CLASE 8
MEDICIÓN DE PERFORMANCE

Introducción

¿Cómo comparar la performance de dos sistemas diferentes en su arquitectura (ISA, Organización y Tecnología)? ¿Cómo analizar el impacto de una mejora?
¿Por qué existen tantos tipos de computadoras?

Comparación con la industria automotriz

- *velocidad máxima (km/h), aceleración (0-100 km/h), derrape (g)*
- *consumo (l/100 km)*
- *precio ($), costo de mantenimiento ($)*

El mercado de computadoras

- *rendimiento, latencia y productividad*
- *tamaño, potencia, autonomía*
- *costo, mantenimiento*

Comparar diseños que tengan el mismo objetivo

Top500
vs.
Green500

Clasificación según la aplicación

Diseño de alto rendimiento

SERVERS [cómputo masivo, gráficos]

Availability, reliability, scalability.

Productividad (throughput).

Diseño de bajo costo

EMBEDDED SYSTEMS [consolas, switches]

Minimización de tamaño, memoria y potencia.

Latencia, tiempo de respuesta a las interrupciones.

Diseño costo/rendimiento

DESKTOP [deben incluir sw!]

LAPTOP (notebooks, netbooks)

HANDHELD (smartphones)

Marketing vs. rendimiento, información incompleta o vaga, medidas inapropiadas, recurrir a la popularidad.

Procesadores AMD EPYC 64C 2 GHz (8,730,112 total cores)

1.102 Exaflop/s (10^{18})

https://top500.org/

Móvil: ARM

- Celulares y reproductores multimedia
- Consolas de mano
- Tablets y PDA

Desktop: Intel/AMD

- PC, laptop
- Consolas de juego de 8ª y 9ª generación

High Performance: +IBM

- Servidores para cálculo masivo
- Consolas de juego de 7ª generación

Expresiones más utilizadas

Si T es el **tiempo de ejecución** de un determinado programa y η el **rendimiento** de la computadora que lo ejecuta, decimos que la computadora A es ***más rápida*** que la computadora B si:

$$T_A < T_B \quad y \quad \eta_A > \eta_B$$

Por lo tanto el rendimiento es inversamente proporcional al tiempo de ejecución del programa involucrado. Puede decirse que:

$$T \alpha \frac{1}{\eta} \Rightarrow \frac{T_B}{T_A} = \frac{\eta_A}{\eta_B}$$

RENDIMIENTO → TPS → 1/Tcpu

Expresiones más utilizadas

Entonces, si **A** es más rápida que **B**, al utilizarla puedo hablar de una **mejora** (**speedup**) S tal que:

$$S = \frac{\eta_A}{\eta_B} = \frac{T_B}{T_A} \qquad S = 1 + \frac{m}{100}$$

A es un m% más rápida que B

Igualando las dos expresiones:

$$S = \frac{T_B}{T_A} = 1 + \frac{m}{100} \qquad m = \frac{T_B - T_A}{T_A} 100 \qquad m = \frac{\eta_A - \eta_B}{\eta_B} 100$$

IMPORTANTE
No confundir los términos **speedup** y **aceleración** (mala traduccción)

Ejemplo 1

Una computadora ejecuta un cierto programa en 10 segundos. Una segunda computadora tarda 15 segundos en ejecutar el mismo programa. ¿Cuánto más rápida es la primera que la segunda? *(en %)*

Ejemplo 1

Una computadora ejecuta un cierto programa en 10 segundos. Una segunda computadora tarda 15 segundos en ejecutar el mismo programa. ¿Cuánto más rápida es la primera que la segunda?

La primera es un 50% (no 33%!!) más rápida que la segunda, o sea que al ser utilizada se obtiene una mejora de 1.5 respecto de la segunda.

$$m = \frac{T_B - T_A}{T_A} 100 \qquad m = \left(\frac{15-10}{10}\right)100 = 50 \qquad S = \frac{15}{10} = 1.5$$

Ejemplo 1

Una computadora ejecuta un cierto programa en 10 segundos. Una segunda computadora tarda 15 segundos en ejecutar el mismo programa. ¿Cuánto más rápida es la primera que la segunda?

