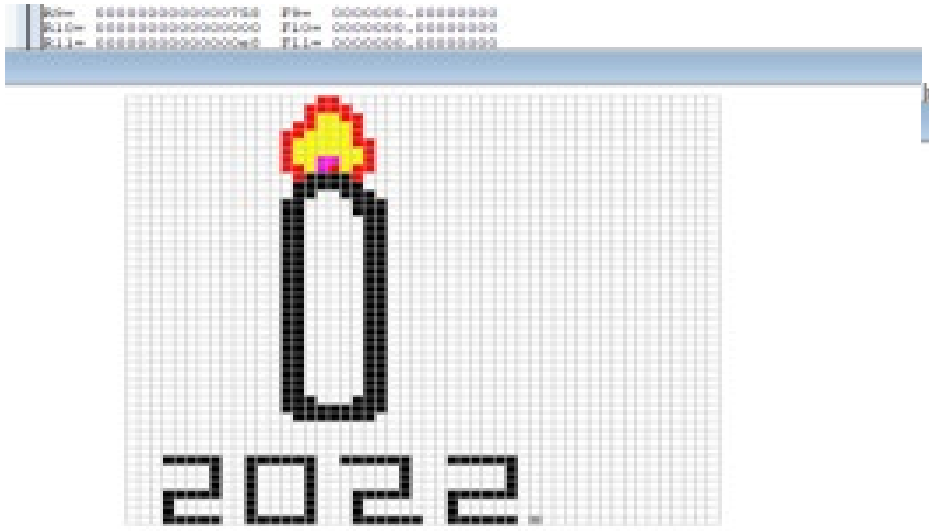
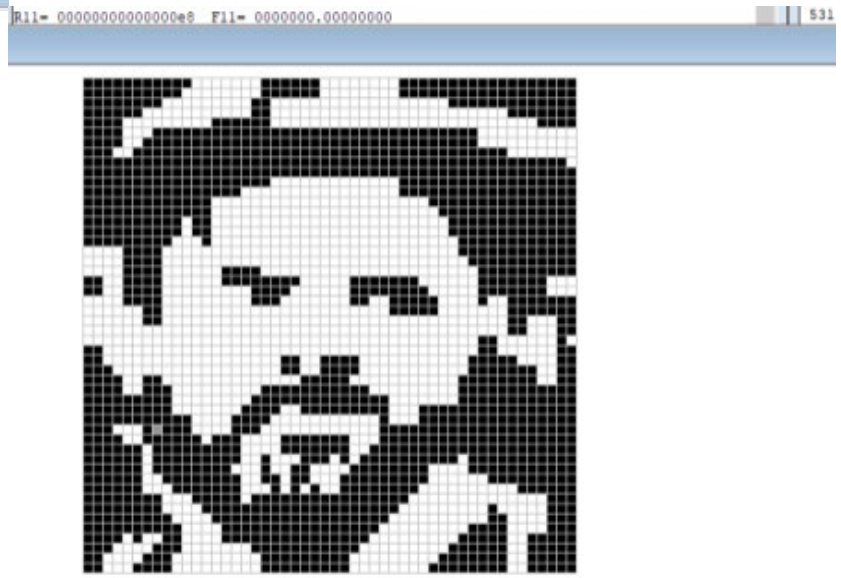
Resumen Segundo parcial





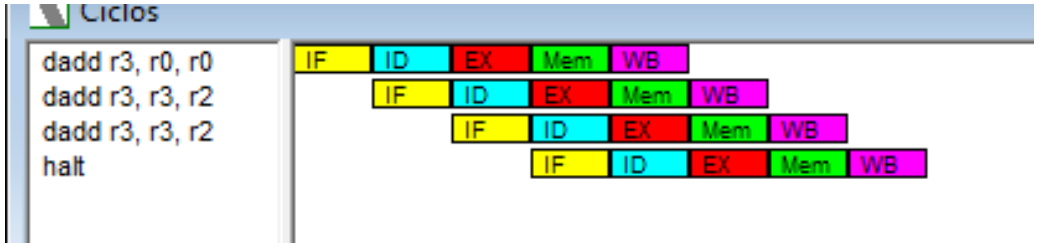
Arn

Etapas de adelantamiento y cosas

Con forwarding

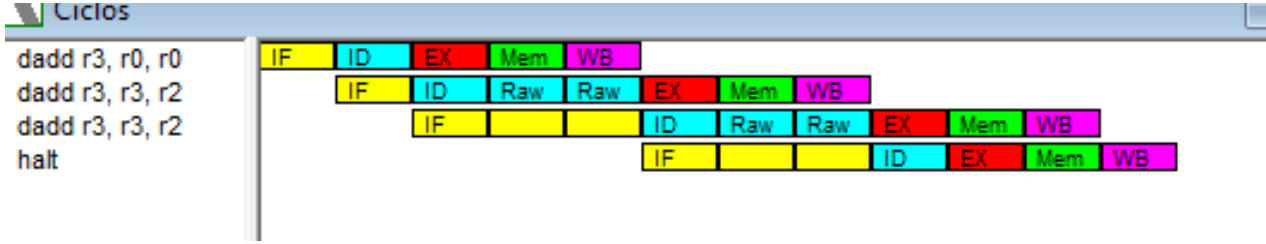
```
dadd r3, r0, r0          IF | ID | EX | MEM | WB
dadd r3, r3, r2               IF | ID | EX | MEM | WB
dadd r3, r3, r2                    IF | ID | EX | MEM | WB
...
```

0 ATASCOS, LA SUMA LA TIENE LISTA EN EX



Sin Forwarding

RECIEN LA OBTENDRIA EN WB



En la etapa IF chequea si tiene que saltar
Con btb On

Tipos de atascos

Atascos

Tipos

Se pueden dar situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde

ESTRUCTURALES

Provocados por conflicto de recursos

DEPENDENCIA DE DATOS

- **RAW:** Read After Write
- **WAR:** Write After Read
- **WAW:** Write After Write

DEPENDENCIA DE CONTROL

Provocados al esperar la decisión de otra instrucción anterior (por ej.: si se realiza o no un salto)

Queremos evitarlos a toda costa

Tipos de soluciones atascos

Podemos solucionarlo de dos formas:

Por software

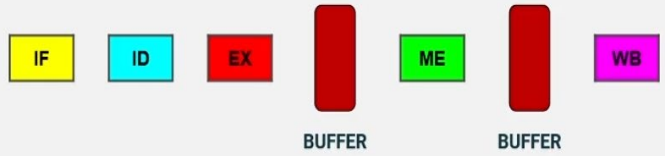
NOP

CODIGO VALIOSO

DADDI R1, R0, 8 DADDI R2, R1, 10 LD R7, NUM1 (R0)	**VS**	DADDI R1, R0, 8 **NOP** DADDI R2, R1, 10 LD R7, NUM1 (R0)	**VS**	DADDI R1, R0, 8 LD R7, NUM1 (R0) DADDI R2, R1, 10
		Con *NOPs*		**Ordenando sentencias**

Por hardware

Si ya tenemos los valores necesarios, podemos "adelantarlos"

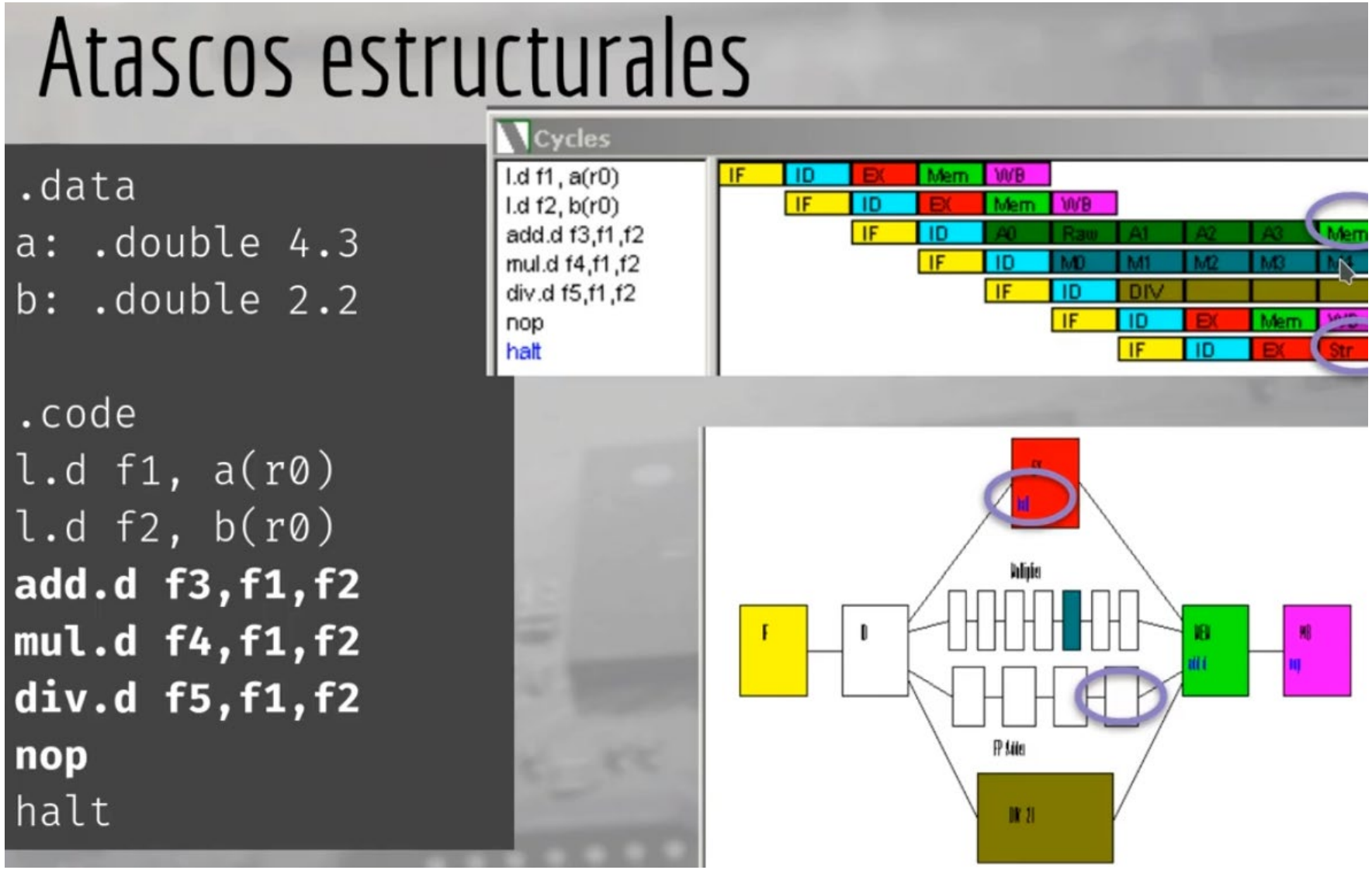


En estos buffers se almacenan los valores para que los puedan usar en las próximas instrucciones

De esta manera no hace falta esperar a las etapas **MEM** y **WB** para usar los valores!

