Optimització de processadors (4):
Super-pipeline & Super-escalars

- Els primers microprocessadors (anys 1970s) estan basats en arquitectures Von Neumann. Amb l'evolució de la tecnologia van fent-se més complicats però sempre amb aquesta arquitectura. Són processadors purament CISC.

- A inicis dels 1980s sorgeixen els primers microprocessadors RISC (arquitectura Harvard, *pipeline*). Hi ha un entusiasme inicial pels RISC.

- Posteriorment convicció que en dissenys RISC avantatge amb inclusió de característiques CISC i viceversa.

- Resultat: Avui dia, dissenys RISC (PowerPC) no són RISC "purs"* i els d'origen CISC recents (Pentium), incorporen característiques RISC.

*S'ha de fer notar que existeixen processadors CISC purs i RISC purs, però no solen ser processadors de gama molt molt alta, sinó per usos o mercats més específics.

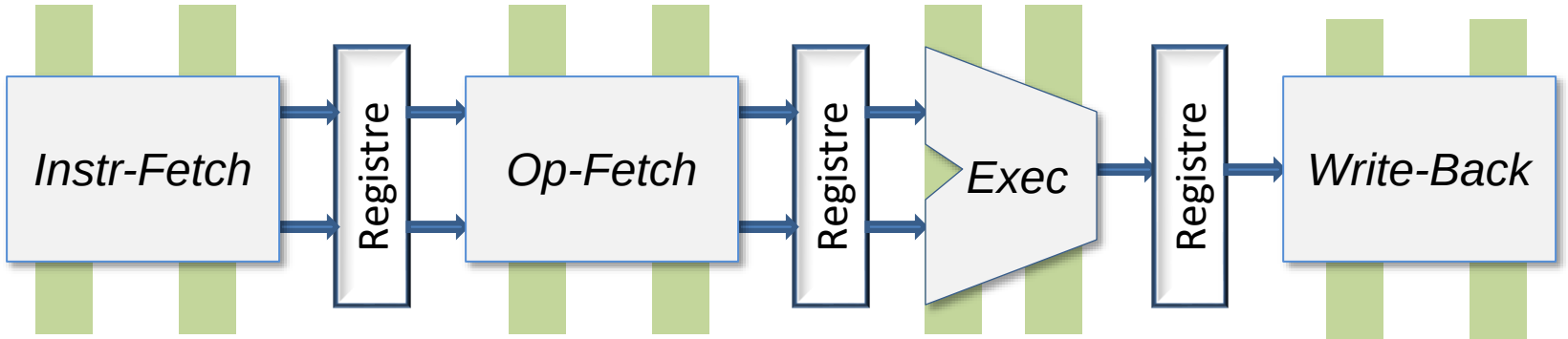
De fet, fins i tot la terminologia es confusa de vegades. Per exemple, l'acrònim RISC vol dir “processador amb un conjunt d’instruccions reduït”. Però al mercat podem trobar processadors CISC amb un conjunt d’instruccions més reduït que el de alguns RISC.

Hi ha altres paràmetres que ens permeten caracteritzar millor si un processador és RISC o CISC. En principi, es consideren típiques RISC les característiques següents:

- Longitud única d’instrucció.
- Nombre petit de modes de direccionament de memòria (típicament <5).
- No s’usa direccionament indirecte, perquè s’ha de fer un accés extra a memòria per aconseguir un operand.
- No hi ha operacions que combinin accés a memòria amb càlculs aritmètics. Per exemple: suma des de memòria, suma a memòria.
- No es direcciona més d’un operand de memòria per instrucció.
- Implementar un gran nombre de registres.
- Operacions Load/Store no suporten alineament arbitrari de dades.
- Un nombre màxim d’usos de la unitat de gestió de memòria (MMU) d’una direcció de dada en cada instrucció

Processador *Super-pipeline*

- Aprofita el fet de que moltes etapes realitzen tasques que requereixen menys de la meitat d'un cicle de clock.
- Es multiplica (per n, sent n el número de subdivisions que es fan de cada etapa) la freqüència de rellotge. Així es poden realitzar n tasques en 1 cicle de rellotge extern.
- El nombre d'etapes en que es divideix indica el grau.
- Augmenta el nombre d'*stalls (però també són més curts).*



Processador Super-Escalar

- Implementació super-escalar és l'arquitectura en que més d'una instrucció (aritmètica entera, punt flotant,..) poden iniciar la seva execució simultàniament i executar-se de manera independent.
- El concepte és diferent al de les etapes d'un *pipeline*. Als escalars les instruccions comencen simultàniament amb recursos independents.
- Bàsicament, tindrem vàries unitats d'execució a les quals enviem a l'hora vàries instruccions.
- En principi, amb aquestes arquitectures es poden aconseguir CPI<1. tenint en compte, a més, que és una tècnica compatible amb la del *pipeline*.

- L'arquitectura super-escalar és propera a l'estructura RISC però pot ser implementada també en processadors CISC.
- Els processadors super-escalars actuals processen entre 2 i 6 instruccions a la vegada
- Tenint en compte que amb aquests processadors (super-escalars + *pipeline*) executarem més instruccions simultàniament, és fàcil imaginar que els problemes que vàrem estudiar als *pipeline* ara es multiplicaran:
  - Més problemes de dependencies de dades.
  - Augmenten també els problemes dels salts condicionals.
  - I en general augmentarà el nombre d'*stalls*

- Hem de pensar que per que un processador escalar sigui funcional, s'han d'introduir elements nous. Per exemple, abans d'enviar una instrucció a una unitat d'execució s'ha de verificar que no necessita dades que encara no es disposen, que la unitat i els recursos necessaris estan disponibles...

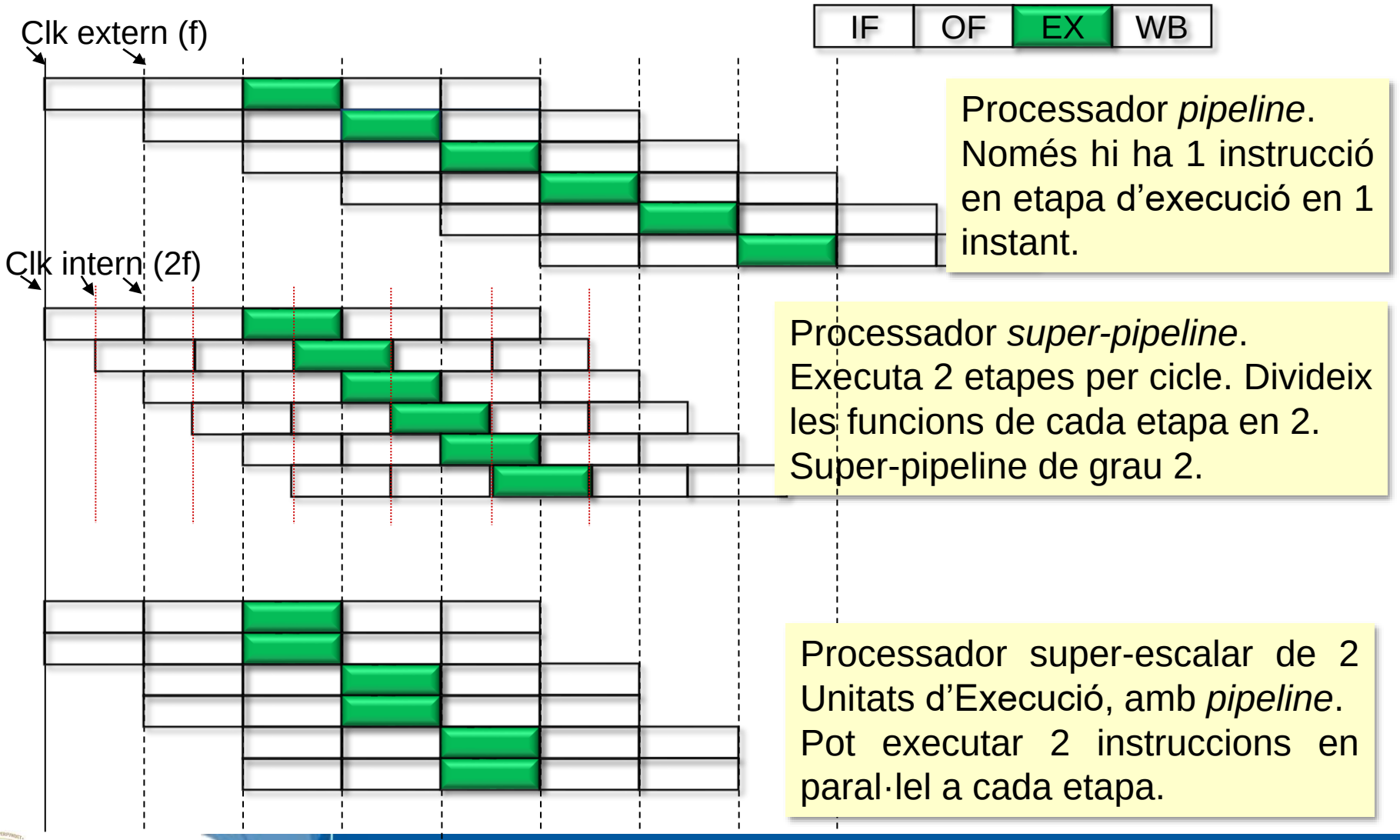
- D'això s'encarrega una/es nova/es unitat/s que es diu/en:

  - Unitat de *Schedulling-Dispatch*.

- A més, els busos han de ser més amples per poder subministrar instruccions i dades a totes les unitats.

  - Mida gran de bus de sistema.

Pipeline, Super-Pipeline i Super-Escalar



Processador *pipeline*. Només hi ha 1 instrucció en etapa d'execució en 1 instant.