La primera es un 50% (no 33%!!) más rápida que la segunda, o sea que al ser utilizada se obtiene una mejora de 1.5 respecto de la segunda.

$$m = \frac{T_B - T_A}{T_A} 100 \qquad m = \left(\frac{15-10}{10}\right)100 = 50 \qquad S = \frac{15}{10} = 1.5$$

Otra forma de verlo: La primera computadora puede ejecutar el programa 6 veces por minuto. La segunda 4 veces por minuto:

$$S = \eta_A / \eta_B = 6 / 4 = 1.5$$

Si cambio la segunda por la primera puedo correr un 50% más de veces el programa.

Expresiones más utilizadas

Ejemplo:
A performance exceeds **B** by **3x**

En la bibliografía más reciente se evitan también las expresiones del tipo *"tanto % más rápido"*. Se intenta utilizar solamente *"n veces" (n times faster)*, o ***n✖***, que coincide con la previa definición de mejora. La enunciación en **veces** es menos confusa:

$$n = \frac{T_B}{T_A}$$

Sin embargo, el término **speedup** sigue siendo muy utilizado, y la expresión *"es n veces **más** rápido que"*... puede confundir.

IMPORTANTE
No utilizar "es tanto más lenta..." y tener cuidado con
"incrementar", "decrementar", "desmejorar"
y otros términos inciertos.

¿Cómo expresar si se mejora sólo una parte?

Ley de Amdahl

Siendo *T* el tiempo de ejecución de una tarea y *f* la fracción de ese tiempo que puede mejorarse en un factor *k*, entonces la mejora total obtenida será:

$$S = \frac{1}{(1-f)+\frac{f}{k}}$$

Demostración:

$$T = (1-f)T + fT$$

$$T_{MEJORADO} = (1-f)T + \frac{f}{k}T$$

$$S = \frac{T}{T_{MEJORADO}}$$

Corolario:

$$S_{MAX} = \frac{1}{(1-f)}$$

Máxima mejora que puede obtenerse si sólo una fracción *f* puede ser mejorada.

Ejemplo 2

Considere un determinado caso de aplicación en que la CPU se utiliza el 50% del tiempo (el resto del tiempo se consume en accesos a memoria y al sistema se entrada/salida).
Suponga que el costo de la CPU representa 1/3 del costo total de la computadora.

a) ¿Mejora la relación Costo/Rendimiento si reemplazo la CPU por otra 5 veces más rápida que cuesta 5 veces más?

b) ¿Hasta cuánto pagaría por la CPU si desea mantener la relación?

c) Si pudiera comprar una CPU infinitamente rápida, ¿de cuánto sería la mejora?

Ejemplo 2

a) La relación costo/rendimiento empeora:

$$S = \frac{1}{(1-0.5)+\frac{0.5}{5}} = 1.66$$

66% más rápido

133% más caro

$$Costo_{NUEVO} = \left(\frac{2}{3}\times 1 + \frac{1}{3}\times 5\right) Costo_{ORIGINAL} = 2.33\, Costo_{ORIGINAL}$$

$$\left(\frac{Costo}{Rendimiento}\right)_{NUEVO} = \frac{2.33}{1.66} \times \left(\frac{Costo}{Rendimiento}\right)_{ORIGINAL}$$

Desmejora

Pero puede ser que ahora sea útil y antes no!

b) 3

c) 2 *(100%)*

Ejemplo 3

Se desea mejorar el rendimiento de un procesador con una nueva ALU que realice las operaciones aritméticas en la mitad de tiempo.
Calcular la mejora en la ejecución de un determinado programa si aproximadamente el 60% del tiempo de ejecución se compone de este tipo de operaciones.
Si el programa tarda 12 segundos en ejecutarse sin la mejora, ¿cuánto tardará con la mejora?

Ejemplo 3

Se desea mejorar el rendimiento de un procesador con una nueva ALU que realice las operaciones aritméticas en la mitad de tiempo.
Calcular la mejora en la ejecución de un determinado programa si aproximadamente el 60% del tiempo de ejecución se compone de este tipo de operaciones.
Si el programa tarda 12 segundos en ejecutarse sin la mejora, ¿cuánto tardará con la mejora?