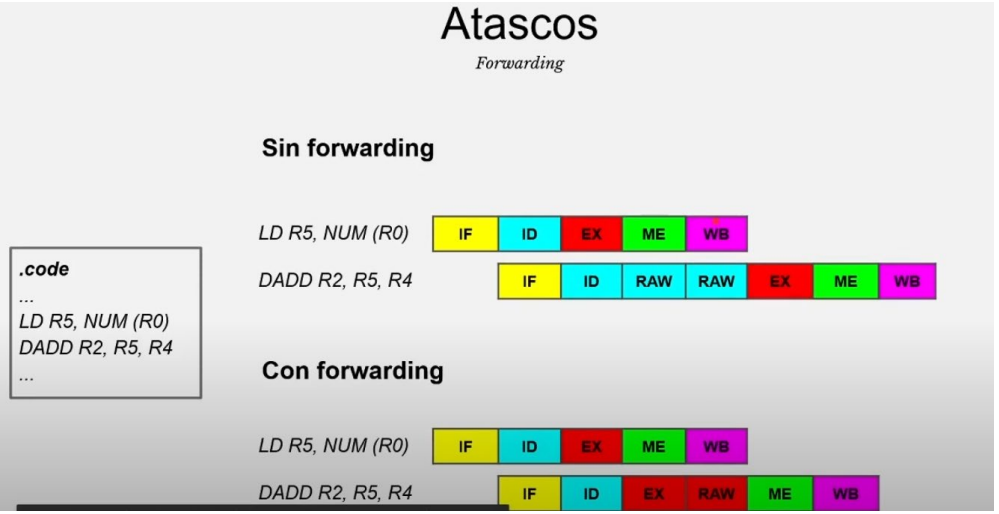
Este adelantamiento de operando lo llamamos **Forwarding**

Atascos Estructurales

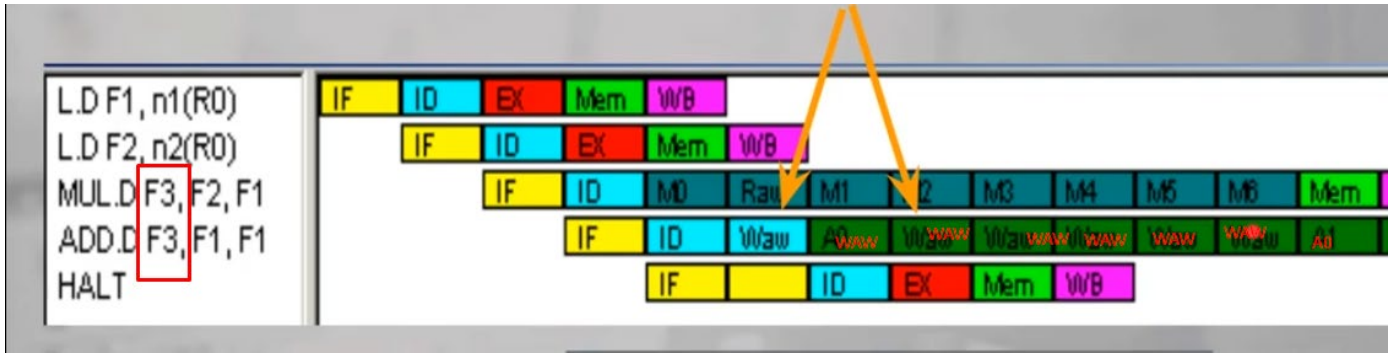


Dependecia de datos

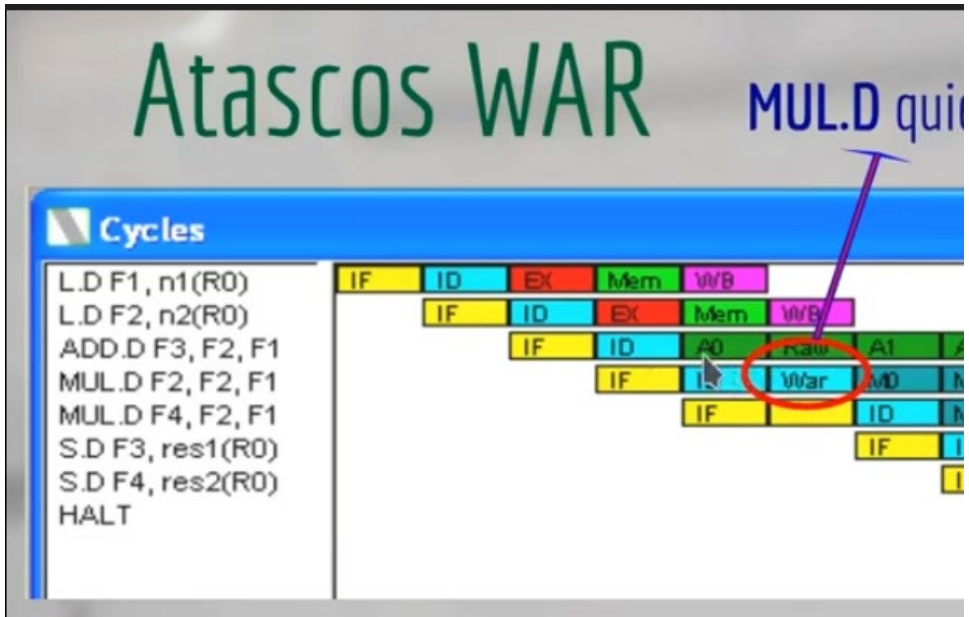
Raw



WAW



Raw está intentando leer algo que todavía no termino de ser escrito en una instrucción anterior
Waw está intentando de escribir en algo que todavía no termino de ser escrito en una instrucción anterior
War: está intentando escribir con un dato que no fue escrito todavía



Write-After-Read, o WAR

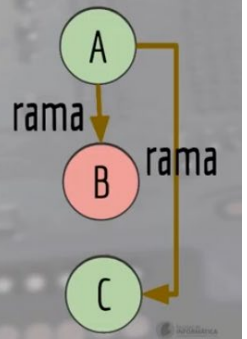
WAW, la cual ocurre cuando dos instrucciones tienen el mismo registro de salida. Sin embargo estas dependencias deben ser resueltas en las arquitecturas mas complejas que veremos mas adelante.

Dependencia de Control

El problema es el ID

```
A: salto a C
B: <cualquier instrucción>
....
C: <cualquier instrucción>
```

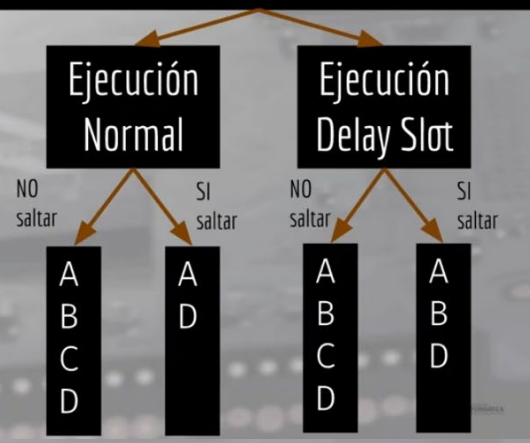
- Instrucción de salto **A**
  - Instrucción siguiente **B**
  - instrucción de salto **C**
- Cuando **A** pasa a la etapa ID, **B** se carga en IF
  - **ID**: se evalúa si la condición de un salto es verdadera o no
    - Verdadera → saltar:
      - Descartar **B** y cargar **C**
      - **Branch Taken Stall (BTS)**
      - (Atasco por rama equivocada)
    - Falsa: seguir con **B** (no hay atasco)



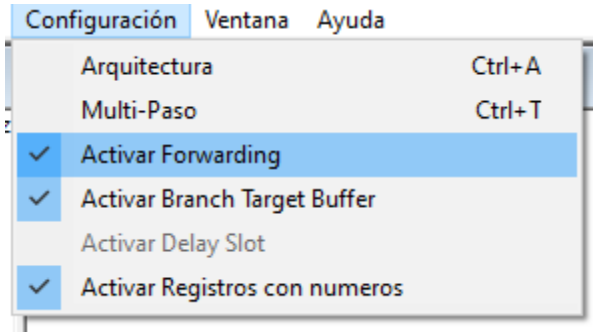
Delay Slot o Salto retardado

- Alternativa al BTB (Branch Target Buffer)
- **CAMBIA LA FORMA EN QUE SE EJECUTAN LOS SALTOS !!!!!!!!**
  - Saltar un ciclo después
  - Ejecutar SIEMPRE la instrucción siguiente al salto
- **0 atascos** garantizados **siempre**
  - Pero hay que ubicar una instrucción después del salto
    - No siempre es posible
    - No mejora el CPI

```
A: Salto condicional a D
B: <instrucción>
C: <instrucción>
D: <instrucción>
```




```
Atascos
0 Atascos RAW
0 Atascos WAW
0 Atascos WAR
0 Atascos Estructurales
0 Atascos Branch Taken
0 Atascos Branch Misprediction
```



Branch Target Buffer (BTB)

BTB	Instrucción: No Saltar	Instrucción: Saltar
No Saltar	0 Atascos	2 Atascos BTS
Saltar	2 Atascos BMS	0 Atascos