Processador *super-pipeline*. Executa 2 etapes per cicle. Divideix les funcions de cada etapa en 2. Super-pipeline de grau 2.

Processador super-escalar de 2 Unitats d'Execució, amb *pipeline*. Pot executar 2 instruccions en paral·lel a cada etapa.

- Si pensem en un processador super-escalar de 2 unitats d'execució (i *pipeline*) podem pensar que podríem aconseguir CPI=0,5 (o millor dit CPI→0,5).
- A més, com que la tecnologia ha millorat molt, no hi ha cap inconvenient per implementar més unitats d'execució (UE) en paral·lel. Si posem 3 UE vol dir que CPI→1/3, amb 4 UE tindríem CPI→1/4...
- Si pensem que un 80486 tenia del ordre de 1.250.000 transistors, i que avui en dia es poden fer xips amb més de mil milions de transistors:
  - Perquè no fem processadors amb mil UE en paral·lel?
- Tecnològicament és factible i aconseguiríem: CPI→1/1.000

Problema de la Unitat de *Schedulling-Dispatch*

- Com hem dit, la unitat de *Dispatch* (hardware) s'encarrega de distribuir les instruccions entre les diferents UE.
- El disseny de la unitat de *Dispatch* és força complex, i molt costós (en àrea) especialment en un processador amb moltes unitats d'execució.
- És evident, que quantes més UE tinguem en paral·lel més problemes de dependencies tindrem i per tant més complexa haurà de ser la unitat de *dispatch*. De fet, la seva complexitat augmenta exponencialment amb el nombre d'unitats d'execució.
- Això fa que no sigui factible fer processadors amb moltes UE.

Solució al problema de la Unitat de Dispatch

Amb la intenció de solucionar el problema de la complexitat de la unitat de *dispatch*, van sorgir els processadors VLIW.

VLIW: (*Very Long Instruction Word*)

- Processador amb n unitats d'execució.
- Longitud d'instrucció llarga (centenars de bits) dividida en n subinstruccions (tantes com UE), especificant moltes operacions que s'executen de forma síncrona i simultània.
- El compilador decideix a quina unitat d'execució va cada subinstrucció (*trace schedulling*).
- Implica: + temps de compilació - temps d'execució.
- El codi de programa generat, ocupa més que en processador super-escalar.

Ex:
Una instrucció VLIW pot contenir 2 operacions amb enters, 2 amb punt flotant, dos referències a memòria i un salt
– Si cada camp té una longitud entre 16 a 24 bits la longitud de la instrucció serà entre 112 a 168 bits

- És evident, que hi haurà problemes de dependència al codi original, que ha de solucionar el compilador. Què farà?:
  - Quan sigui possible reordenarà* les instruccions (subinstruccions de la instrucció VLIW).
  - Quan no sigui possible, introduirà instruccions Noop.
- També introduirà instruccions Noop, com a subinstruccions, quan una unitat d'execució estigui ocupada.

* Sempre que es canvia l'ordre original de les instruccions, es diu que es fa una execució fora d'ordre (out of order execution).

Limitacions dels processadors VLIW

- Una de les limitacions és l'ample del bus per transmetre les instruccions VLIW.
- L'altre gran limitació és que aquests processadors no són efectius per tasques genèriques. Són efectius només per tasques molt simètriques (si no, les dependències solen ser moltes i generen molts "*stalls*") .
- Les tasques per les que són més eficaços els processadors VLIW són les de multimèdia. A aquestes aplicacions, es sol fer el mateix càlcul per moltes dades diferents (gestionar una pantalla, un fitxer de vídeo o àudio...).
- Això fa que encara que no ens siguin molt coneguts, si que són molt utilitzats a equips multimèdia (lectors i gravadors de DVD, MP3, MP4...).

Com hem vist, existeixen molts tipus de processadors, amb diferents arquitectures, nivells de complexitat, i cada un amb les seves aplicacions i mercat:

- Amb arquitectura Von Neumann, o Harvard.
- Amb *pipeline*.
- Super-Escalars.
- VLIW.
- CISC-RISC...

Volem fer una classificació de forma que puguem emmarcar tots els possibles tipus de processadors.

Classificació de Flynn (1966)

En realitat, no és pròpiament una classificació de processadors sinó de computadors.

Aquesta clasificació està basada en la relació entre instruccions i dades, i estableix els 4 possibles tipus:

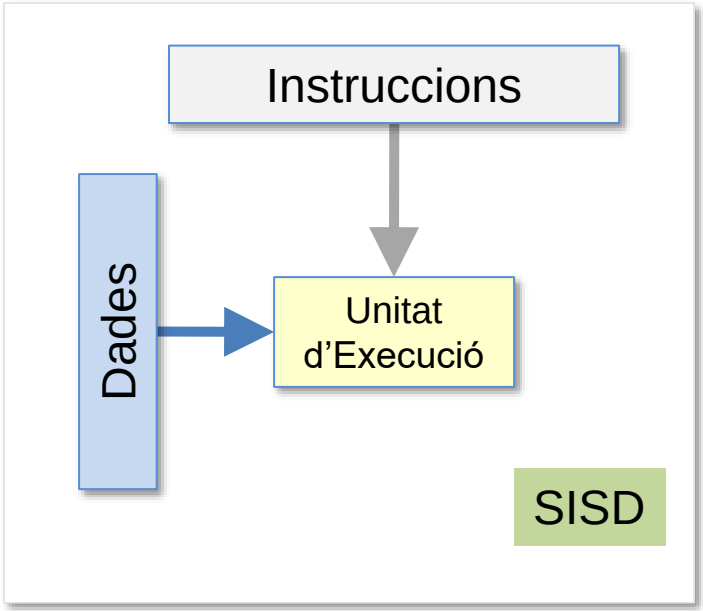
- SISD Una instrucció, un flux de dades
- SIMD Una instrucció, múltiples dades
- MISD Múltiples instruccions, un flux de dades
- MIMD Múltiples instruccions, múltiples flux dades.

Els processadors actuals tenen diversos tipus d'instruccions. Per ex. el Pentium 4, Atlhon, Core 2, Phenom, Core i7... tenen instruccions SIMD i algunes SISD. Els multicore a més són també MIMD.

Processadors SISD

Single Instruction, Single Data Stream

- Un sol processador
- Un sol flux d'instruccions
- Les dades es guarden en una sola memòria

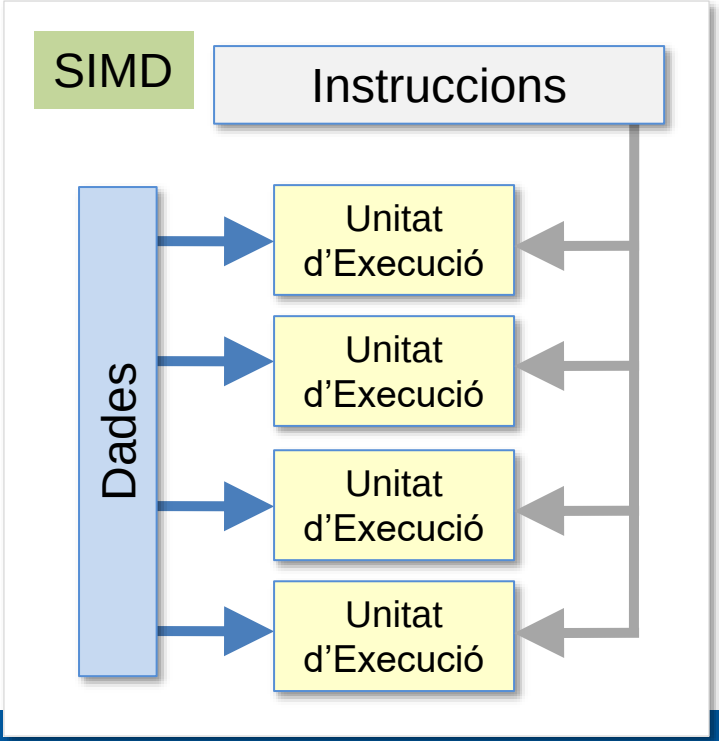


Cada instrucció fa una operació amb una dada (o dos segons els operands).

Processadors SIMD

Single Instruction, Multiple Data Stream

- Una sola instrucció controla la execució simultània i sincronitzada d'un número d'elements de procés.
- Cada instrucció és executada amb diferents conjunts de dades per diferents processadors o Unitats d'execució.
- Van molt bé per a processament paral·lel o vectorial.

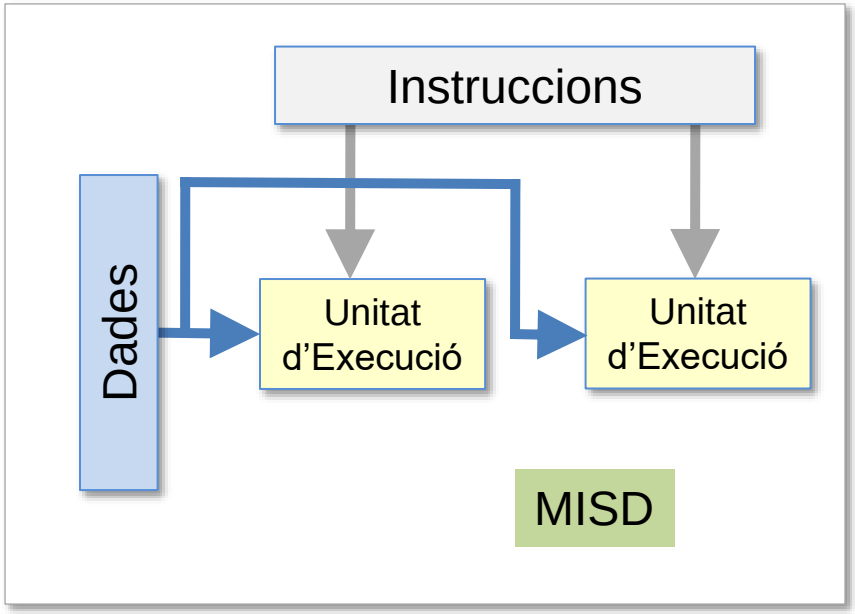


Exemples d'instruccions d'aquests tipus són les MMX (per multimèdia) o les SSE_i.

