	1E	3	4D	3	5E
LRU: 1-0-3-2		LRU: 2-1-0-3		LRU: 1-2-3-0		LRU: 3-2-1-0	

Tabla 1: Estado de las memorias CAM

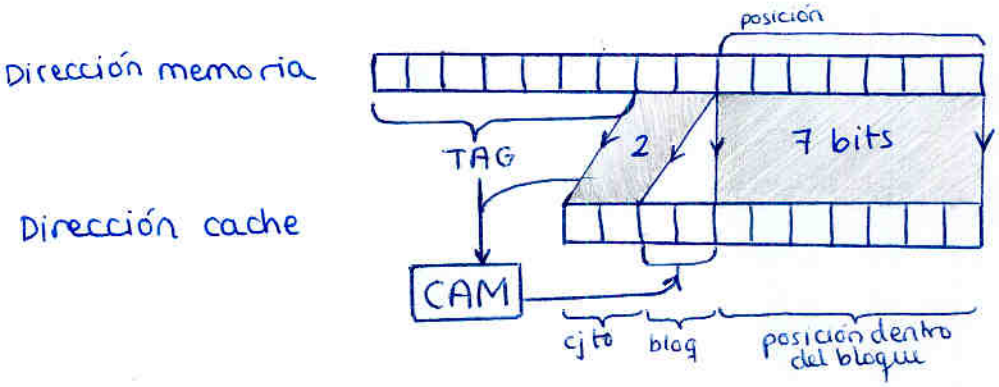
(2,5 puntos)

Nombre y apellidos:

5. direcciones de memoria 16 bits

cache 2KB → direcciones de cache 11 bits

4 conjuntos con 4 bloques cada uno
⇒ 2 bits para el conjunto
⇒ 2 bits para el bloque dentro del conjunto
⇒ 7 bits para la posición dentro del bloque



Acceso 1: 0001111001110000

0x0F | cjto 0 | 1110000

CAM → bloque 1 | 0001

← dirección cache

Acierto
LRU$_0$: 0-3-2-1

Acceso 2: 0010 0000 0101 0000

0x10 | cjto 0

CAM → bloque 3

0011 1010000

Acierto
LRU$_0$: 0-2-1-3

Acceso 3: 0010 0000 1101 0000

0x10 | cjto 1

CAM → fallo → reemplazo bloque 2

0110 1010000

Fallo
se sustituye el bloque 2, por ser el menos recientemente utilizado
se pone tag 0x10
LRU$_1$: 1-0-3-2

Acceso 4: 0001 1110 0111 0111

0x0F | cjto 0

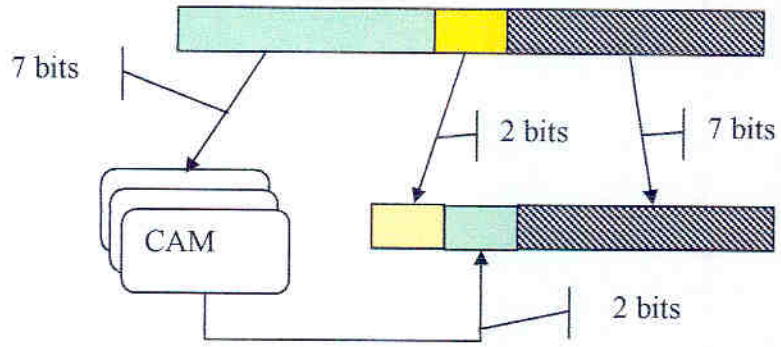
CAM → bloque 1

0001 1110111

Acierto
LRU$_0$: 0-2-3-1

etc...

Solución: De la tabla podemos observar que disponemos de una cache con cuatro conjuntos y cuatro líneas (bloques en cache) por conjunto. Esto significa que necesitamos 2 bits para direccional el conjunto y otros 2 para el bloque (línea) dentro del conjunto. Si la dirección de memoria cache es de 11 bits, quedan 7 bits para la posición dentro del bloque.
Aplicando esta información a la dirección del procesador (16 bits) significa que el TAG será de 16-7-2= 7 bits,
El esquema de asignación es:



Por lo tanto los accesos que se realizan a los siguientes conjuntos con los siguientes TAGS:

N°Acceso	TAG	Conjunto
1	0001111 (0x0f)	00
2	0010000 (0x10)	00
3	0010000 (0x10)	01
4	0001111 (0x0f)	00
5	0010000 (0x10)	00
6	0010000 (0x10)	01
7	0010000 (0x10)	11
8	0010000 (0x10)	00
9	0010000 (0x10)	01

Los resultados de estos accesos y las acciones a realizar son:

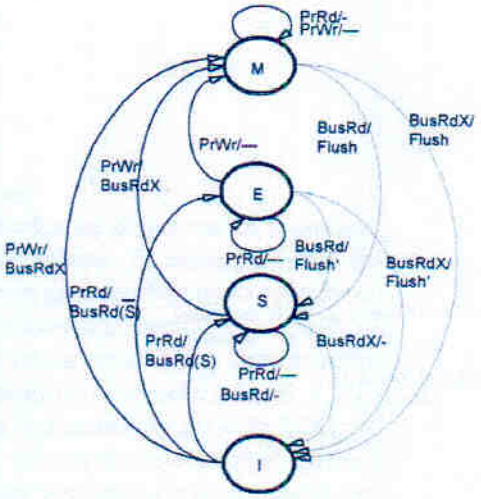
N°Acceso	Result.	LRU	Acción
1	Acierto	0-3-2-1	-
2	Acierto	0-2-1-3	-
3	Fallo	1-0-3-2	Sustitución línea 2 cjto 1. Nuevo TAG: 0010000
4	Acierto	0-2-3-1	-
5	Acierto	0-2-1-3	-
6	Acierto	1-0-3-2	-
7	Fallo	2-1-0-3	Sustitución línea 3 cjto3. Nuevo TAG: 0010000
8	Acierto	0-2-1-3	-
9	Acierto	1-0-3-2	-

Nombre y apellidos:

ACSOI

Enero 2006

5. Un sistema multiprocesador simétrico de 3 procesadores utiliza un sistema de antememorias coherente basado un bus snoopy y protocolo MESI de la figura. Suponiendo que la cache de cada procesador contiene un único bloque de cuatro palabras y que las palabras W0-W3 pertenecen al mismo bloque X y que W4-W7 pertenecen al bloque Y, se pide:
Analizar la siguiente secuencia de accesos indicando, que sucede en cada una de las caches, en que estado se encontrará el bloque después de la ejecución de cada instrucción y que bloque contiene. Si es un acierto (A) o es un fallo (F). Indicar también las operaciones de bus que se generan. No hay que poner las operaciones relacionadas con remplazamientos ni el estado I
IMPORTANTE: Una casilla en blanco en la columna bloque implica que todos los bloques de ese procesador están en estado I.



Fallo / Acierto

Línea S

	P1				S	P2				S	P3				S	origen de datos del bus
	Inst	Bloque	Estado	Op.Bus		Inst	Bloque	Estado	Op.Bus		Inst	Bloque	Estado	Op.Bus		
1	st W6	Y	M/F	BusRdx	-		—	-	-	0		—	—	—	0	Mem
2		Y	I	Flush	1	st W7	Y	M/F	BusRdX	-		—	—	—	0	P1
3					0		Y	S	Flush	1	ld W6	Y	S/F	BusRd	-	P2
4					0	ld W2	X	E/F	BusRd	-		Y	S		0	Mem
5						ld W3	X	E/A	—	-		Y	S		-	-
6	ld W2	X	S/F	BusRd	-		X	S	Flush'	1		Y	S		0	P2
7		X	S			ld W0	X	S/A	—			Y	S			-
(*) 8		X	I		-	st W2	X	M/A	BusRdX	1		Y	S		-	— (*) ← o mem (depende de la implem.)
9					0		X	S	Flush*	1	ld W2	X	S/F	BusRd	-	P2
10	st W2	X	M/F	BusRdX	0		X	I	—	-		X	I	—	-	mem
11		X	S	Flush	1	ld W3	X	S/F	BusRd	-					0	P1
12		X	S		0		X	S		0	ld W5	Y	E/F	BusRd	-	Mem
13	st W2	X	M/A	BusRdX	1		X	I				Y	E			-
14		X	S	Flush	1	ld W3	X	S/F	BusRd	-		Y	E		0	P1
15		X	S				X	S			st W7	Y	M/A	BusRdX	1	

2 ptos

↑ a la cache le cabe un UNICO bloque

Examen Julio 2005

ACSOI

Julio 2005

1. Sea la siguiente secuencia de órdenes que se ejecutan en un shell tipo BASH:

```
Orden parmetro > ftemp
Orden par2 < ftemp > ftemp2
grep "cut*re" <ftemp2 >result_file
```

Re escríbela de las siguientes formas que:

a) El archivo result_file contenga lo mismo que con las tres líneas y sólo se escriba una línea de órdenes y no se creen archivos temporales

b) Los mismo que a) pero que además la salida de error de todas la ordenes se concatene en un único fichero de error llamado "errores.txt"

0,5 ptos

a)

```
Orden parámetro | Orden par2 | grep "cut*re" > result_file
```

b)

```
(Orden parámetro | Orden par2 | grep "cut*re" > result_file) 2> errores.txt
```

2. La empresa DIGIPRINT dispone de una imprenta de altas prestaciones con un sistema de impresión monitorizado por un procesador en tiempo real. El sistema corrige el color de las copias conforme las imprime. Cada copia impresa supone tres operaciones:

- Un proceso P de generación de una copia impresa que incorpora el escaneado de la hoja generada. La duración de este proceso es invariable, pues depende de un sistema mecánico y óptico fijo.
- Un proceso A de análisis de la imagen obtenida en la fase P. Este proceso es realizado por un sistema de n procesadores digitales de señal (*DSP*) que operan en paralelo. La productividad del proceso es proporcional al número de procesadores *DSP* instalados.
- Un proceso C de ajuste de parámetros de impresión que debe de realizarse a partir de los datos suministrados por A y que debe realizarse antes de imprimir una nueva hoja. Este proceso lo realiza un procesador que funciona a 500 MHz.