$$S = \frac{1}{(1-f)+\frac{f}{k}}$$

Dos formas de pensarlo:

a) S = 1/(0.4+0.6/2) = 1.42
b) Tn = 12/1.4286 = 8.4 s

*b) Tn = 12*0.6/2+12*0.4 = 8.4 s*
a) S = 12/8.4 = 1.42

Mejora 42% == 1.42x

Ejemplo 4

Considere dos procesadores con diferentes estrategias para el manejo de saltos condicionales:
Procesador A: En una única instrucción que toma 2 ciclos realiza la verificación de la condición y el salto. El resto de las instrucciones toman 1 ciclo.
Procesador B (mejorado?): La frecuencia de operación es 25% más elevada. Pero utiliza dos instrucciones para realizar el salto. La primera es una comparación que prepara los flags de condición; esta instrucción toma 1 ciclo. A continuación los flags deben ser examinados por una segunda instrucción de 2 ciclos que produce el salto.
Hasta qué proporción de saltos condicionales en el total del código representa el procesador B una verdadera mejora respecto del procesador A.

Ejemplo 4

Considere dos procesadores con diferentes estrategias para el manejo de saltos condicionales:
Procesador A: En una única instrucción que toma 2 ciclos realiza la verificación de la condición y el salto. El resto de las instrucciones toman 1 ciclo.
Procesador B (mejorado?): La frecuencia de operación es 25% más elevada. Pero utiliza dos instrucciones para realizar el salto. La primera es una comparación que prepara los flags de condición; esta instrucción toma 1 ciclo. A continuación los flags deben ser examinados por una segunda instrucción de 2 ciclos que produce el salto.
Hasta qué proporción de saltos condicionales en el total del código representa el procesador B una verdadera mejora respecto del procesador A.

Si hay 0% de saltos condicionales la mejora es 1.25x
Si hay 100% de saltos condicionales $ta=2*N/f$ *y* $tb=(N*2+N)/(1.25f)=2.4*N/f$
(B es peor que A) la "mejora" es 2/2.4 = 0.83x

Importante:

$Nb > Na$

Ejemplo 4

Considere dos procesadores con diferentes estrategias para el manejo de saltos condicionales:
Procesador A: En una única instrucción que toma 2 ciclos realiza la verificación de la condición y el salto. El resto de las instrucciones toman 1 ciclo.
Procesador B (mejorado?): La frecuencia de operación es 25% más elevada. Pero utiliza dos instrucciones para realizar el salto. La primera es una comparación que prepara los flags de condición; esta instrucción toma 1 ciclo. A continuación los flags deben ser examinados por una segunda instrucción de 2 ciclos que produce el salto.
Hasta qué proporción de saltos condicionales en el total del código representa el procesador B una verdadera mejora respecto del procesador A.

Si hay 0% de saltos condicionales la mejora es 1.25x
*Si hay 100% de saltos condicionales ta=2*N/f y tb=(N*2+N)/(1.25f)=2.4*N/f*
(B es peor que A) la "mejora" es 2/2.4 = 0.83x

Importante:

$Nb > Na$

Si ***P*** *es la porción de saltos condicionales respecto del total [0-1]*
*CPIa=P*2+(1-P)*1*
*CPIb=P*2+P*1+(1-P)*1*
S=1 cuando CPIb/CPIa=1.25 -> P=1/3

Ejemplo 4

Considere dos procesadores con diferentes estrategias para el manejo de saltos condicionales:
Procesador A: En una única instrucción que toma 2 ciclos realiza la verificación de la condición y el salto. El resto de las instrucciones toman 1 ciclo.
Procesador B (mejorado?): La frecuencia de operación es 25% más elevada. Pero utiliza dos instrucciones para realizar el salto. La primera es una comparación que prepara los flags de condición; esta instrucción toma 1 ciclo. A continuación los flags deben ser examinados por una segunda instrucción de 2 ciclos que produce el salto.
Hasta qué proporción de saltos condicionales en el total del código representa el procesador B una verdadera mejora respecto del procesador A.

Si hay 0% de saltos condicionales la mejora es 1.25x
Si hay 100% de saltos condicionales $ta=2*N/f$ *y* $tb=(N*2+N)/(1.25f)=2.4*N/f$
(B es peor que A) la "mejora" es $2/2.4 =$ 0.83x

Si ***P*** *es la porción de saltos condicionales respecto del total [0-1]*
$CPIa=P*2+(1-P)*1$
$CPIb=P*2+P*1+(1-P)*1$
$S=1$ *cuando* $CPIb/CPIa=1.25 \rightarrow P=1/3$

Ver AVR: es del tipo B pero toma 2 ciclos si salta y 1 ciclo si no salta. Ps proporción nueva.
¿Cómo se calcula en ese caso?