- BMS: Branch Misprediction Stall (Atasco por Predicción Incorrecta
  - Como el BTS pero cuando predecís saltar
- ¿Por qué 2 atascos?
  - 1 por cargar la instrucción incorrecta
  - 1 para actualizar la tabla/buffer de BTB

Atascos ejemplos y explicaciones

Atascos

RAW

Escribir despues de leer

RAW significa “Read After Write”

Se produce cuando una instrucción necesita leer un dato que todavía no está disponible

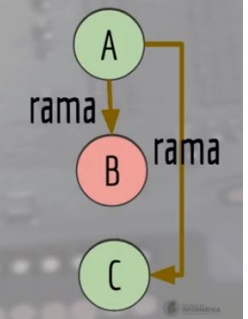
```
.data
NUM1: .word 15

.code
; Inicializamos un registro y le sumamos 10
DADDI R1, R0, 8
DADDI R2, R1, 10
; Después hacemos otra cosa
LD R7, NUM1 (R0)
HALT
```

SIN FORWARDIN
RECIEN EN LA ETAPA WB VA ESTAR DISPONIBLE...
SE ATASCA EN LA ETAPA ID

El problema es el ID

- Instrucción de salto **A**
  - Instrucción siguiente **B**
  - instrucción de salto **C**
- Cuando **A** pasa a la etapa ID, **B** se carga en IF
  - **ID**: se evalúa si la condición de un salto es verdadera o no
    - Verdadera → saltar:
      - Descartar **B** y cargar **C**
      - **Branch Taken Stall (BTS)**
      - (Atasco por rama equivocada)
    - Falsa: seguir con **B** (no hay atasco)

```
A: salto a C
B: <cualquier instrucción>
....
C: <cualquier instrucción>
```



Estoy tratando de leer algo que la intruccion anterior Todavia no escribio

La suma(Resultados) se guarda en la etapa de Wb(sin forwarding)

Con Forwardin se guarda en la etapa Ex en el buffer

BTS (BRANCH TAKEN STALL) seria un atasco por tomar la rama equivocada, en este caso se produce por tener una instruccion de salto y un HALT cerca… el HALT entra en la etapa de IF, pero si quedan saltos por hacer , va a entrar al LOOP y se va a tener que descargar el IF del HALT y se van a perder ciclos o generar dicho atasco BTS..
BTS HAY, SI SE CUMPLE LA CONDICION, TENIENDO Q DESCARGAR LA INTRUCCION Q SE HABIA CARGADO

ID SE CALCULA EL SALTO (R2 ES 0? NO ENTONCES SALTO)
EX SE REALIZA EL SALTO

Atascos De saltos Ejemplos

Branch Target Buffer (BTB)

- Nuevo circuito de la CPU
  - Predicción de saltos
    - BEQZ, BNEZ, BEQ, BNE, J, JR
- Guarda dirección del último salto
- Algoritmo de predicción para cargar próx inst
  - Si nunca se ejecutó el salto
    - Cargar siguiente instrucción
  - Sino
    - Cargar instrucción de la tabla

```
        bnez r1,seguir
        nop
seguir:halt
```

bnez r1,seguir	IF	ID
???		IF

No saltar

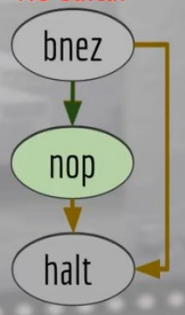


Tabla BTB	
Instrucción	Último salto
bnez	nop

- Caso 1 (tabla vacía)

Siempre arranca en No saltes, para cada intruccion de salto tiene un cajita

Si no le pego a la prediccion, Misprection stall

Branch Target Buffer (BTB)

BTB		Instrucción: No Saltar	Instrucción: Saltar
	No Saltar	0 Atascos	2 Atascos BTS
	Saltar	2 Atascos BMS	0 Atascos

- BMS: Branch Misprediction Stall (Atasco por Predicción Incorrecta
  - Como el BTS pero cuando predecís saltar
- ¿Por qué 2 atascos?
  - 1 por cargar la instrucción incorrecta
  - 1 para actualizar la tabla/buffer de BTB

Técnicas de Hardware

- **Branch Target Buffer (BTB)**
  - Utiliza una tabla/buffer para "predecir" el salto
  - Estrategia
    - Si saltó la vez anterior, predecir que salta
    - Si NO saltó la vez anterior, predecir que NO salta
  - Almacenamiento de la tabla
    - 1 bit por cada salto del programa
- **Delay Slot**
  - Cambia la manera en que se ejecutan los saltos
    - Retarda el salto 1 ciclo
  - Desventaja
    - Los programas tienen que modificarse para seguir funcionando

En la etapa IF chequea si tiene que saltar
Con btb On

```
DADDI R2, R0, 3
LOOP:  DADDI R2, R2, -1
       BNEZ R2, LOOP
NOP ; Instrucción problemática
```

Cuando esta en no saltar la cajita [0]no
If r2<>0.. La caga 1bts, por rama equiv
La prox salta, cambiar eso 1 bts +

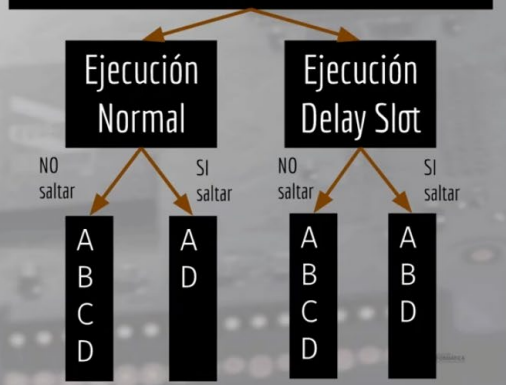
Cuando esta en saltar la cajita[1]si
Salta, no habia q saltar 1 mispredic
Y actualiza la table 1 + de mispredc