Inicialmente, se ajusta el sistema para que imprima N hojas por segundo. Para conseguirlo, se observa que son necesarios un mínimo de 20 procesadores *DSP* para que el trabajo se realice en el plazo disponible. Con dicho número de DSP, los procesos P, A y C ocupan (respectivamente) el 30%, el 40% y el 30% del tiempo de operación total del sistema y no sobra tiempo para comenzar a imprimir una nueva hoja.

El equipo completo cuesta 10,000 euros. Se pretende ahora multiplicar por 2 la productividad global (que se mide en copias por segundo). Para ello, se puede reemplazar el procesador actual por otro con un reloj a 1 GHz, que ejecuta el mismo programa que el anterior (con el mismo CPI), por un coste de 1000 euros, y si es necesario, la adquisición de procesadores DSP a razón de 100 euros la unidad.

Calcula utilizando la ley de Amhdal:

a) Si no se sustituye el procesador, ¿cuál es el mínimo número de procesadores *DSP* necesarios para conseguir la nueva productividad?

b) Si se sustituye el procesador por el modelo más rápido, ¿cuál será el mínimo número de procesadores *DSP* necesarios para conseguir dicha productividad?

c) Qué configuración consigue la productividad propuesta con el coste mínimo, teniendo en cuenta que cada hoja a imprimir es un trabajo independiente. Considera que eliminar procesadores DSP no afecta al coste.

Justifica cualitativa y cuantitativamente todas las respuestas.1,5 ptos

NOMBRE:

a)
La Ley de Amdhal dice que por mucho que mejoremos una parte de un sistema, la parte no mejorada impone un tope en la mejora global. Aplicando la ley de Amdhal al caso del sistema de impresión, si mejoramos los DSP la fórmula quedara como sigue:

$$T' = (1-0.4)\ T + 0.4 * T * S^{-1}$$

Aunque $S \rightarrow \infty$, el tiempo global después de mejorar los DSP (T') nunca será menor que 0,6T, por lo tanto no se puede conseguir una mejora de 2.
Esta opción no permite conseguir la ganancia deseada por sí sola.

b)
En este caso vamos a mejorar el 70% del sistema. Por lo tanto el tiempo después de la mejora será igual a la suma de los tiempos de cada parte:

$$T' = (1-0,7)\ T + 0,5 * 0,3 * T + S_d^{-1} * 0.4 * T$$

Siendo S_a la mejora aplicada a los DSP. Sustituyendo para averiguar S_d:

$$T' = T(0,45 + S_d^{-1} * 0.4)$$

Como queremos una mejora global de 2:

$$S_d = 0,4/0.05 = 8$$

Luego necesitaremos 160 DSP, o lo que es lo mismo, comprar 140 DSP.

c) Solución para obtener la mejora de 2 con coste mínimo

La opción a) es inviable.
La opción b) supone un coste de 1000 € para el procesador más (140*100€) para los DSP, lo cual suma 15000 €.
Si lo que queremos es mejorar la productividad global de la empresa y como los trabajos de impresión son independientes, la opción más barata sería comprar un segundo sistema de impresión como el original, COSTE: 10000€

3. En los siguientes tipos de instrucciones del DLX, ¿qué formas de direccionamiento se soportan? (pon un ejemplo y las limitaciones de cada uno de ellos)

	Indexado	Inmediato	Rel. a PC	Absoluto
Aritmética	NO	SI ADD R3,R3,#val val en 16 bits.	NO	NO
Salto condicional	NO	NO	SI BNEZ RN,desplaz desplaz es un valor de 16 bits	NO

1 pto

NOMBRE:

4. Un procesador se ha segmentado en K etapas. Se supone que la ganancia máxima de prestaciones al ejecutar un programa será igual al número de etapas (K). Se han hecho pruebas y la ganancia es menor que K.
    a. Escribir la expresión de la ganancia (SpeedUp) obtenida teniendo en cuenta que el programa es el mismo y se compone de N instrucciones. Partir de la expresión del tiempo de ejecución de un programa.
    b. Cuales pueden ser las causas para que la ganancia sea menor que K.
    c. ¿Se puede conseguir una ganancia mayor que K?

Justificar las respuestas

1 pto

a)

El tiempo de ejecución de un programa se puede expresar como $T = I * CPI * Tc$, siendo I el número de instrucciones, CPI el número medio de ciclos de reloj por instrucción y Tc el periodo de reloj del procesador utilizado. Por lo tanto, la mejora será:

$$S = \frac{T}{Tseg} = \frac{I_1 * CPI_1 * Tc_1}{I_2 * CPI_2 * Tc_2} = \frac{CPI_1 * Tc_1}{CPI_2 * Tc_2}$$

Si el programa es el mismo $I_1=I_2$.

b)

Existen varias causas que se pueden combinar para que la ganancia sea menor que K:

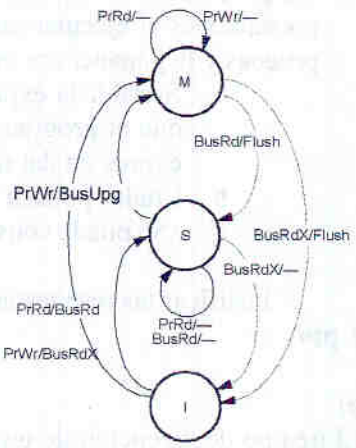
- $CPI_2>1$: El CPI2 normalmente será mayor que 1 por los riesgos de la segmentación
- $CPI_1<K$: No todas las instrucciones del no segmentado tardan K ciclos de reloj, por ejemplo en el DLX, con K=5 el CPI del no segmentado es algo más de 4 ya que sólo load tarda 5 ciclos.
- $Tc_1>Tc_2$: Por culpa del HW que se ha añadido al segmentar (registros) o que las etapas del segmentado no tengan la misma duración.

c)

Sólo con la segmentación no sería posible, pero si además de segmentar hacemos una implementación superescalar, el CPI2 puede ser menor que 1.

NOMBRE:

5. Un procesador "multicore" equivalente a un sistema multiprocesador simétrico de 3 procesadores utiliza un sistema de antememorias coherente basado un bus snoopy y protocolo MSI. Suponiendo que la cache de cada "core" contiene un único bloque de cuatro palabras y que las palabras W0-W3 pertenecen al mismo bloque X y que W4-W7 pertenecen al bloque Y, se pide:
Analizar la siguiente secuencia de accesos indicando, que sucede en cada una de las caches, en que estado se encontrará el bloque después de la ejecución de cada instrucción y que bloque contiene. Si es un acierto o es un fallo. Indicar también las operaciones de bus que se generan. No hay que poner las operaciones relacionadas con remplazamientos ni el estado I. Cada fila de la tabla debe contener información de qué bloque hay en cada una de las antememorias y en que estado se encuentra para ese instante de tiempo (justo después de la ejecución de la instrucción), y qué operación hace en el bus cada controlador de cache, un guión (-) significaría que el controlador de ese "core" no usa el bus en ese ciclo.



	Core1				Core2				Core3				origen de datos
		Bloque	Estado	Op.Bus		Bloque	Estado	Op.Bus		Bloque	Estado	Op.Bus	
1	st W0	X	M/Fallo	BusRdx				-				-	Mem
2		X	M					-	st W7	Y	M/Fallo	BusRdx	Mem
3	ld W1	X	M/Ac.	-				-		Y	S	-	-
4		X	S	Flush	ld W2	X	S/Fallo	BusRd		Y	S	-	Core1
5		X	S	-	ld W7	Y	S/Fallo	BusRd		Y	S	Flush	Core3
6	ld W6	Y	S/Fallo	BusRd		Y	S	-		Y	S	-	Mem
7		Y	S	-	ld W0	X	S/Fallo	BusRd		Y	S	-	Mem
8		Y	S	-	st W2	X	M/Ac.	BusUpg		Y	S	-	-
9	ld W2	X	S/Fallo	BusRd		X	S	Flush		Y	S	-	Core2
10		X	S	-		X	S	-	ld W6	Y	S/Ac.	-	-
11		X	S	-	ld W5	Y	S/Fallo	BusRd		Y	S	-	Mem
12		X	S	-		Y	S	-	ld W5	Y	S/Ac.	-	-
13	st W5	Y	M/Fallo	BusRdx		-	-	-		-	-	-	Mem
14		Y	M	-	ld W3	X	S/Fallo	BusRd		-	-	-	Mem
15		Y	S	Flush		X	S	-	ld W7	Y	S/Fallo	BusRd	Core1
16		Y	S	-	ld W6	Y	S/Fallo	BusRd		Y	S	-	Mem
17		Y	S	-		Y	S	-	ld W2	X	S/Fallo	BusRd	Mem

NOTA: A todos los efectos un "core" en este ejercicio es equivalente a un procesador en un SMP-

2 ptos

NOMBRE:

6. Sea una unidad segmentada con los siguientes operadores multiciclo:
   Load en coma flotante (Tev= 2 IR=1/2 ciclos)
   Sumador/restador (Tev=2 IR=1/1 ciclos)
   Multiplicador (Tev=4 IR=1/1 ciclos)
   Divisor (Tev=8,IR=1/4 ciclos)

Para la siguiente secuencia de instrucciones:

```
ADDF    F2,F4,F8
LDF     F4,100(R1)
LDF     F8,200(R1)
MULTF   F10,F4,F8
DIVF    F10,F10,F2
ADDF    F6,F10,F2
STF     100(R1),F6
ADDF    F10,F6,F2
ADDF    F6,F4,F8
SUBF    F6,F6,F2
LDF     F2,100(R1)
ADDF    F8,F2,F10
```

Si se utiliza una unidad de ejecución con Tomasulo en la que todos los operadores están libres y que los valores de los registros son:

F4 = 1 F8=2

Los demás registros están a cero, incluyendo R1.

