

UT 1. Introducció a l'Arquitectura dels Computadors

Tema 1.1 Concepte d'Arquitectura de Computadors

J. Flích, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departament d'Informàtica de Sistemes i Computadors
Universitat Politècnica de València

DOCÈNCIA VIRTUAL

Finalitat:
Prestació del servei públic d'educació superior
(art. 1 LOU)

Responsable:
Universitat Politècnica de València.

Drets d'accés, rectificació, supressió, portabilitat, limitació o oposició al tractament conforme a polítiques de privacitat:
<http://www.upv.es/contenidos/DDP/>

Propietat intel·lectual:
Us exclusiu en l'entorn d'aula virtual.
Queda prohibida la difusió, distribució o divulgació de la informació de les classes i particularment la seva compartició en xanxes socials o serveis dedicats a compartir apunts.

La infracció d'aquesta prohibició pot generar responsabilitat disciplinària, administrativa o civil

UNIVERSITAT POLITÈCNICA DE VALÈNCIA



UNIVERSIDAD
POLITECNICA
DE VALENCIA

DISCA

Índex

- 1 Concepte d'Arquitectura**
- 2 Els requeriments d'un computador**
- 3 Tecnologia, consum i cost**
- 4 Evolució del rendiment dels processadors**
- 5 Classes de computadors**
- 6 Màster en Enginyeria de Computadors i Xarxes**

Bibliografia

-  John L. Hennessy and David A. Patterson.
Computer Architecture, Sixth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6
edition, 2018.

Índex

- 1 Concepte d'Arquitectura**
- 2 Els requeriments d'un computador
- 3 Tecnologia, consum i cost
- 4 Evolució del rendiment dels processadors
- 5 Classes de computadors
- 6 Màster en Enginyeria de Computadors i Xarxes

1. Concepte d'Arquitectura

El concepte i els nivells clàssics

Defineix la funcionalitat del hardware del computador a partir d'uns requeriments, fent un compromís entre productivitat, tecnologia disponible i cost.

L'arquitectura de computadors distingeix tres nivells:

Joc d'instruccions (ISA, per *Instruction Set Architecture*).

És tot allò que ha de coneixer el programador en assemblador: la descripció de les instruccions i del seu ús, l'organització lògica de la memòria, etc.

Organització del processador

És la descripció dels elements lògics que duen a terme el cicle d'instrucció: registres, descodificadors, operadors aritmètic-lògics, interfície amb les memòries, etc.

Realització o Tecnologia base

És la realització del dispositiu en forma de transistors, connexions, etc.

Definició moderna

- Durant el període clàssic dels computadors (fins els anys 70), cada nivell era tractat aïlladament per un especialista distint.
- Actualment, l'arquitectura dels computadors és una disciplina vertical, que comprèn tots tres nivells, que dissenya màquines programables perquè executen correcta i eficaçment un conjunt (previsible o no) de programes.
- Tots tres nivells són interdependents: les decisions preses en cadascun poden influir sobre la resta.

El treball de l'Enginyer de Computadors

- partint dels requeriments desitjats
- considerant les limitacions tecnològiques, energètiques i de cost
- fer el millor disseny

→ quantificar prestacions, cost i altres atributs i comparar

Índex

- 1** Concepte d'Arquitectura
- 2** Els requeriments d'un computador
- 3** Tecnologia, consum i cost
- 4** Evolució del rendiment dels processadors
- 5** Classes de computadors
- 6** Màster en Enginyeria de Computadors i Xarxes

2. Els requeriments d'un computador

Per a dissenyar un processador, l'arquitecte ha de considerar

- La classe de computador que cal definir
- El grau de compatibilitat amb altres computadors ja existents

Codi font Disseny més flexible,

Cal desenvolupar nous compiladors

Binari /ISA definit (poca flexibilitat)

no hi cal software nou

- Els requeriments del sistema operatiu

Espai d'adreçament Limita la mida de les aplicacions

Gestió de la memòria Paginació, segmentació, ...

Protecció

Gestión de procesos Suport multitasca

2. Els requeriments d'un computador

(cont.)

■ Estàndards del mercat

Coma flotant

IEEE 754

Bus E/S

SATA, SCSI, PCI ...

Sistema operatiu

Linux, Windows, Mac OSX ...

Xarxes

Supорт a Ethernet, InfiniBand ...

Llenguatges

C, C++, Java, FORTRAN, ...

Índex

- 1 Concepte d'Arquitectura**
- 2 Els requeriments d'un computador**
- 3 Tecnologia, consum i cost**
- 4 Evolució del rendiment dels processadors**
- 5 Classes de computadors**
- 6 Màster en Enginyeria de Computadors i Xarxes**

3. Tecnologia, consum i cost

El treball del dissenyador ha de tindre en compte:

- La tecnologia disponible
- Limitacions en relació a potència i energia
- Cost de fabricació

3. Tecnologia, consum i cost

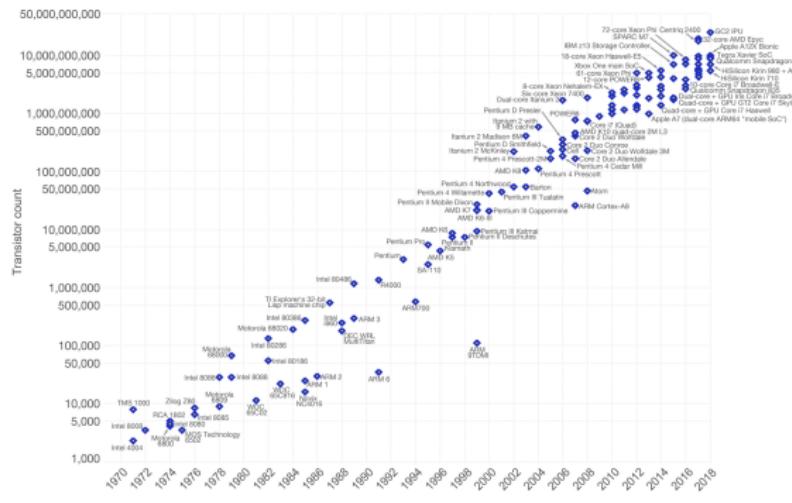
Tecnologia dels circuits integrats: La llei de Moore

El nombre de transistors per xip creix un 40-55% per any (es duplica cada 18 mesos).

Dues causes:

- La densitat de transistors creix un 35% per any
 - El diàmetre de la neula creix un 10–20% per any

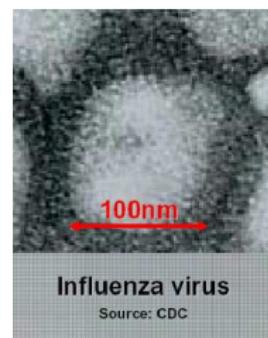
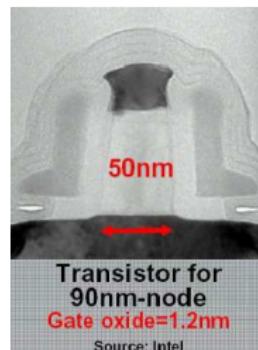
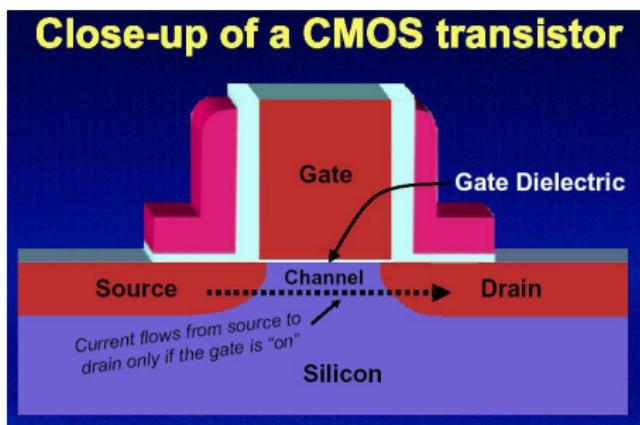
Tendència difícilment sostenible (mireu [DARPA ERISummit 2018](#)).



3. Tecnologia, consum i cost

Tecnologia dels circuits integrats: *Feature size*

Grandària del transistor:

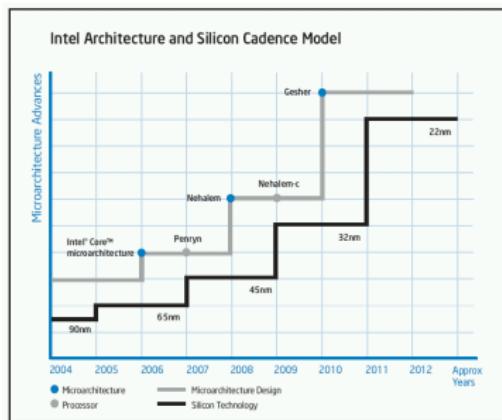
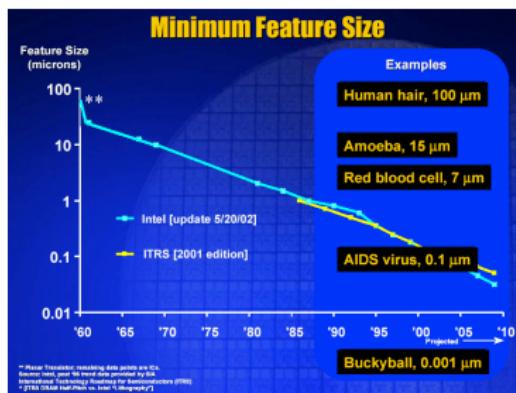


- El nombre de transistors creix quadràticament amb la reducció de les mides del transistor.
- La velocitat dels transistors augmenta linealment en reduir-ne les mides.

3. Tecnologia, consum i cost

Tecnologia dels circuits integrats: *Feature size (cont.)*

Evolució de la grandària del transistor



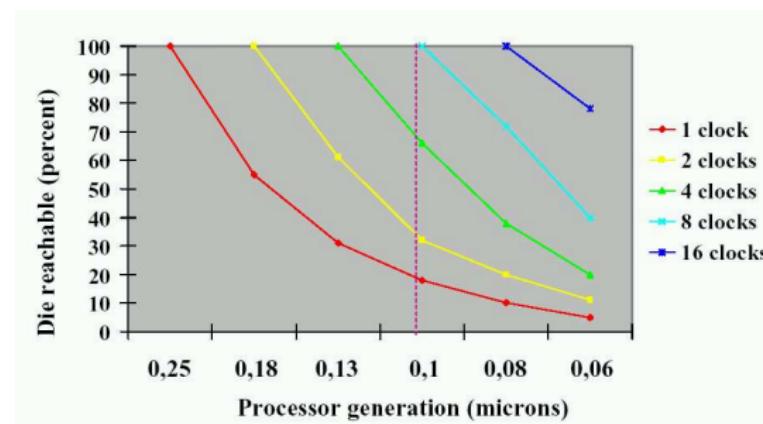
3. Tecnologia, consum i cost

Tecnologia dels circuits integrats: Retards en les interconnexions

El retard d'una connexió depén de la resistència i capacitat del conductor ($R \cdot C$).

- En reduir la mida del transistor es redueix la secció i augmenta R .
- Encara que la capacitat associada a la superfície del conductor baixa, la capacitat d'acoblament entre línies és major i augmenta C .

→ La fracció de xip accessible en un cicle de rellotge es redueix: 100% amb $0.25\mu\text{m}$ a 5% con $0.06\mu\text{m}$.