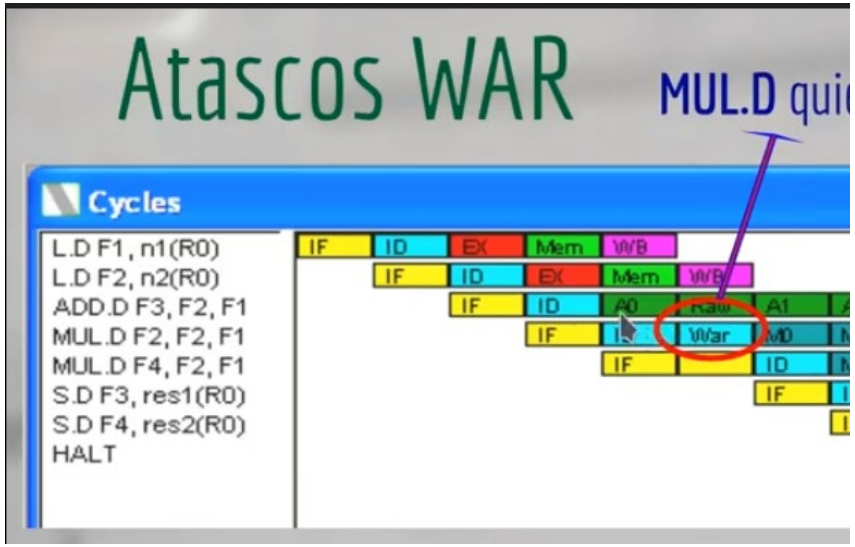
Atascos

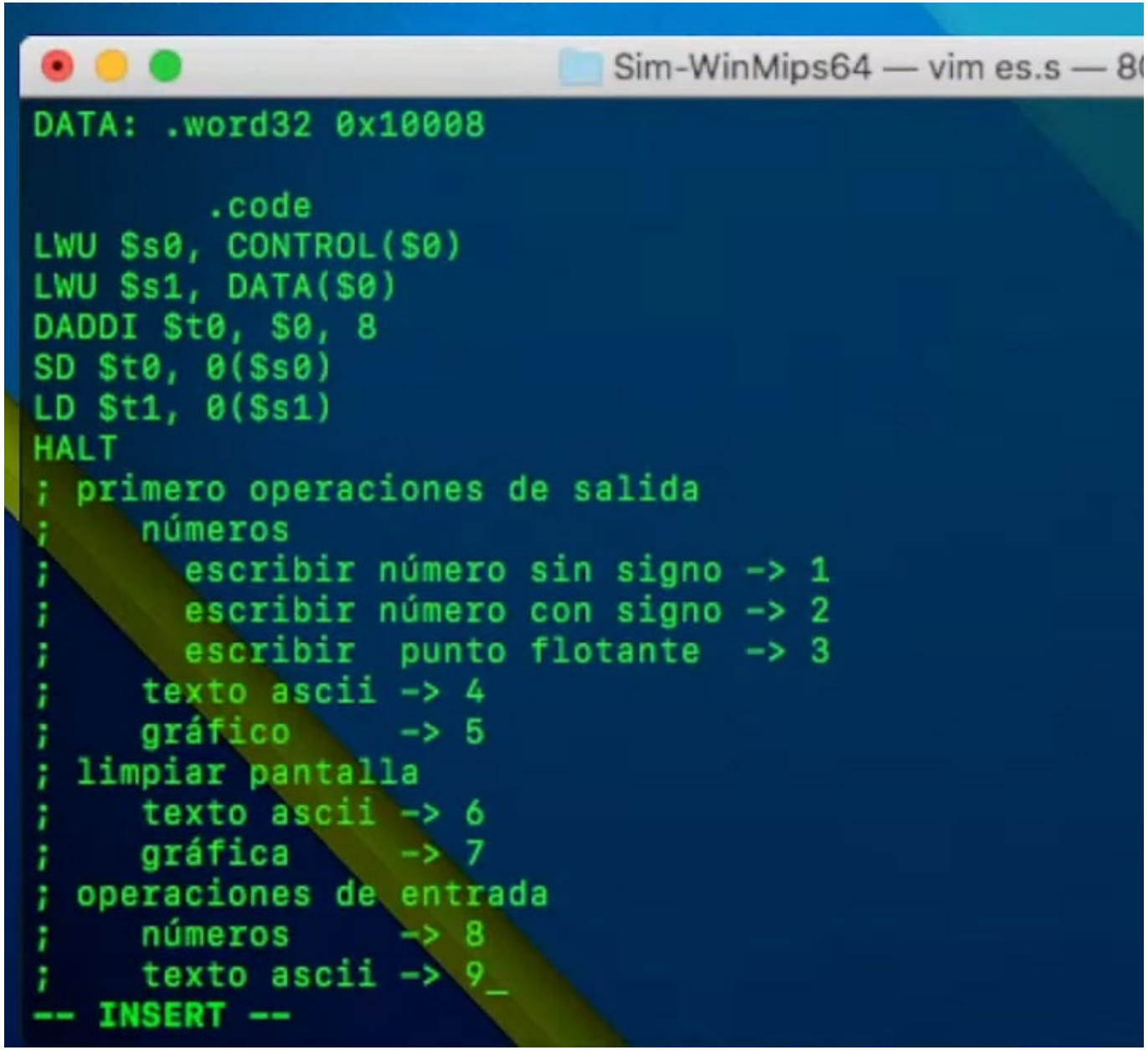
Delay Slot o Salto retardado

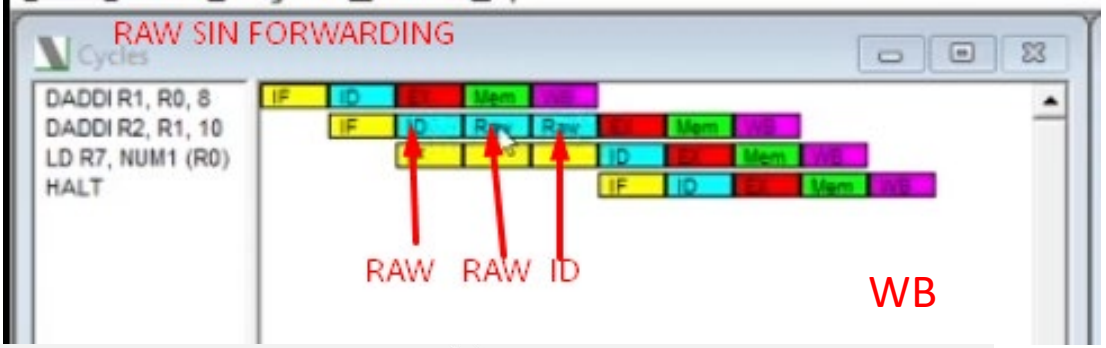
- Alternativa al BTB (Branch Target Buffer)
- CAMBIA LA FORMA EN QUE SE EJECUTAN LOS SALTOS !!!!!!!!!
  - Saltar un ciclo después
  - Ejecutar SIEMPRE la instrucción siguiente al salto
- **0 atascos** garantizados **siempre**
  - Pero hay que ubicar una instrucción después del salto
    - No siempre es posible
    - No mejora el CPI

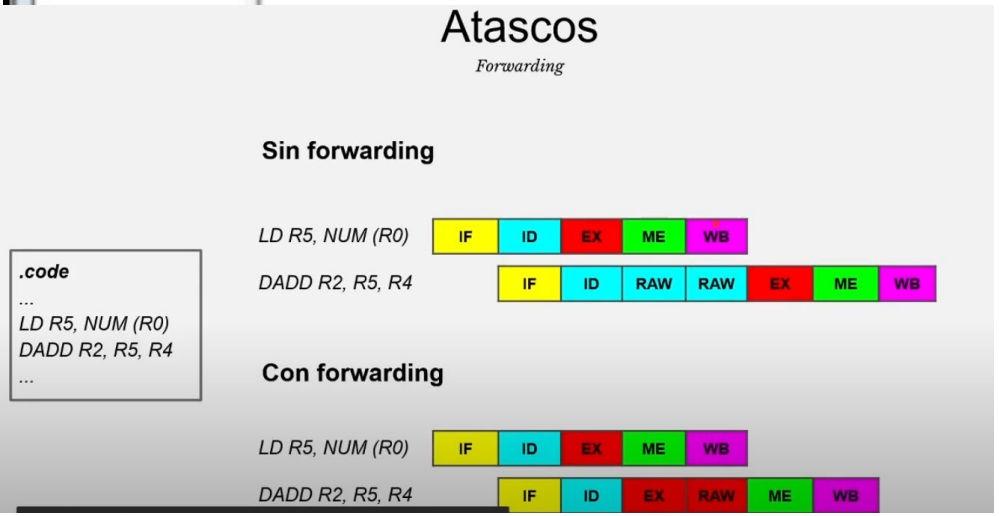
```
A: Salto condicional a D
B: <instrucción>
C: <instrucción>
D: <instrucción>
```