En memoria:

Mem[100]=3 Mem[200]=6

Se pide:

- Esquema de ejecución con Tomasulo sin ningún tipo de cortocircuitos.
  Se contesta en la hoja adjunta, indicando la evolución de las marcas en los operadores virtuales, en los registros y en la memoria.
  No se pueden utilizar más operadores virtuales que los indicados.

NOTAS:

El ciclo 1 de ejecución se corresponde con la llegada a la etapa de Issue de la primera instrucción, esta etapa todavía no se ha ejecutado, por lo que habrá que describir su efecto en los registros, operadores virtuales o memoria.

Realizar el esquema de ejecución hasta acabar la ejecución de las instrucciones o llenar una hoja de resultados (23 ciclos).

En las casillas de las etapas ISSUE aparece la siguiente información:

Instrucción I	...	ISS nombre_operacion operando1,operando2	...

En las casillas de los operadores virtuales debe aparece la siguiente información:

Op Virtual	...	Nombre_Operación operando_1 openrando_2	...

2ptos

NOMBRE:

NOMBRE:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
M[200]	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
M[100]	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
F2	a1	a1	a1	3	3	3	3	3	3	3	3	a2	a2	a2	a2	a2	a2	a2	a2	a2	6	6	6
F4	1	lo1	lo1	lo1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
F6	0	0	0	0	0	a2	a2	a2	a4	a5	a5	a5	a5	9	9	9	9	9	9	9	9	9	9
F8	2	2	lo2	lo2	lo2	lo2	3	3	3	3	3	3	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1
F10	0	0	0	m1	d1	d1	d1	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3	a3
Dv1						d1 m1 3	d1 m1 3	d1 m1 3	d1 m1 3	d1 m1 3													
Mv1					m1 3 lo2	m1 3 lo2																	
Av1							a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a2 d1 3	a1 a2 a3	a1 a2 a3	a1 6 a3	a1 6 a3	a1 6 a3
Av2									a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3	a3 a2 3			
Lv1				lo2 M[100]				st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2	st1 M[100] a2			
Lv2																							

NOMBRE:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
ADDF	ISS a1 1,2	A1	A2	WB a1=3																			
LDF	ID	ISS lo1 M[200]	L	L	WB lo1=3																		
LDF	IF	ID	ISS lo2 M[100]	Lv1	L	L	WB lo2=3																
MULTF		IF	ID	ISS m1 lo1,lo2	Mv1	Mv1	M1	M2	M3	M4	WB m1=9												
DIVF			IF	ID	ISS d1 m1,3	Dv1	Dv1	Dv1	Dv1	Dv1	D1	D1	D1	D1	D2	D2	D2	D2	WB d1=3				
ADDF				IF	ID	ISS a2 d1,3	Av1	Av1	Av1	Av1	Av1	Av1	Av1	Av1	Av1	Av1	Av1	Av1	A1	A2	WB a2=6		
STF					IF	ID	ISS st1 M[100], a2	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	Lv1	L	L	WB
ADDF						IF	ID	ISS a3 a2,3	Av2	Av2	Av2	Av2	Av2	Av2	Av2	Av2	Av2	Av2	Av2	Av2	A1	A2	A2
ADDF							IF	ID	ISS a4 3,3	A1	A2	WB a4=6											
SUBF								IF	ID	ISS a5 a4,3	ISS	A1	A2	WB a5=9									
LDF									IF	ID	ID	ISS F2←a2											
ADDF										IF	IF	ID	ISS a1 a2,a3	ISS	ISS	ISS	ISS	ISS	Av1	Av1	Av1	Av1	Av1

NOMBRE:

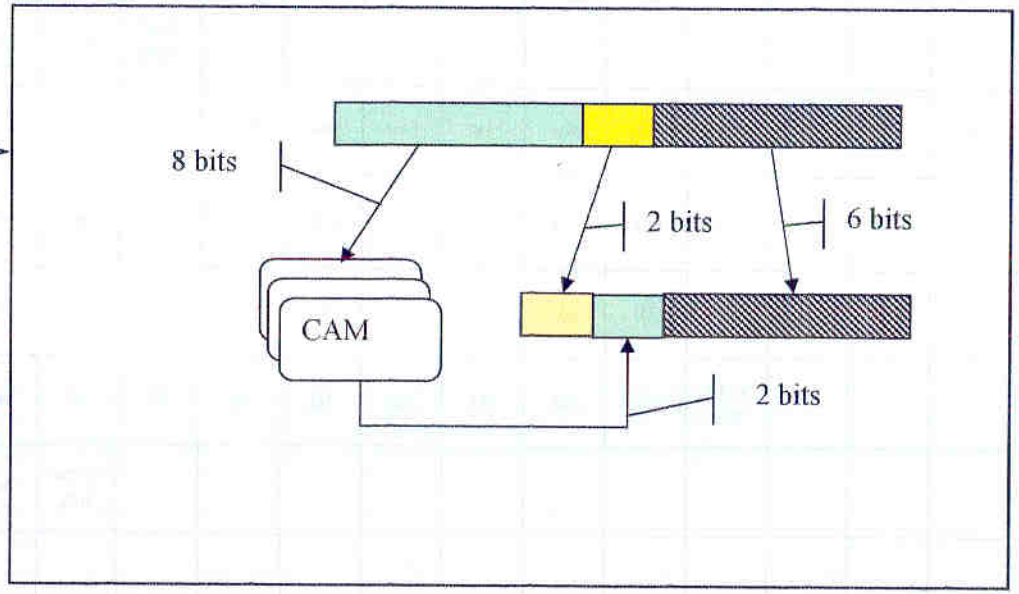
7. Un computador genera direcciones de memoria de 16 bits, está provisto de una memoria cache de 1KB, asociativa por conjuntos.
La cache se encuentra inicialmente en la situación mostrada en la tabla 1 (en el campo LRU el primer bloque representa el último accedido).

CONJUNTO 0		CONJUNTO 1		CONJUNTO 2		CONJUNTO 3	
Bloque	Tag	Bloque	Tag	Bloque	Tag	Bloque	Tag
0	39	0	29	0	02	0	01
1	0F	1	77	1	10	1	64
2	1E	2	20	2	7F	2	03
3	10	3	1E	3	4D	3	5E
LRU: 1-0-3-2		LRU: 2-1-0-3		LRU: 1-2-3-0		LRU: 3-2-1-0	

Tabla 1: Estado de las memorias CAM

Se pide:

a) Esquema de asignación de direcciones



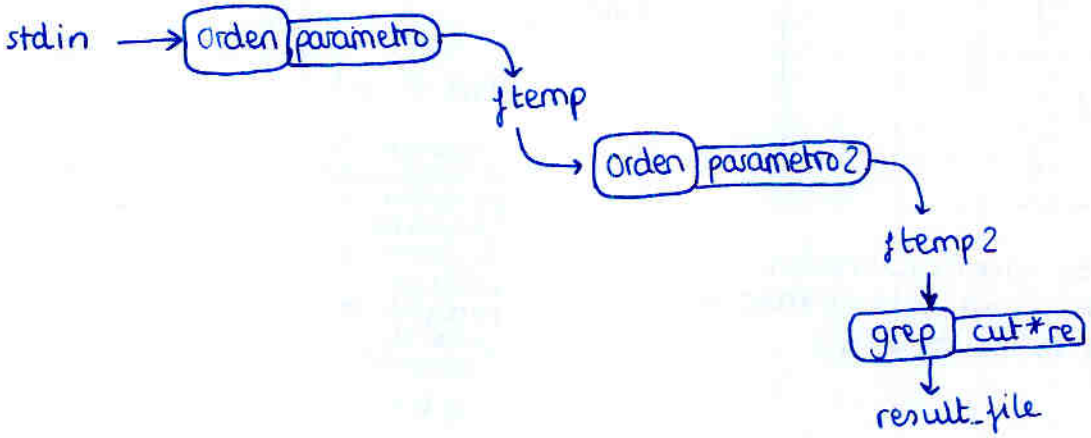
b) En la siguiente tabla se muestran 9 direcciones de memoria que se acceden de forma consecutiva, rellenar la tabla indicando para cada caso el TAG y el conjunto asociado a dicha dirección, si es acierto o fallo, como estaba la LRU antes y después del acceso y en caso de que sea un fallo la línea y conjunto a la que sustituirá el nuevo bloque. Además modificar en la tabla 1 los TAGS para reflejar los cambios, tachando el TAG que ya no es válido y escribiendo el nuevo.