3. Tecnologia, consum i cost

Consum i calor: Potència dissipada per un transistor

Potència total = Potència dinàmica + Potència estàtica

Potència dinàmica $P_d = \frac{1}{2} \cdot C \cdot V^2 \cdot f$

- P_d és proporcional a la freqüència f
- Cal reduir la tensió d'alimentació V , encara que hi ha un valor mínim per a una freqüència donada.
- En 24 anys, V ha passat de 12 a 1.1 V, i P_d s'ha reduït en un factor de $\frac{12^2}{1.1^2} = 119 \times$
- el marge de reducció restant és baix $\frac{1.1^2}{0.7^2} = 2.5 \times$

Potència estàtica $P_s = I_{fuga} \cdot V$

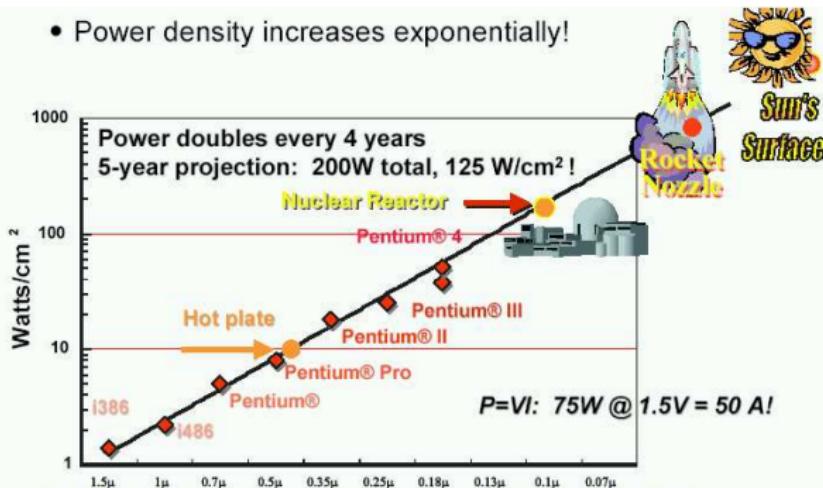
- Com menor és el transistor, $\uparrow I_{fuga}$
- Com major nombre de transistors, major contribució de P_s .

3. Tecnologia, consum i cost

Consum i calor: Potència dissipada per un transistor (cont.)

L'augment del nombre de transistors i de la freqüència predomina sobre la reducció de potència deguda a la tensió i capacitat

→ De 0.01 W en els primers microprocessadors a prop de 130 W en un Itanium2.



3. Tecnologia, consum i cost

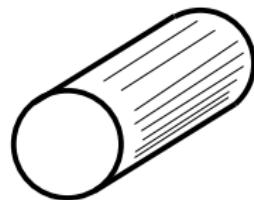
Consum i calor: Potència dissipada per un transistor (cont.)

Implicacions:

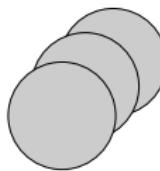
- Distribució del corrent al microprocessador. Els microprocessadors moderns tenen centenars de patetes per a l'alimentació.
- Evacuació del calor generat.
- Desenvolupament de nous materials per reduïr el corrent de fuga. Millora del dielèctric (*high-k* de Intel).

Cost

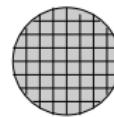
- Cost d'un circuit integrat.



BARRA DE SILICIO



OBLEA



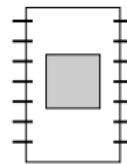
DADOS



DADO
(defectos)



DADO



CIRCUITO INTEGRADO

Cost final $\approx f(\text{superfície dau}^4)$

Superfície dau = f (complexitat del disseny)

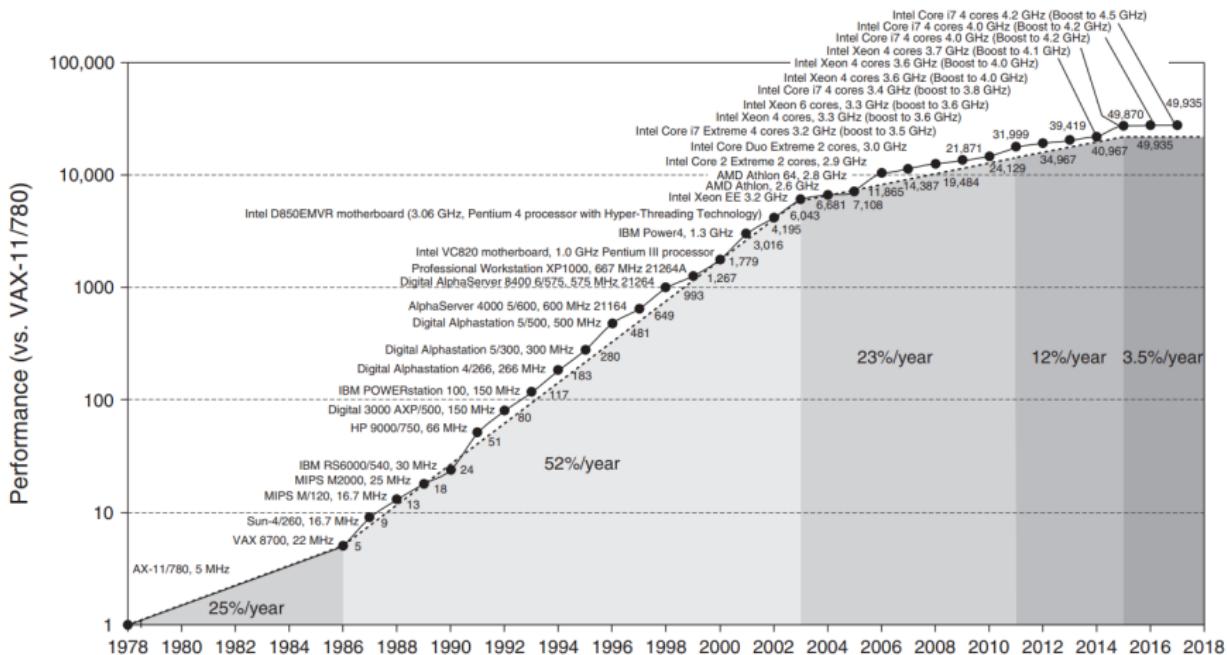
Cost (cont.)

- Factors que redueixen el cost dels components:
 - Corva d'“aprenentatge”: El cost d'un component disminueix amb el temps en augmentar la productivitat (baixa la taxa dels components defectuosos).
 - Volum de vendes: Duplicar el volum de vendes abaixa el cost un 10 %.
- Cost del disseny
 - El cost d'una fàbrica de xips és inversament proporcional al *feature size*.
 - La grandària de l'equip de dissenyadors és inversament proporcional al *feature size*: de 3 persones en l'Intel 4004 a més de 300 en els processadors moderns.

Índex

- 1 Concepte d'Arquitectura
- 2 Els requeriments d'un computador
- 3 Tecnologia, consum i cost
- 4 Evolució del rendiment dels processadors
- 5 Classes de computadors
- 6 Màster en Enginyeria de Computadors i Xarxes

4. Evolució del rendiment dels processadors



4. Evolució del rendiment dels processadors

- Període I: 1978–1986. Creixement anual de prestacions: 25%
 - Millores tecnològiques.
- Període II: 1986–2003. Creixement anual de prestacions: 52%
 - Millores tecnològiques.
 - **Millores arquitecturals:** arquitectures RISC (*Reduced Instruction Set Computers*), instruction-level parallelism (ILP), memòries cau
- Període III: 2003–2012. Creixement anual de prestacions: 23%
 - Millores tecnològiques.
 - S'arriba als límits en ILP, no es redueix la latència de la memòria, elevada dissipació i consum de potència,
→ per millorar cal incrementar el paralelisme
 - **DLP** *Data-Level Parallelism*. Una operació sobre múltiples dades
 - **TLP** *Thread-Level Parallelism*. Varies tasques en paralel
- Període IV: 2011–2015. Les prestacions creixen un 12% anual per limitacions tecnològiques i pel límit del paralelisme (Llei d'Amdahl)

4. Evolució del rendiment dels processadors

- Període V: 2015–2018. Creiximent estancat (3,5% anual)
 - Fí de la Llei de Moore
 - La tecnologia no ofereix millors significatives en eficiència energètica
 - Límits del paral.lelisme
- Situació actual:
 - Millora de prestacions i eficiència energètica mitjançant especialització → incloure acceleradors de tasques específiques (*domain-specific*). Per exemple, Google Tensor Processing Unit (TPU) multiplica per 80 les prestacions / Watt comparat amb una CPU per a tasques d'inferència en xarxes neuronals.
 - Recerca i desenvolupament de noves tecnologies de fabricació:

<https://francis.naukas.com/2019/08/29/rv16x-nano-un-microprocesador-risc-de-16-bits-con-14702-transistores-de-nanotubos-de-carbono/>.

4. Evolució del rendiment dels processadors

Millores arquitectòniques

Alguns exemples:

Arquitectura RISC: Instruccions senzilles que s'executen ràpidament.
Hardware simple.

Segmentació: Descomposició del cicle d'instrucció en fases que progresen concurrentement.

Explotació de l'ILP: Execució d'instruccions en ordre distint al del programa.

Paralelisme:

DLP *Data-Level Parallelism.* La mateixa operació sobre múltiples dades.

TLP *Thread-Level Parallelism.* Vàries tasques en paral·lel.

Moltes d'aquestes millores només són possibles quan el nombre de transistors encabits en el xip és suficient.

4. Evolució del rendiment dels processadors

La tecnologia disponible motiva algunes idees arquitectòniques

Alguns exemples:

Memòries cau La diferència de velocitat entre el processador i la memòria principal va motivar la introducció de la memòria *cau*. El creixement sostingut d'aquesta diferència ha provocat l'aparició de nivells (L1, L2, L3) de *cau* en els dissenys.

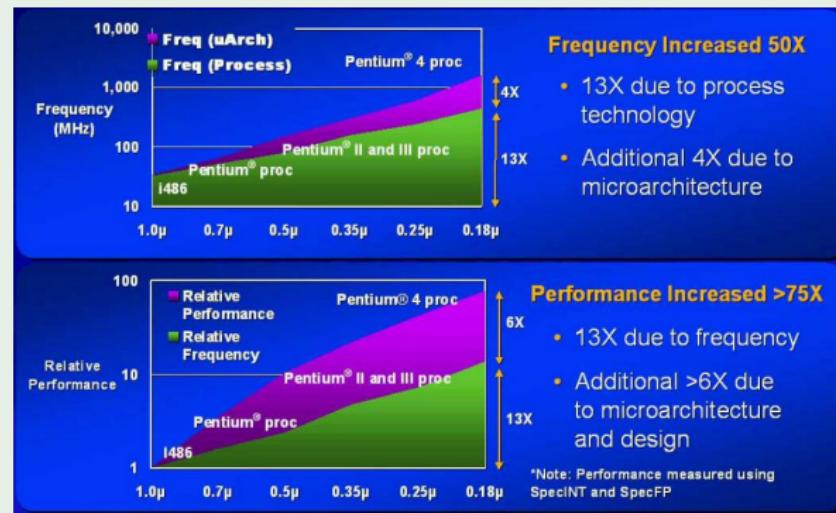
Etapes per a propagar senyals El retard creixent dels cables va motivar la inclusió en el Pentium 4 de dues etapes de segmentació adreçades només a la propagació dels senyals pels cables.

Multiprocessadors en un xip Un processador sofisticat funcionant a molta freqüència consumeix massa i dissipa massa calor. En el seu lloc, s'ubiquen processadors més senzills que funcionen a menor freqüència i menor tensió.

4. Evolució del rendiment dels processadors

: La importància de l'arquitectura

Exemple: processadors d'Intel



→ Las mejores arquitecturas han permitido acelerar en un factor de 7 el que se conseguiría con solo mejoras tecnológicas.

Índex

- 1 Concepte d'Arquitectura**
- 2 Els requeriments d'un computador**
- 3 Tecnologia, consum i cost**
- 4 Evolució del rendiment dels processadors**
- 5 Classes de computadors**
- 6 Màster en Enginyeria de Computadors i Xarxes**

Classes de computadors

El mercat dels computadors evoluciona segons la tecnologia base i els hàbits de consum de la societat.

En cada moment, la indústria de la informàtica proposa una sèrie de dispositius possibles. Per exemple:

- En la dècada de 1970, mentre s'estaven constraint els primers microprocessadors, hi havia dues categories de computadors
 - *Mainframes*: grans computadors, caríssims i només accessibles per a grans corporacions.
 - *Minicomputers*: computadors de talla mitjana, molt utilitzats en les universitats



Classes de computadors

En 2012, gràcies al progrés en el disseny dels microprocessadors, de les comunicacions i de les interfícies humanes, les aplicacions dels computadors s'han ampliat

- Un processador i la seua memòria poden ocupar molt poc espai i formar part del disseny d'un dispositiu d'ús corrent.
- Un computador pot estar format per una gran quantitat de processadors amb memòria que sumen les seues prestacions

Els tipus de computador més rellevants ara són:

Personal mobile device

Inclou: *smartphones*, Palm, Tablet, etc.

Característiques generals.

- Consum d'energia molt limitat
 - Depenen de la bateria
 - No hi ha ventilació forçada
- Disseny orientat al temps de resposta garantit (*responsiveness* i *predictability*)
 - Han de mantindre una interfície multimèdia
 - Han de processar quadres de video o blocs d'audio a temps (requisits *real-time*).
- Capacitat de la memòria principal reduïda perquè:
 - codi compacte
- Memòria secundària de tecnologia *flash*.

