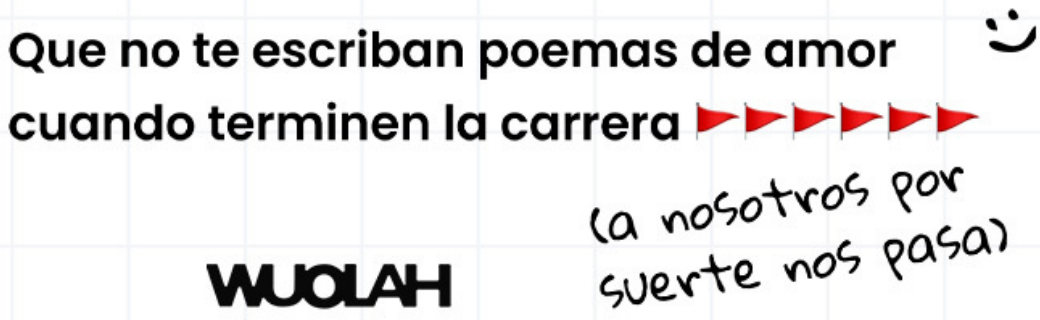
Resumen-wuolah.pdf

EPS_UAM_4o

Arquitectura de Ordenadores

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid



Chapter 1 - Computer Abstractions and Technology

CPU Performance:

Performance = 1/Execution Time

"X is n time faster than Y"

$$n = \frac{\text{Rendimiento(X)}}{\text{Rendimiento(Y)}} = \frac{T_{EJEC}(Y)}{T_{EJEC}(X)}$$

$$CPI = \%_{TotalInstructions} * Clock\ Cycles\ of\ this\ operation\ in\ the\ emulation$$

$$CPU\ Time\ (T_{CPU}) = Instruction\ Count\ x\ Cycles\ Per\ Instruction\ x\ \frac{1}{Frequency}$$

$$= \frac{IC\ x\ CPI}{Freq}$$

Cycles Per Instruction (CPI) es una media. Si hay diferentes tipos o clases de instrucciones con diferentes tiempos tendrán que ser tomadas en cuenta:

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

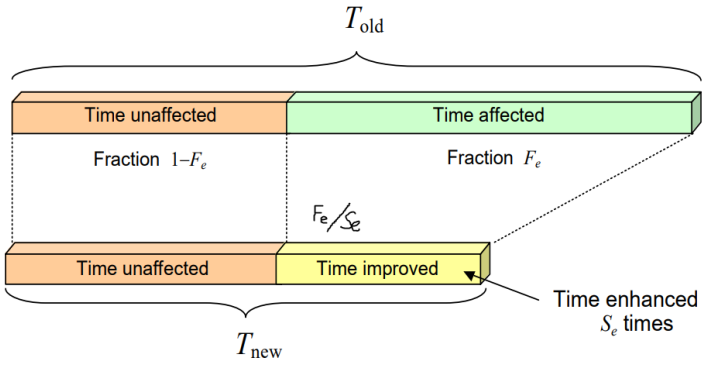
System speedups:

Amdahl´s Law is used to find the speedup to an overall system when only part of the system is improved: $F_e = \frac{T_x\ (Before\ improvement!!)}{T_{CPU\ (Before\ improvement!!)}} = \frac{IC_{CPU}\ x\ CPI_x\ x\ \frac{1}{Freq_{CPU}}}{IC_{CPU}\ x\ CPI_{CPU}\ x\ \frac{1}{Freq_{CPU}}}$; $S_e = \frac{Old\ Clock\ Cycles}{New\ Clock\ Cycles}$