N°Acc	Dir Hexa	Dir. Binario	TAG	Conjunto	Ac/Fallo	LRU antes	LRU desp.	Línea/cto. Sustituida
1	0x1E 70	00011110 01 110000	1E	1	Ac	1-0-3-2	3-1-0-2	-
2	0x20 50	00100000 01 010000	20	1	Ac	3-1-0-2	2-3-1-0	-
3	0x20 D0	00100000 11 010000	20	3	Fallo	3-2-1-0	0-3-1-2	0/3
4	0x1E 77	00011110 01 110111	1E	1	Ac	2-3-1-0	3-2-1-0	-
5	0x20 64	00100000 01 100100	20	1	Ac	3-2-1-0	2-3-1-0	-
6	0x20 D0	00100000 11 010000	20	3	Ac	0-3-2-1	0-3-2-1	-
7	0x21 ED	00100001 11 101101	21	3	Fallo	0-3-2-1	1-0-3-2	1/3
8	0x20 78	00100000 01 111000	20	1	Ac	2-3-1-0	2-3-1-0	-
9	0x20 D0	00100000 11 010000	20	3	Ac	1-0-3-2	0-1-3-2	-

2ptos

Examen
Julio 2005

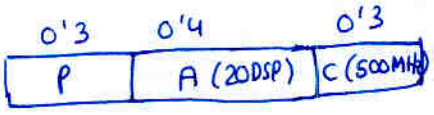
Problema 1



a) orden parametro | orden parametro2 | grep cut*re > result_file

b) (orden parametro | orden parametro2 | grep cut*re > result_file) 2> errores.txt

Problema 2



a) Si no se sustituye el procesador

$$T_{nuevo} = \left(0'3 + 0'3 + \frac{0'4}{S_{DSP}}\right) > 0'6$$

No se puede lograr la mitad de tiempo

b) si sustituimos el procesador

$$T_{nuevo} = \left(0'3 + \frac{0'3}{2} + \frac{0'4}{S_{DSP}}\right) = 0'5$$

$$\left(0'45 + \frac{0'4}{S_{DSP}} = 0'5$$

$$0'4 = 0'05 \cdot S_{DSP}$$

$$S_{DSP} = \frac{0'4}{0'05} = 8$$

Necesitamos reducir la etapa A a 8 veces el tiempo anterior, por tanto deberemos pasar de 20 DSPs a 160 DSP's

c) La única viable es la opción c

cambiar µP : 1000 €
140 DSP's : 100 x 140 = 14 000 €
} Total 15000 €

3.

	Indexado	Inmediato	Rel. a PC	Absoluto
Antmetica		ADD R3, R1, #val (16 bits)		
Salto condicional			BEQZ R1, val (aunque en el compilador ponemos la etiqueta destino, en código máquina pone el desplaz. 16 bits)	

Sólo los saltos incondicionales pueden saltar a un valor inmediato (que además es de 24 bits)

4.

a) $$\text{SpeedUp} = \frac{T_{\text{no segment}}}{T_{\text{segment}}} = \frac{I_1 \cdot CPI_1 \cdot T_{\text{ciclo}_1}}{I_2 \cdot CPI_2 \cdot T_{\text{ciclo}_2}} = \frac{CPI_1}{CPI_2} \cdot \frac{T_{\text{ciclo}_1}}{T_{\text{ciclo}_2}} \leq k$$

b) La ganancia es igual a K bajo las siguientes condiciones ideales:

(1) · El CPI sin segmentar es igual a K

(2) · El CPI segmentado es igual a 1

(3) · $T_{\text{ciclo}_1} = T_{\text{ciclo}_2}$

Pero en la práctica

(1) El CPI sin segmentar puede ser menor que K ya que algunas instrucciones pueden saltarse algunas etapas

(2) El CPI segmentado será normalmente mayor que 1 debido a riesgos (estructurales, de datos y de control) causados por la segmentación

(3) T_{ciclo_2} puede ser mayor que T_{ciclo_1} por culpa de los registros que se deben añadir al segmentar

c) Con la segmentación no se puede lograr un speedup mayor que K, el máximo es K bajo las condiciones ideales de b)

Si usamos sin embargo un procesador superescalar (replica hardware) se puede conseguir un CPI menor que 1 y por tanto speedUp > K

Examen
Junio 2006

NOMBRE:

1. **(1,5 puntos)** Supongamos que estamos trabajando en un equipo Unix, en el directorio **/home/acso1/usuario**. Asumimos que tenemos todos los permisos necesarios. Disponemos del fichero **examen.sh.**

`mkdir examen`
`cd examen`
`touch f1 > f2`
`mv f1 ..`
`pwd`
`cd ..`
`pwd`
`ls examen`
	`echo "" > out.txt`
	`for i in $*`
	`do`
	`  A=`grep $i examen.sh | tail -1``
	`  echo $A >> out.txt`
	`  # Ayuda: tail -n obtiene las`
	`  # n últimas líneas de su`
	`  # entrada estándar`
	`done`
	`set $A`
	`shift`
	`echo $1`
	`cat out.txt | wc -w`
	`head -2 out.txt`
	`# Ayuda: head -n`
	`# obtiene las n primeras líneas de su entrada`

examen.sh	**examen2.sh**

a) ¿Cuál sería el resultado obtenido en pantalla tras ejecutar en la línea de órdenes?:

```
$ ./examen.sh
```

```
/home/acso1/usuario/examen
/home/acso1/usuario
f2
```

Supongamos que en el mismo directorio disponemos de un segundo fichero, **examen2.sh**, con el contenido mostrado en la tabla anterior

b) ¿Cuál sería el resultado obtenido en pantalla tras ejecutar en la línea de órdenes?:

```
$ ./examen2.sh cd pwd mkdir
```

Se han dado por buenas dos soluciones:

`examen`
`5`
(linea en blanco)
`cd ..`
	`Examen`
	`5`
	`cd ..`
	`pwd`

Realmente la correcta, al ejecutar el programa, es la primera, ya que el "`echo "" > out.txt`" no sólo crea el fichero out.txt, sino que inserta una línea en blanco. No obstante, no se pretendía que fuese una trampa, y la solución que no contempla la línea en blanco también es válida.

c) ¿Cuál sería el contenido del fichero out.txt?

Igualmente, se dan por buenas las dos opciones:

(linea en blanco)
`cd ..`
`pwd`
`mkdir examen`
	`cd ..`
	`pwd`
	`mkdir examen`

NOMBRE:

d) Si sustituimos la línea: A=`grep $i examen.sh | tail -1`
por esta otra: A="grep $i examen.sh | tail -1"
¿cuál sería el resultado obtenido en pantalla?

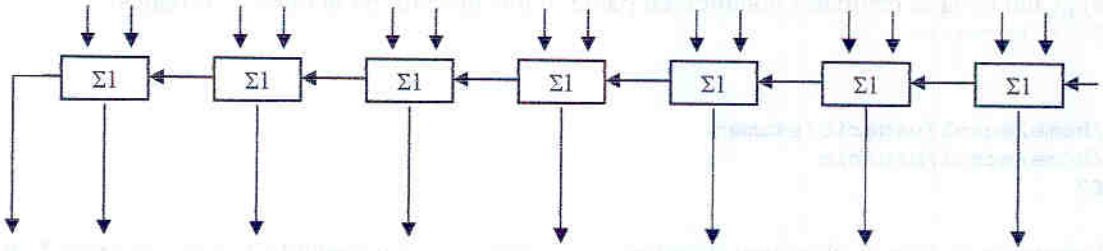
Se vuelven a dar por buenas las dos opciones:

`mkdir`
`18`
(linea en blanco)
`grep cd examen.sh | tail -1`
	`Mkdir`
	`18`
	`grep cd examen.sh | tail -1`
	`grep pwd examen.sh | tail -1`

Observad que las dobles comillas hacen que la cadena se tome como literal, excepto las variables, que sí son sustituidas por su valor. Por tanto, $i toma el valor de cada uno de los parámetros.

2. **(1,5 pto)**Dado el sumador segmentado de 7 bits visto en clase, con T_{suma}= 3 tp y $T_{acarreo}$= 2 tp, Diseña una versión segmentada para que, con el T_{ciclo} mínimo posible, se consiga un:

a) Speed-Up teórico máximo de 6. Indica el T_{ciclo} del sumador segmentado.
b) Speed-Up teórico máximo de 3. Indica el T_{ciclo} del sumador segmentado.
Se supone que los tiempos de los registros son despreciables.



Para poder calcular el Speed-Up hay que saber el tiempo de suma del operador no segmentado. Este es:

$$T_{SUMA\ NSeg} = 6\ T_{acarreo} + T_{suma} = 15\ tp$$

a) Si segmentamos en 7 etapas, $T_{ciclo} = T_{suma} = 3$ tp , con lo que el Speed-Up máximo teórico (cuando nº operaciones → ∞) es:

$$SUp = 15 / 3 = 5$$

El modo de segmentarlo puede verse en los apuntes de clase del Tema 5.

b) Si, al segmentar, agrupamos 2 sumadores de 1 bit en una etapa, $T_{ciclo} = T_{suma} + T_{acarreo} = 3 + 2 = 5$ tp, con lo que el Speed-Up máximo teórico (cuando nº operaciones → ∞) es:

$$SUp = 15 / 5 = 3$$

Hay varias posibles soluciones en las que, como mucho, no habrá más de 2 sumadores de 1 bit agrupados en una sola etapa. Como lo que interesa es el tiempo

de ciclo (que será el mismo para todas las etapas, aunque sólo tengan un sumador), el nº de etapas del sumador segmentado puede variar en las distintas soluciones.