Sistema empotrat

Inclou: Electrònica diversa, sistemes embarcats d'automoció, electrodomèstics, navegació (aviació, marítima, espacial), etc

Diferències amb PMD:

- Ventall amplíssim de dissenys i prestacions
- Programari de fàbrica (no hi ha programes de tercera parts)

Computador personal

Inclou: Portàtils, Netbooks, Sobretaula.

Característiques generals:

- Ús divers
- Optimitza preu-prestacions (càlcul i gràfics)
- Sol ser el banc de proves dels nous dissenys

Servidor

Computador que ofereix serveis (de dades, de correu, d'impressió, etc.) dins d'una xarxa.

Sol treballar en la base de la infraestructura informàtica d'una corporació.

Característiques generals:

- La disponibilitat és crítica
- Disseny escalable
- Orientat al temps de resposta garantit (*responsiveness*)
- Prima la productivitat



Cluster

És una col·lecció de computadors, cadascun amb el seu sistema operatiu, interconnectats amb una xarxa, visible des de l'exterior com un computador únic

Característiques generals:

- L'alimentació i la refrigeració són una part important de la inversió
- Escalabilitat via xarxa

Casos:

- Cluster a gran escala
- Supercomputadors



Cluster a gran escala

Dóna suport als grans serveis d'Internet: xarxes socials, buscadors, plataformes de comerç, compartició d'arxius, etc.

- Cal gran amplada de banda d'Internet i gran volum d'emmagatzemament secundari
- La disponibilitat és crítica
- Components barats, redundància. Prima la relació prestacions/cost
- Disseny orientat a la seguretat del funcionament (*dependability*)

Supercomputadors

Són màquines dissenyades per a obtenir altíssimes prestacions a qualsevol preu.

- Executen grans aplicacions científiques de forma distribuïda amb poca interacció amb l'usuari
- Les prestacions de la xarxa són crítiques
- Gran productivitat d'aritmètica en coma flotant



Índex

- 1 Concepte d'Arquitectura**
- 2 Els requeriments d'un computador**
- 3 Tecnologia, consum i cost**
- 4 Evolució del rendiment dels processadors**
- 5 Classes de computadors**
- 6 Màster en Enginyeria de Computadors i Xarxes**

Interessat en aprendre més? RISC-V

- RISC V és una arquitectura de conjunt d'instruccions (ISA) de maquinari lliure basat en un disseny de tipus RISC
 - Habilita als arquitectes (de la indústria o acadèmia) per a dissenyar, fabricar i vendre xips i programari de RISC-V
 - Té molt suport programari: compiladors (chisel, GCC) i depuradors, simulador de placa base, etc.
- Convergència dels ISA en les diferents classes de computadors
 - És útil per a una àmplia gamma de dispositius: supercomputadors, clústers, PC, portàtils, tauletes, mòbils
 - Exemples processadors que suporten RISC V de codi obert
 - Rocket: implementa una arquitectura simple per a dispositius mòbils amb baix consum energètic
<https://github.com/chipsalliance/rocket-chip>
 - BOOM: implementa un processador d'altres prestacions. Dissenyat en la Univ. Berkeley
<https://github.com/riscv-boom/riscv-boom>

Interessat en aprendre més?

- Títol oficial, 1 any de durada (60 ECTS).
 - Més informació en <http://mic.disca.upv.es>.
- Assignatures relacionades:
 - Arquitectura i Tecnologia dels Processadors Multinucli
 - Claus de les prestacions dels processadors multinucli.
 - Disseny dels processadors multinucli actuals.
 - Conceptes avançats sobre caus compartides.
 - Memòria principal amb controladors de memòria compartits.
 - Xarxes en xip
 - Arquitectures heterogènies
 - GPUs, embedded (Jetson, Coral, ...)
 - FPGAs, sistòlics
 - Xarxes en xip

Interessat en aprendre més? (cont.)

- Arquitectura de Xarxes d'Altes Prestacions
 - Disseny i construcció de xarxes d'altes prestacions.
 - Técniques de control de flux, tràfeg, tolerància a fallades, etc.
 - Reducció del consum en xarxes d'altes prestacions.
 - Disseny de routers.
- Configuració, Administración i Utilització de Clusters
 - Disseny i configuració de clusters.
 - Sistemes d'emmagatzemament.
 - Equilibrat de càrrega.
 - Clusters d'alta disponibilitat.

UT 1. Introducció a l'Arquitectura dels Computadors

Tema 1.2 Anàlisi de prestacions

J. Flich, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departament d'Informàtica de Sistemes i Computadors
Universitat Politècnica de València

DOCÈNCIA VIRTUAL

Finalitat:
Prestació del servei públic d'educació superior
(art. 1 LOU)

Responsable:
Universitat Politècnica de València.

Drets d'accés, rectificació, supressió, portabilitat, limitació o oposició al tractament conforme a polítiques de privacitat:
<http://www.upv.es/contenidos/DDP/>

Propietat intel·lectual:
Us exclusiu en l'entorn d'aula virtual.
Queda prohibida la difusió, distribució o divulgació de la informació de les classes i particularment la seva compartició en xanxes socials o serveis dedicats a compartir apunts.

La infracció d'aquesta prohibició pot generar responsabilitat disciplinària, administrativa o civil

UNIVERSITAT POLITÈCNICA DE VALÈNCIA



UNIVERSIDAD
POLITECNICA
DE VALENCIA

DISCA

Índex

- 1 Definició de prestacions**
- 2 Principis quantitatius del disseny de computadors**
- 3 La mesura de prestacions**
- 4 Altres mesures de prestacions**

Bibliografia

-  John L. Hennessy and David A. Patterson.
Computer Architecture, Sixth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6
edition, 2018.

Índex

1 Definició de prestacions

2 Principis quantitatius del disseny de computadors

3 La mesura de prestacions

4 Altres mesures de prestacions

1. Definició de prestacions

Temps i productivitat

Dos estils o punts de vista

Visió de l'usuari: *L'usuari vol acabar prompte*

↓ Temps de resposta (*response time*) o temps d'execució (*execution time*): Temps per a completar un treball.

Visió de l'administrador del sistema: *L'administrador del sistema vol molts treballs per unitat de temps.*

↑ Productividad (*throughput*): Operacions o execucions completades per unitat de temps.

Relació freqüent entre els dos estils de mesura:

$$\text{Productivitat} = \frac{1}{\text{Temps d'execució}}$$

Comparacions

La productivitat sol expressar-se de forma relativa.

- Quan cal comparar dos computadors X i Y , es tria el més lent (suposem Y) perquè actue com a referència
- Quan s'examina un conjunt de dissenys X_1, \dots, X_n , es tria un computador Y (inclòs o no en el conjunt) perquè actue de referència comuna

En una comparació elemental entre dos computadors X i Y , cal triar una càrrega (un treball de prova) i mesurar-ne les prestacions en tots dos. Segons l'estil de la mesura, hi obtindrem:

- Temps d'execució: T_X i T_Y
- Productivitats: P_X i P_Y

1. Definició de prestacions

Comparacions (cont.)

La relació S es calcula:

$$S = \frac{T_Y}{T_X} = \frac{P_X}{P_Y} = 1 + \frac{n}{100}$$

Fraseologia:

- “X és S vegades més ràpid que Y”
- “X és $n\%$ més ràpid que Y”

Índex

- 1 Definició de prestacions**
- 2 Principis quantitatius del disseny de computadors**
- 3 La mesura de prestacions**
- 4 Altres mesures de prestacions**

Equació del temps d'execució

$$T_{\text{execució}} = \frac{\text{seg}}{\text{programa}} = \frac{\text{nre. instr.}}{\text{programa}} \times \frac{\text{cicles}}{\text{instr.}} \times \frac{\text{segons}}{\text{cicle}} = I \times CPI \times T$$

■ Tots tres paràmetres estan relacionats:

- $I = f(\text{joc d'instruccions, compilador})$
- $CPI = f(\text{joc d'instruccions, organització})$
- $T = f(\text{tecnologia, organització})$

⇒ No és possible reduir cadascun d'ells sense afectar els altres.

Equació del temps d'execució (cont.)

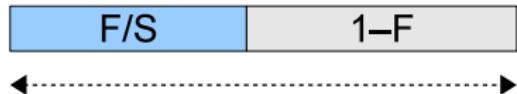
Exemples:

- Un joc d'instruccions més ric pot reduir I , però pot complicar la organització i la implementació, augmentant CPI o T .
- Una organització més complexa pot reduir CPI , però pot ser creixerà el retard del circuits i augmentarà T .

2. Principis quantitatius del disseny de computadors

La llei d'Amdahl

Descriu com afecta el canvi d'una part d'un procés en el total



$$t' = F/S \cdot t + (1 - F) \cdot t$$

- F és la fracció de temps que canvia
- S és l'acceleració que s'hi aplica

Si el procés necessitava un temps t abans del canvi, ara hi caldrà

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

La llei d'Amdahl (cont.)

L'acceleració global S' serà ara

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

La fracció del temps que no millora ($1 - F$) imposa una cota superior a l'acceleració que s'hi pot assolir:

$$S'_\infty = \lim_{S \rightarrow \infty} S' = \frac{1}{1 - F}$$

2. Principis quantitatius del disseny de computadors

La llei d'Amdahl (cont.)

La llei d'Amdahl pot generalitzar-se per a múltiples fraccions (n), quan s'aplica una acceleració local distinta a cadascuna. Així,

$$S' = \frac{1}{\frac{F_1}{S_1} + \frac{F_2}{S_2} + \dots + \frac{F_n}{S_n}}$$

on $F_1 + F_2 + \dots + F_n = 1$.

D'altra banda, una acceleració local S_i pot ser el resultat de la composició de diverses (m_i) acceleracions locals independents. Així,

$$S_i = S_{i,1} \times S_{i,2} \times \dots \times S_{i,m_i}$$

La llei d'Amdahl (cont.)

Exemple: El temps d'execució d'un programa P es descompon en dues fraccions F_1 i F_2 , on F_2 és paral·lelitzable. Originalment P s'executa en un processador amb dos nuclis que treballen a 2.2 GHz. Per a accelerar-ne l'execució penseu canviar el processador a un model amb 4 nuclis que funciona a 3.3 GHz. Quina seria la fórmula per a obtenir l'acceleració global de l'actualització?

Solució:

$$S' = \frac{1}{\frac{F_1}{S_1} + \frac{F_2}{S_2}}$$

$$S_1 = \frac{3.3}{2.2} = 1.5 \quad S_2 = \frac{3.3}{2.2} \times \frac{4}{2} = 3$$

Exemples d'aplicació de la llei d'Amdahl

- En la programació: El temps d'execució d'un programa es concentra en una part del codi.
Príncipi de localitat de referència: el 90% del temps s'executa el 10% del codi
⇒ Convé optimitzar la part del codi més freqüent
- el disseny del joc d'instruccions: quines són les instruccions més freqüents?
- els sistemes amb múltiples processadors: quina fracció dels programes es pot executar en paral·lel?
- en general, al disseny de les parts del computador

2. Principis quantitatius del disseny de computadors

La relació prestacions–cost

- La relació prestacions–cost és el quotient entre les prestacions assolides per un sistema donat i el seu cost (genèric: preu, consum, ...)
- Permet comparar varíes alternatives de disseny i triar-ne la que tinga major quotient

Exemple: Cal millorar un disseny de referència amb unes prestacions P_0 i un cost C_0 i n'hi ha tres opcions:

- A Reduir el cost en un 20 % i les prestacions un 50 %
- B Incrementar les prestacions en un 80 % i el cost en un 20 %
- C Duplicar tant les prestacions com el cost

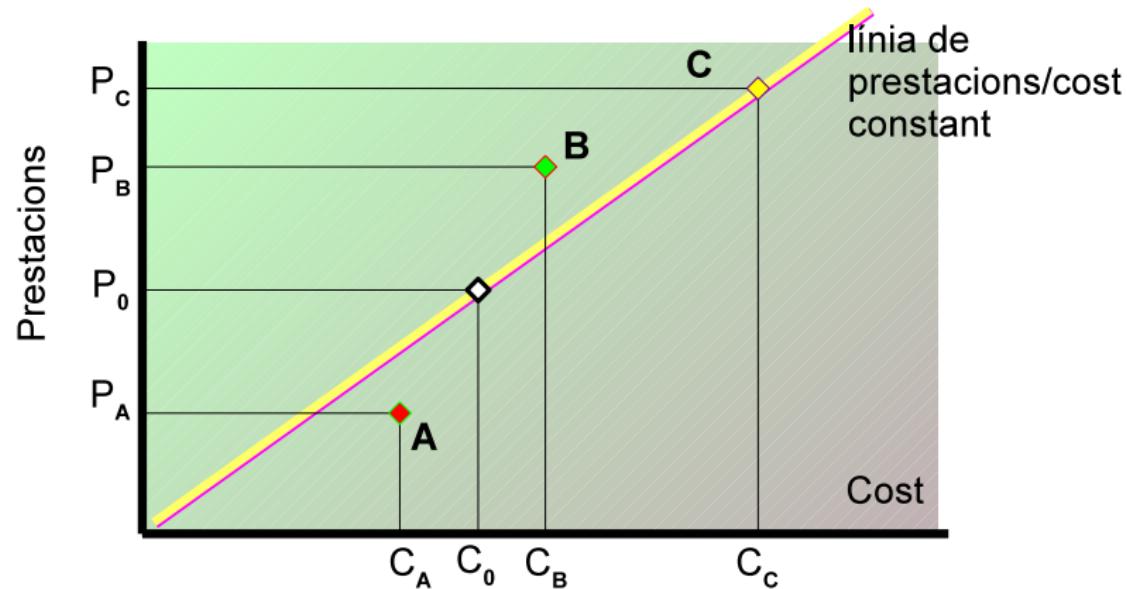
La millor opció és B i la pitjor A, perquè $\frac{P_B}{C_B} > \frac{P_0}{C_0} > \frac{P_A}{C_A}$

L'opció C és indiferent, perquè $\frac{P_0}{C_0} = \frac{P_C}{C_C}$

2. Principis quantitatius del disseny de computadors

La relació prestacions–cost (cont.)