Processadors MISD

Multiple Instruction, Single Data Stream

- Varies unitats funcionals efectuen diferents operacions amb una sola dada.

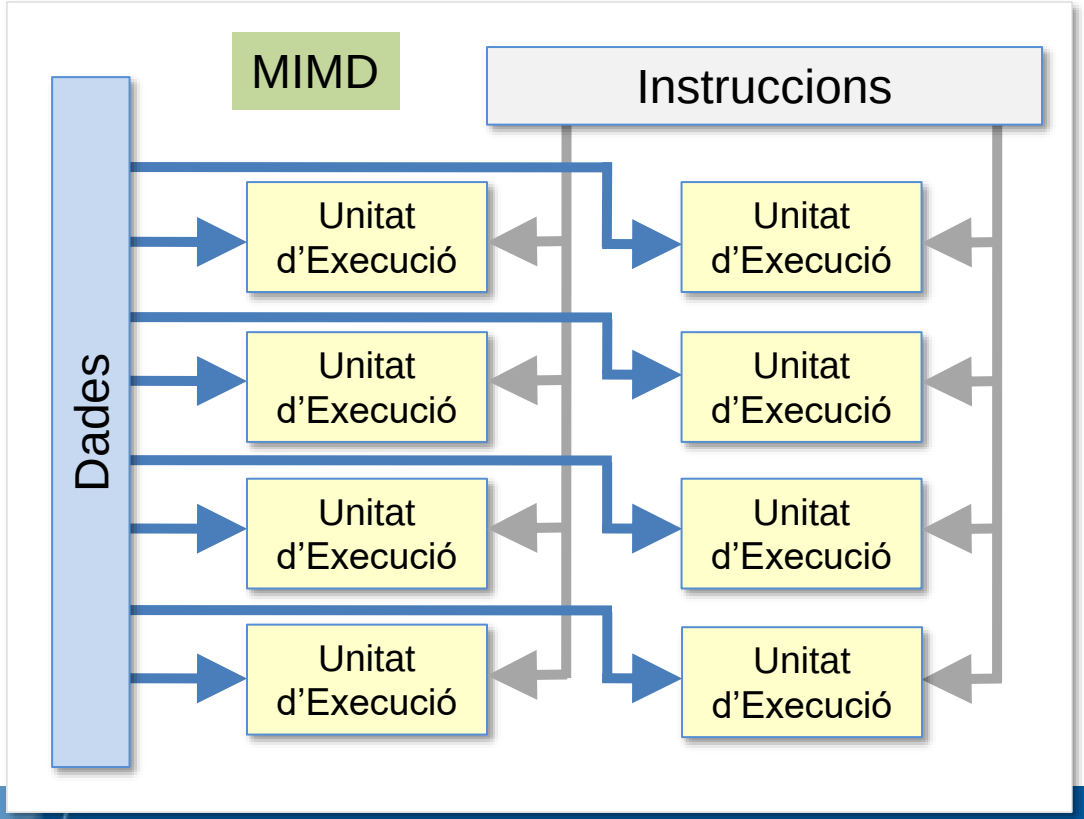


- Els processadors amb *pipeline* es consideren d'aquest tipus. Encara que els puristes diuen que no ho són ja que la dada és diferent per a cada instrucció, tot i que al processador s'executen vàries instruccions simultàniament i només hi ha una dada a cada instant.
- Processadors redundants.

Processadors MIMD

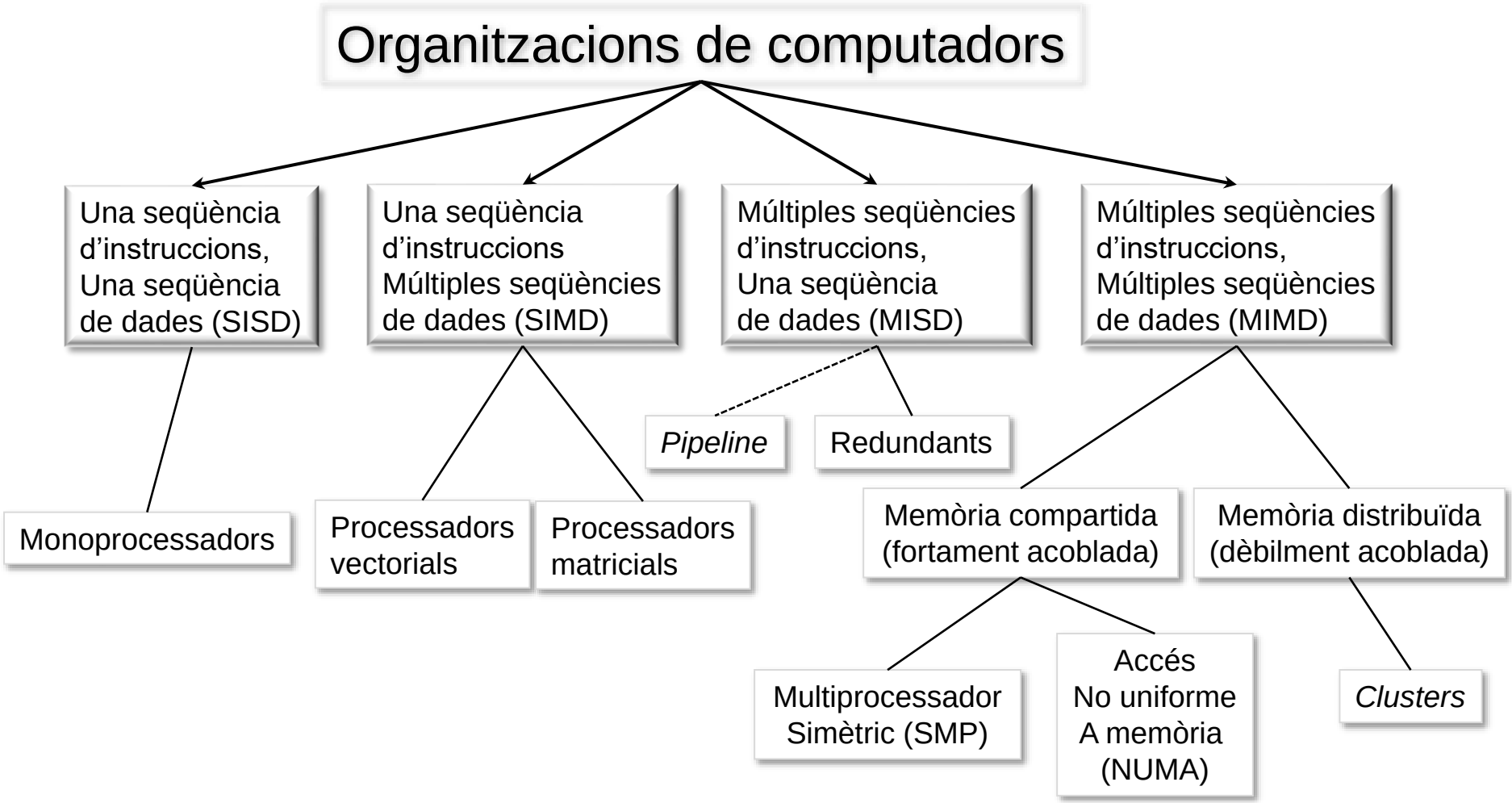
Multiple Instruction, Multiple Data Stream

- Conjunt de Processadors, "*Cores*", o Unitats d'Execució.
- Simultàniament executa diferents seqüències d'instruccions amb diferents conjunts de dades.
- Poden ser de memòria compartida o distribuïda.



Exemples de processadors d'aquest tipus són els *Multicore* i els *Multithreading*.

TAXONOMIA de les ARQUITECTURES de PROCESSADORS



Una mica d'Història d'Arquitectures

- 70's: Al 1971 es fabrica el primer microprocessador que va sortir al mercat. A mida que millora la tecnologia la tendència és suportar tanta funcionalitat com fos possible (CISC).

- 80's: S'implementen els primers processadors RISC. Es tendeix a considerar la trajectòria complerta d'aplicacions a Hardware amb anàlisi quantitativa sobre cost i prestacions (RISC) amb *pipeline* d'instruccions.

- 90's: Tendència un altre cop cap a processadors més complexos. Els grans avenços tecnològics amb desenvolupaments VLSI (*Very Large Scale of Integration*) possibiliten el disseny de processadors amb moltes unitats funcionals operant concurrentment. Processadors que exploten el Paral·lelisme a Nivell d'Instruccions (ILP: *Instruction Level Parallelism*). 2 classes de processadors ILP:
  - Superescalars i
  - VLIW

- Els processadors super-escalars usats a PCs i WSs esdevenen molt complexos degut al seu suport per gestionar (*issuing*) múltiples instruccions per cicle i execució especulativa.