$$Speedup_{Overall} = \frac{1}{(1-F_e) + \frac{F_e}{S_e}}$$

F_e: Fraction enhanced

S_e: *Speedup* enhanced. Ratio of improvement



or

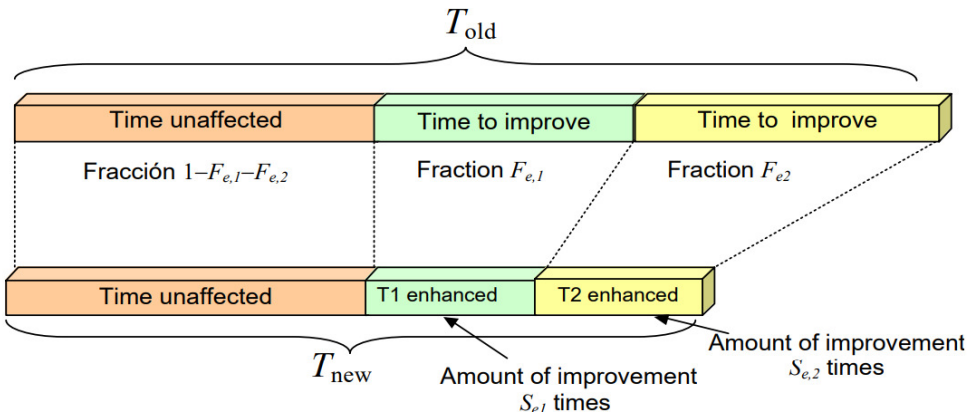
Execution time after improvement = Execution time of unaffected part +

(Execution time of affected part / Amount of Improvement)

Speedup Overall = Time before / Time after

Amdahl´s Law for several improvements (Acc is Se (Speedup enhanced)):

$$Acc_{global} = \frac{1}{1 - \sum_{i=1}^{n} F_{m,i} + \sum_{i=1}^{n} \frac{F_{m,i}}{Acc_{m,i}}}$$



MIPS (Millions of Instructions Per Second): No tiene en cuenta las diferencias en el conjunto de operaciones entre ordenadores ni las diferencias en la complejidad de las instrucciones.

$$MIPS = \frac{Instruction\ Count}{CPU\ Time\ x\ 10^6} = \frac{IC}{IC\ x\ CPI\ x\ \frac{1}{Freq}\ x\ 10^6} = \frac{Freq}{CPI\ x\ 10^6}$$

MFLOPS (Millions of Floating-point Operations Per Second): Efectiva y precisa forma de medir rendimiento

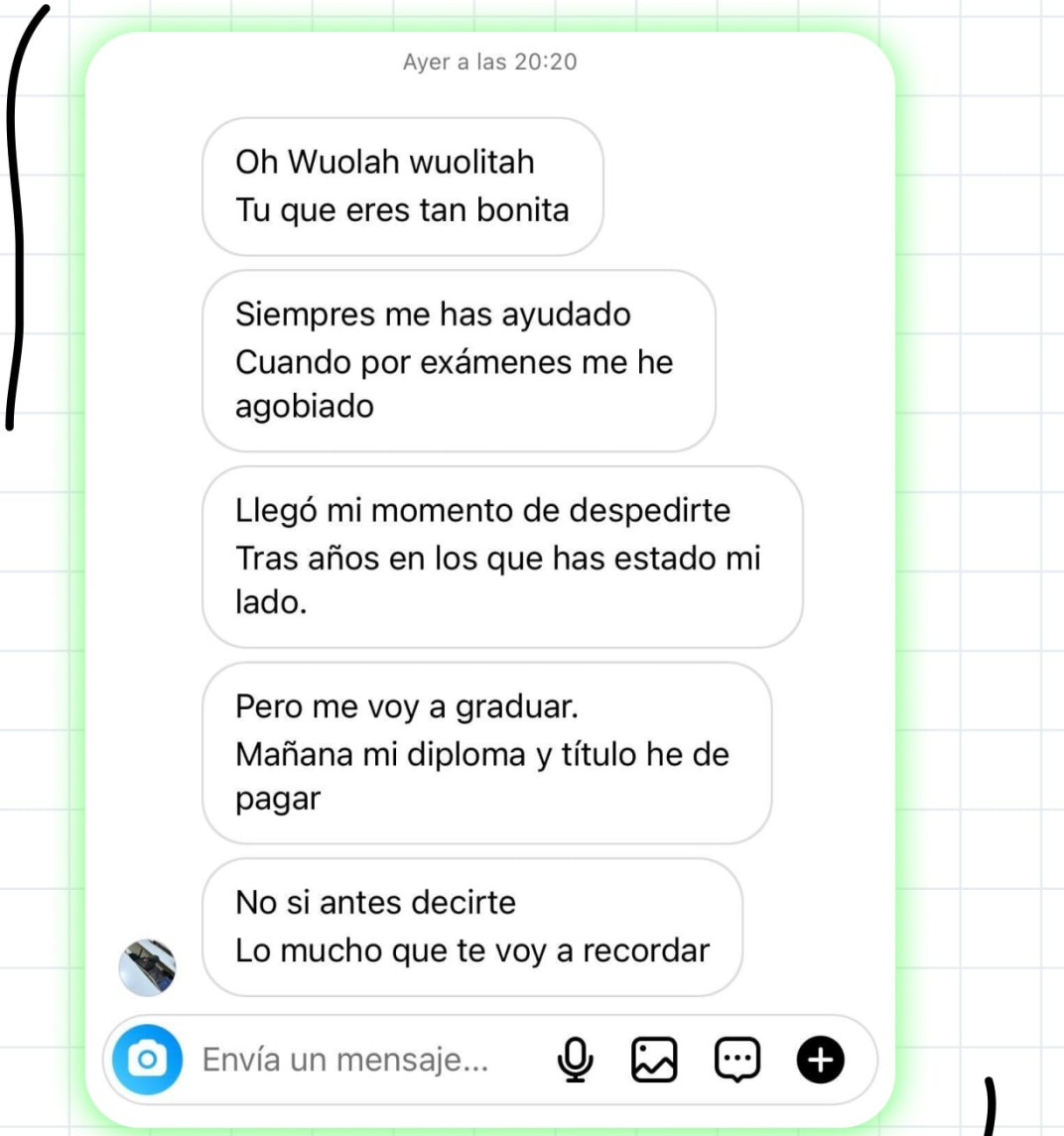
$$MFLOPS = \frac{Floating\ Point\ Instruction\ Count}{CPU\ Time\ x\ 10^6}$$