Representació gràfica de l'exemple:



Índex

- 1 Definició de prestacions
- 2 Principis quantitatius del disseny de computadors
- 3 La mesura de prestacions
- 4 Altres mesures de prestacions

3. La mesura de prestacions

Workloads

Per a mesurar prestacions, caldrà triar programes de prova.
Cal disposar del codi font per tal de compilar-lo per a cada computador
sota anàlisi
La millor opció:

- Programes reals

Altres opcions, desacreditades:

- Nuclis (*kernels*). Fragments de codi obtinguts de programes reals.
Exemples: *Livermore Loops* i *Linpack*.
- *Toy benchmarks*. Exercicis acadèmics de programació amb resultats d'execució coneguts. Exemples: *Quicksort*, *Puzzle*, etc.
- *Synthetic Benchmarks*. Programes escrits amb el propòsit de representar el programa mitjà. Exemples: *Whetstone*, *Dhrystone*, etc.

3. La mesura de prestacions

Benchmark suites

Composició: programes reals sense interactivitat i kernels.

Especialitzats a mesurar les prestacions dins d'un perfil d'ús.

Exemples: SPEC (CPU, gràfics, servidors, etc), MediaBench (multimèdia) TPC-xx (Transaccions), EEMBC (empotrats) ...

Actualització: Els programes incluits en el paquet han de representar el treball que fa un usuari típic *en el moment* → s'actualitzen periòdicament.

Reproducibilitat: Les mesures han de ser reproduïbles
→ cal indicar tots els detalls:

hardware: processador, memòria cau, memòria principal, disc, ...

software: sistema operatiu, programes i versions, dades d'entrada, opcions d'execució, ...

3. La mesura de prestacions

Exemple: SPEC CPU95-CINT95 Benchmarks

Benchmark	Application Area	Specific Task
099.go	Game playing &	Plays the game Go against itself
124.m88ksim	Simulation	Simulates the M88100 processor running test programs
126.gcc	Programming & compilation	Compiles pre-processed source into optimized SPARC assembly code
129.compress	Compression	Compresses large text files (about 16MB) using adaptive Lempel-Ziv coding
130.li	Language interpreter	Lisp interpreter
132.jpeg	Imaging	Performs jpeg image compression with various parameters
134.perl	Shell interpreter	Performs text and numeric manipulations (anagrams/prime number factoring)
147.vortex	Database	Builds and manipulates three interrelated databases

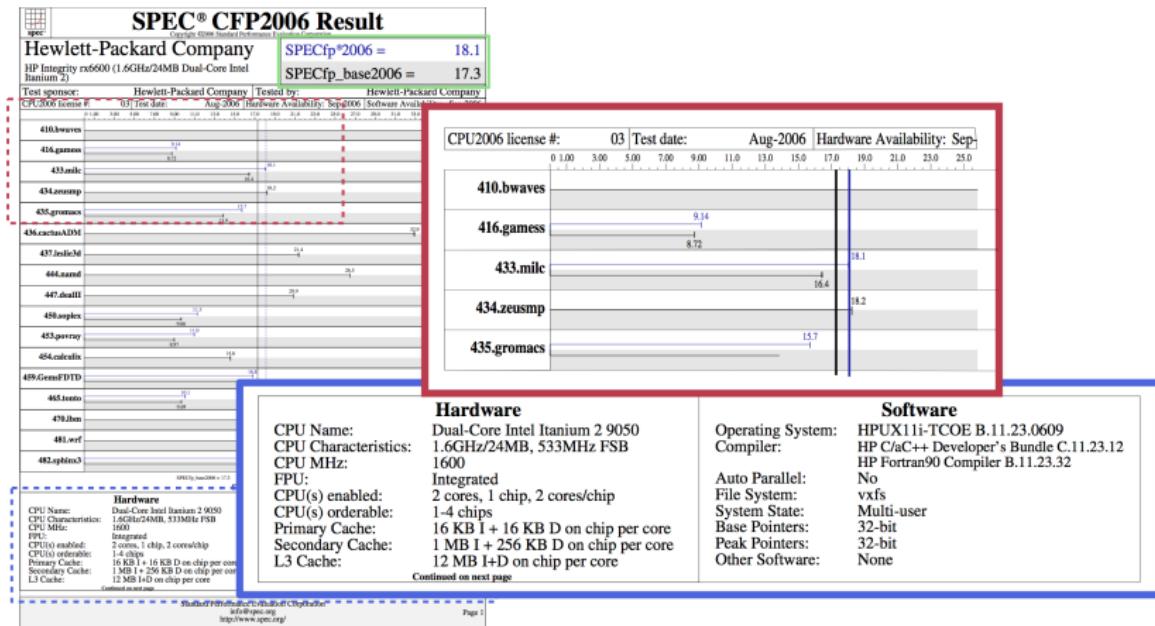
3. La mesura de prestacions

Ejemplo: SPEC CPU95-**CFP95** Benchmarks

Benchmark	Application Area	Specific Task
101.tomcatv	Fluid Dynamics & Geometric Translation	Generation of 2D coordinate system around general geometric domains
102.swim	Weather Prediction	Solves shallow water equations using finite difference approximations (SP)
103.su2cor	Quantum Physics	Masses of elementary particles are computed in the Quark-Gluon theory
104.hydro2d	Astrophysics	Hydrodynamical Navier Stokes equations are used to compute galactic jets
107.mgrid	Electromagnetism	Calculation of a 3D potential field
110.applu	Fluid Dynamics/Math	Solves matrix system with pivoting
125.turb3d	Simulation	Simulates turbulence in a cubic area
141.apsi	Weather Prediction	Calculates statistics on temperature and pollutants in a grid
145.fpppp	Chemistry	Performs multi-electron derivatives
146.wave	Electromagnetics	Solve's Maxwell's equations on a cartesian mesh

3. La mesura de prestacions

Exemple: resultats



3. La mesura de prestacions

Comparació de computadors

Com obtenir una mesura resum de l'execució de molts programes?

→ Característica d'una bona mitjana de temps: el valor mitjà ha de ser directament proporcional al temps total d'execució.

- Temps total d'execució:

$$T_T = \sum_{i=1}^n \text{Temps}_i$$

- Mitjana aritmètica.

$$T_A = \frac{1}{n} \sum_{i=1}^n \text{Temps}_i$$

3. La mesura de prestacions

Comparació de computadors (cont.)

- Temps d'execució ponderat.

$$T_W = \sum_{i=1}^n w_i \times \text{Temps}_i$$

on w_i representa la freqüència del programa i en la càrrega de treball.

- (SPEC) la mitjana geomètrica dels temps d'execució normalitzades a un computador de referència $\rightarrow R$ vegades més ràpid que la referència:

$$R = \sqrt[n]{\prod_{i=1}^n \frac{\text{Temps}_{\text{ref}}}{\text{Temps}_i}}$$

Índex

- 1 Definició de prestacions
- 2 Principis quantitatius del disseny de computadors
- 3 La mesura de prestacions
- 4 Altres mesures de prestacions

MIPS

MIPS=Millons d'instruccions por segon

$$\begin{aligned}\text{MIPS} &= \frac{\text{nre d'instruccions executades}}{T_{\text{execució}} \times 10^6} = \\ &= \frac{I}{I \times CPI \times T \times 10^6} = \frac{1}{CPI \times T \times 10^6} = \frac{f}{CPI \times 10^6}\end{aligned}$$

- És una magnitud intuïtiva, directament proporcional a les prestacions.
- No té en compte el nombre d'instruccions executades:
- Depén del programa que es considere. Diferents programes executen distintes instruccions, amb distinta complexitat i temps d'execució → però el programa executat no se sol indicar!

MIPS (cont.)

- Depén del joc d'instruccions. El mateix programa executa diferent nombre d'instruccions en cada màquina, segons la complexitat del seu joc d'instruccions → no valen per a comparar màquines amb joc d'instruccions molt distints.
- Poden ser inversament proporcionals a les prestacions!
Exemple: comparant un computador amb i sense coprocessador de coma flotant → més prestacions: **amb** coprocessador; més MIPS: **sense** coprocessador.

4. Altres mesures de prestacions

MIPS (cont.)

Siga un programa que executa n milions d'instruccions amb coprocessador i $m > n$ milions d'instruccions sense coprocessador. Els temps requerits per a una instrucció de coprocessador és t_c , i per la resta és t , sent $t_c > t$. Tenim:

	Sense coprocessador	Amb coprocessador
Nre instr:	m instr	n instr
T_{ejec} :	mt	nt_c
MIPS:	$\frac{m}{mt \times 10^6}$	$\frac{n}{nt_c \times 10^6}$

Como $t_c > t \rightarrow \frac{1}{t} > \frac{1}{t_c} \Rightarrow \text{MIPS}_{\text{sense copro}} > \text{MIPS}_{\text{amb copro}}$

MFLOPS

Milions d'operacions en coma flotant per segon.

$$\text{MFLOPS} = \frac{\text{nre d'op. en coma flotant del programa}}{T_{\text{execució}} \times 10^6}$$

- Comptabilitza operacions en lloc d'instruccions: El mateix programa executant-se en diferents arquitectures farà nombre distint d'instruccions, però les mateixes operacions en CF.
- No és aplicable a programes que no facen ús de coma flotant, como són els compiladors o als processadors de textos.

MFLOPS (cont.)

- En realitat, depén del repertori d'instruccions de coma flotant de la màquina, que no sempre és el mateix. Exemple: CRAY-2 no té div; M68882 sí i a més a més sqr, sin i cos.
→ el nombre d'operacions en coma flotant a nivell de màquina del mateix programa ja no és una constant.
Solució: Contabilitzar les operacions de coma flotant del *codi font*.
- Programes diferents executen distintes operacions en coma flotant i totes les operacions en coma flotant no tenen el mateix cost.

UT 1. Introducció a l'Arquitectura dels Computadors

Tema 1.3 Disseny dels jocs d'instrucció

J. Flich, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departament d'Informàtica de Sistemes i Computadors
Universitat Politècnica de València

DOCÈNCIA VIRTUAL

Finalitat:
Prestació del servei públic d'educació superior
(art. 1 LOU)

Responsable:
Universitat Politècnica de València.

Drets d'accés, rectificació, supressió, portabilitat, limitació o oposició al tractament conforme a polítics de privacitat:
<http://www.upv.es/contenidos/DDP/>

Propietat intel·lectual:
Us exclusiu en l'entorn d'aula virtual.
Queda prohibida la difusió, distribució o divulgació de la informació de les classes i particularment la seva compartició en xanxes socials o serveis dedicats a compartir apunts.