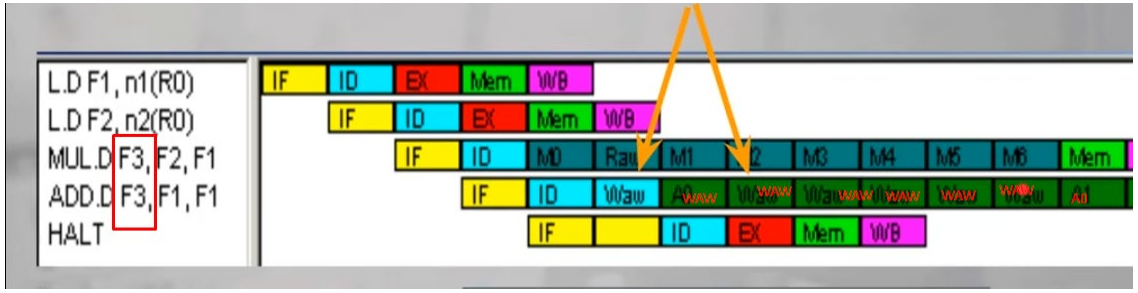


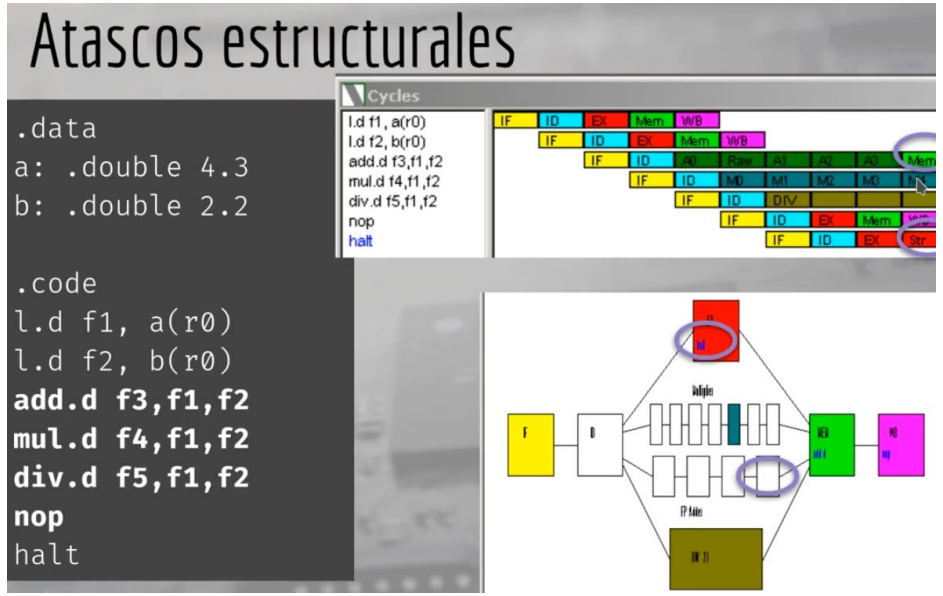




Atascos Raws, nop, ordenamientos de sentencias
O por Hardware por mis prediccion

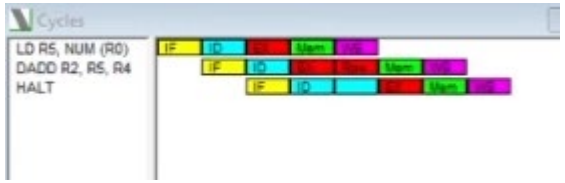
WAW





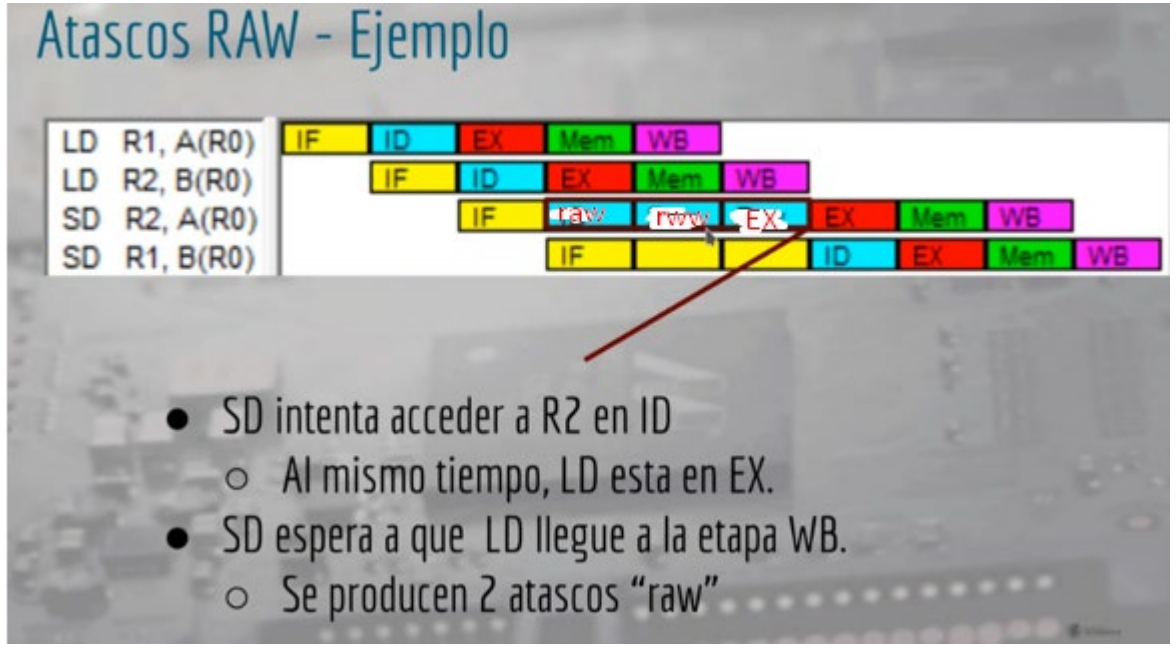
Mas de lo mismo
Raw profundizado

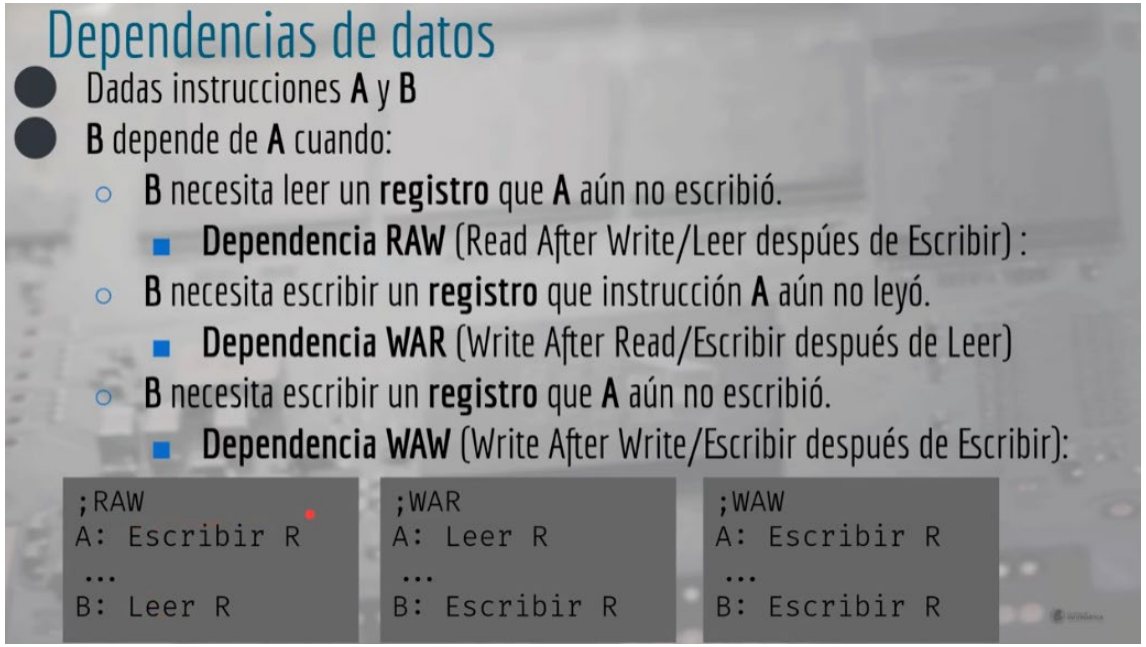
Raw



Con forwarding

Se produce raw porque dadd quiere utilizer r5, pero r5 todavia el valor de num, recien en MM lo tengo disponible





Dependencias RAW (*posibles* atascos RAW)

```
daddi r1,r2,4 ;escribe
...
daddi r3,r1,r2; lee
```

```
daddi r1,r2,4 ;escribe
...
bneq r1,r2, ETIQUETA
```

```
daddi r1,r0,4;escribe
...
sd r1,A(r0); lee
```

```
ld r1,A(r2)
...
daddi r2,r3,r1
```

```
daddi r1,r2,4;escribe
...
sd r2,A(r1); lee
```

```
ld r1,A(r2)
...
sd r1,B(r2)
```

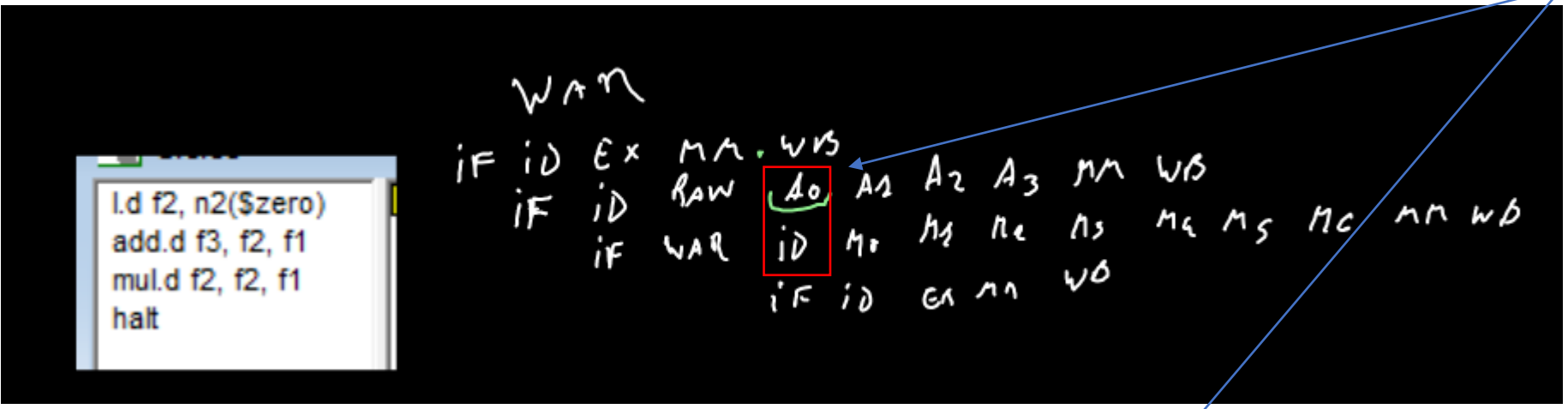
```
daddi r1,r2,4;escribe
...
ld r2,A(r1); lee
```

```
ld r1,A(r2)
...
bnez r1, ETIQUETA
```

Adelantamiento de operandos o **Forwarding**

1. Se puede acceder a resultados antes de WB
   - Acceso a registros temporales (intermedios)
2. Instrucciones se atascan solo cuando van a utilizar los operandos y no están
   - Etapa ID no verifica los operandos
     - Salvo para saltos condicionales
   - Instrucciones aritmético/lógicas se atascan en **EX**
   - Instrucciones **LD/SD** se atasca en **EX** (si no está el desplazamiento)
   - Instrucción **SD** se atasca en **MEM** (si no está el valor a guardar)

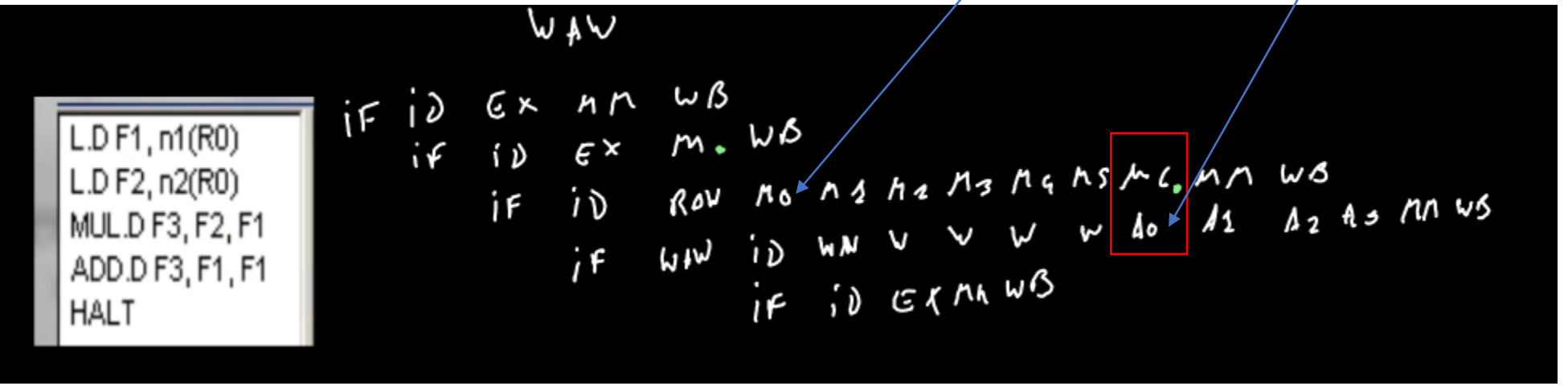
WAW Y WAR + PROFUDIZADO



Llega a ex y tengo mis operandos para calcular

Intruccion anterior a la posterior:
Flaco no escribes que todavia
No lo lei

Arranca q no hay peligro, ya termino



Intruccion anterior a posterior:
Para loco no soy tan rapido
Atascate o vas a escribirme
cualquier cosa

Imprimir pixeles

Entrada/Salida

Pantalla gráfica. Ejemplo 2

```
.data
coorX: .byte   24 ; Coordenada X
coorY: .byte   24 ; Coordenada Y
color:  .byte   255, 0, 255, 0 ; Máximo rojo + máximo azul = magenta
CONTROL: .word 0x10000
DATA: .word 0x10008

.code
lwu  $s0, CONTROL ($zero)     ; $s0 = dirección de CONTROL
lwu  $s1, DATA ($zero)        ; $s1 = dirección de DATA
daddi $t0, $zero, 7           ; $t0 = 7 -> función 7: limpiar pantalla gráfica
sd $t0, 0 ($s0)               ; CONTROL = 7 (limpia la pantalla gráfica)
lbu $t0, coorX ($zero)        ; $t0 = valor de coordenada X
sb $t0, 5 ($s1)               ; DATA + 5 recibe el valor de coordenada X
lbu $t1, coorY ($zero)        ; $t1 = valor de coordenada Y
sb $t1, 4 ($s1)               ; DATA + 4 recibe el valor de coordenada Y

lwu $t2, color ($zero)  ; $t2 = color
sw $t2, 0 ($s1) ; Pongo color en DATA
daddi $t0, $zero, 5
sd $t0, 0 ($s0) ; Pinta el píxel
HALT
```

```
.data
  control: .word32 0x10000
  data: .word32 0x10008

  coordX: .byte 4
  coordY: .byte 4
  color: .byte 0,0,0,0 #COLOR NEGRO  R G B A

.code
  lwu $s0, control($zero)
  lwu $s1, data($zero)

  #limpiamos pantalla grafica
  daddi $t0, $zero, 7
  sd $t0, 0($s0)

  #siempre me traigo las coordenadas con LBU
  #mando coordenada X
  lbu $t0, coordX($zero)
  sb $t0, 5($s1)

  #coordenada Y
  lbu $t1, coordY($zero)
  sb $t0, 4($s1)

  #cargo el color
  lwu $t2, color($zero)
  sw $t2, 0($s1)
  daddi $t0, $zero, 5
  sd $t0, 0($s0)        #pinta el pixel
halt
```