Power: Se busca reducir el consumo de energía reduciendo la carga, el voltaje y la frecuencia siempre que sea posible:

$$Power = Capacitive\ load \times Voltage^2 \times Frequency$$

Que no te escriban poemas de amor cuando terminen la carrera 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)



WUOLAH

Suppose a new CPU has

- 85% of capacitive load of old CPU
- 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

Chapter 2 - The Pipelined Processor

MIPS INSTRUCTIONS

op: operación básica de la instrucción, tradicionalmente llamada código de operación u opcode.

rs: el registro del primer operando fuente.

rt: el registro del segundo operando fuente.

rd: el registro del operando destino, donde se almacena el resultado de la operación.

shamt: cantidad de desplazamientos (*shift amount*). (La sección 2.6 explica instrucciones de desplazamiento y este término, que no será utilizado hasta entonces y por tanto el campo contiene cero).

funct: función. Selecciona la variante específica de la operación en el campo *op* y a veces es llamado *código de función*.

R-type:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Ex.: add, and, div, jr, mult, or, slt, sub, xor. En R-type, op = 0.

add rd, rs, rt (donde op = 0 y funct = 100000)

I-type:

op	rs	rt	constante o dirección
6 bits	5 bits	5 bits	16 bits

Ex.: lw, sw, addi, andi, beq, lui, ori, slti, xori. lw y sw usan dirección, las demás constante.

addi rt, rs, imm (donde op = 001000)

En lw -> op = 35 y rt es el registro destino: lw Reg.Dest, Offset(Reg.Source)

En sw -> op = 43 y rt es el registro fuente: sw Reg.Source, Offset(Reg.Dest)

A MIPS instruction is 32 bits (always). A MIPS memory address is 32 bits (always).

An instruction that refers to memory uses a base register and an offset. The base register is a general-purpose register that contains a 32-bit address. The offset is a 16-bit signed integer contained in the instruction. The sum of the address in the base register with the (sign-extended) offset forms the memory address.