La infracció d'aquesta prohibició pot generar responsabilitat disciplinària, administrativa o civil

 **UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA**



UNIVERSIDAD
POLITECNICA
DE VALENCIA

DISCA



Índex

- 1 Generalitats sobre els jocs d'instruccions**
- 2 Classes de jocs d'instruccions**
- 3 Registres i tipus**
- 4 Codificació**
- 5 L'adreçament de la memòria**
- 6 Control de flux**
- 7 El joc d'instruccions del MIPS64**
- 8 Instruccions SIMD**

Bibliografia

-  Bostjan Cigan.
SIMD SSE instructions in C, part one, April 2012.
-  John L. Hennessy and David A. Patterson.
Computer Architecture, Fourth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4
edition, 2006.
-  John L. Hennessy and David A. Patterson.
Computer Architecture, Fifth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5
edition, 2012.
-  Microsoft.
MMX, SSE, and SSE2 intrinsics, April 2013.

Bibliografia (cont.)

-  Microsoft.
Streaming SIMD extensions 4 instructions, April 2013.
-  Microsoft.
Supplemental streaming SIMD extensions 3 instructions, April 2013.

Índex

1 Generalitats sobre els jocs d'instruccions

2 Classes de jocs d'instruccions

3 Registres i tipus

4 Codificació

5 L'adreçament de la memòria

6 Control de flux

7 El joc d'instruccions del MIPS64

8 Instruccions SIMD

1. Generalitats sobre els jocs d'instruccions

Factors de disseny del joc d'instruccions

- El joc d'instruccions és la interfície entre els programes i la ruta de dades
- Les instruccions són el producte de la compilació
 - ☞ Si una instrucció del joc no la gasten els compiladors, és inútil
- L'experiència amb la compilació fa pensar que
 - ☞ Els programes poden ser molt complexes, però la majoria són molt senzills
- Les instruccions simples poden executar-se ràpidament sobre una ruta de dades simple
 - ☞ Les instruccions complexes necessiten rutes de dades complexes
 - ☞ Amb instruccions simples, → baixen CPI i T

1. Generalitats sobre els jocs d'instruccions

Propietats dels jocs d'instruccions

Príncipi bàsic de la compilació

Cas més freqüent: **codi eficient**; resta de casos: **codi correcte**

Regularitat/Ortogonalitat

Sempre que tinga sentit, les operacions, modes d'adreçament i els tipus de dades han de ser independents.

- 👉 Això simplifica la generació de codi, sobretot quan la compilació es descompon en fases.

Oferir primitives i no solucions

A evitar: incloure solucions que suporten directament construccions d'alt nivell

- 👉 Només funcionaran amb un llenguatge.
- 👉 Tindran més o menys funcionalitat de la necessària.

El joc ha de proporcionar primitives perquè el compilador genere el codi òptim.

1. Generalitats sobre els jocs d'instruccions

Propietats dels jocs d'instruccions (cont.)

Príncipi "un o tot"

O només hi ha una manera de fer una determinada cosa, o totes les formes són possibles.

- 👉 Simplificar el cost del càlcul de cada alternativa.

Exemple: condicions de salt

- $>, =$ Una forma de generar totes les condicions
- $>, >=, <, <=, =, <>$ Totes les formes de generar les condicions

No interpretar dinàmicament allò conegit durant la compilació

Les instruccions que operen amb valors coneguts en temps de compilació obliguen a solucions ineficients.

Índex

1 Generalitats sobre els jocs d'instruccions

2 Classes de jocs d'instruccions

3 Registres i tipus

4 Codificació

5 L'adreçament de la memòria

6 Control de flux

7 El joc d'instruccions del MIPS64

8 Instruccions SIMD

2. Classes de jocs d'instruccions

Paradigma actual

Criteri clàssic: emmagatzemament dels operands en la CPU. Les instruccions operen sobre unes dades i generen un resultat. Datos i resultats estan en la memòria i *en la CPU*.

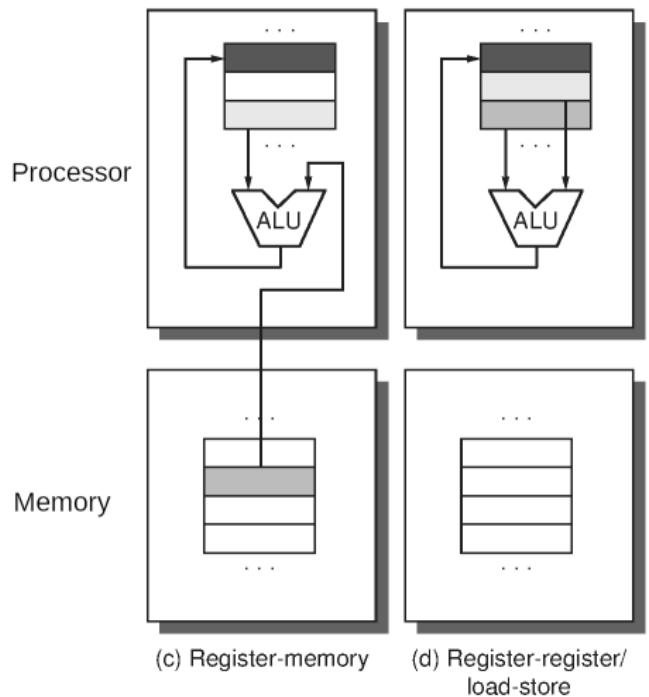
Registres de propòsit general (Ex.: MIPS r2000, x86-64).

Dades i resultats estan en un *banc de registres* o en memòria.

Tots els operands han d'anomenar-se **explícitamente**.

2. Classes de jocs d'instruccions

Paradigma actual (cont.)



2. Classes de jocs d'instruccions

Paradigma actual- Exemple

Codi per a $C = A + B$

Registres de propòsit 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	

- El paradigma de processador amb memòria adreçable i banc de registres de propòsit general permet la compilació eficient

2. Classes de jocs d'instruccions

Classificació de les màquines de registres de propòsit general

Dos paràmetres significatius:

- 1 Nombre n d'operands en una instrucció UAL típica (2 o 3).
 - Si $n = 2$, un dels operands és font i destinació al temps.
 - Si $n = 3$, hi ha els dos operands i el destinatari del resultat.
- 2 Nombre $m \leq n$ d'adreces a la memòria en una instrucció UAL.

Casos:

- Si $m = 0$, model *reg-reg* o *load/store* (típic $m = 0, n = 3$).
 - Instruccions de càlcul: dades i resultats en registre.
 - Les úniques instruccions d'accés a la memòria son del tipus *load* i *store*.
- Si $m < n$, model *reg-mem* (típic $m = 1, n = 2$).
- Si $m = n$, model *mem-mem* (típic $m = 3, n = 3$).

L'elecció afecta al temps d'execució: $T_{ej} = I \times CPI \times T$

2. Classes de jocs d'instruccions

Jocs d'instruccions vigents

Dos tipus de joc d'instrucció majoritaris:

IA (*Intel Architecture*) Pròpia dels processadors d'Intel i compatibles (AMD)

RISC Pròpia dels processadors MIPS, ARM, SPARC, POWER (encara vigents) i HP-PA, Alpha, (desapareguts)

Característiques essencials:

	Intel Architecture	RISC
Model prog.	<i>R-M</i> Registre-memòria	<i>L/S</i> Load/Store
Registres	Pocs, de diverses longituds	Molts, de la mateixa longitud
Format d'instr	Cadena de bytes	Fix (32 bits)

Models R-M i L/S

RISC Model L/S

- 1 Les instruccions de càlcul operen només amb registres
- 2 Hi ha instruccions especialitzades *load* i *store* que transporten les dades entre els registres i la memòria
- 3 La quantitat de treball que fan les instruccions del joc és paregut (o un càlcul o un accés a la memòria)

IA Model R-M

- 1 Les instruccions de càlcul poden operar entre registres o amb un operand en la memòria
- 2 Les instruccions de càlcul tenen dos operands
- 3 Hi ha una instrucció MOV que transporta dades $R \Leftrightarrow M$ i $R \Leftrightarrow R$
- 4 La quantitat de treball que han de fer les instruccions és molt divers

Models R-M i L/S (cont.)

Com afecta al temps d'execució?

RISC Model L/S

- 1 Un programa té més instruccions per fer el mateix treball
→ $I \uparrow$
- 2 Format més senzill i més fàcil de descodificar → $T \downarrow$
- 3 Quantitat de treball per instrucció homogeni → $CPI \downarrow T \downarrow$

IA Model R-M

- 1 Un programa té menys instruccions per a fer el mateix treball
→ $I \downarrow$
- 2 Format variable, més difícil de descodificar → $T \uparrow$
- 3 Quantitat de treball per instrucció molt diferent → $CPI \uparrow T \uparrow$

2. Classes de jocs d'instruccions

Models R-M i L/S (cont.)

Exemple

Intel 32 bits	Equivalent en MIPS-32
ADD EAX, EBX	add \$t0,\$t0,\$t1
ADD EAX, a	lw \$t1,a add \$t0,\$t0,\$t1
ADD a, EAX	lw \$t1,a add \$t1,\$t1,\$t0 sw \$t1,a

Índex

- 1 Generalitats sobre els jocs d'instruccions
- 2 Classes de jocs d'instruccions
- 3 Registres i tipus
- 4 Codificació
- 5 L'adreçament de la memòria
- 6 Control de flux
- 7 El joc d'instruccions del MIPS64
- 8 Instruccions SIMD

3. Registres i tipus

Registres i tipus d'operand

RISC Els registres són nombrosos i tots tenen la mateixa longitud.

Per exemple: MIPS r2000: 32 registres d'enters (32 bits) i 32 de coma flotant (32 bits)

Cada instrucció de càlcul opera amb el contingut complet dels registres implicats

Les instruccions L/S fan la conversió de tipus d'enter que calga

IA Pocs registres, i adaptats als tipus de dades disponibles

Cada instrucció aritmètica té un codi d'operació per a cada tipus

En x86-64 hi ha quatre versions da la instrucció `add` per a operands de 8, 16, 32 i 64 bits

👉 Es fàcil canviar el llarg de paraula en RISC

3. Registres i tipus

Registres i tipus d'operand (cont.)

Exemple: els registres IA-32

- 8 registres de 32 bits EAX ... EDI
- la part baixa de quatre d'ells pot ser tractada com 4 registres de 16 bits o 8 registres de 8 bits

Ampliació a 64 bits:

- Els 8 registres de 32 bits EAX ... EDI són la part baixa de 8 registres de 64 bits RAX ... RDI
- Hi ha 8 registres de 64 bits addicionals

	31	15	8 7	0
EAX	AX	AH	AL	
ECX	CX	CH	CL	
EDX	DX	DH	DL	
EBX	BX	BH	BL	
ESP	SP			
EBP	BP			
ESI	SI			
EDI	DI			

3. Registres i tipus

Registres i tipus d'operand (cont.)

Exemple: els registres MIPS

- 32 registres de 32 bits: R0 ... R31
- Conveni d'ús per part dels compiladors

Name	Number	Use	Preserved across a call?
\$zero	0	The constant value 0	N.A.
\$at	1	Assembler temporary	No
\$v0-\$v1	2-3	Values for function results and expression evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS kernel	No
\$gp	28	Global pointer	Yes
\$sp	29	Stack pointer	Yes
\$fp	30	Frame pointer	Yes
\$ra	31	Return address	Yes

3. Registres i tipus

Registres i tipus d'operand (cont.)

Exemple

Codi font

```
byte a,b,c;
```

```
...
```

```
c = a + b;
```

IA-32

```
a db ...
```

```
...
```

```
MOV AL,a
```

```
ADD AL,b
```

```
MOV c,AL
```

MIPS r2000

```
a: byte ...
```

```
...
```

```
lb $t0,a
```

```
lb $t1,b
```

```
add $t2,$t0,$t1
```

```
sb $t2,c
```

Índex

- 1 Generalitats sobre els jocs d'instruccions
- 2 Classes de jocs d'instruccions
- 3 Registres i tipus
- 4 Codificació
- 5 L'adreçament de la memòria
- 6 Control de flux
- 7 El joc d'instruccions del MIPS64
- 8 Instruccions SIMD

Estratègies de codificació

Les instruccions s'emmagatzemen en la memòria d'acord amb un format, on s'indica l'operació i els operands.

Format fix vs. variable:

Fix

Totes les instruccions ocupen el mateix nombre de bits.

- Facilita la cerca d'instruccions i la seua descodificació.
- En algunes instruccions sobren bits en el format.