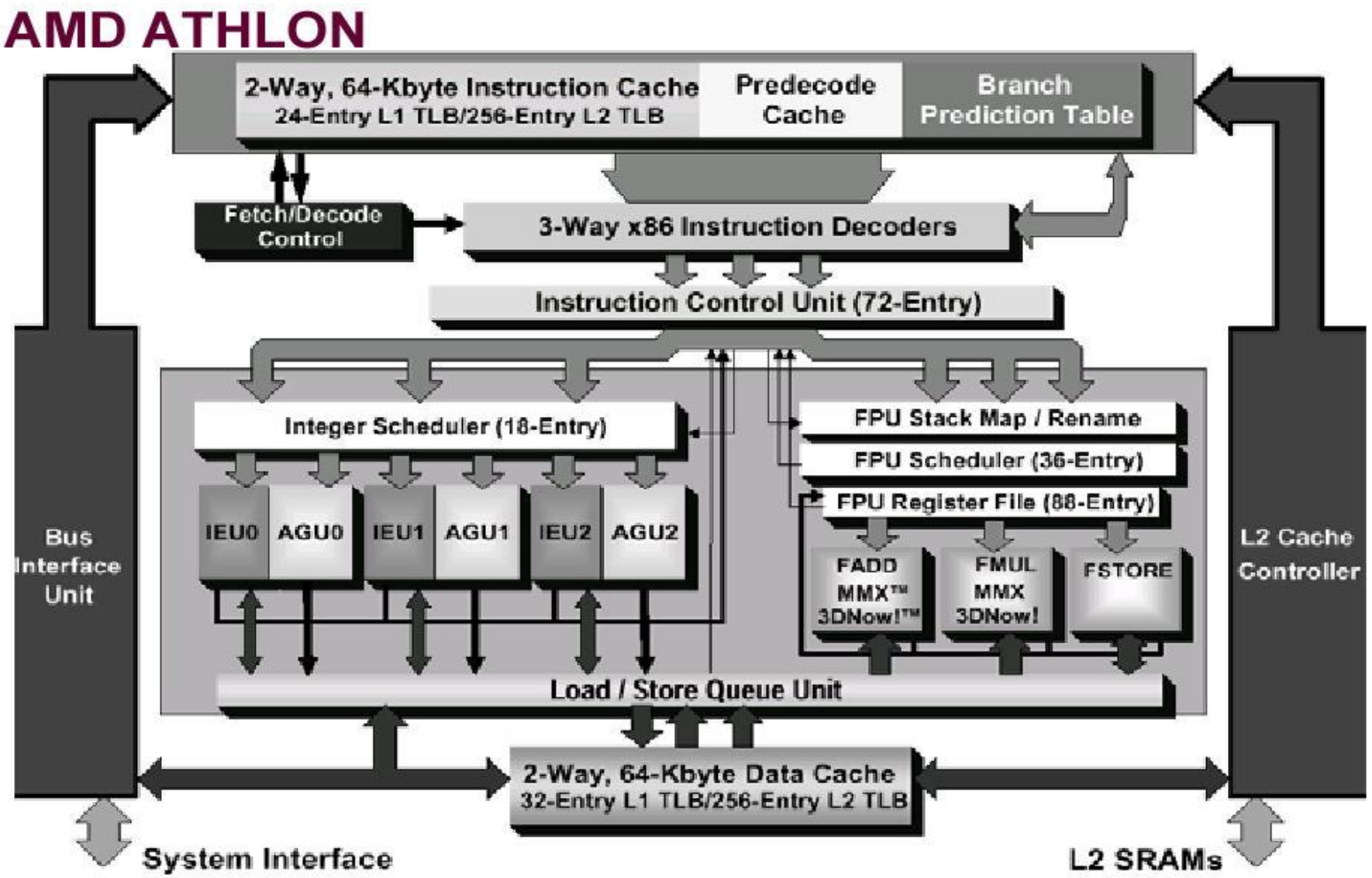
- Les unitats de control i execució d'aquests processadors (Pentium Pro, MIPS R10000, PowerPC 620, HP 8000, i el Alpha 21264) esdevenen molt complexes, amb moltes etapes del *pipeline* i de control d'execució.

- 2000's: Continua la mateixa tendència. El gran increment de la densitat d'integració permet avançar en punts com la:
  - Integració de diversos nivells de caché al processador.
  - Implementació de tecnologies multi-fil (*multithreading*).
  - Implementació de més d'un *Core* al mateix xip.

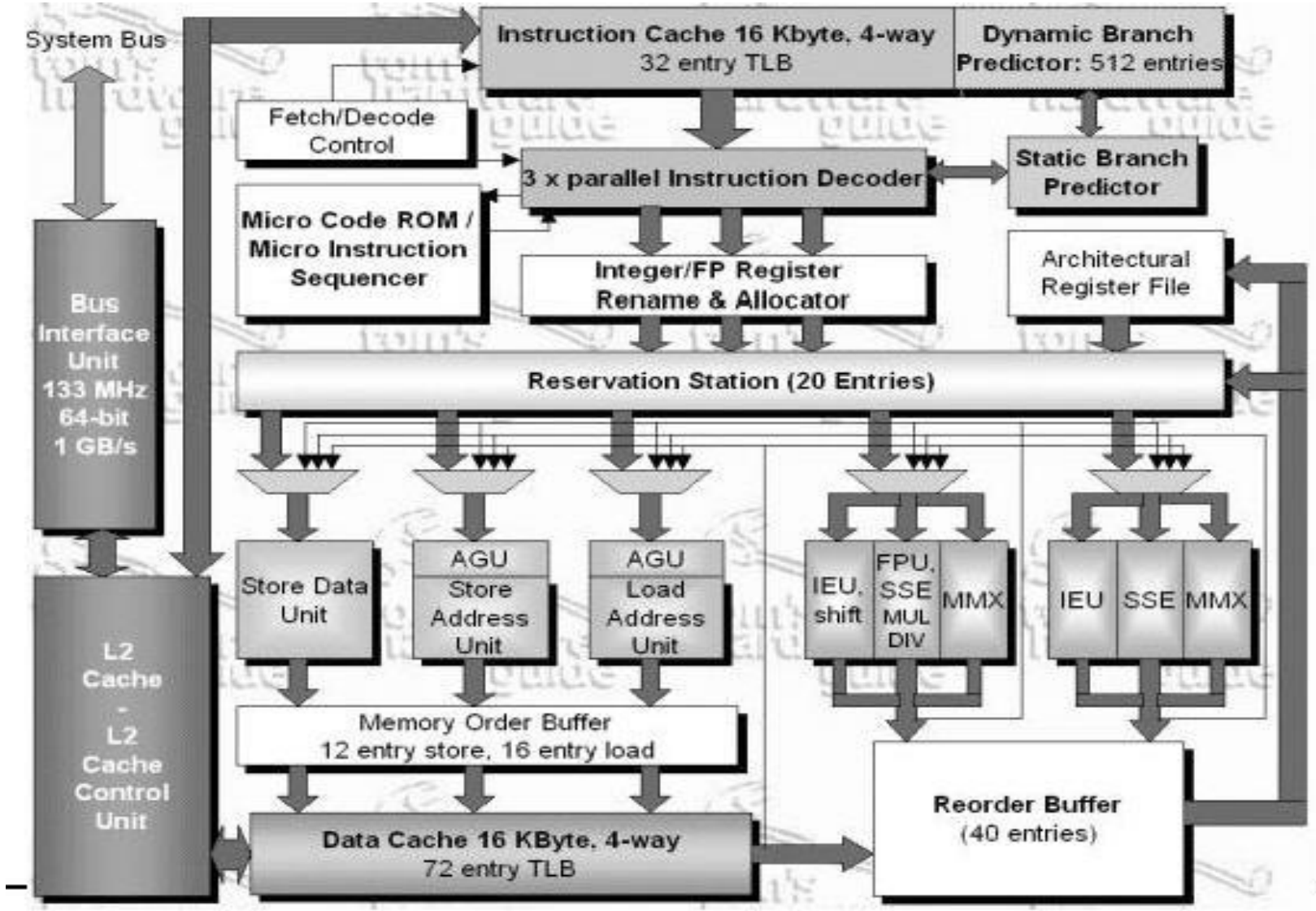
- Això pel que es refereix als processadors "cars" i orientats a mercat de propòsit general d'altes prestacions, no adequats per sistemes "*embedded*" de baix cost.

- Els Processadors VLIW esquiven la lògica de control i decodificació complexa i el cost associat, al preu de no ser compatibles amb els seus predecessors, i no ser òptims per tasques genèriques.
- Els processadors VLIW tenen millors propietats d'escalabilitat. Són més flexibles i adaptables a funcions per aplicacions específiques.
- Bons candidats per sistemes *embedded* de baix cost i altes prestacions per aplicacions més específiques.
- Processadors VLIW interessants que han aparegut al mercat són:
  - Trimedia i Nexperia de Philips.
  - Mpact de Chromatic.
  - TMS320C6x de Texas Instruments.
- A més, existeixen milers de models de processadors de molts tipus diferents, pensats per múltiples aplicacions. Des de models extremadament senzills i econòmics, a alguns amb cert tipus de "perifèrics" incorporats molt especialitzats.