3. **(1pto)** En un procesador DLX con operaciones multiciclo como el visto en clase, se va a ejecutar el siguiente código, previa optimización del mismo. Para ello, el compilador deberá averiguar las dependencias que existan entre las instrucciones. Indica, en la siguiente tabla, qué dependencias existen entre las instrucciones que se muestran en la primera columna y el resto de las instrucciones del fragmento de código proporcionado. Basta con escribir el nº de aquellas instrucciones con las que tenga dependencias, en la columna que indica su tipo. Si no tiene dependencias de algún tipo, escribir "NO" en la celda correspondiente a dicho tipo.

```
1.              BNEZ R2, salto
2.              ADD  R3,R3,#8
3.              LDD   F3, 0(R3)
4.      salto:  SUBD F2, F1, F3
5.              ADDD F5, F2, F1
6.              ADDD F2, F7, F5
7.              DIVD F7, F4, F6
```

Instrucción	Estructural	de Datos	Antidependencia	de Salida	de Control
3	NO	2,4,5,6	NO	NO	1
4	5,6	2,3,5,6	NO	6	1
5	4,6	2,3,4,6	6	NO	1

4. **(1,5 pto)** ¿Qué dependencias de la instrucción 3 (ver tabla anterior) se convierten en riesgos si disponemos de un DLX segmentado sin Tomasulo y sin cortocircuitos. Justificad las respuestas.

Dep. de Control: SI, se convierte en un riesgo de control, ya que está a una distancia de menos de 4 instrucciones de la instrucción de salto.

Dep. de Datos:

Con la instr. 2: SI, riesgo de datos, porque lee R3 antes de que la 2 lo actualice.

Con la instr. 4: SI, riesgo de datos, porque 4 lee F3 antes de que la 3 lo actualice.

Con las instr. 5 y 6: SI, riesgo de datos, si no se solucionan las anteriores. Si se hace, no lo habrá.

NOMBRE:

5. **(1,5 ptos)** Un computador genera direcciones de memoria de 20 bits, está provisto de una memoria cache de 16KB, asociativa por conjuntos.
La cache se encuentra inicialmente en la situación mostrada en la tabla 1 (el campo U representa el último bloque/línea accedido).

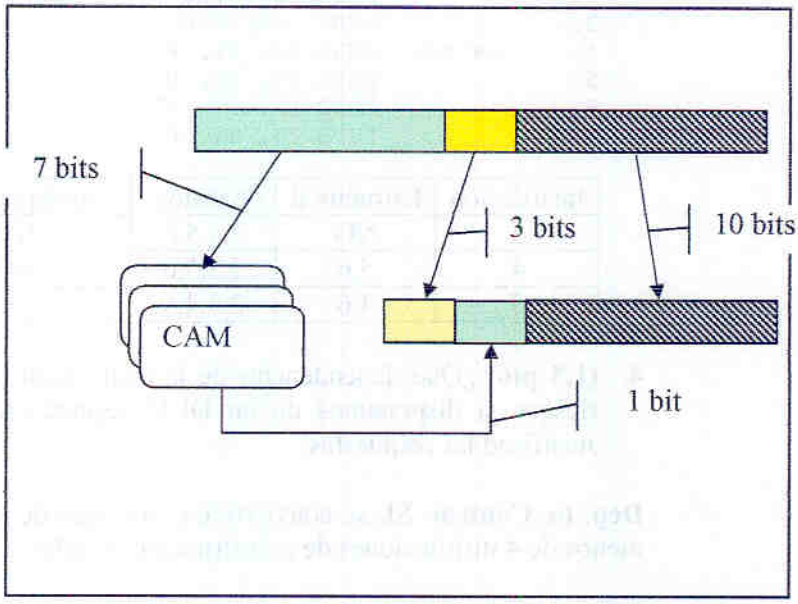
CONJUNTO 0		CONJUNTO 1		CONJUNTO 2		CONJUNTO 3	
Tag	Bloque	Tag	Bloque	Bloque	Tag	Bloque	Tag
39	0	29	0	0	02	0	10
0F	1	10	1	1	10	1	24
U: 0		U: 1		U: 1		U: 1	
CONJUNTO 4		CONJUNTO 5		CONJUNTO 6		CONJUNTO 7	
1E	0	20	0	0	2F	0	03
10	1	1E	1	1	3D	1	3E
U: 1		U: 0		U: 0		U: 1	

Tabla 1: Estado de las memorias CAM

Se pide:

a) Esquema de asignación de direcciones

b) En la siguiente tabla se muestran 9 direcciones de memoria que se acceden de forma consecutiva, rellenar la tabla indicando para cada caso el TAG y el conjunto asociado a dicha dirección, si es acierto o fallo el valor de U antes y después del acceso (se utiliza LRU en caso de fallo). Además, MODIFICAR en la tabla 1 los TAGS para reflejar los cambios, tachando el TAG que ya no es válido y escribiendo el nuevo.

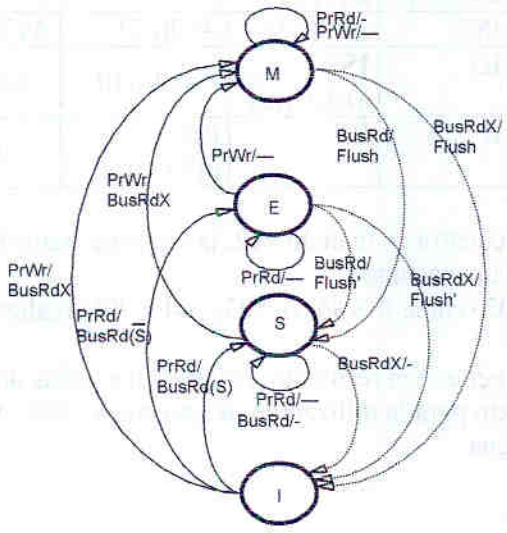


Nº Acc	Dir Hexa	Dir. Binario	TAG	Conjunto	Ac/Fallo	U antes	U desp.
1	0x1E703	0001111 001 1100000011	0F	1	Fallo	1	0
2	0x20502	0010000 001 0100000010	10	1	Ac	0	1
3	0x20D03	0010000 011 0100000011	10	3	Ac	1	0
4	0x1E770	0001111 001 1101110000	0F	1	Ac	1	0
5	0x20640	0010000 001 1001000000	10	1	Ac	0	1
6	0x60D01	0110000 011 0100000001	30	3	Fallo	0	1
7	0x21EDF	0010000 111 1011011111	10	7	Fallo	1	0
8	0x2078F	0010000 001 1110001111	10	1	Ac	1	1
9	0x20D0F	0010000 011 0100001111	10	3	Ac	1	0

NOMBRE:

6. (**2 ptos**)Un sistema multiprocesador simétrico de 3 procesadores utiliza un sistema de antememorias coherente basado un bus snoopy y protocolo MESI de la figura. Suponiendo que en la cache de cada procesador **caben los dos bloques (X e Y)** de cuatro palabras y que las palabras W0-W3 pertenecen al mismo bloque X y que W4-W7 pertenecen al bloque Y, se pide:
   Analizar la siguiente secuencia de accesos indicando, que sucede en cada una de las caches, en que estado se encontrará cada bloque (M,E,S,I) después de la ejecución de cada instrucción (**no se pueden dejar casillas de bloques en blanco**) y en caso de que proceda si es un acierto (A) o es un fallo (F). Indicar también las operaciones de bus que se generan. Indicar también el estado de la línea S que cada controlado de cache activa cuando haya operaciones BusRd. Por último, cuando haya una operación de bus indicar el origen de los datos (P1,P2,P3 o MEM) La línea 1 de la tabla ya está resuelta a modo de ejemplo.

	P1	Bloque	Bloque		S	P2	Bloque	Bloque		S	P3	Bloque	Bloque		S	origen de datos
	Inst	X	Y	Op.Bus		Inst	X	Y	Op.Bus		Inst	X	Y	Op.Bus		
1	ld W6	I	E/F	BusRd	-		I	I	-	0		I	I	-	0	MEM
2		I	E	-	0	ld W0	E/F	I	BusRd	-		I	I	-	0	MEM
3		I	S	Flush'	1		E	I	-	0	ld W6	I	S/F	BusRd	-	P1
4		I	S	-	-	st W2	M/A	I	-	-		I	S	-	-	-
5		I	S	-	1	ld W6	M	S/F	BusRd	-		I	S	-	1	MEM
6	ld W2	S/F	S	BusRd	-		S	S	Flush	1		I	S	-	0	P2
7		S	S	-	-	ld W3	S/A	S	-	-		I	S	-	-	-
8		I	S	-	-	st W2	M/A	S	BusRdx	-		I	S	-	-	-/MEM
9		I	S	-	0		S	S	Flush	1	ld W2	S/F	S	BusRd	-	P2
10	st W2	M/F	S	BusRdx	-		I	S	-	-		I	S	-	-	MEM



NOMBRE:

7. (**1 pto**)Un DLX segmentado, con instrucciones multiciclo sobre las que se aplica el algoritmo de TOMASULO, y los siguientes operadores de coma flotante:

a. Acceso a Memoria, Tev = 2, IR=1 cada 2 ciclos-
b. Sumador, Tev = 2, IR = 1 cada ciclo.
c. Multiplicado, Tev = 4, IR = 1 cada ciclo.