$CPIb=P*\mathbf{2}+P*1+(1-P)*1$

$(Ps*2+(1-Ps)*1)$

El tiempo de ejecución puede calcularse

Cuando las cuentas se complican puede utilizarse un **simulador**.
Pero en ambos casos es necesario disponer previamente de la solución, o al menos de un "borrador" del núcleo del algoritmo.

Como vimos, el tiempo de ejecución de un programa puede calcularse como:

$$T_{CPU} = \frac{CICLOS\ \ DE\ \ CLOCK\ \ DEL\ \ PROGRAMA}{f_{CLOCK}} = \frac{N\ .CPI}{f_{CLOCK}}$$

Siendo N el número (dinámico) de instrucciones, CPI el número de ciclos de reloj promedio por instrucción y f la frecuencia del reloj.

CPI es una medida estadística!

$$CPI = \sum CPI_i \frac{N_i}{N}$$

CUIDADO: si i es la familia de instrucciones, CPI_i es fácil de calcular. Pero, ¿qué es N_i?

Instrucción	Frecuencia	CPIi
ALU	43.00%	1
LOAD	21.00%	2
STORE	12.00%	2
JUMP	24.00%	2

$\Rightarrow$ $CPI = 1.57$

El tiempo de ejecución puede medirse

El rendimiento es inversamente proporcional al tiempo de ejecución del programa involucrado. ¿Cómo mido ese tiempo? ¿Puede calcularse? ¿Puede simularse?

En una PC con linux

```
$ time test
8.1u  7.3s  25.2  61%

$ time test
real  0m2.739s      <- tiempo transcurrido (wall)
user  0m2.304s      <- tiempo de ejecución
sys   0m0.016s      <- llamadas al sistema operativo
```

El tiempo user (+sys?) es mi T_{CPU}. El resto del tiempo esperando I/O o ejecutando otros programas en el multitarea.

No sólo es necesario disponer de la implementación, sino también del sistema completo funcionando.

Puede no ser un problema si se trata, por ejemplo, de una aplicación Linux/Windows, pero en un sistema dedicado...

¿Cómo puede medirse en un microcontrolador?

¿Cómo seleccionar la CPU antes de tener la solución?

La solución óptima depende de la CPU utilizada.
Hasta que no tenga la solución no puedo medirla o calcularla.

Probar todas las combinaciones no parece viable...
Tal vez sea suficiente disponer de una buena caracterización del problema.

Benchmarks

¿Es posible establecer una métrica única de performance?

Medición del rendimiento de un sistema (o uno de sus componentes) mediante la ejecución de un programa (o un conjunto de programas) con el objetivo de poder comparar los resultados con máquinas similares.

Bibliografía recomendada

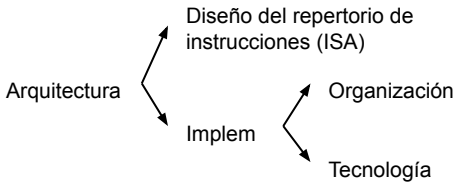
"Measuring Computer Performance: A practitioner's guide", David J. Lilja

"The essentials of computer organization and architecture",
Linda Null. Capítulo 10: Performance Measurement and Analysis

https://en.wikipedia.org/wiki/Benchmark_(computing)
(la entrada en español está incompleta)

La tarea de ejecutar una prueba originalmente se reducía a estimar el tiempo de proceso que lleva la ejecución de un programa (medida por lo general en miles o millones de operaciones por segundo). Con el correr del tiempo, la mejora en los compiladores y la gran variedad de arquitecturas y situaciones existentes convirtieron a esta técnica en toda una especialidad. La elección de las condiciones bajo la cual dos sistemas distintos pueden compararse entre sí es especialmente ardua. La publicación de los resultados suele ser objeto de largos debates cuando éstos se abren a la comunidad.

Benchmarks



$$t = \frac{N \times CPI}{f_{clock}}$$

Algunas opciones simples pero no muy útiles:

Parámetros (ej. frecuencia de clock)
Permiten comparar dentro de una familia, no entre diferentes organizaciones.