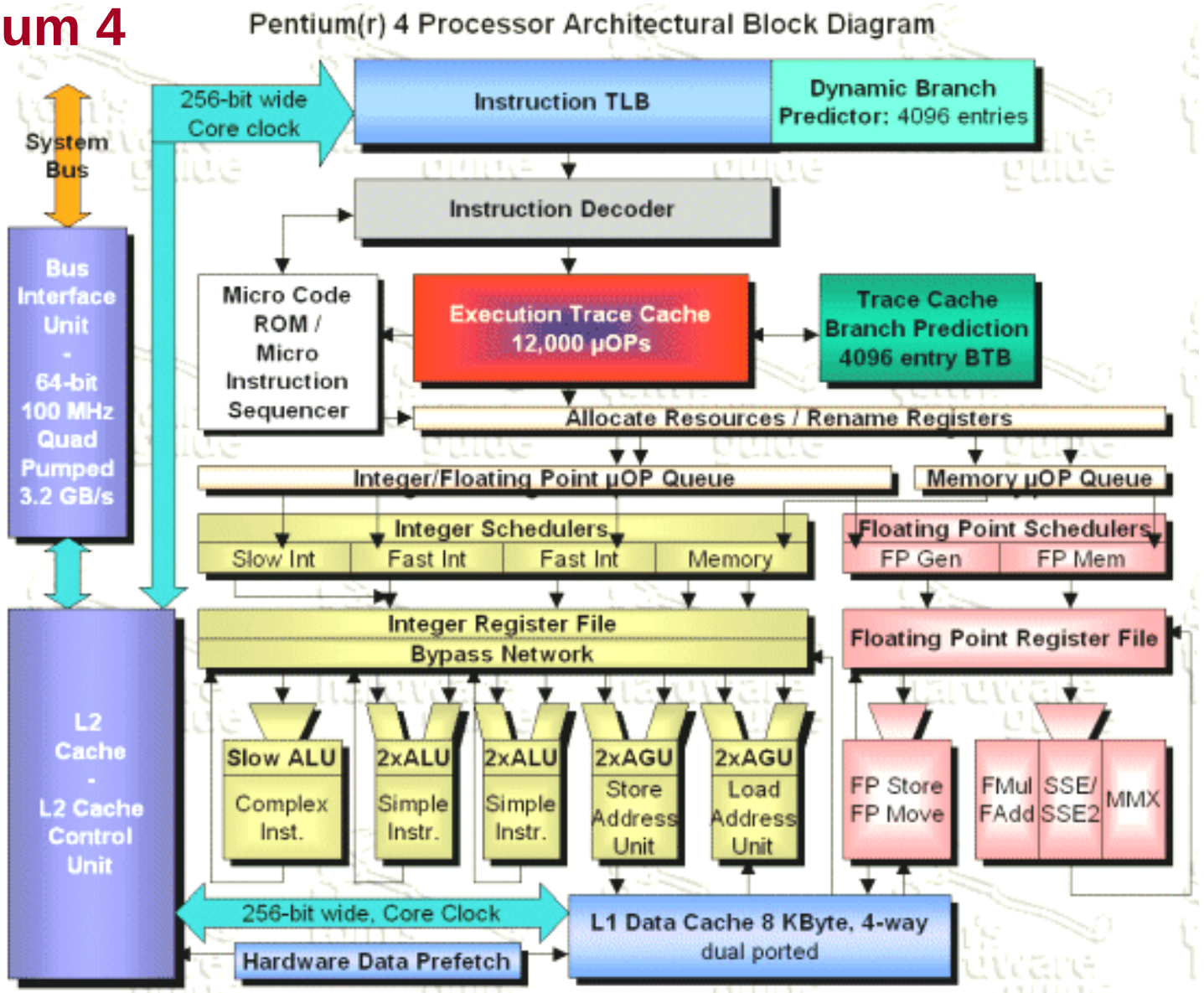
EXEMPLES D'ESTRUCTURES DE PROCESSADORS


AMD ATHLON
2-Way, 64-Kbyte Instruction Cache
24-Entry L1 TLB/256-Entry L2 TLB
Predecode Cache
Branch Prediction Table
Fetch/Decode Control
3-Way x86 Instruction Decoders
Instruction Control Unit (72-Entry)
Integer Scheduler (18-Entry)
IEU0
AGU0
IEU1
AGU1
IEU2
AGU2
FPU Stack Map / Rename
FPU Scheduler (36-Entry)
FPU Register File (88-Entry)
FADD MMX™ 3DNow!™
FMUL MMX 3DNow!
FSTORE
Load / Store Queue Unit
Bus Interface Unit
L2 Cache Controller
2-Way, 64-Kbyte Data Cache
32-Entry L1 TLB/256-Entry L2 TLB
System Interface
L2 SRAMs

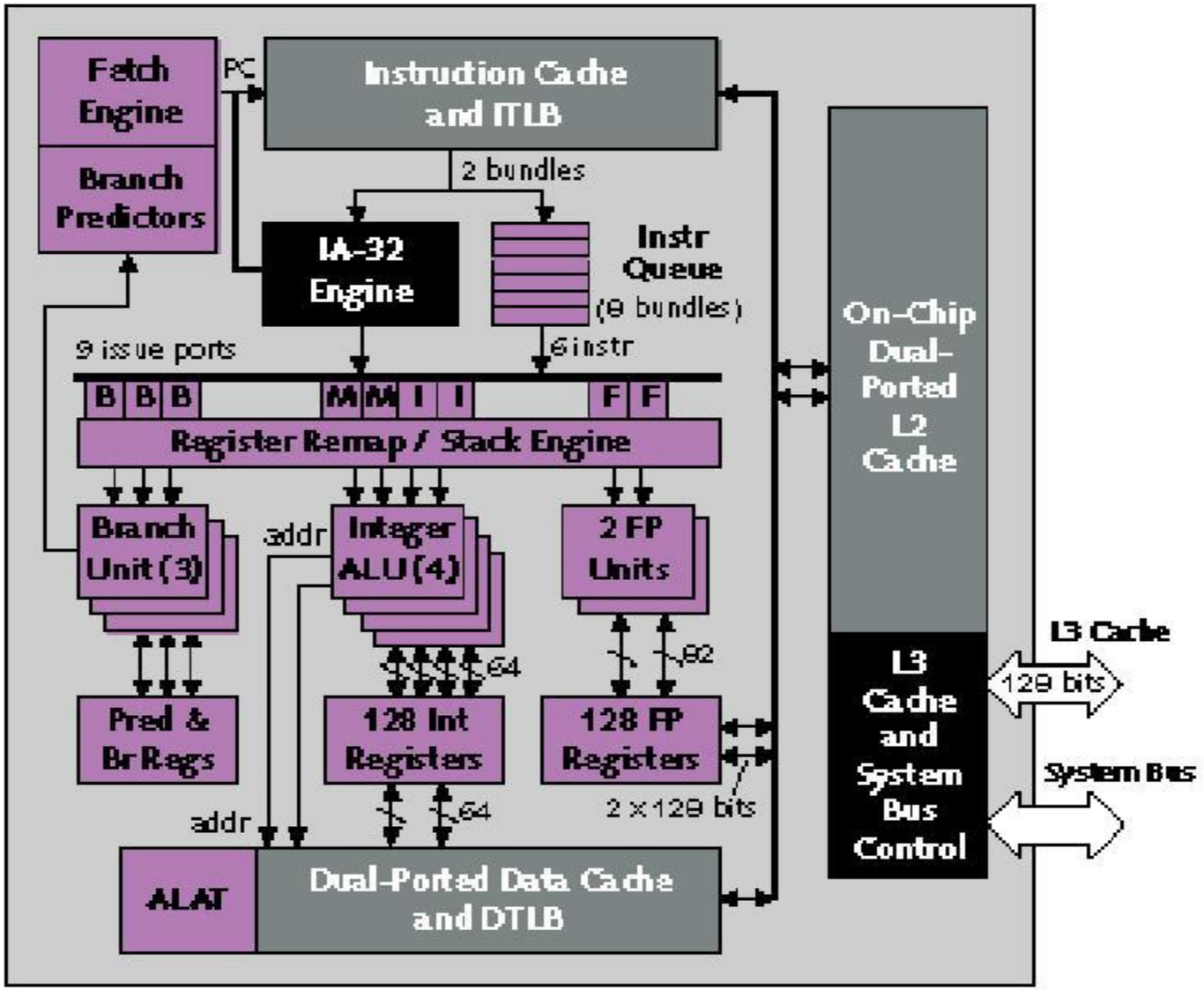
Pentium III


System Bus
Instruction Cache 16 Kbyte, 4-way
32 entry TLB
Dynamic Branch Predictor: 512 entries
Fetch/Decode Control
3 x parallel Instruction Decoder
Static Branch Predictor
Micro Code ROM / Micro Instruction Sequencer
Integer/FP Register Rename & Allocator
Architectural Register File
Bus Interface Unit 133 MHz 64-bit 1 GB/s
Reservation Station (20 Entries)
Store Data Unit
AGU Store Address Unit
AGU Load Address Unit
IEU, shift
FPU, SSE MUL DIV
MMX
IEU
SSE
MMX
L2 Cache - L2 Cache Control Unit
Memory Order Buffer
12 entry store, 16 entry load
Reorder Buffer
(40 entries)
Data Cache 16 KByte, 4-way
72 entry TLB