Pila

Solución con Pila

Anidada

```
.code
daddi $sp,$sp,0x400
jal subrutina1
halt

subrutina2: nop
            jr $ra
```

```
subrutina1: daddi $sp,$sp,-8 ;push $ra
            sd $ra, 0($sp)

            jal subrutina2
            nop

            ld $ra, 0($sp) ;pop $ra
            daddi $sp,$sp,8
            jr $ra
```

```
#mayusculas y minusculas rango de letras
#41H-5AH / 61H-7AH no lo use en este
.data
vocal: .asciiz "AEIOUaeiou"
caracterES: .asciiz "saracatunga"
contV: .word 0 #1 si es vocal, 0 si no lo es

.code
daddi $sp, $zero, 0x400 #pila
#daddi $a1, $zero, vocal #direc vocal
daddi $a2, $zero, caracterES #direc caracterES
#lbu $a0, caracter($zero) #carga el caracter en $a0

jal validador #llama a validador
sd $v0, contV($zero) #guarda el resultado en la variable Es
halt


validador: daddi $sp, $sp, -8 #push
sd $ra, 0($sp) #push
dadd $v0, $zero, $zero #inicializa el resultado en 0
lopardo1: lbu $t5, 0($a2) #carga el caracter en $t5
beqz $t5, terminaTodo

#aca tengo que guardar sp - r31xxxxxxxxxxxxxxxxxxxx
jal es_vocal

daddi $a2, $a2, 1 #incrementa el puntero

j lopardo1

terminaTodo: ld $ra, 0 ($sp) #pop
daddi $sp, $sp, 8
jr $ra

es_vocal: daddi $a1, $zero, vocal #direc vocal
lopardo: lbu $t0, 0($a1) #carga el primer caracter de la cadena vocal
beqz $t0, fin #me quede sin vocales a comparar, salgo
beq $t5, $t0, es #comparo el caracter con la vocal
daddi $a1, $a1, 1 #paso a la siguiente vocal
j lopardo #vuelvo a comparar

es: daddi $v0, $v0, 1 #si es vocal, incremento el resultado

fin: jr $ra
```

Punto flotante

Comparaciones y flags

- c.**lt**.d f_i, f_j
  - Pone FP en 1 si $f_i < f_j$
- c.**le**.d f_i, f_j
  - Pone FP en 1 si $f_i <= f_j$
- c.**eq**.d f_i, f_j
  - Pone FP en 1 si $f_i = f_j$
- bc1t **<etiqueta>**
  - Salta a **<etiqueta>** si FP=1
  - idem bc1f si FP=0

comparaciones se guardan en "FA"

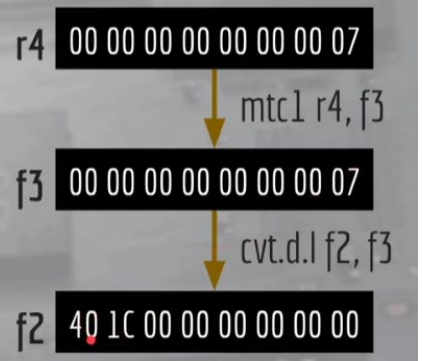
ESE FA, SE ACCEDE CON ESTA INSTRUCCION

```
# Calcular max entre A y B
# Guardar en C
.data
A: .double 4.5
B: .double 3.2
C: .double 0
.code
        l.d f1, A(r0)
        l.d f2, B(r0)
        c.lt.d f1, f2
        bc1t bmayor
        ; a es mayor
        s.d f1,C(r0)
        j fin
bmayor: s.d f2,C(r0)
fin:    halt
```

Decimal a flotante

Conversión de punto fijo a flotante

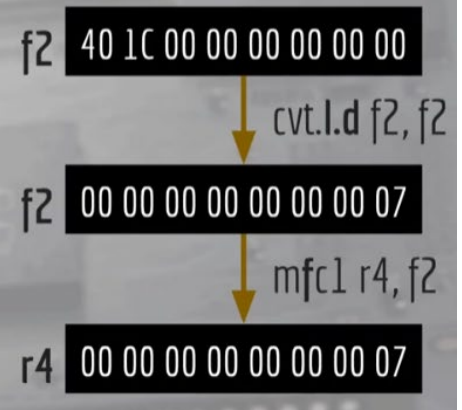
- Valor en registro **r**
- Pasar a registro **f**
- Ejemplo
  - Pasar el valor de **r4** a **f2**
- Dos instrucciones
  - mtc1 r4, f3
    - **Copia** los **bits**
  - cvt.d.l f2, f3
    - **Convierte bits** de fijo a flotante



Flotante a decimal

Conversión de punto flotante a fijo

- Valor en registro **f**
- Pasar a registro **r**
- Ejemplo
  - Pasar el valor de **f2** a **r4**
- Dos instrucciones
  - cvt.**l.d** f2, f2
    - **Convierte bits** de flotante a fijo
  - mfc1 r4, f2
    - **Copia** los **bits**



Ejemplo con lo anterior

Conversión de punto fijo a flotante

- Valor en registro **r**
- Pasar a registro **f**
- Ejemplo
  - Pasar el valor de **r4** a **f2**
  - **Sin pasar por f3**
    - mtc1 r4, **f2**
    - cvt.d.l f2, **f2**

```
# Convertir el valor de A
# a flot y guardarlo en B
.data
A: .word 5
B: .double 0

.code
ld r4, A(r0)
mtc1 r4, f2
cvt.d.l f2, f2
s.d f2, B(r0)
halt
```

Conversión de punto flotante a fijo

- Valor en registro **f**
- Pasar a registro **r**
- Ejemplo
  - Pasar el valor de **f2** a **r4**
- Dos instrucciones
  - cvt.**l.d** f2, f2
    - **Convierte bits** de flotante a fijo
  - mfc1 r4, f2
    - **Copia** los **bits**

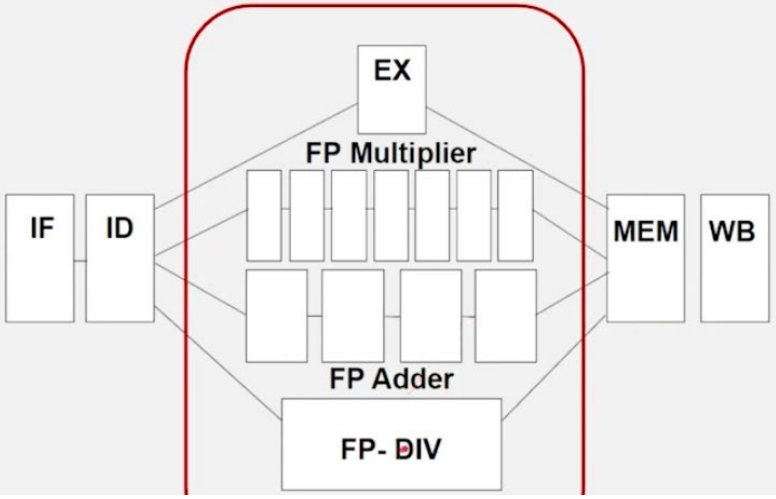
```
# Convertir el valor de A
# a fijo y guardarlo en B
.data
A: .double 5
B: .word 0

.code
l.d f2, A(r0)
cvt.l.d f2, f2
mfc1 r4, f2
s.d r4, B(r0)
halt
```

Ciclos

Punto Flotante

Instrucciones pesadas

Recordar que: **no todas las etapas tardan lo mismo**



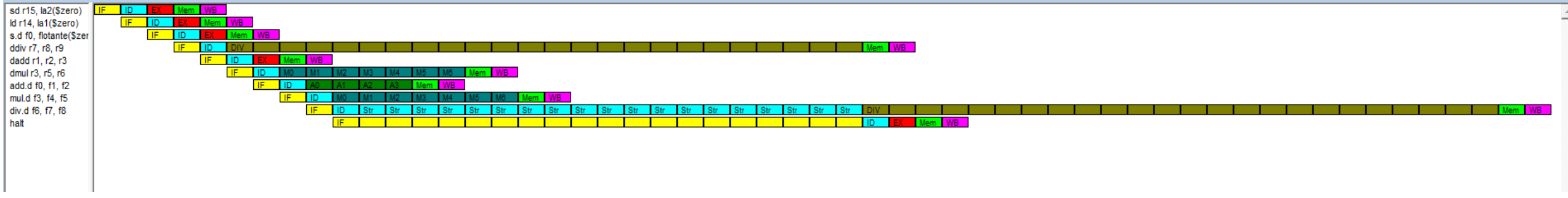
- Generales = 1 ciclo
- Multiplicar en Pto. F. = 7 ciclos
- Sumar en Pto. F. = 4 ciclos
- Dividir en Pto. F. = 24 ciclos