MIPS
Depende de la implementación del repertorio de instrucciones: CPI promedio.
Podría resultar inversamente proporcional al rendimiento (ej. operaciones FP).

$$MIPS = \frac{N}{T_{CPU}\, 10^6} = \frac{f_{CLOCK}}{CPI\, 10^6}$$

$$MFLOPS = \frac{N_{FP}}{T_{CPU}\, 10^6}$$

MFLOPS
Precaución con los diferentes repertorios de instrucciones (ej. 68882 de Motorola tenía seno y raíz cuadrada en punto flotante).
Precaución con los cálculos parciales (ej. no es lo mismo una suma completa que la normalización de una suma).

Programas reales (ej. Spice, CAD, gráficos)
Deben estar disponibles o deben instalarse. Versiones.

floating-point: Whetstone
string processing: Dhrystone
integer multiplication: Coremark

Benchmarks sintéticos

Programas simples, escritos originalmente en ALGOL 60 o FORTRAN y traducidos a C. Utilizados para comprobar el comportamiento ante un tipo específico de carga.

Benchmarketing

Los más tradicionales (todos tienen entradas en Wikipedia):

- **Whetstone** (1972) FP [MWIPS]
- **Linpack** (1979) FP álgebra lineal [FLOPS]
- **Dhrystone** (1984) ENTEROS [iteraciones/seg]

Los más nuevos:

- **CoreMark** desarrollado por el Embedded Microprocessor Benchmark Consortium (EEMBC). Soluciona algunos problemas del anterior (algoritmos reales en vez de ser completamente sintético). Reglas estrictas para correrlo.
  - list processing (find and sort)
  - matrix manipulation (common matrix operations)
  - state machine (determine if an input stream contains valid numbers)
  - CRC.

NOTACION: CoreMark 1.0 : N / C / P / M
N Number of iterations per second with seeds 0,0,0x66,size=2000)
C Compiler version and flags
P Parameters such as data and code allocation specifics
M Type of Parallel algorithm execution (if used) and number of contexts
Ejemplo: CoreMark 1.0 : 128 / GCC 4.1.2 -O2 -fprofile-use / Heap in TCRAM / FORK:2

En arquitecturas más sofisticadas debe observarse el efecto del caché: al ser programas pequeños no miden la performance de la memoria.
El resultado es sensible a las técnicas de compilación y optimizaciones.
El “balance” de carga puede no ser representativo.

Benchmarks sintéticos avanzados

Consorcios de fabricantes de hardware y software

EEMBC (1997) “Embedded Microprocessor Benchmark Consortium”
(www.eembc.org) Orientado a sistemas embebidos.

SPEC (1988) “Standard Performance Evaluation Corporation”
(www.spec.org) Orientado a la CPU de propósitos generales, tiempo, latencia.

TPC (1990) “Transaction Processing Performance Council”
(www.tpc.org) Orientado a transacciones, throughput.

EMBC EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM

1988 – 2018 30 YEARS spec®

TPC®

DMIPS

- **DMIPS** = Dhrystone (veces/s) / **1757** (también llamado **Vax-MIPS**), mide **performance relativa**
- **DMIPS/MHz** refleja la **organización** de la máquina (también llamado **Vax-MIPS/MHz**)
- **DMIPS/Watt** = DMIPS/MHz x MHz/Watt, recordar que $P=CV^2f$, mide **eficiencia**

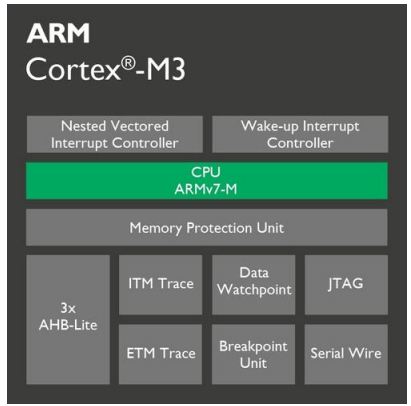
VAX 11/780
La primera computadora que realizó 1 MIPS
Performance Dhrystone: 1757 veces por segundo

arm Developer

Para comparar, ARM elige medir la performance de sus microcontroladores (familia Cortex-M) en **DMIPS/MHz** y **CoreMark/MHz**. Su sitio web contiene una generosa cantidad de información (comparado con otros fabricantes):
https://developer.arm.com/ip-products/processors/cortex-m