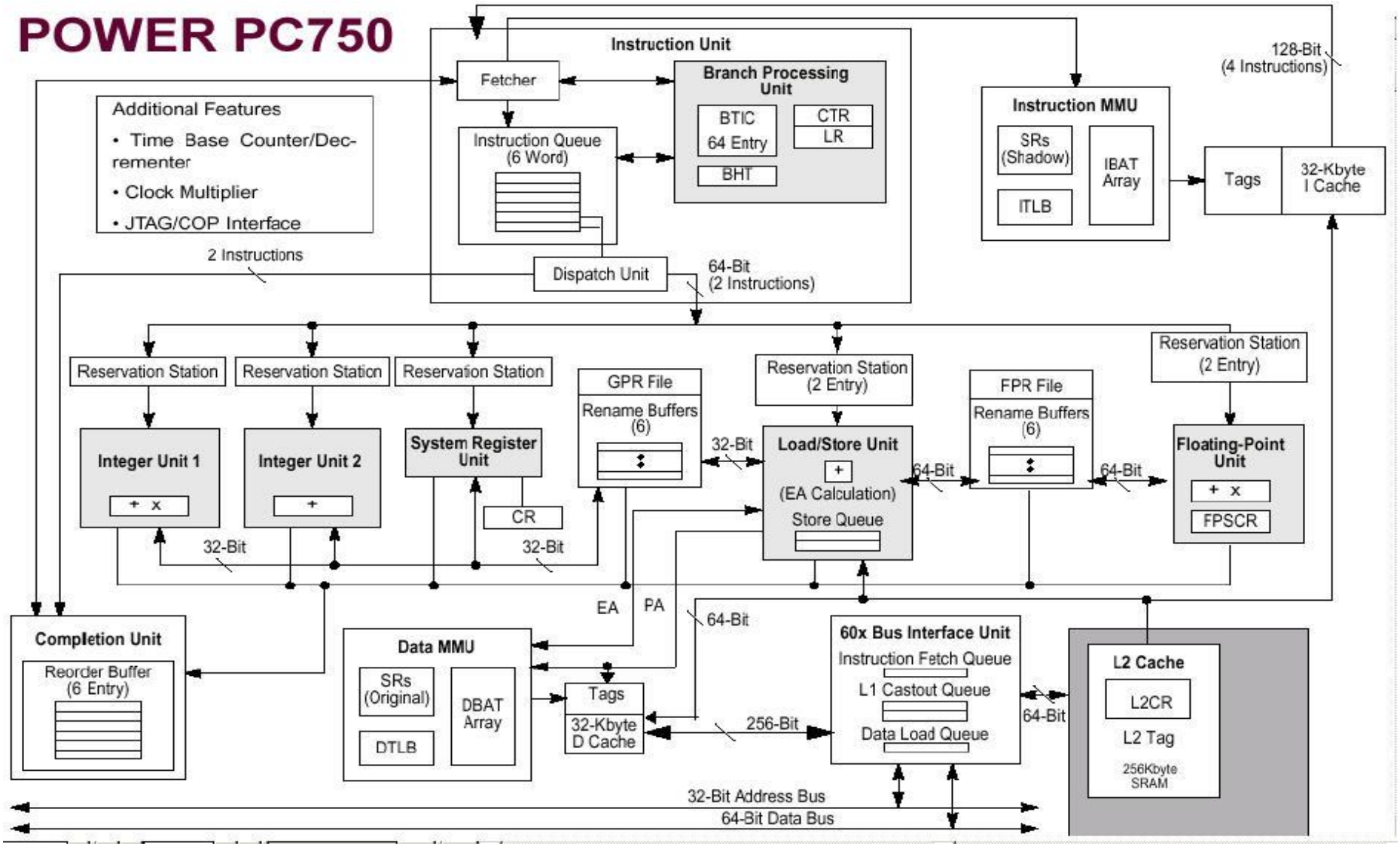
Pentium 4


Pentium(r) 4 Processor Architectural Block Diagram
System Bus
256-bit wide Core clock
Instruction TLB
Dynamic Branch Predictor: 4096 entries
Instruction Decoder
Bus Interface Unit - 64-bit 100 MHz Quad Pumped 3.2 GB/s
Micro Code ROM / Micro Instruction Sequencer
Execution Trace Cache 12,000 µOPs
Trace Cache Branch Prediction 4096 entry BTB
Allocate Resources / Rename Registers
Integer/Floating Point µOP Queue
Memory µOP Queue
Integer Schedulers
Slow Int
Fast Int
Fast Int
Memory
Floating Point Schedulers
FP Gen
FP Mem
Integer Register File
Bypass Network
Floating Point Register File
L2 Cache - L2 Cache Control Unit
Slow ALU
Complex Inst.
2xALU
Simple Instr.
2xALU
Simple Instr.
2xAGU
Store Address Unit
2xAGU
Load Address Unit
FP Store
FP Move
FMul
FAdd
SSE/ SSE2
MMX
256-bit wide, Core Clock
Hardware Data Prefetch
L1 Data Cache 8 KByte, 4-way dual ported

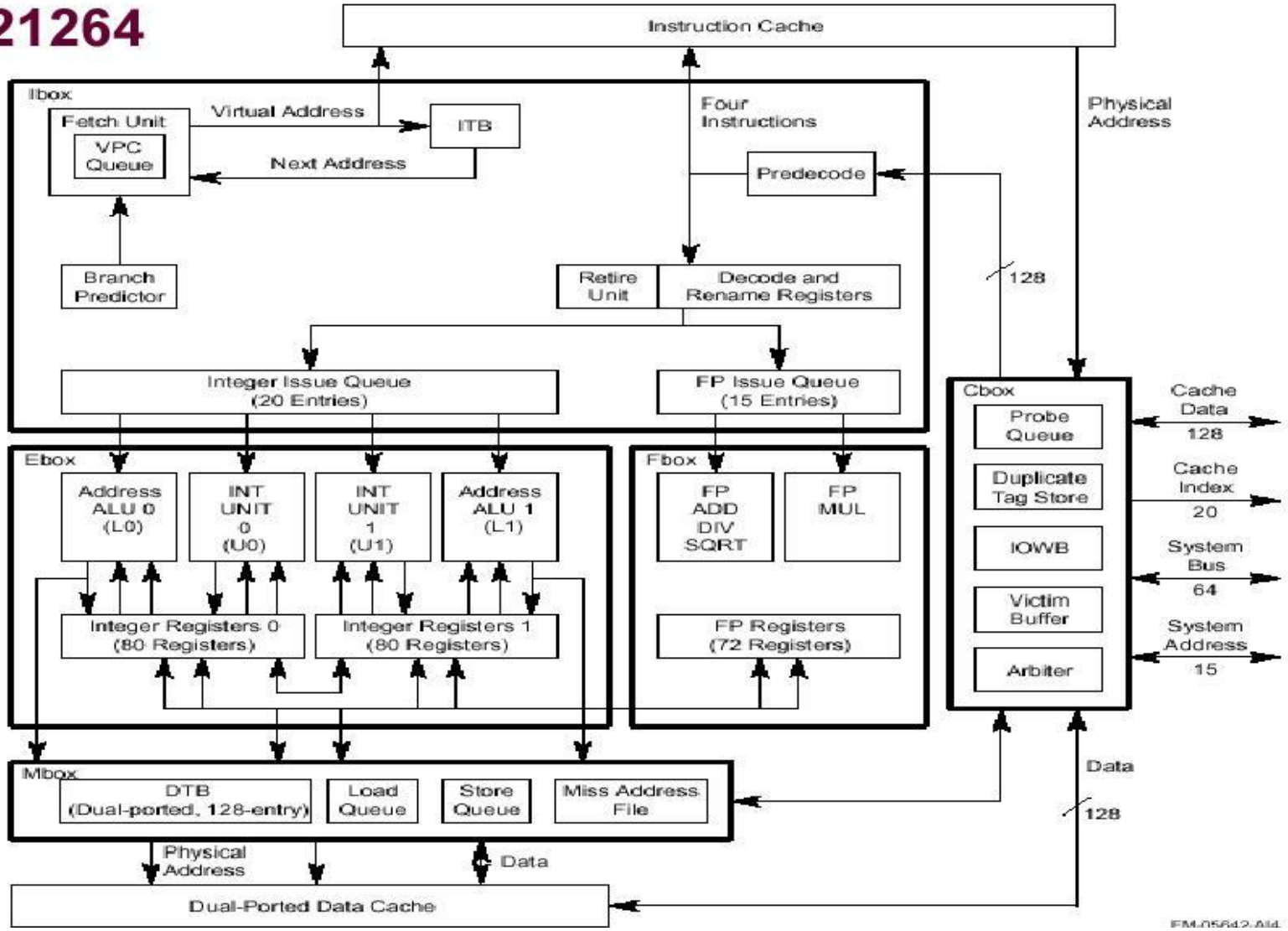
INTEL IA-64


Fetch Engine
Branch Predictors
PC
Instruction Cache and ITLB
2 bundles
IA-32 Engine
Instr Queue
(8 bundles)
6 instr
9 issue ports
B B B M M I I F F
Register Remap / Stack Engine
Branch Unit (3)
addr
Integer ALU (4)
2 FP Units
64
82
Pred & Br Regs
128 Int Registers
128 FP Registers
2 x 128 bits
addr
64
ALAT
Dual-Ported Data Cache and DTLB
On-Chip Dual-Ported L2 Cache
L3 Cache and System Bus Control
L3 Cache
128 bits
System Bus