Ex.: Write the instruction that loads the word at address 0x00400060 into register $8. Assume that register $10 contains 0x00400000. -> **lw $8, 0x60($10)**

Examples:

1 R-type

`add $t0,$s1,$s2` donde $t0 = 8, $s1 = 17, $s2 = 18

0	17	18	8	0	32
000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
op	rs	rt	rd	sh	func

2 BEQ

`beq $11, $0, 3` means jump to the third instruction ahead from `beq` if `$11 == $0`. For instance:

```
beq $11, $0, 3
instruction 1
instruction 2
instruction 3 < the target
```

the number `3` will be first sign extended and then added to the program counter `$pc` as:

```
$pc = $pc + 3 * 4
```

or simply:

```
$pc = $pc + 3 << 2
```

3 Shift left logical (<<)

$0000\ 0000\ 0000\ 0000\ 000\ 0000\ 0000\ 0000\ 1001_{dos} = 9_{diez}$

```
sll  $t2,$s0,4   # reg $t2 = reg $s0 << 4 bits
```

$0000\ \ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{dos} = 144_{diez}$

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

El desplazamiento a la izquierda de i bits nos da el mismo resultado que multiplicar por 2^i .

4 Sign Extend

The immediate operand of this instruction is 16 bits (as are all MIPS immediate operands). However, when extended to a 32-bit operand by the ALU it is sign extended: The value of the left-most bit of the immediate operand (bit 15) is copied to all bits to the left (into the high-order bits).

5 Jump

000010	dirección
31-26	25-0

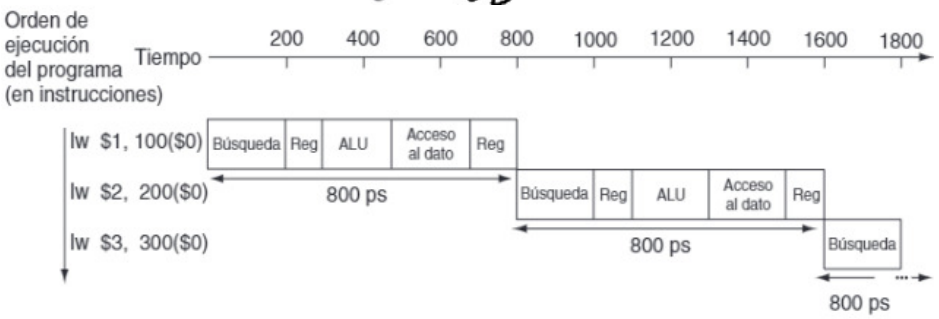
La dirección se hace Shift Left 2 y al nuevo PC se le asigna PC+4[31-28]JAdress[27-0].

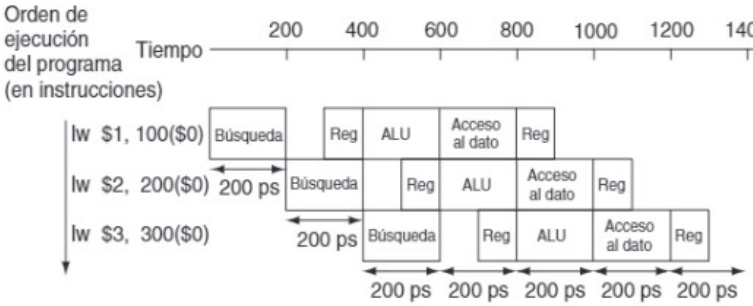
Señales de control:

Instrucción	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
Formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

PIPELINED PROCESSOR

1. Buscar una instrucción en memoria. IF
2. Leer los registros mientras se descodifica la instrucción. El formato de las instrucciones MIPS permite que la lectura y descodificación ocurran de forma simultánea. ID
3. Ejecutar una operación o calcular una dirección de memoria. EX
4. Acceder a un operando en la memoria de datos. MEM
5. Escribir el resultado en un registro. WB





Riesgos del pipeline (Pipeline Hazards)

1 Riesgos estructurales (Structural hazard)

El hardware no admite una cierta combinación de instrucciones durante el mismo ciclo. Si el pipeline de la figura anterior tuviera una cuarta instrucción, se vería que en un mismo ciclo la primera instrucción está accediendo a datos de memoria mientras que la cuarta está buscando una instrucción de la misma memoria. Sin disponer de dos bancos de memoria, nuestra segmentación podría tener riesgos estructurales.