Variable

El nombre de bits varia

- Optimitza l'espai ocupat per les instruccions i, per tant, pels programes.
- Complica la cerca i descodificació d'instruccions

Estratègies de codificació (cont.)

Nombre de bits del format:

El nombre de bits destinat al format imposa un límit a l'espai reservat a cadascun dels camps, cosa que limita el nombre de valors codificables:

- nre. d'instruccions (codis d'operació),
- nre. de registres,
- espai adreçable en la memòria
- etc.

Estratègies de codificació (cont.)

Nombre de formats d'instrucció

Com s'assignen els bits dels camps del format als requeriments de les instruccions?

- Format únic

La correspondència entre els bits del format i els camps sempre és la mateixa.

- Facilita la descodificació de la instrucció.
- Hi sobren bits quan les instruccions no requereixen tots els camps previstos.

- Diversos formats.

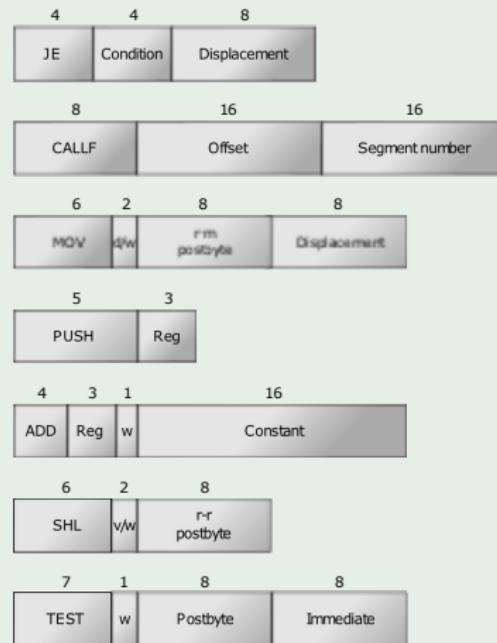
Cada format pot tenir camps distints

- Permet ajustar millor els bits ocupats per la instrucció

Exemples

Codificació IA-32

- Format variable:
una instrucció pot ocupar entre 1 i 17 bytes
- La descodificació és seqüencial: cal conéixer el valor dels bits d'un camp per a descodificar el següent



4. Codificació

Exemples (cont.)

Codificació MIPS64

- Format fix de 32 bits.
- Poden descodificar-se els camps en paral·lel.
- Tres variants del format: R, I i J
- Nre. de bits. Fins a 2^6 codops + 2^6 extensions (format R); fins a 2^5 registres; desplaçaments i immediats de 2^{16} .

I-type instruction



LOAD/STORE: LD rt,Imm(rs) SD rt,Imm(rs)

ALU con constantes DADD rt,rs,Imm

Saltos condicionals BEQZ rs,Imm(PC) BEQ rs,rt,Imm(PC)

Saltos incondicionals JR rs JALR rs

R-type instruction



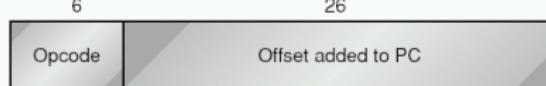
ALU reg-reg DADD rd,rs,rt
func es una extensión del Cod. op.

Indica la operación a realizar en la ALU

Desplazamientos DSLL R1,R2,#shamt

Transferencia entre regs. MOVN rd,rs,rt MFC0 rt,rd

J-type instruction



Jump and jump and link
Trap and return from exception

Índex

- 1 Generalitats sobre els jocs d'instruccions
- 2 Classes de jocs d'instruccions
- 3 Registres i tipus
- 4 Codificació
- 5 L'adreçament de la memòria
- 6 Control de flux
- 7 El joc d'instruccions del MIPS64
- 8 Instruccions SIMD

5. L'adreçament de la memòria

Interpretació de les adreces

- Les unitats físiques de lectura/escriptura (les paraules) estan formades per $W = 2^w$ unitats adreçables (bytes)
- La paraula d'adreça $A = a \cdot W$ conté els bytes d'adreça $A, A + 1, \dots, A + W - 1$
- Dues alternatives, sense cap conseqüència important:

IA Little endian

El byte d'adreça A ocupa la posició menys significativa de la paraula

Adreces
de paraula

0
4

3	2	1	0
7	6	5	4

RISC Big endian

El byte d'adreça A ocupa la posició més significativa de la paraula

Adreces
de paraula

0
4

0	1	2	3
4	5	6	7

Alineament

- Tots els jocs donen accés a unitats de $1, 2, \dots 2^w$ bytes
En el MIPS-32: byte, *halfword*, etc.
- Dues alternatives:

RISC Accés alineat

L'adreça de la unitat formada per 2^i bytes és múltiple de 2^i
En el MIPS-32: l'adreça d'un *halfword* és sempre parella, i
l'adreça d'un *word* és múltiple de 4

IA Accés no alineat

No hi ha restricció
Una instrucció pot accedir a bytes continguts en dues paraules consecutives, i la seua execució exigirà dos accessos físics a la memòria

Modes d'adreçament

Com s'especifiquen els operands de les instruccions?

⇒ modes d'adreçament.

Convé disposar-hi de modes d'adreçament sofisticats?

- Reducció del nombre d'instruccions dels programes
⇒ $I \downarrow$
- Hardware més complex ⇒ $CPI \uparrow$ i/o $T \uparrow$.

5. L'adreçament de la memòria

Modes d'adreçament (cont.)

Exemples de modes

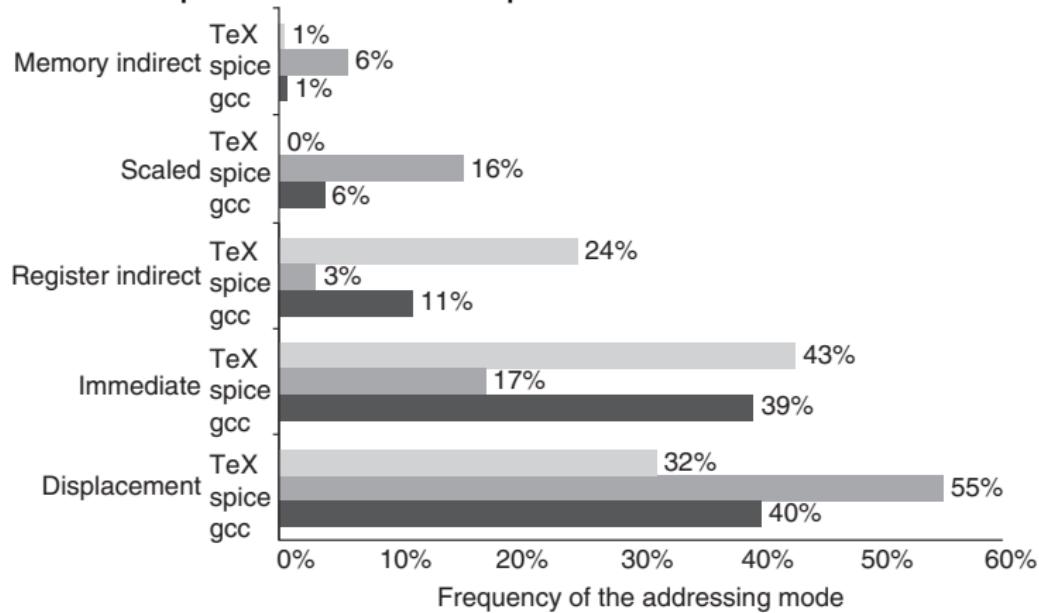
Mode	Exemple	Significat
Directe a registre	add r1,r2,r3	$r1 \leftarrow r2 + r3$
Immediat	add r1,r2,#1	$r1 \leftarrow r2 + 1$
Directe o Absolut	lw r1,(1000)	$r1 \leftarrow \text{Mem}[1000]$
Registre indirecte	lw r1,(r2)	$r1 \leftarrow \text{Mem}[r2]$
Desplaçament	lw r1,100(r2)	$r1 \leftarrow \text{Mem}[100+r2]$
Indexat	lw r1,(r2+r3)	$r1 \leftarrow \text{Mem}[r2+r3]$
Indirecte a mem.	lw r1,@(r2)	$r1 \leftarrow \text{Mem}[\text{Mem}[r2]]$
Autoincrement	lw r1,(r2)+	$r1 \leftarrow \text{Mem}[r2]$ $r2 \leftarrow r2 + d$
Autodecrement	lw r1,-(r2)	$r2 \leftarrow r2 - d$ $r1 \leftarrow \text{Mem}[r2]$
Escalat	lw r1,100(r2)(r3)	$r1 \leftarrow \text{Mem}[100+r2+r3 * d]$

d es la mida de l'operand en bytes

5. L'adreçament de la memòria

Modes d'adreçament: Ús dels modes

Mesures preses en un computador clàssic amb tots els modes



5. L'adreçament de la memòria

Modes d'adreçament: Jocs vigents

IA x86-64 inclou, entre altres, el mode escalat

$[Reg]+[Reg] \times d + \text{desplaçament}$

Combina lliurement els modes amb els codis d'operació.

Amb els modes més complexos, una instrucció pot tindre molt de treball:

Exemplo:

Intel 32 bits	Equivalent en MIPS-32
add EAX, [BX][SI].X	add \$t0,\$t0,\$t1 lw \$t2,X(\$t0) add \$t3,\$t3,\$t2
add EAX,a	lw \$t1,a add \$t0,\$t0,\$t1

No hi ha modes autoincrementats ni autodecrementats

5. L'adreçament de la memòria

Modes d'adreçament: Jocs vigents (cont.)

RISC Per a simplificar:

Mode immediat

- Les instruccions de càlcul tenen dues versions:

R-format : $Rd \leftarrow Rs \text{ op } Rt.$

Exemple: ADD

I-format : $Rd \leftarrow Rs \text{ op } X$

Exemple: ADDI

Rang de valors immediats: solució de compromís entre constants grans i l'espai en el format. 16 bits en el MIPS. Cal una instrucció per permetre el treball amb constants més grans.

Per exemple,

`lui $1, #valor $\Rightarrow R1 \leftarrow valor << 16$`

Modes d'adreçament: Jocs vigents (cont.)

Mode desplaçament

- Per a accedir a la memòria principal, només hi ha el mode d'adreçament: $X(R_n)$
 - Fent $X=0$ s'hi obté el mode registre indirecte
 - Fent $R_n = \$zero$ queda el mode absolut
- Rang de desplaçament:
compromís entre desplaçaments grans i l'espai disponible en el format. 16 bits en el MIPS.

Alguns processadors RISC disposen de modes indexats

Índex

- 1 Generalitats sobre els jocs d'instruccions
- 2 Classes de jocs d'instruccions
- 3 Registres i tipus
- 4 Codificació
- 5 L'adreçament de la memòria
- 6 Control de flux
- 7 El joc d'instruccions del MIPS64
- 8 Instruccions SIMD

Classes d'instrucció de control

Tipus:

- Salts condicionals (*branch*),
- Salts incondicionals (*jump*),
- Crida i retorn de procediment (*call/return*)

Estadístiques d'ús:

- Salts incondicionals, *call* i *return* representen $\frac{1}{3}$, i són sempre efectius.
- Salts condicionals. Altres $\frac{1}{3}$ corresponen amb bucles (efectius prop del 100%). La resta ($\frac{1}{3}$) són efectius en un 50 %. [→] $\frac{5}{6}$ són efectius (salten) i $\frac{1}{6}$ són no efectius (no salten).

Modes d'adreçament en el salt

Relatiu al PC

- La destinació del salt sol estar prop de la instrucció actual
→ les adreces relatives necessiten pocs bits.
- Lo habitual es reservar almenys 8 bits per al desplaçament.
Valors típics són 16–20 bits en salts condicionals i 26 bits en incondicionals.

Indirecte a registre

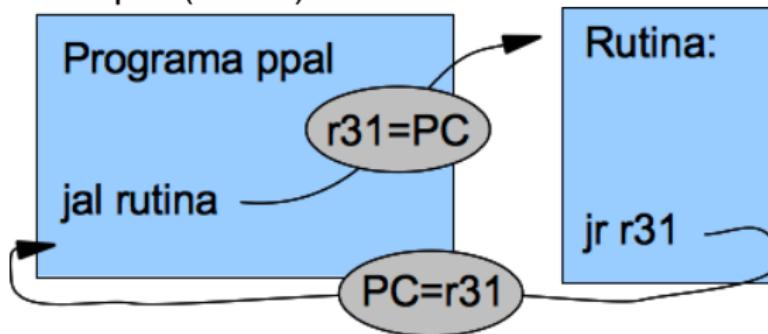
Útil si la destinació del salt és desconegut durant la compilació

- Sentències que seleccionen una d'entre vàries alternatives (i.e., *case*, *switch*).
- Mètodes virtuals en llenguatges orientats a objectes.
- Pas de funcions com paràmetres a altra funció.
- Llibreries enllaçades dinàmicament.