Intrucciones de registros enteros

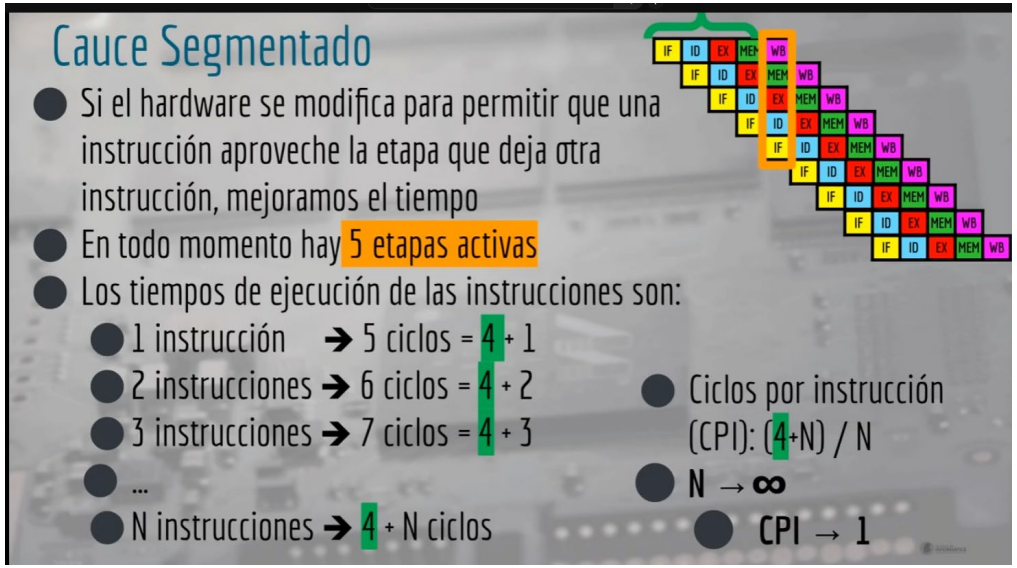
todas son de 5 ciclos, si no temenos atascos

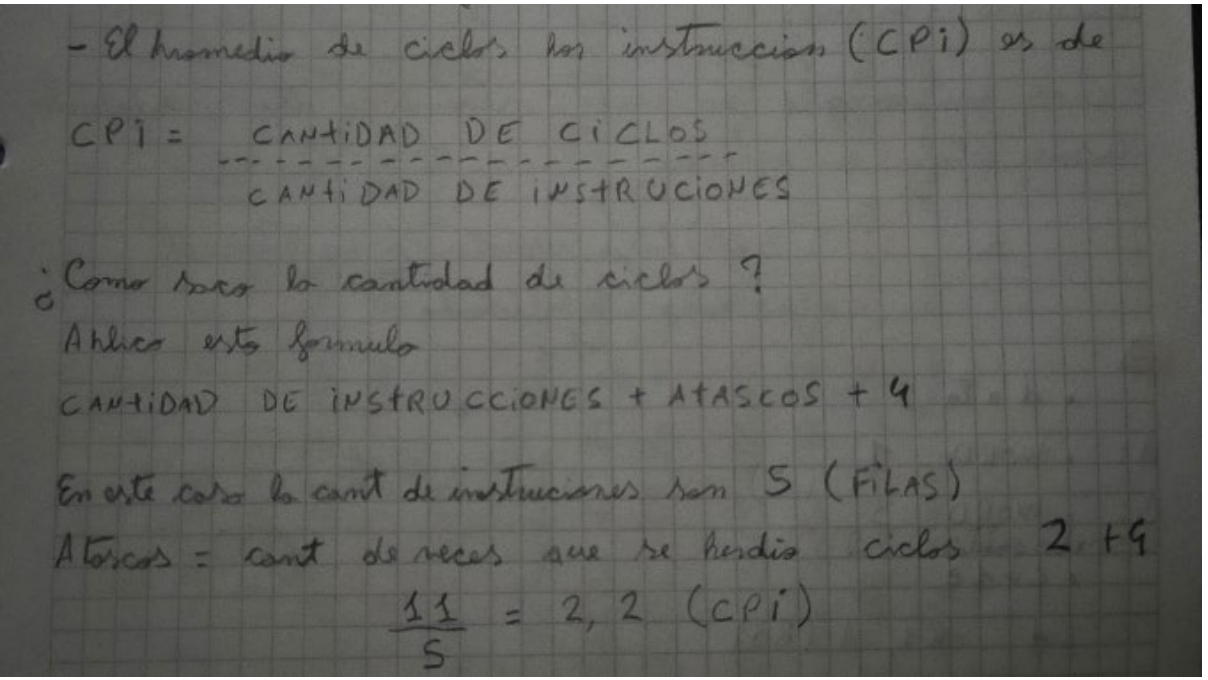
```
.data
  la1: .word 0
  la2: .word 2
  flotante: .double 0.0
.code
  #en Punto flotante es lo mismo
  sd r15, la2($zero)
  ld r14, la1($zero)
  s.d f0, flotante($zero)
  #normal
  ddiv r7, r8, r9
  dadd r1, r2, r3   #suoma/resta
  dmul r3, r5, r6


  #punto flotante
  add.d f0, f1, f2  #suoma/resta
  mul.d f3, f4, f5
  div.d f6, f7, f8


halt
```

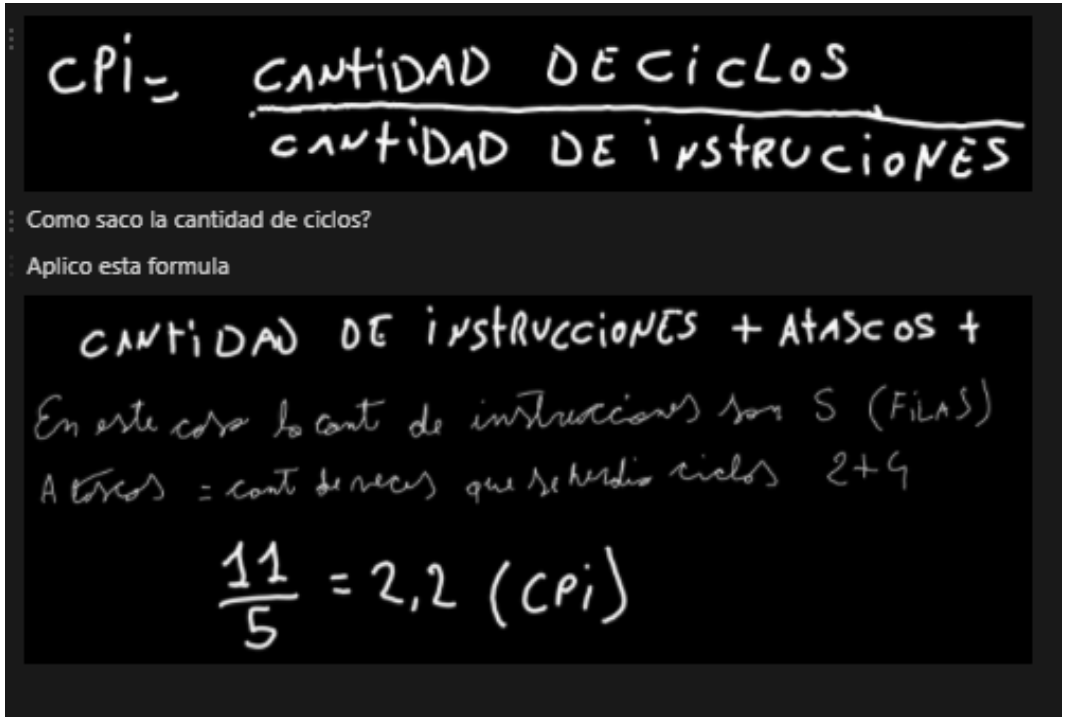
Pasan conflictos de recursos en P.F pero no va al caso, esto se hizo para ver los ciclos



Calcular CPI, si no es Punto flotante





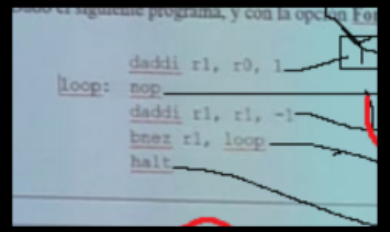


ECUACION PARA CALCULAR CPI=

4 + CANT INSTRUCCIONES + CANT ATASCOS TOTALES / CANT INSTRUCCIONES

(EL HALT CUENTA COMO INSTRUCCION)

Si es punto flotante y la que nunca falla



CANT DE CICLOS + CANT + INSTRUCCIONES + 5 / CANT DE INSTRUCCIONES = 5 + 3 + 5 / 5

O CAGARSE A PIÑAS

IF ID EX MM WB
IF ID EX MM WB
IF RAW ID EX MM WB
IF RAW RAW ID EX MM WB
IF ID EX MM WB

CICLOS / CAN DE ISTRUC = 13/5

Segmentación en MIPS64

Búsqueda (IF)

- Se accede a memoria por la instrucción
- Se incrementa el PC

Decodificación / Búqueda de operandos (ID)

- Se decodifica la instrucción
- Se accede al banco de registros por los operandos
- Se calcula el valor del operando inmediato con extensión de signo *(si hace falta)*
- Si es un salto, se calcula el destino y si se toma o no

Ejecución / Dirección efectiva (EX)

- Si es una instrucción de procesamiento, se ejecuta en la ALU
- Si es un acceso a memoria, se calcula la dirección efectiva
- Si es un salto, se almacena el nuevo PC

Acceso a memoria / terminación del salto (MEM)

- Si es un acceso a memoria, se accede

Almacenamiento (WB)