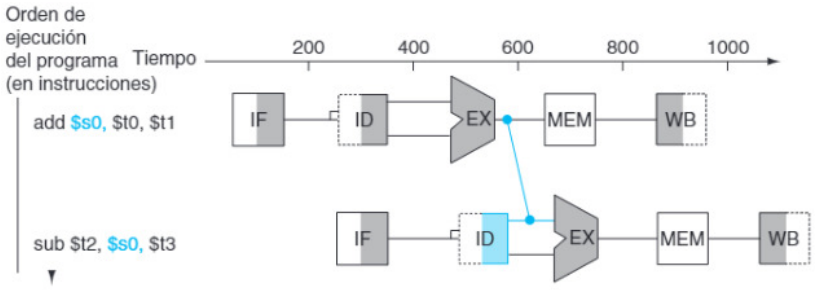
2 Riesgos de datos (Data hazard)

Los riesgos de datos ocurren cuando el pipeline se debe bloquear debido a que un paso de ejecución debe esperar a que otro paso sea completado. Ej.:

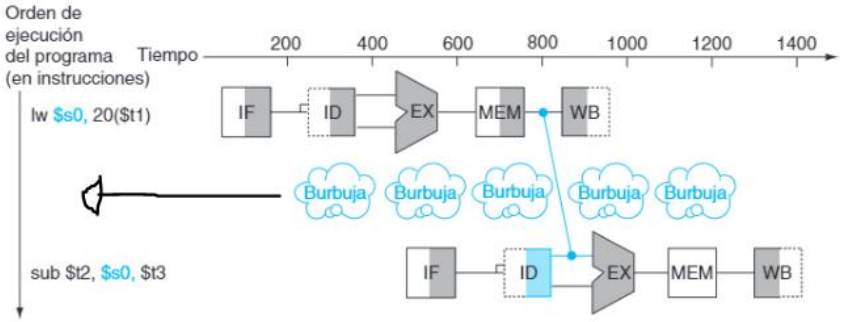
```
add     $s0, $t0, $t1
sub     $t2, $s0, $t3
```

Solución:

Forwarding: La principal solución se basa en la observación de que no es necesario esperar a que la instrucción se complete antes de intentar resolver el riesgo de datos. Para la secuencia de código anterior, tan pronto como la ALU calcula la suma para la instrucción add, se puede suministrar el resultado como entrada de la resta.



En el forwarding los datos sólo pueden ir hacia delante temporalmente hablando. Si no se puede, se tiene que hacer un pipeline stall (burbuja) para darle tiempo a la siguiente instrucción:



3 Riesgos de control (Control hazard)

El tercer tipo de riesgo se llama riesgo de control y surge de la necesidad de tomar una decisión basada en los resultados de una instrucción mientras las otras se están ejecutando. Ocurre con los beq.

Solución:

Bloquear inmediatamente después de haber ido a buscar un salto, y esperar a que el pipeline determine el resultado del salto y conozca cuál es la instrucción que se debería buscar.

Solución 2:

Predicción de saltos (branch prediction). Se puede predecir de varias formas como que siempre se predice que no va a hacer el salto, entonces sólo sería eficiente cuando realmente no lo hace. Otro modo de predicción sería predecir que unos sí y otros no, según el tipo de estructura que sea como por ejemplo en un loop se predeciría que sí va a haber salto. Otro modo es la predicción dinámica basada en el historial de saltos anteiores (90% de precisión).

¿Te está sirviendo de ayuda este resumen?

Este resumen es una demostración del documento completo, que contiene **25** páginas en su totalidad, si quieres verlas todas escribe a juanlu.sc56@gmail.com, respondo en poco tiempo 😊

Mi nombre es Juanlu, soy un estudiante que ya ha superado todos los créditos de las asignaturas del grado de Ingeniería Informática en la EPS UAM.

Ofrezco apuntes, clases y apoyo en prácticas (tengo todas las prácticas hechas), contacta conmigo en juanlu.sc56@gmail.com.