Se encuentre en el siguiente estado:

Registros:

	F1	F2	F3	F4	F5	F6	F7
Valor	3.1→m10	(*l*2)→a10	(*m*1)→v3	(*a*4)→v1	(*a*1)	(*m*3)	(*a*2)

Operadores virtuales

Operador virtual (estación reserva)	Datos almacenados			
	702	703	704	705
Lv1	(*l*2,M[2300F2])			
Mv1	(*m*3,*a*7,35.1)	(*m*3,*a*7,35.1)	(*m*3,*a*7,35.1)	(*m*3,*a*7,35.1)
Mv2			(m10,3.1,a1)	(m10,3.1,a1)
Av1	(*a*1,*a*4,*a*3)	(*a*1,*a*4,*a*3)	(*a*1,*v1*,*a*3)	(*a*1,*v1*,*a*3)
Av2	(*a*2,234.1,121)	(*a*2,234.1,121)	(*a*2,234.1,121)	(*a*2,234.1,121)
Av3	(*a*7,*l*0,57.1)	(*a*7,*v0*,57.1)		
Av4		(a10,3.1,a4)	(a10,3.1,v1)	(a10,3.1,v1)

Ejecución

		701	702	703	704	705
		L(*l*0)	L(*l*0)	WB(l0→v0)		
		M1(*m*1)	M2(*m*1)	M3	M4	WB(m1→v3)
		Av2(*a*3)	A1(*a*3)	A1	A2	A2
		A1(*a*4)	A2(*a*4)	A2	WB(a4→v1)	
		Av1(*a*1)	Av1(*a*1)	Av1(*a*1)	Av1(*a*1)	Av1(*a*1)
		Av3(*a*7)	Av3(*a*7)	Av3(*a*7)	A1	A1
	…	Mv1(*m*3)	Mv1(*m*3)	Mv1(*m*3)	Mv1(*m*3)	Mv1(*m*3)
		Lv1(*l*2)	Lv1(*l*2)	L	L	L
		ISS	Av2(*a*2)	Av2(*a*2)	Av2(*a*2)	Av2(*a*2)
ADDF F2,F1,F4		ID	**ISS** a10,3.1,a4	Av4(*a*10)	Av4(*a*10)	Av4(*a*10)
MULTF F1,F1,F5		IF	ID	ISS m10,3.1,a1	MV2	MV2

En este momento se encuentra en el ciclo 702, la etapa de Issue de la instrucción ADDF F2,F1,F4 todavía no se ha ejecutado.

Completar los ciclos 702 (etapa ISS sólo), 703, 704 y 705, realizando la evolución de las marcas.

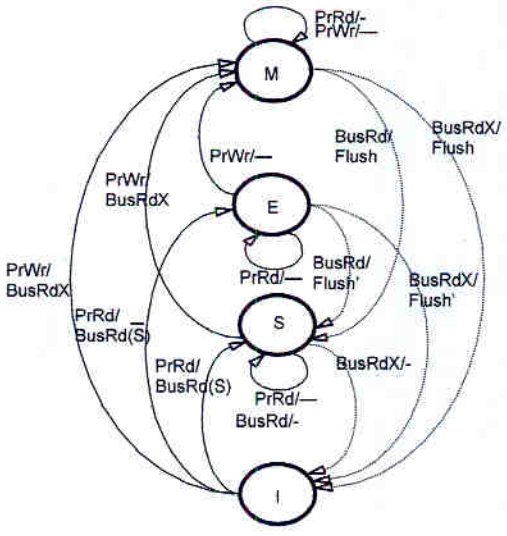
Cuando una operación genere un resultado, poner en las tablas una *v* de valor.

Cuando haya un conflicto para la utilización de una etapa (WB, A1…) tiene preferencia la instrucción más antigua

NOMBRE:

6. (**2 ptos**)Un sistema multiprocesador simétrico de 3 procesadores utiliza un sistema de antememorias coherente basado un bus snoopy y protocolo MESI de la figura. Suponiendo que en la cache de cada procesador **caben los dos bloques (X e Y)** de cuatro palabras y que las palabras W0-W3 pertenecen al mismo bloque X y que W4-W7 pertenecen al bloque Y, se pide:
   Analizar la siguiente secuencia de accesos indicando, que sucede en cada una de las caches, en que estado se encontrará cada bloque (M,E,S,I) después de la ejecución de cada instrucción (**no se pueden dejar casillas de bloques en blanco)** y en caso de que proceda si es un acierto (A) o es un fallo (F). Indicar también las operaciones de bus que se generan. Indicar también el estado de la línea S que cada controlado de cache activa cuando haya operaciones BusRd. Por último, cuando haya una operación de bus indicar el origen de los datos (P1,P2,P3 o MEM) La línea 1 de la tabla ya está resuelta a modo de ejemplo.

	P1	Bloque	Bloque		S	P2	Bloque	Bloque		S	P3	Bloque	Bloque		S	origen de
	Inst	X	Y	Op.Bus		Inst	X	Y	Op.Bus		Inst	X	Y	Op.Bus		datos
1	ld W6	I	E/F	BusRd	-		I	I	-	0		I	I	-	0	MEM
2		I	E		0	ld W2	E/F	I	BusRd	-		I	I		0	Mem
3		I	S	Flush'	1		E	I		0	ld W6	I	S/F	BusRd	-	P1
4		I	S			st W2	M/Ac.	I	-			I	S			-
5		I	S		1	ld W6	M	S/F	BusRd			I	S		1	Mem
6	ld W2	S/F	S	BusRd	-		S	S	Flush	1		I	S		0	P2
7		S	S			ld W3	S/Ac	S	-			I	S			
8		I	S			st W2	M/Ac	S	BusRdX	1		I	S			Mem ó -
9		I	S		0		S	S	Flush	1	ld W2	S/F	S	BusRd	-	P2
10	st W2	M/F	S	BusRdX			I	S		1		I	S		1	Mem



ARQUITECTURA DE COMPUTADORES Y SISTEMAS OPERATIVOS I
Convocatoria Enero 2005 21-Enero-2005

6. Escribe los resultados que muestran por pantalla las siguientes líneas de órdenes.
"wc –w" es para contar sólo las palabras.
Suponer que el fichero user_name en el directorio home de cada usuario contiene una palabra)

0,5 pto

```
echo Hola `cat $HOME/user_name` | wc -w
```

2

```
echo Hola "cat $HOME/user_name" | wc -w
```

3

7. Sea una unidad segmentada con los siguientes operadores multiciclo:
Load en coma flotante (Tev= 2 IR=1/2 ciclos)
Sumador/restador (Tev=2 IR=1/1 ciclos)
Multiplicador (Tev=5 IR=1/1 ciclos)

Se resuelven los riesgos de control con un "**delay slot**" de 1 instrucción.
Sea el siguiente bloque de ejecución:

```
Loop: LDD    F2,a(R4)
      LDD    F6,b(R4)
      ADDD   F8,F2,F6
      MULTD  F14,F6,F2
      SD     c(R4),F8
      SUB    R4,R4,#8
      BNEZ   R4,Loop
      ADD    F10,F10,F14
```

- ¿Cuántos ciclos tarda en ejecutarse esta secuencia de instrucciones utilizando el algoritmo de Tomasulo?
- ¿Qué speed-up se alcanza respecto a una unidad secuencial?

Suponed que el bucle se ejecuta sobre un vector de N componentes.
En la respuesta deberá constar el esquema de ejecución para dos iteraciones de bucle (2 hojas de resultados).
NOTA: En la primera hoja caben 21 ciclos de ejecución.
NOTA: En la primera columna de la segunda hoja se repetirá el ciclo 21, el último de la primera hoja, de forma que la cuadrícula correspondiente a la primera fila en la primera columna no esté vacía.
(2ptos)

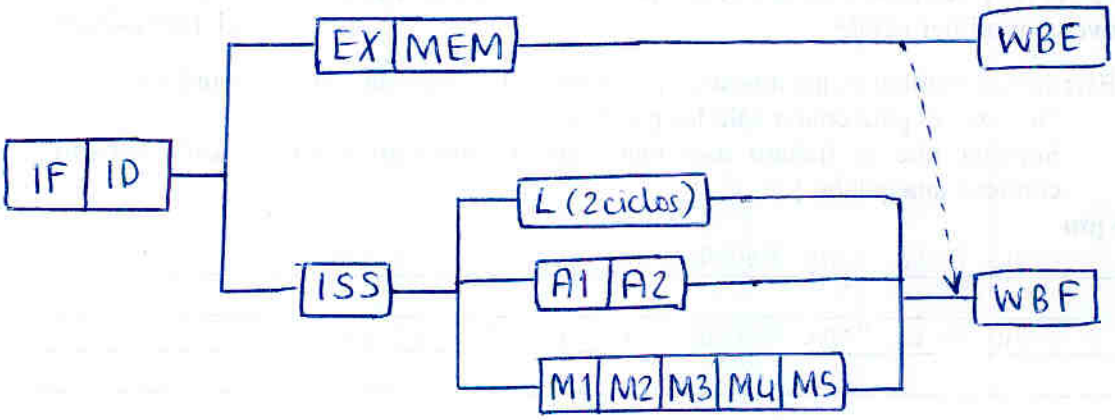
En la unidad no segmentada los costes de las instrucciones serán:

LDD	= IF + ID + EX + L + L + WB	= 6 CICLOS
SDD	= IF + ID + EX + L + L	= 5 CICLOS
MULTD	= IF + ID +M1+M2+M3+M4+M5+WB	= 8 CICLOS
ADDD	= IF + ID +A1+A2 +WB	= 5 CICLOS
SUB R1..	= IF + ID +EX +WB	= 4 CICLOS
BNEZ	= IF +ID +EX + MEM	= 4 CICLOS