Modes d'adreçament en el salt (cont.)

Salt amb enllaç (*jump and link*)

- Necessari per a les crides a subrutina (*call*).
- És un salt amb mode relatiu al PC o indirecte a registre que guarda l'adreça de retorn.
- El retorn de la subrutina (*return*) es fa mitjançant un salt en mode indirecte a registre, fent servir un registre.
- Exemple (MIPS):



Les condicions de salt

¿Com especificar la **condició** de salt?

Vàries alternatives:

1 Codis de condició (80x86, PowerPC, SPARC).

- El joc d'instruccions defineix un "estat" que canvia segons el resultat de l'última operació aritmètica.
- Habitualment són uns codis de condició o *flags*: *C* (carry), *Z* (zero), *N* (negatiu), etc.
- La instrucció de salt només ha de comprovar la condició corresponent.
- Exemple (Intel): Saltar a ETI si $EAX \leq EBX$

```
CMP EAX,EBX # Resta sense escriure el resultat  
# Si EAX <= EBX, queda Z=1 o N=1  
JLE ETI # Salta si Z=1 o N=1
```

Les condicions de salt (cont.)

- Inconvenients:

- La generació dels codis de condició no és trivial i a més a més cal espai en el xip.
- El fet que totes les instruccions modifiquen els codis de condició planteja problemes en reordenar el codi o en executar múltiples instruccions aritmètiques simultàniament.

2 Comprovació explícita (MIPS). El resultat de les operacions es comprova explícitament per mitjà d'instruccions específiques. No cal tindre codis de condició.

Exemple

; Comparació, salt	; Comparació + salt
dadd r1,r2,#1	dadd r1,r2,#1
slt r10,r1,r3	blt r1,r3,salt
bnez r10, salt	

Índex

- 1 Generalitats sobre els jocs d'instruccions
- 2 Classes de jocs d'instruccions
- 3 Registres i tipus
- 4 Codificació
- 5 L'adreçament de la memòria
- 6 Control de flux
- 7 El joc d'instruccions del MIPS64
- 8 Instruccions SIMD

Càcul

<i>Arithmetic/logical</i>	<i>Operations on integer or logical data in GPRs; signed arithmetic trap on overflow</i>
DADD, DADDI, DADDU, DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB, DSUBU	Subtract, signed and unsigned
DMUL, DMULU, DDIV, DDIVU, MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND, ANDI	And, and immediate
OR, ORI, XOR, XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV	Shifts: both immediate (DS_) and variable form (DS_V); shifts are shift left logical, right logical, right arithmetic
SLT, SLTI, SLTU, SLTIU	Set less than, set less than immediate, signed and unsigned

<i>Floating point</i>	<i>FP operations on DP and SP formats</i>
ADD.D, ADD.S, ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D, SUB.S, SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D, MUL.S, MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D, MADD.S, MADD.PS	Multiply-add DP, SP numbers, and pairs of SP numbers
DIV.D, DIV.S, DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C._.D, C._.S	DP and SP compares: “_” = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

Accés a la memòria i control

<i>Data transfers</i>	<i>Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR</i>
LB, LBU, SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH, LHU, SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW, LWU, SW	Load word, load word unsigned, store word (to/from integer registers)
LD, SD	Load double word, store double word (to/from integer registers)
L.S, L.D, S.S, S.D	Load SP float, load DP float, store SP float, store DP float
MFC0, MTC0	Copy from/to GPR to/from a special register
MOV.S, MOV.D	Copy one SP or DP FP register to another FP register
MFC1, MTC1	Copy 32 bits to/from FP registers from/to integer registers
<i>Control</i>	<i>Conditional branches and jumps; PC-relative or through register</i>
BEQZ, BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ, BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T, BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN, MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J, JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL, JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode

Índex

- 1 Generalitats sobre els jocs d'instruccions
- 2 Classes de jocs d'instruccions
- 3 Registres i tipus
- 4 Codificació
- 5 L'adreçament de la memòria
- 6 Control de flux
- 7 El joc d'instruccions del MIPS64
- 8 Instruccions SIMD

Precedents

El càlcul intensiu amb vectors és una de les aplicacions importants en la història dels computadors.

- Té aplicacions en el control, en la simulació de processos físics, en el processament d'imatges, etc.

Durant els anys 1970 a 2000 (aprox), es dissenyaren supercomputadors “vectorials” amb instruccions que operaven amb vectors.

- Una instrucció vectorial podia operar amb vectors de dimensió donada (típicament 64 o 256) formats per elements de certa grandària (32 o 64 bits)
- Els tipus de dades que manejaven eren, molt sovint, CF en simple o doble precisió

Quan l'escala d'integració ho va permetre, els microprocessadors ampliaren el seu joc amb instruccions SIMD i van substituir als computadors vectorials.

Dades empaquetades

Les instruccions SIMD (*Single Instruction - Multiple Data*) operen sobre registres que contenen diverses dades empaquetades.

- Molts tipus de dades utilitzables: enters de 8, 16, 32, etc. bits i CF en simple i doble precisió.
- El nombre de dades contingut en un registre és

$$n = \frac{\text{longitud del registre}}{\text{longitud d'una dada}}$$

- Els registres tenen longitud fixa, però el joc d'instruccions permet empaquetar dades de diferent llarg
Exemple: amb registres de 64 bits es poden empaquetar

- 8 dades del tipus *byte*
- 4 dades del tipus *halfword*
- 2 dades dels tipus *word* o *float*
- 1 dada del tipus *double*

8. Instruccions SIMD

Dades empaquetades (cont.)

Una instrucció SIMD fa n operacions idèntiques.

Exemple de dues operacions de suma SIMD amb registres de 64 bits:

add_pb R3, R1, R2

(8 dades del tipus byte)

64 bits									
R1	00	A0	00	04	00	70	A0	00	
+									
R2	00	70	FF	FF	00	60	B0	00	
=									
R3	00	10	FF	03	00	D0	50	00	

add_ph R3, R1, R2

(4 dades del tipus *halfword*)

64 bits				
R1	00A0	0004	0070	A000
+				
R2	0070	FFFF	0060	B000
=				
R3	0110	0003	00D0	5000

Instruccions habituals

Instruction category	Alpha MAX	HP PA-RISC MAX2	Intel Pentium MMX	Power PC AltiVec	SPARC VIS
Add/subtract		4H	8B,4H,2W	16B, 8H, 4W	4H,2W
Saturating add/sub		4H	8B,4H	16B, 8H, 4W	
Multiply			4H	16B, 8H	
Compare	8B (>=)		8B,4H,2W (=,>)	16B, 8H, 4W (=,>,>=,<,<=)	4H,2W (=,not=>,<=)
Shift right/left		4H	4H,2W	16B, 8H, 4W	
Shift right arithmetic		4H		16B, 8H, 4W	
Multiply and add				8H	
Shift and add (saturating)		4H			
And/or/xor	8B,4H,2W	8B,4H,2W	8B,4H,2W	16B, 8H, 4W	8B,4H,2W
Absolute difference	8B			16B, 8H, 4W	8B
Maximum/minimum	8B, 4W			16B, 8H, 4W	
Pack (2n bits --> n bits)	2W->2B, 4H->4B	2*4H->8B	4H->4B, 2W->2H	4W->4B, 8H->8B	2W->2H, 2W->2B, 4H->4B
Unpack/merge	2B->2W, 4B->4H		2B->2W, 4B->4H	4B->4W, 8B->8H	4B->4H, 2*4B->8B
Permute/shuffle		4H		16B, 8H, 4W	

Evolució de les instruccions SIMD en els processadors Intel

- 1997 Pentium **MMX** amb vuit registres (MM0-MM7) de 64 bits.
- 1999 Pentium-III amb joc **SSE**. Noves instruccions i registres de 128 bits (XMM0-XMM7).
- 2001 El Pentium 4 amb SSE2 afegeix noves instruccions.
- 2004 SSE3
- 2006 SSSE3 i SSE4
- 2008 **AVX**, amb registres de 256 bits reanomenats com YMM0-YMM15
- 2011 AVX2
- 2015 AVX-512 amb 32 registres ZMM de 512 bits cadascun.

Tendència

En cada nova generació de processadors, un percentatge significatiu dels transistors addicionals es dedica a instruccions vectorials cada vegada més potents i a bancs de registres vectorials més grans.

8. Instruccions SIMD

Banco de registros vectoriales AVX-512

Esquema dels registres AVX-512 (ZMM) com extensió dels registres AVX (YMM) i SSE (XMM)

511	256	255	128	127	0
ZMM0		YMM0		XMM0	
ZMM1		YMM1		XMM1	
...		
ZMM15		YMM15		XMM15	
ZMM16					
...					
ZMM31					

En processadors amb suport als Intel AVX-512, les instruccions SSE i AVX operen sobre els 128 o 256 bits de menor pes dels primers 16 registres ZMM.

Exemple 1: SAXPY

El càlcul de $\vec{Y} = a\vec{X} + \vec{Y}$ és molt freqüent en el càlcul numèric
Es denomina bucle SAXPY (amb simple precisió) o DAXPY (en doble precisió)

```
void SAXPY(int n, float a, float *X, float *Y) {  
    int i;  
    for(i=0; i<n; i++)  
        Y[i] = a*X[i] + Y[i];  
}
```

Amb instruccions no vectorials

n iteracions, n multiplicacions i n sumes.

Exemple 1: SAXPY amb instruccions SIMD

Esquema de la solució:

- Les components del vector ocupen 32 bits (simple precisió)
- Registres de 128 bits: en cadascun cap un bloc de quatre components del vector.
- Suposem tres registres SIMD, de noms `a_reg`, `X_reg` i `Y_reg`
- Cal inicialitzar el registre SIMD `a_reg` amb quatre còpies del valor `a`
- Suposem les instruccions:

`load_ps reg, dir`

Llig un bloc de la memòria

`add_ps rd, rf1, rf2`

Suma vectorial de 4 components

`mul_ps rd, rf1, rf2`

Producte vectorial de 4 components

`store_ps reg, dir`

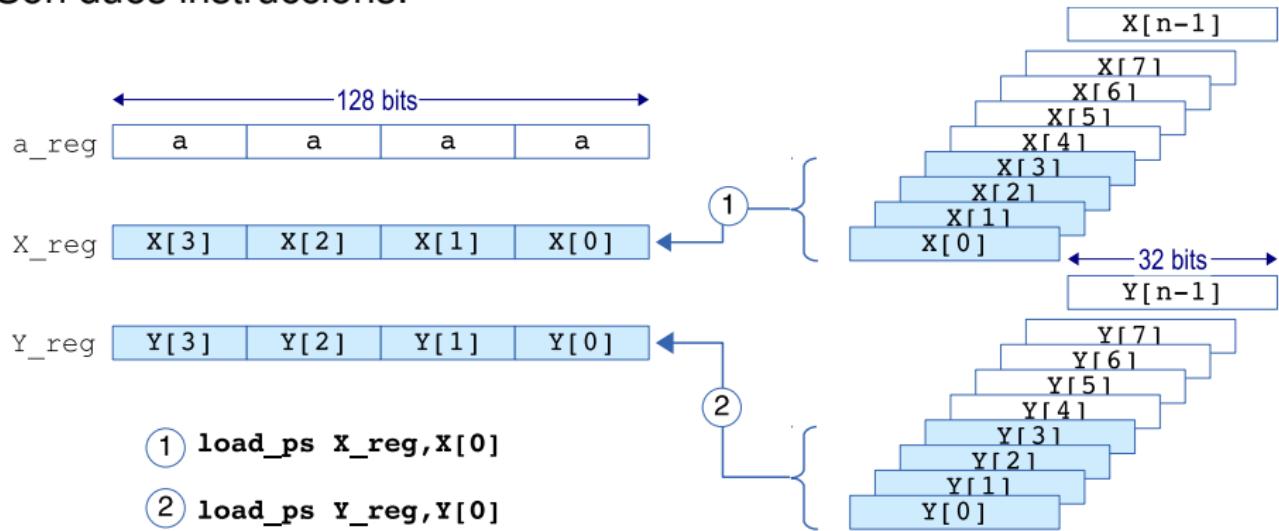
Escriu un bloc en la memòria

8. Instruccions SIMD

Exemple 1: SAXPY amb instruccions SIMD (cont.)