Cortex-M Comparison Table

Feature	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M23	Cortex-M3	Cortex-M4	Cortex-M33	Cortex-M35P	Cortex-M7
Instruction Set Architecture	Armv6-M	Armv6-M	Armv6-M	Armv8-M Baseline	Armv7-M	Armv7-M	Armv8-M Mainline	Armv8-M Mainline	Armv7-M
	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2	Thumb, Thumb-2
DMIPS/MHz range*	0.87-1.27	0.95-1.36	0.8	0.98	1.25-1.89	1.25-1.95	1.5	1.5	2.14-3.23
CoreMark®/MHz*	2.33	2.46	1.85	2.64	3.34	3.42	4.02	4.02	5.01
Pipeline Stages	3	2	3	2	3	3	3	3	6
Memory Protection Unit (MPU)	No	Yes (option)	No	Yes (option) (2 x)	Yes (option)	Yes (option)	Yes (option) (2 x)	Yes (option) (2 x)	Yes (option)
Maximum MPU Regions	0	8	0	16	8	8	16		
Trace (ETM or MTB)	No	MTB (option)	No	MTB (option) or ETMv3 (option)	ETMv3 (option)	ETMv3 (option)	MTB (option) and/or ETMv4 (option)		
Digital Signal Processing (DSP)	No	No	No	No	No	Yes	Yes (option)		
Floating Point Hardware	No	No	No	No	No	Yes (option SP)	Yes (option SP)		



Otras opciones para medir performance:
Forward Discrete Cosine Transform / FFT

	Cortex-A15 vs Cortex-A7 Performance	Cortex-A7 vs Cortex-A15 Energy Efficiency
Dhrystone	1.9x	3.5x
FDCT	2.3x	3.8x
IMDCT	3.0x	3.0x
MemCopy L1	1.9x	2.3x
MemCopy L2	1.9x	3.4x

Ejemplo de aplicación de Dhrystone

El código del benchmark **Dhrystone** está disponible desde 1984. Mide la performance en operaciones con enteros (no PF), en veces (iteraciones) por segundo. Se compila con gcc y se ejecuta con algunas precauciones (ver comentarios en el código). No debe alterarse el código. No está permitido utilizar *inlining* de funciones ni otras "trampas".

Ejemplo: Dhrystone v2.1 ejecutado en un kit de desarrollo que contiene un procesador **Cortex-M3** operando a una frecuencia de reloj de 120 MHz (M3 es un procesador con ISA ARMv7). ARM tiene una nota de aplicación sobre cómo correr este benchmark (Application Note 273).

```
$ ./dhry
Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
Please give the number of runs through the benchmark: 1000000
Execution starts, 1000000 runs through Dhrystone
...
...
Dhrystones per Second: 263360.9
$
```

Para Linux, está incluído en **UnixBench** (con reg)

Recordar que 1 DMIPS = 1757 Dhrystones per second (performance de la VAX-11/780)

Interpretación del resultado

Mejora relativa en la performance

La VAX-11 ejecutaba 1757 iteraciones del Dhrystone por segundo. Cortex-M3 ejecuta 263360. Entonces:

DMIPS = Dhrystones por segundo / 1757 = 263360 / 1757 = **150**

DMIPS es la **mejora** respecto de la VAX-11, y ya que la VAX-11 tenía 1 DMIPS (supuestamente fue la primera con 1 MIPS), el Cortex-M3 tiene un "equivalente" a 150 millones de instrucciones por segundo (qué instrucciones? las de la VAX corriendo Dhrystone).

Mejora en la tecnología

En gran parte, este avance se debe a mejoras **tecnológicas** (velocidad de reloj). La VAX-11 corría a **5** MHz y el Cortex a **120** MHz. Por lo tanto hay una mejora en el reloj de **120/5** = **24**. Pero evidentemente eso no es todo...

Mejora en la organización y el repertorio de instrucciones

Para independizar de la tecnología se expresa el resultado relativo al reloj:

Cortex-M3: DMIPS / MHz = 150 / 120 MHz = **1.25**

VAX-11: DMIPS / MHz = 1 / 5 MHz = **0.2** (notar que la VAX-11 no tenía 1 DMIPS/MHz)

Existe una **mejora** de **1.25/0.2** = **6.25** que se debe a varios factores:

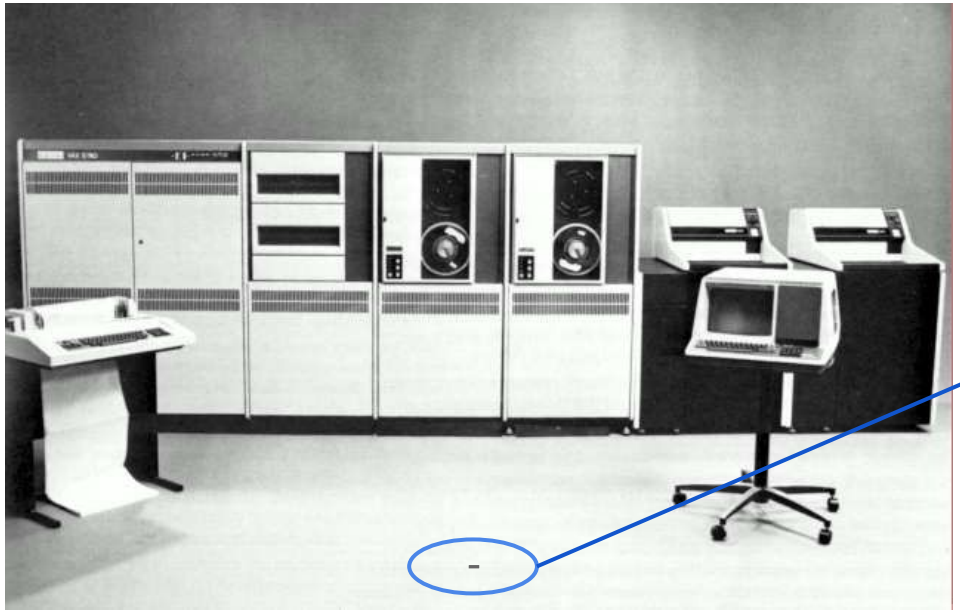
a) **Organización:** segmentación del cauce de instrucciones.
b) **ISA:** Ejecución condicional, barrel shifter.
c) **Mejora en los compiladores**
(Dhrystone es sensible a este punto;
si se tratara de un buen benchmark, no lo sería).

Las mejoras se multiplican: S = 24 x 6.25 = 150

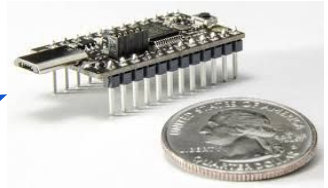
Otra aplicación de los DMIPS/MHz: criticar la arquitectura x86...
Un Pentium 4 @ 3.6GHz (2004) presenta aproximadamente 4200 DMIPS. Dividiendo por la frecuencia de reloj, resultan 1.16 DMIPS/MHz. Entonces la **organización** del Cortex M3 (2004) es "mejor", porque tiene 1.25. Pero OJO la performance es 4200 vs. 150... diferentes tecnologías de implementación.

Un MCU de hoy tiene 150 veces la performance de una moderna computadora de la década del 70

¡¡MUCHO CUIDADO CON LA BIBLIOGRAFÍA!!



150×



Para pensar:
La VAX fue central durante la carrera espacial y armamentista de los 80.
Un MCU se utiliza hoy para ajustar la posición del asiento de un auto de alta gama.

AVR DMIPS/MHz

Las implementaciones existentes de AVR no tienen suficiente memoria RAM para correr Dhrystone

http://www.ecrostech.com/Other/Resources/Dhrystone.htm

AVR (ATmega128) - Dhrystone adaptado para 2K RAM
8487 Dhrystone/sec @14.75 MHz
576 Dhrystone/MHz
4.83 DMIPS
0.33 DMIPS/MHz

http://fpgalibre.sourceforge.net/SASE2011/poster_avr.pdf

AVR (fpga + RAM) - Dhrystone versión 2.1V
10436 Dhrystone/sec @16 MHz
652 Dhrystone/MHz
5.94 DMIPS
0.37 DMIPS/MHz

NOTA 1: la VAX-11 **NO** tenía 1 DMIPS/MHz.
Llegaba a 1 MIPS pero operaba a 5 MHz, o sea 0.2 DMIPS/MHz.
AVR tiene 0.33 DMIPS/MHz.

NOTA 2: la **I** de MIPS son "instrucciones del tipo Dhrystone", son las mismas para todos los procesadores. Pero este procesador es de 8 bits!
La VAX-11 tenía ALU y registros de 32 bits.

NOTA 3: la VAX-11 costaba en 1977 aprox $20,000, lo que actualizado por inflación sería aprox $100,000.
Hoy un ATmega328 cuesta menos de $2.