En total serán: 43 ciclos por cada pasada del bucle = 43N ciclos
En el caso de Tomasulo se tardan 9N+7 ciclos en el peor caso, y 9N+6 en el mejor. La diferencia no influye en el caso asintótico.
La mejora introducida por Tomasulo será:

$$S = \frac{43N}{9N+6} \xrightarrow[N\to\infty]{} S = \frac{43}{9}$$

NOMBRE:

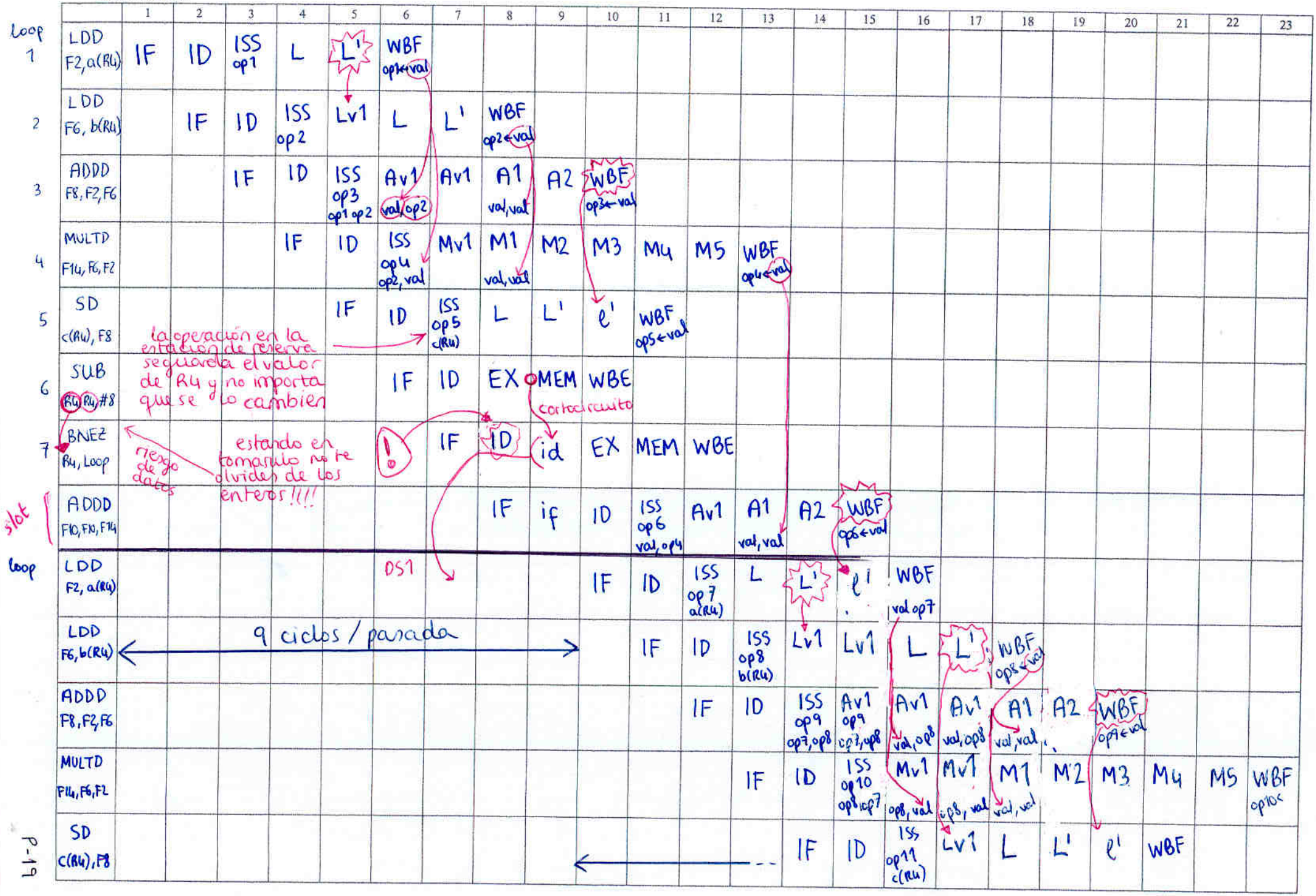


F2 :	~~val~~ ~~op1~~ ~~val~~ ~~op7~~ ~~val~~ op13
F6 :	~~val~~ ~~op2~~ ~~val~~ ~~op8~~ val
F8 :	~~val~~ ~~op3~~ ~~val~~ ~~op9~~ val
F14 :	~~val~~ ~~op4~~ ~~val~~ op10
F10 :	~~val~~ ~~op6~~ ~~val~~ op12

Número de ciclos para N pasadas al bucle : $\begin{cases} 9N+6 & \text{segmentado} \\ 43N & \text{sin segmentar} \end{cases}$

$$S = \frac{43N}{9N+6} \underset{N\to\infty}{=} \frac{43}{9} = 5 - \frac{2}{9}$$

NOMBRE:


Loop
1
LDD F2,a(R4)
IF ID ISS op1 L L' WBF op1←val
2
LDD F6, b(R4)
IF ID ISS op2 Lv1 L L' WBF op2←val
3
ADDD F8, F2, F6
IF ID ISS op3 op1 op2 Av1 val,op2 Av1 A1 val,val A2 WBF op3←val
4
MULTD F14, F6, F2
IF ID ISS op4 op2,val Mv1 M1 val,val M2 M3 M4 M5 WBF op4←val
5
SD c(R4), F8
IF ID ISS op5 c(R4) L L' e' WBF op5←val
la operación en la estación de reserva seguarda el valor de R4 y no importa que se lo cambien
6
SUB R4 R4 #8
IF ID EX MEM WBE
cortocircuito
7
BNEZ R4, Loop
riesgo de datos
estando en tomasulo no te olvides de los enteros!!!
! ID id EX MEM WBE
slot
ADDD F10, F4, F14
IF if ID ISS op6 val, op4 Av1 A1 val, val A2 WBF op6←val
Loop
LDD F2, a(R4)
DS1 IF ID ISS op7 a(R4) L L' e' WBF val op7
LDD F6, b(R4)
9 ciclos / parada
IF ID ISS op8 b(R4) Lv1 Lv1 L L' WBF op8←val
ADDD F8, F2, F6
IF ID ISS op9 op7, op8 Av1 op9 op7, op8 Av1 val, op8 Av1 val, op8 A1 val, val A2 WBF op9←val
MULTD F14, F6, F2
IF ID ISS op10 op8, op7 Mv1 op8, val Mv1 op8, val M1 val, val M2 M3 M4 M5 WBF op10<
SD c(R4), F8
IF ID ISS op11 c(R4) Lv1 L L' e' WBF

NOMBRE:

	15	16	17	18	19	20	21	22	23	10	11	12	13	14	15	16	17	18	19	20	21	22	23
MULTD F14,F6,F12	ISS op 10 op8, op7	Mv1 op8, val	Mv1 op8, val	M1 val, val	M2	M3	M4	M5	WBF op10 ← v														
SD C(R4), F8	ID	ISS op 11 C(R4)	Lv1	L	L'	l'	WBF op11 ← v																
SUB R4, R4, #8	IF	ID	EX	MEM	WBE																		
BNEZ R4, loop		IF	ID	id	EX	MEM	WBE																
ADDD F10, F0, F14			IF	ij	ID	ISS op 12 val, op10	Av1	Av1	A1 val, val	WBF													
loop LDD F2, a(R4)					IF	ID	ISS op 13 a(R4)	L	L'	l'	WBF												
- - - -	9 ciclos/parada							⋮	el resto igual														

6 ciclos adicionales si fuera la última parada al bucle

Laboratorio ACSO I. Práctica 4: Evaluación de la segmentación en el DLX

Eloi Rico Blasco, Francisco José Rodríguez Fortuño

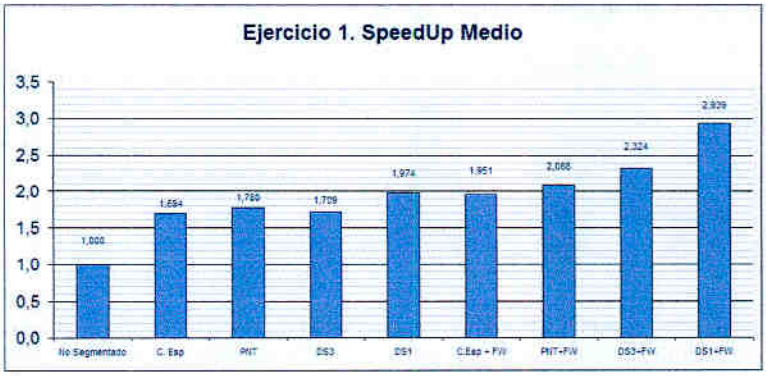
8 de enero de 2007

0.1. Primer ejercicio

Tras optimizar el código suministrado utilizando desenrollado de bucles y reordenación de algunas líneas de código, y tras optimizar de nuevo el código para los casos especiales de Delay Slot, obtuvimos los siguientes resultados:

Resultados Ejercicio 1	No Segmentado	C. Esp	PNT	DS3	DS1	C.Esp + FW	PNT+FW	DS3+FW	DS1+FW
Ciclos(-10*1000)	970	576	533	568	495	504	453	425	334
Ciclos(-10*6011)	4798	2815	2757	2804	2412	2427	2358	2029	1613
Ciclos(1280*1000)	970	576	533	568	495	504	453	425	334
Ciclos(1280*6011)	4798	2815	2757	2804	2412	2427	2358	2029	1613
CPImed	4,005	2,364	2,251	1,369	1,947	2,053	1,919	1,007	1,308
SpUmed	1,000	1,694	1,780	1,709	1,974	1,951	2,088	2,324	2,939