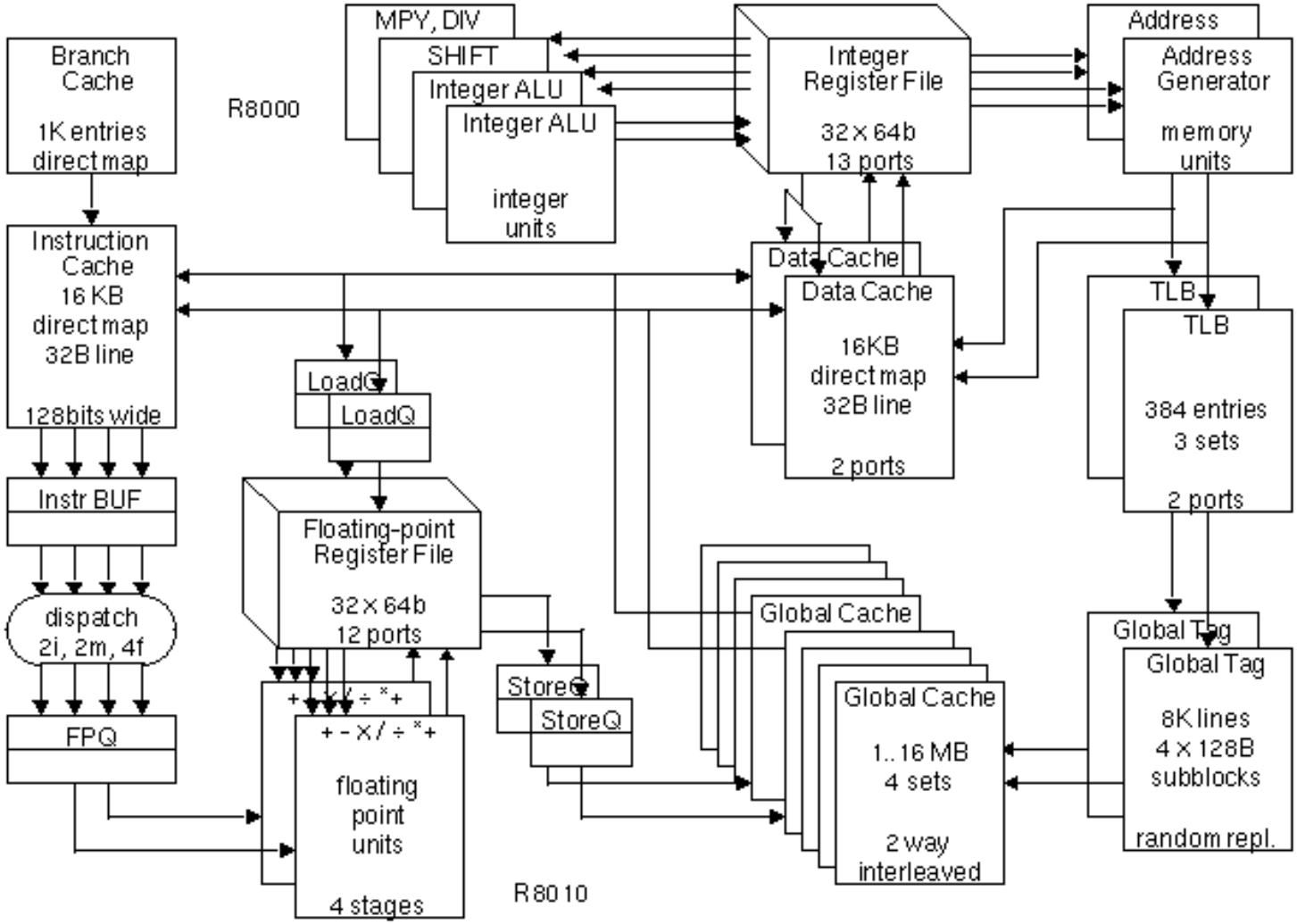
EXEMPLES D'ESTRUCTURES DE PROCESSADORS



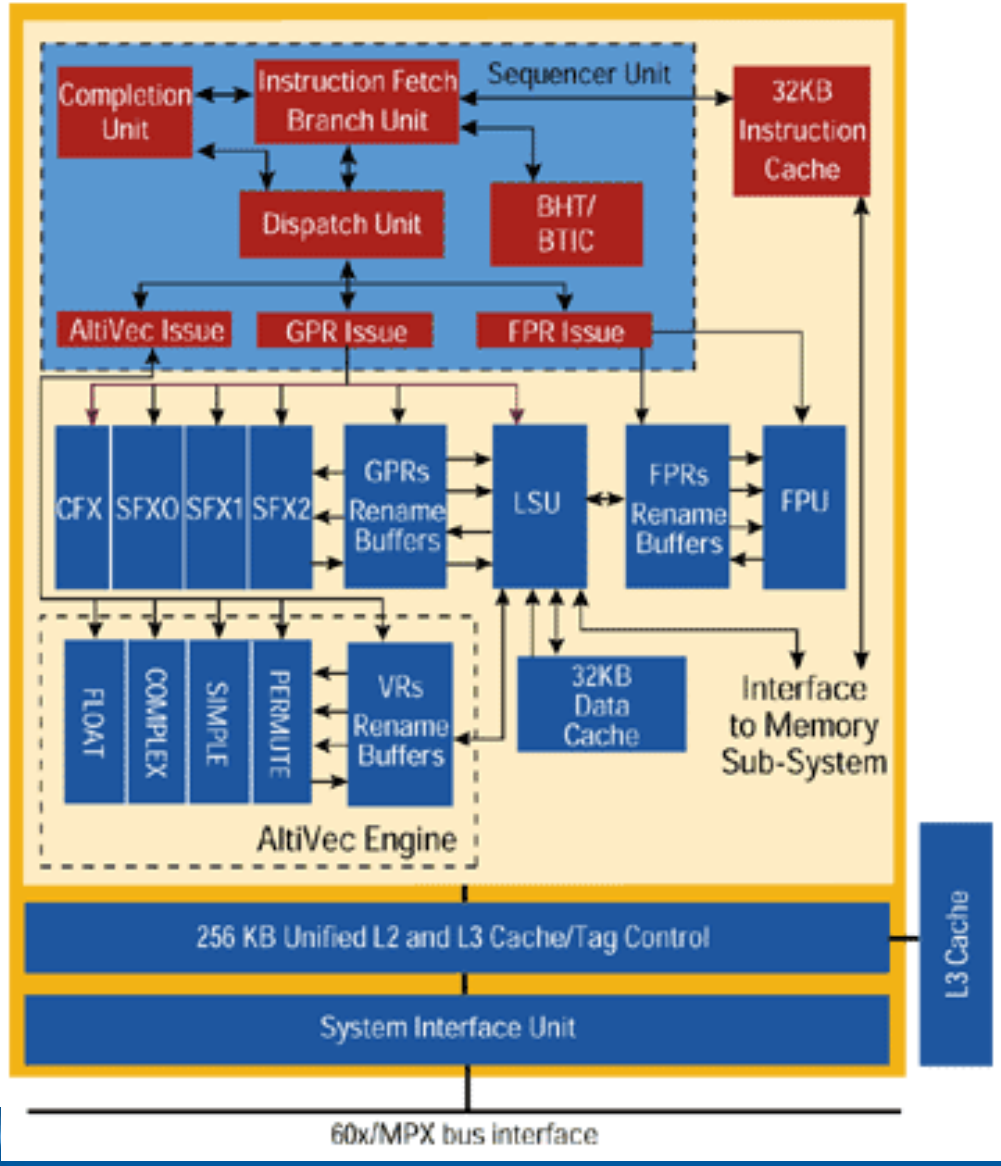
ALPHA 21264


Instruction Cache
Ibox
Fetch Unit
VPC Queue
Virtual Address
ITB
Next Address
Four Instructions
Predecode
Physical Address
Branch Predictor
Retire Unit
Decode and Rename Registers
128
Integer Issue Queue (20 Entries)
FP Issue Queue (15 Entries)
Ebox
Address ALU 0 (L0)
INT UNIT 0 (U0)
INT UNIT 1 (U1)
Address ALU 1 (L1)
Integer Registers 0 (80 Registers)
Integer Registers 1 (80 Registers)
Fbox
FP ADD DIV SQRT
FP MUL
FP Registers (72 Registers)
Cbox
Probe Queue
Duplicate Tag Store
IOWB
Victim Buffer
Arbiter
Cache Data
128
Cache Index
20
System Bus
64
System Address
15
Mbox
DTB (Dual-ported, 128-entry)
Load Queue
Store Queue
Miss Address File
Data
128
Physical Address
Data
Dual-Ported Data Cache