- Se almacena el resultado *(si lo hay)* en el banco de registros

Conveciones

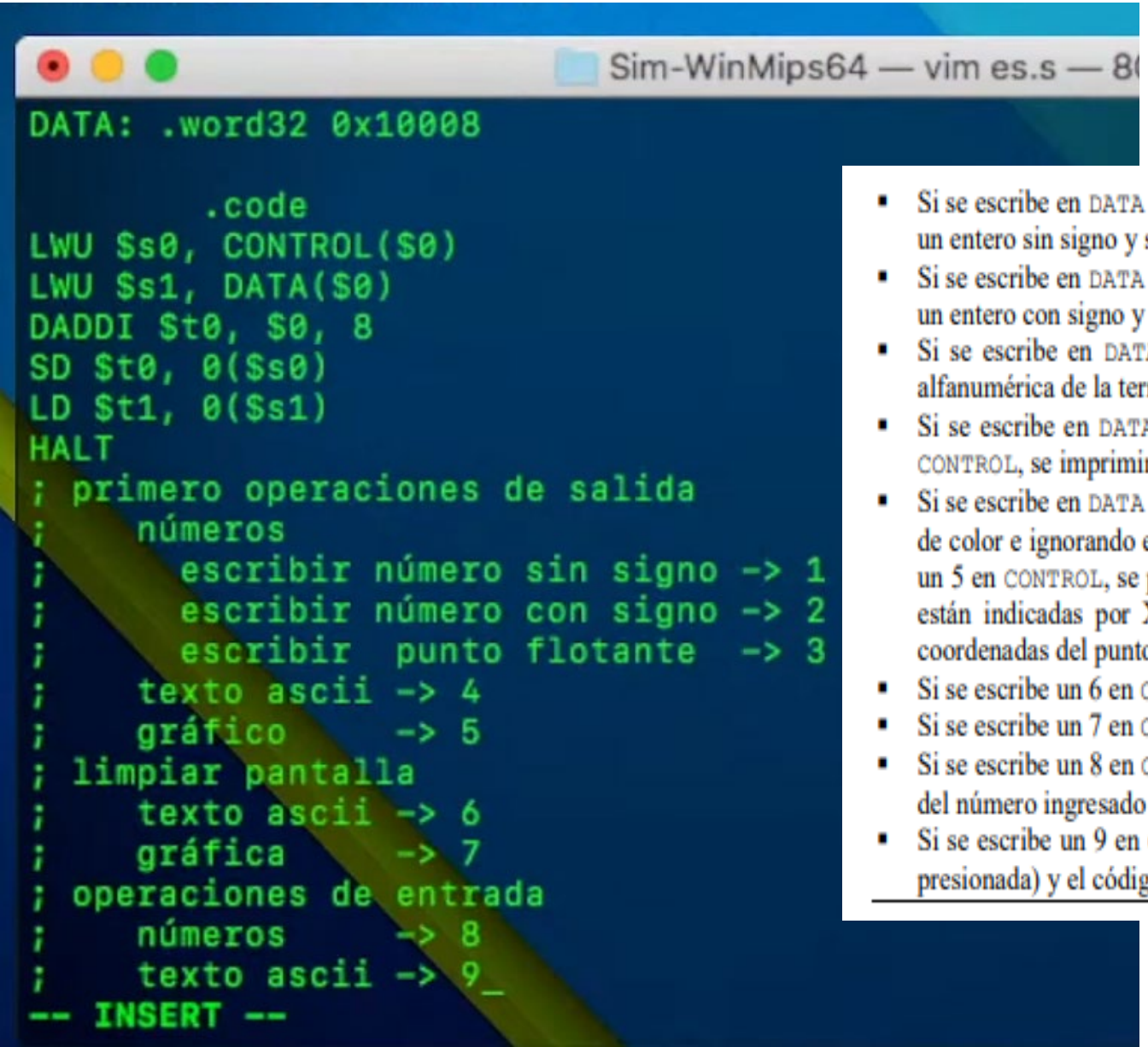
Las verdes son los que tengo/tienen que ser salvados

Convenciones

Ante la cantidad de registros y consideraciones que debemos tener en cuenta, se establecieron **convenciones**:

AHORA	DESCRIPCIÓN	ANTES
$zero	Siempre tiene el valor 0 y no se puede cambiar	(r0)
$ra	*Return Address* – Dir. de retorno de subrutina. Debe ser salvado	(r31)
$v0-$v1	Valores de retorno de la subrutina llamada	(r2-r3)
$a0-$a3	Argumentos pasados a la subrutina llamada	(r4-r7)
$t0-$t9	Registros temporarios	(r8-r15 y r24-r25)
$s0-$s7	Registros que deben ser salvados	(r16-r23)
$sp	*Stack Pointer* – Puntero al tope de la pila. Debe ser preservado	(r29)
$fp	*Frame Pointer* – Puntero de pila. Debe ser salvado	(r30)
$at	*Assembler Temporary* – Reservado para ser usado por el ensamblador	(r1)
$k0-$k1	Para uso del kernel del sistema operativo	(r26-r27)
$gp	*Global Pointer* – Puntero a zona de memoria estática. Debe ser salvado	(r28)

Codigos



- Si se escribe en DATA un número entero y se escribe un 1 en CONTROL, se interpretará el valor escrito en DATA como un entero sin signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un número entero y se escribe un 2 en CONTROL, se interpretará el valor escrito en DATA como un entero con signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un número en punto flotante y se escribe un 3 en CONTROL, se imprimirá en la pantalla alfanumérica de la terminal el número en punto flotante.
- Si se escribe en DATA la dirección de memoria del comienzo de una cadena terminada en 0 y se escribe un 4 en CONTROL, se imprimirá la cadena en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un color expresado en RGB (usando 4 bytes para representarlo: un byte para cada componente de color e ignorando el valor del cuarto byte), en DATA+4 la coordenada Y, en DATA+5 la coordenada X y se escribe un 5 en CONTROL, se pintará con el color indicado un punto de la pantalla gráfica de la terminal, cuyas coordenadas están indicadas por X e Y. La pantalla gráfica cuenta con una resolución de 50x50 puntos, siendo (0, 0) las coordenadas del punto en la esquina inferior izquierda y (49, 49) las del punto en la esquina superior derecha.
- Si se escribe un 6 en CONTROL, se limpia la pantalla alfanumérica de la terminal.
- Si se escribe un 7 en CONTROL, se limpia la pantalla gráfica de la terminal.
- Si se escribe un 8 en CONTROL, se permitirá ingresar en la terminal un número (entero o en punto flotante) y el valor del número ingresado estará disponible para ser leído en DATA.
- Si se escribe un 9 en CONTROL, se esperará a que se presione una tecla en la terminal (la cuál no se mostrará al ser presionada) y el código ASCII de dicha tecla estará disponible para ser leído en DATA.

Cosas a tener encuenta

6. ¿Cuantas opciones para generar salidas en la pantalla alfanumérica tiene WinMIPS64? (1 pto)

4

1-enteros s/s ; 2-enteros c/s 3-flotante ; 4-string

Desplazamiento
.ascciz de a 1
.word de a 8
.word32 de a 4

8. Complete con la cantidad de ciclos que tarda cada una de las siguientes instrucciones teniendo en cuenta que no se producen atascos. **(0.5 pto c/u)**

LB R1, ETIQUETA (R0)	5	ADD.D F5, F6, F7	8	F ID A0 A1 A2 A3 M W
S.D F1, NUMERO (R0)	5	DIV.D F8, F9, F10	28	F ID D0 D1 D2..D23 M W

9. Bajo la convención para el uso de registros, indique que representan los siguientes tipos. **(1pto c/u)**

```
#par e impar
.data
.code
  daddi r2, $zero, 3 #cargar num a verifcar
  andi r4, r2, 1     #si ese numero es impar va dejar un 1 en r1,
                     #si es par va a dejar un 0
halt
```

3. Indicar que tipo de atasco por dependencia de datos se producirá durante la ejecución de un programa que incluya las siguientes instrucciones, consecutivas entre sí, y explicar el motivo. Asumir que no hay atascos previos y que Forwarding está habilitado. **(1pto)**

mul.d f1, f3, f2 D

add.d f3, f2, f4 D

F D M0 M1 M2 M3 M4 M5 M6 M W

F D A0 A1 A2 A3 M W

NO HAY ATASCOS

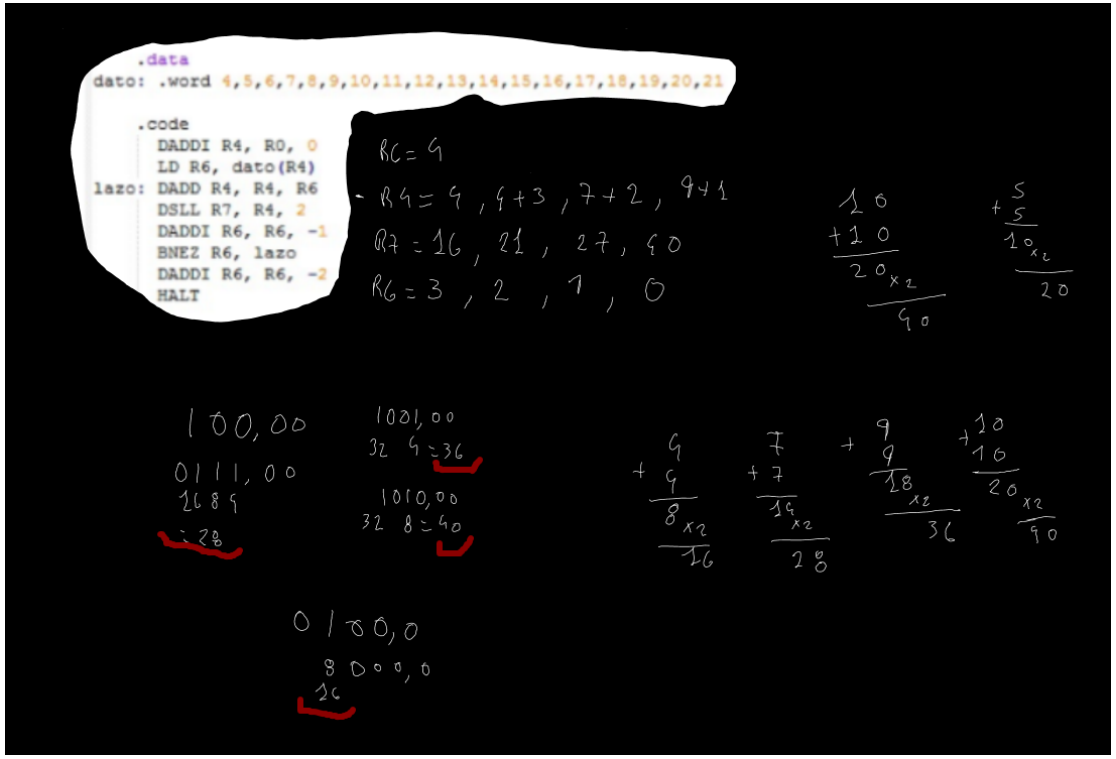
CICLOS = COLUMNAS | INTRUCCIONES = FILAS

3. Enumere los *tipos* de atascos que pueden encontrarse en el simulador Winmips. (1 pto)

dependencia de datos
dependencia de control
dependencia estructural

Arn

Mas cosas



lb ; byte
lbu ; byte sin extensión de signo
lh ; 2 bytes
lhu ; 2 bytes sin extensión de signo
lw ; 4 bytes
lwu ; 4 bytes sin extensión de signo
ld ; 8 bytes

Caracteres con lbu $t0, caracter($zero)

waw tambien pasa sin punto flotante, War todavía no lo pude verificar