Comença el bucle llegint 4 components de cada vector per a deixarles en un registre del banc SIMD

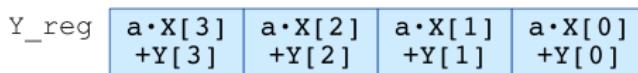
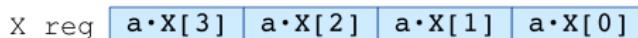
Són dues instruccions:



8. Instruccions SIMD

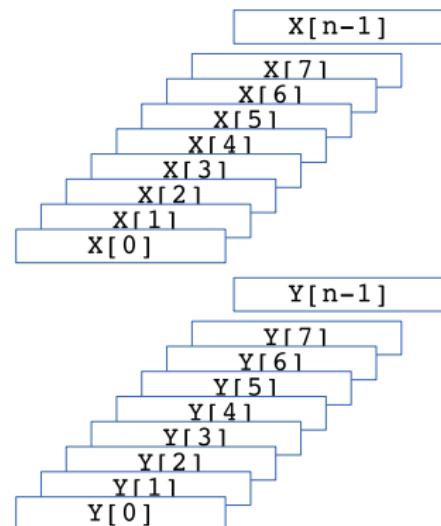
Exemple 1: SAXPY amb instruccions SIMD (cont.)

Una única instrucció permet fer les quatre multiplicacions $a \cdot X[i]$.
Altra instrucció fa les quatre sumes:



③ `mul_ps x_reg, x_reg, a_reg`

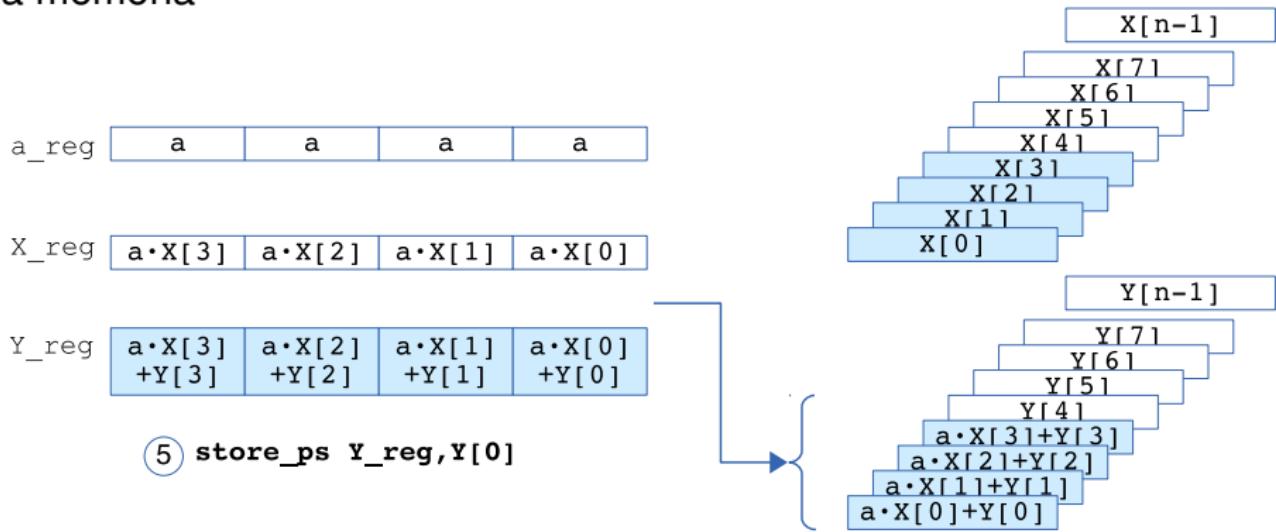
④ `add_ps y_reg, x_reg, y_reg`



8. Instruccions SIMD

Exemple 1: SAXPY amb instruccions SIMD (cont.)

La quinta instrucció actualitza les quatre primeres components $Y[i]$ en la memòria



Exemple 1: SAXPY amb instruccions SIMD (cont.)

Una iteració del bucle amb instruccions SIMD equival a quatre iteracions amb instruccions no vectorials.

Amb instruccions SIMD

Cal iterar $n/4$ vegades

En total: $n/2$ instruccions de càrrega, $n/4$ instruccions de multiplicació $n/4$, instruccions de suma i $n/4$ instruccions d'emmagatzemament

Suport del compilador

La compilació es basa en els tipus de dades i en les expressions per a generar codi.

- Si el programador expressa el codi de forma no vectorial un compilador clàssic no inserirà les instruccions SIMD

Tres maneres d'inserir instruccions SIMD en el codi:

Vectorització automàtica

El programador no explicita el paral·lelisme

El compilador avançat extrau el paral·lelisme del codi font escalar

Fer ús explícit dels tipus de dades SIMD

El programador defineix les variables d'interés amb un tipus específic i les inclou en expressions comuns del llenguatge

Intrinsic functions

El llenguatge disposa d'una biblioteca de funcions SIMD

El programador las fa servir explícitament.

Compilació de l'exemple 1

Vectorització automàtica

El codi font no conté cap referència a la vectorització

```
void SAXPY(int n, float a, float *X, float *Y) {  
    int i;  
    for(i=0; i<n; i++)  
        Y[i] = a*X[i] + Y[i];  
}
```

L'opció de compilació -O3

gcc -O3 saxpy.c -o saxpy

optimitzarà el codi (si pot) inserint instruccions SIMD

8. Instruccions SIMD

Compilació de l'exemple 1 (cont.)

Amb tipus de dades SIMD

El tipus “`_m128`” permet especificar blocs de 4 elements.

```
void saxpy(int n, float a, float *X, float *Y) {  
    __m128 *x_ptr, *y_ptr;  
    int i;  
    x_ptr = (__m128 *) X;  
    y_ptr = (__m128 *) Y;  
    for (i=0; i<n/4; i++)  
        y_ptr[i] = a*x_ptr[i] + y_ptr[i];  
}
```

El compilador dedueix que la expressió “`a*x_ptr[i] + y_ptr[i]`” es compila amb instruccions SIMD.

Compilació de l'exemple 1 (cont.)

Intrinsic functions

Hi ha biblioteques que permeten inserir codi específic.

Les funcions “`_mm_load_ps`”, “`_mm_add_ps`”, “`_mm_mul_ps`” y “`_mm_store_ps`” es tradueixen en les instruccions SIMD adients.

```
void saxpy(int n, float a, float *X, float *Y) {  
    __m128 x_vec, y_vec, a_vec;  
    int i;  
    a_vec = _mm_set1_ps(a);  
    for (i=0; i<n; i+=4) {  
        x_vec = _mm_load_ps(&X[i]);  
        y_vec = _mm_load_ps(&Y[i]);  
        x_vec = _mm_mul_ps(a_vec, x_vec);  
        y_vec = _mm_add_ps(x_vec, y_vec);  
        _mm_store_ps(&Y[i], y_vec);  
    }  
}
```

Exemple 2: Producte escalar

Codi convencional

```
float Scalar(int n, float *X, float *Y) {  
    int i;  
    float prod = 0.0;  
  
    for(i=0; i<n; i++)  
        prod += X[i] * Y[i];  
    return prod;  
}
```

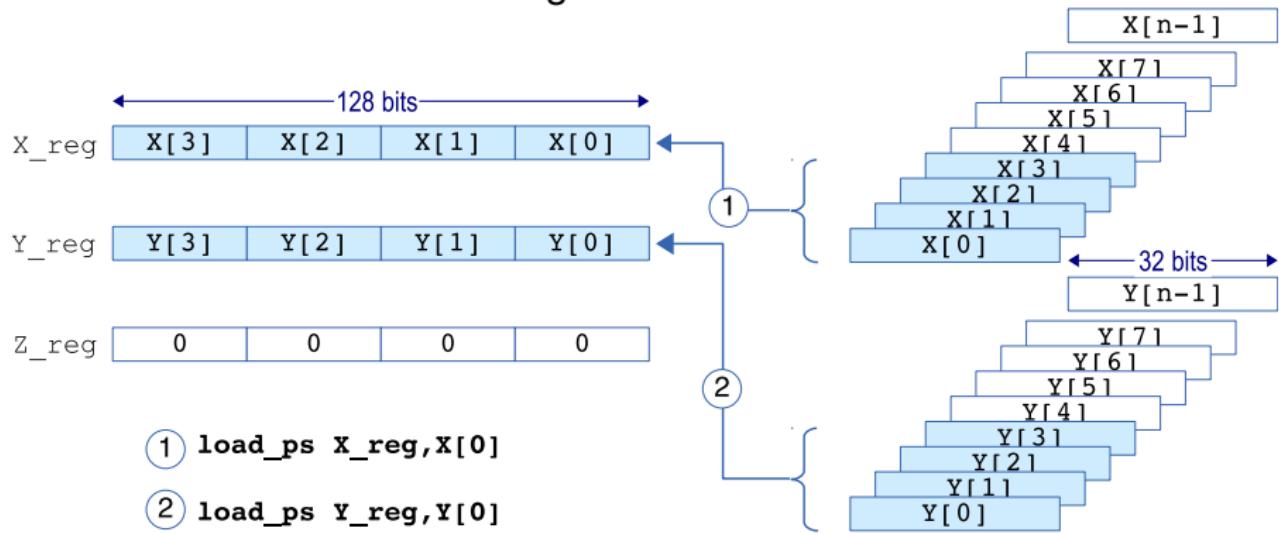
Sense instruccions SIMD

- n iteracions,
- n multiplicacions i
- n sumes.

8. Instruccions SIMD

Exemple 2: Producte escalar (cont.)

Vectorització: Cada iteració llegirà dos blocs...



8. Instruccions SIMD

Exemple 2: Producte escalar (cont.)

...i farà quatre productes i quatre sumes.

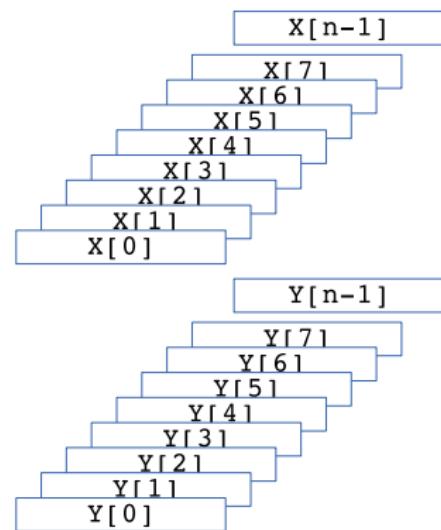
X_reg X[3] X[2] X[1] X[0]

Y_reg X[3] X[2] X[1] X[0]
 • Y[3] • Y[2] • Y[1] • Y[0]

Z_reg X[3] X[2] X[1] X[0]
 • Y[3] • Y[2] • Y[1] • Y[0]

3) `mul_ps Y_reg, X_reg, Y_reg`

4) `add_ps Z_reg, Z_reg, Y_reg`



Exemple 2: Producte escalar (cont.)

Al final de les $n/4$ iteracions, en el registre Z_reg quedaran, per a $i = 0 \dots (\frac{n}{4} - 1)$:

$$\sum(X_{4i+3} \cdot Y_{4i+3}), \sum(X_{4i+2} \cdot Y_{4i+2}), \sum(X_{4i+1} \cdot Y_{4i+1}) \text{ y } \sum(X_{4i} \cdot Y_{4i})$$

El producte escalar resulta de sumar els quatre valors.

Amb instruccions SIMD

- $n/4$ iteracions,
- $n/4$ multiplicacions i
- $4 + n/4$ sumes.

Exemple 2: Producte escalar (cont.)

Código SIMD

```
float Scalar(int n, float *X_reg, float *Y_reg) {  
    float prod = 0.0;  
    int i;  
    __m128 X_reg, Y_reg, Z_reg;  
    Z_reg = _mm_setzero_ps();  
    for(i=0; i<n; i+=4) {  
        X_reg = _mm_load_ps(&m1[i]);  
        Y_reg = _mm_load_ps(&m2[i]);  
        Y_reg = _mm_mul_ps(X_reg, Y_reg);  
        Z_reg = _mm_add_ps(Z_reg, Y_reg);  
    }  
    for(i=0; i<4; i++)  
        prod += Z_reg[i];  
    return prod;  
}
```