Representando el SpeedUp medio para cada configuración:



La configuración con que se obtiene mejor CPI resulta ser Delay Slot 3 con cortocircuitos, sin embargo la configuración que presenta un mejor SpeedUp es Delay Slot 1 con cortocircuitos. ¿A qué se debe ésto? Estudiando las distintas configuraciones más detalladamente, se tienen los siguientes resultados para el número total de instrucciones ejecutadas (que resulta depender únicamente del primer operando):

P-21

Ejercicio 1	No Segm	C Esp	PNT	DS3	DS1	C Esp + FW	PNT+FW	DS3+FW	DS1+FW
num instrucciones 1000	242	242	242	420	258	242	242	420	258
num instrucciones 6011	1199	1199	1199	2024	1221	1199	1199	2024	1221

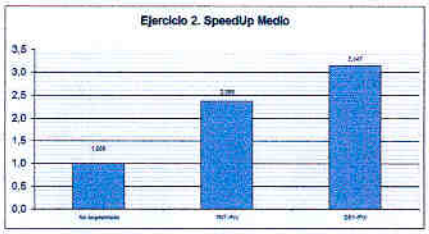
Vemos que Delay Slot 3 tiene un número mucho mayor de operaciones que las demás configuraciones, esto es debido a que en el programa se nos han quedado varios "slots" que no hemos podido rellenar, y por tanto son operaciones adicionales que incrementan el tiempo total de ejecución. Éste efecto no se considera en el cálculo del CPI, el cual considera a todas las instrucciones por igual aunque sean inútiles, y por ello obtiene una puntación tan buena.

0.2. Segundo Ejercicio

Optimizando el código ejMult.s para las configuraciones Delay Slot 1 + FW y Predict Not Taken + FW y realizando las medidas necesarias, obtuvimos los siguientes resultados:

Resultados Ejercicio 2	No segmentado	PNT+FW	DS1+FW
Ciclos(-10*1000)	150	56	48
Ciclos(-10*6011)	150	56	48
Ciclos(1280*1000)	282	137	89
Ciclos(1280*6011)	282	137	89
NumInstr(-10)	37	37	39
NumInstr(1280)	70	70	72
CPImed	4,041	1,735	1,233
SpUmed	1,000	2,368	3,147

Representando la gráfica del SpeedUp:



En éste tercer ejercicio, la configuración de máximo CPI y máxima velocidad resulta coincidir, y se da para la configuración Delay Slot 1 + Cortocircuitos, ya que hemos logrado rellenar una gran cantidad de slots y por tanto estamos siendo más eficientes que con PNT.

0.3. Tercer Ejercicio

Primero modificamos el fichero "media.s" para que implementase el método de ordenación "burbuja", optimizándolo para los casos PNT y DS1 tanto como

pudimos.

Luego usamos el código en C proporcionado en el enunciado para crear el fichero "heapsort.s", ésta fue la parte más difícil, ya que había que introducir algunos cambios en el algoritmo que tuvieran en cuenta que los registros que funcionasen como índices debían empezar en 0 e indexar en múltiplos de 4 (ya que indexan "words" en memoria), a diferencia de las variables utilizadas como índices en el programa en C, que comienzan en 1 e indexan en múltiplos de 1.

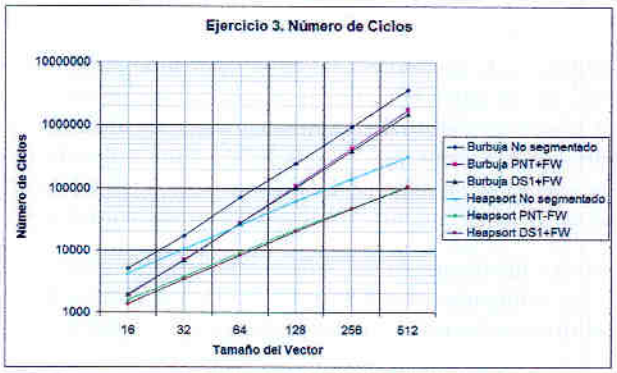
Teniendo eso en cuenta finalmente hicimos funcionar el programa y pudimos optimizarlo para las configuraciones DS1 y PNT. Tras realizar todas las medidas necesarias [1] obtuvimos los resultados presentados en la siguiente tabla:

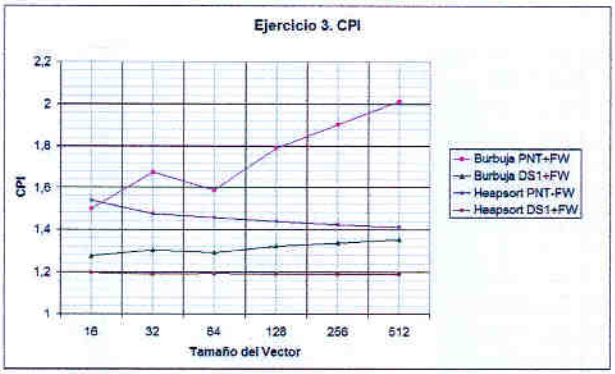
Resultados Ejercicio 3	Burbuja			Heapsort		
	No segmentado	PNT+FW	DS1+FW	No segmentado	PNT+FW	DS1+FW
Ciclos(16)	5039	1830	1910	4221	1557	1348
Ciclos(32)	17215	6965	6813	10522	3726	3378
Ciclos(64)	70651	27212	27537	25651	8957	8313
Ciclos(128)	249183	107635	101491	61913	21366	20169
Ciclos(256)	934595	428218	389860	141538	48309	46278
Ciclos(512)	3528243	1708353	1505756	322474	109188	105789
NumInstrucciones(16)	1221	1221	1494	1012	1012	1126
NumInstrucciones(32)	4163	4163	5220	2526	2526	2834
NumInstrucciones(64)	17126	17126	21287	6149	6149	6961
NumInstrucciones(128)	60199	60199	76712	14854	14854	16910
NumInstrucciones(256)	225360	225360	291160	33953	33953	38794
NumInstrucciones(512)	849100	849100	1111700	77349	77349	88659
CPI(16)	4,127	1,499	1,278	4,171	1,539	1,197
CPI(32)	4,135	1,673	1,305	4,165	1,475	1,192
CPI(64)	4,125	1,589	1,294	4,172	1,457	1,194
CPI(128)	4,139	1,788	1,323	4,168	1,438	1,193
CPI(256)	4,147	1,900	1,339	4,169	1,423	1,193
CPI(512)	4,155	2,012	1,354	4,169	1,412	1,193
SpeedUp(16)	1,000	2,754	2,638	1,000	2,711	3,131
SpeedUp(32)	1,000	2,472	2,527	1,000	2,824	3,115
SpeedUp(64)	1,000	2,596	2,566	1,000	2,864	3,086
SpeedUp(128)	1,000	2,315	2,455	1,000	2,898	3,070
SpeedUp(256)	1,000	2,183	2,397	1,000	2,930	3,058
SpeedUp(512)	1,000	2,065	2,343	1,000	2,953	3,048
CPImed	4,138	1,743	1,316	4,169	1,457	1,194
SpUmed	1,000	2,397	2,488	1,000	2,863	3,085

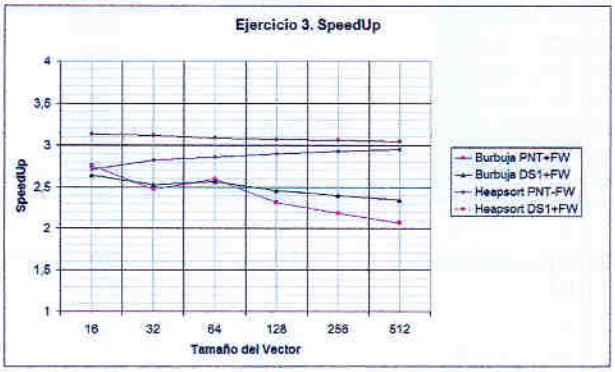
A partir de la tabla se pueden construir las siguientes gráficas para el número de ciclos, SpeedUp y CPI:

[1] Nota: Para obtener el SpeedUp de cada configuración es necesario conocer el tiempo de ejecución del caso no segmentado, para ello a su vez es necesario conocer el número de ciclos en el caso no segmentado. Puesto que el simulador no puede simular el caso no segmentado, logramos obtener el número de ciclos aplicando la fórmula: nºciclos = 4*nºoperaciones + nº loads.

El problema está en conocer el nº de loads, ya que hay loads en el interior de los bucles. Para solucionar ésto hemos usado una versión del algoritmo con todo idéntico salvo un registro que se incrementa en uno cada vez que se hace un load, obteniendo así el número total de loads durante la ejecución.







Se puede concluir que el algoritmo de ordenación que proporciona mejores prestaciones es sin duda alguna el "heapsort". Para empezar, obtiene un CPI menor a 1,2 mientras que burbuja un CPI superior a 1,3.

Además "heapsort" tarda menos de una décima parte de lo que tarda el algoritmo de burbuja para ordenar 512 elementos, en cuanto al SpeedUp "heapsort" es claramente superior en ambas configuraciones, y lo que es mejor: *la diferencia se acentúa según crece el tamaño del vector*, obteniendo "heapsort" cada vez mayor ventaja en número de ciclos y SpeedUp.