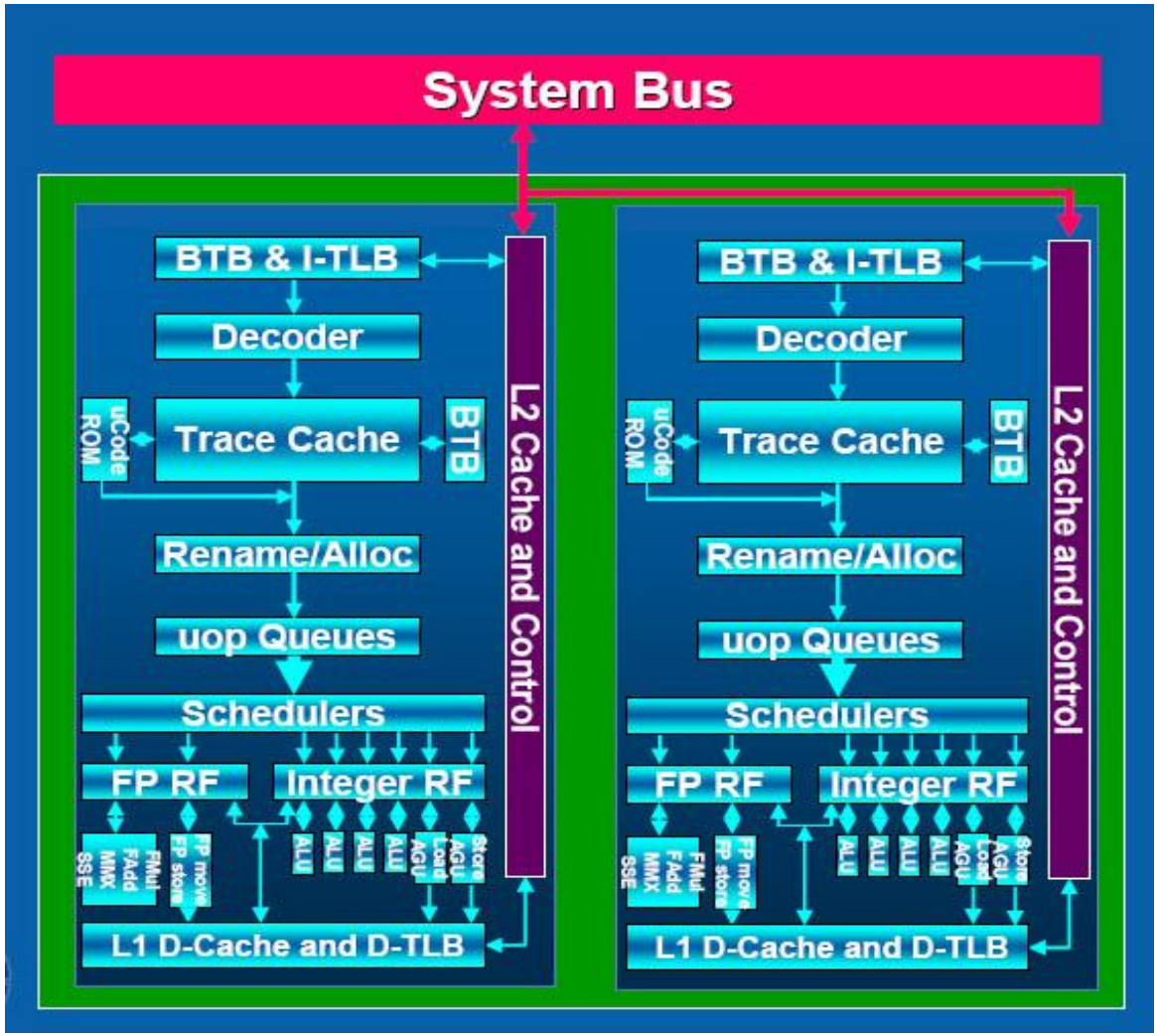
MIPS R8000


R8000
Branch Cache
1K entries direct map
Instruction Cache
16 KB
direct map
32B line
128bits wide
Instr BUF
dispatch
2i, 2m, 4f
FPQ
MPY, DIV
SHIFT
Integer ALU
Integer ALU
integer units
Integer Register File
32 x 64b
13 ports
Address
Address Generator
memory units
Data Cache
Data Cache
16KB
direct map
32B line
2 ports
TLB
TLB
384 entries
3 sets
2 ports
LoadQ
Floating-point Register File
32 x 64b
12 ports
StoreQ
+ - X / ÷ *+
floating point units
4 stages
Global Cache
Global Cache
1..16 MB
4 sets
2 way interleaved
Global Tag
Global Tag
8K lines
4 x 128B subblocks
random repl.
R8010

Motorola MPC7455 (G4)

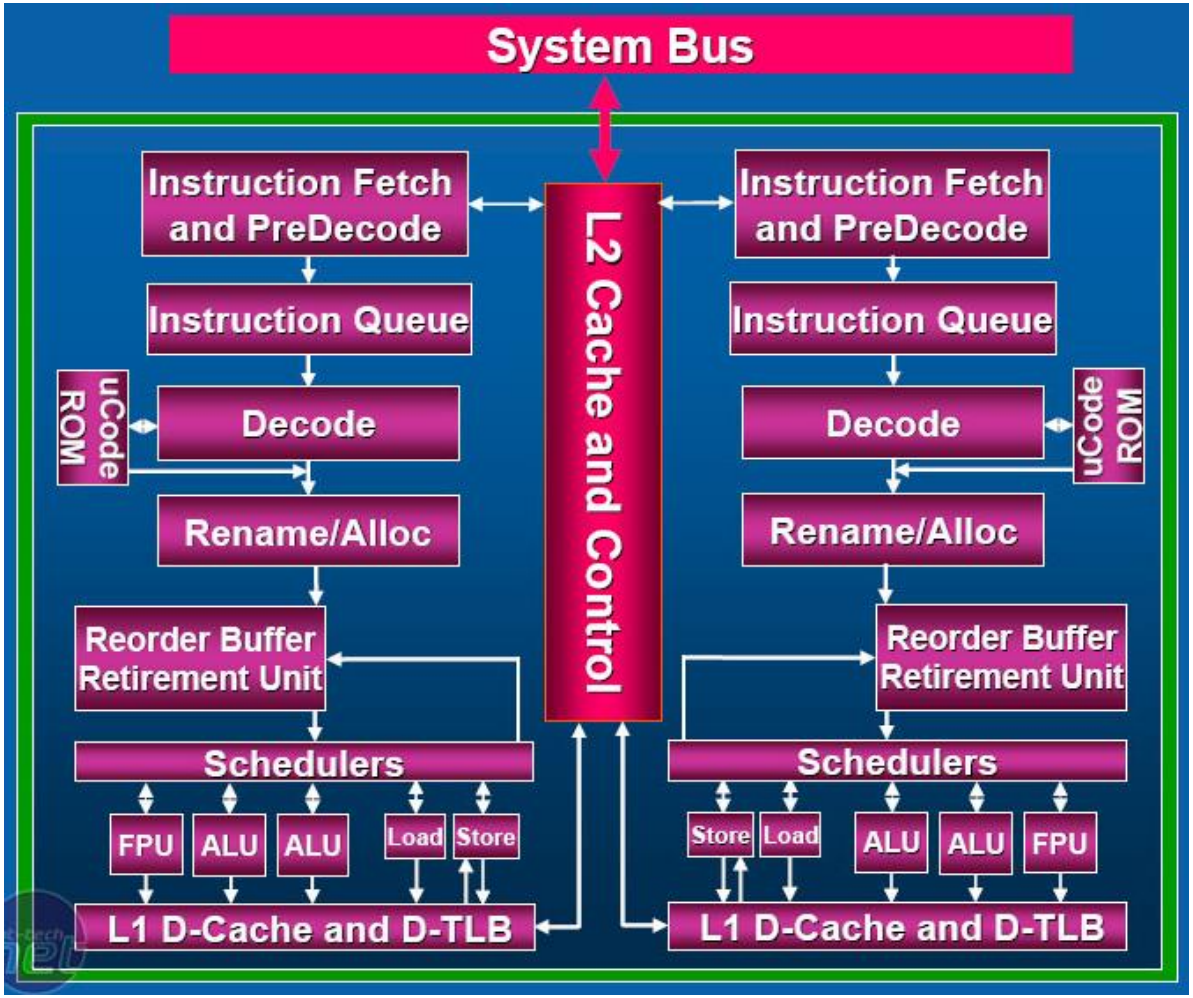


Intel Pentium D



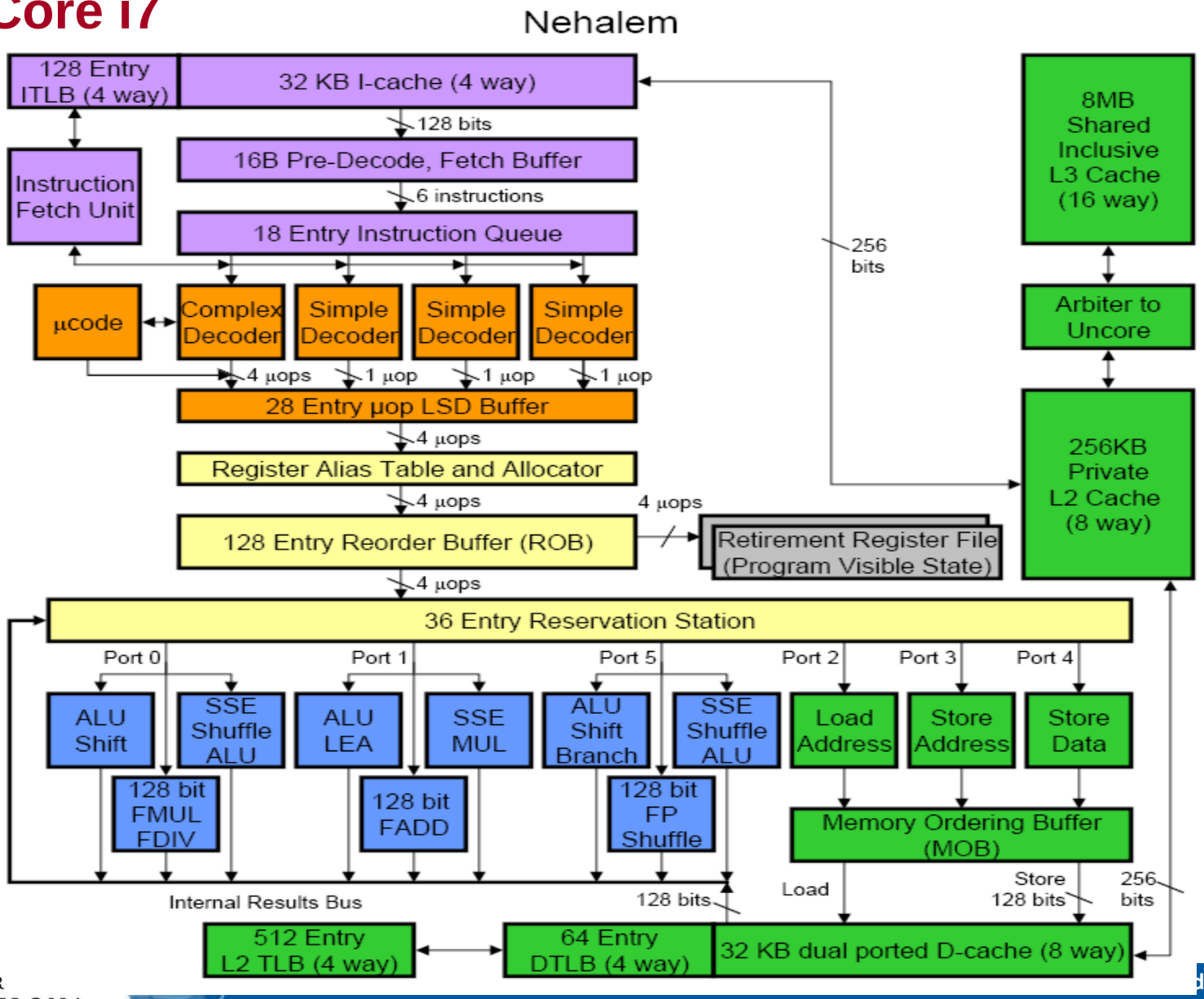
Presler

Intel Core 2

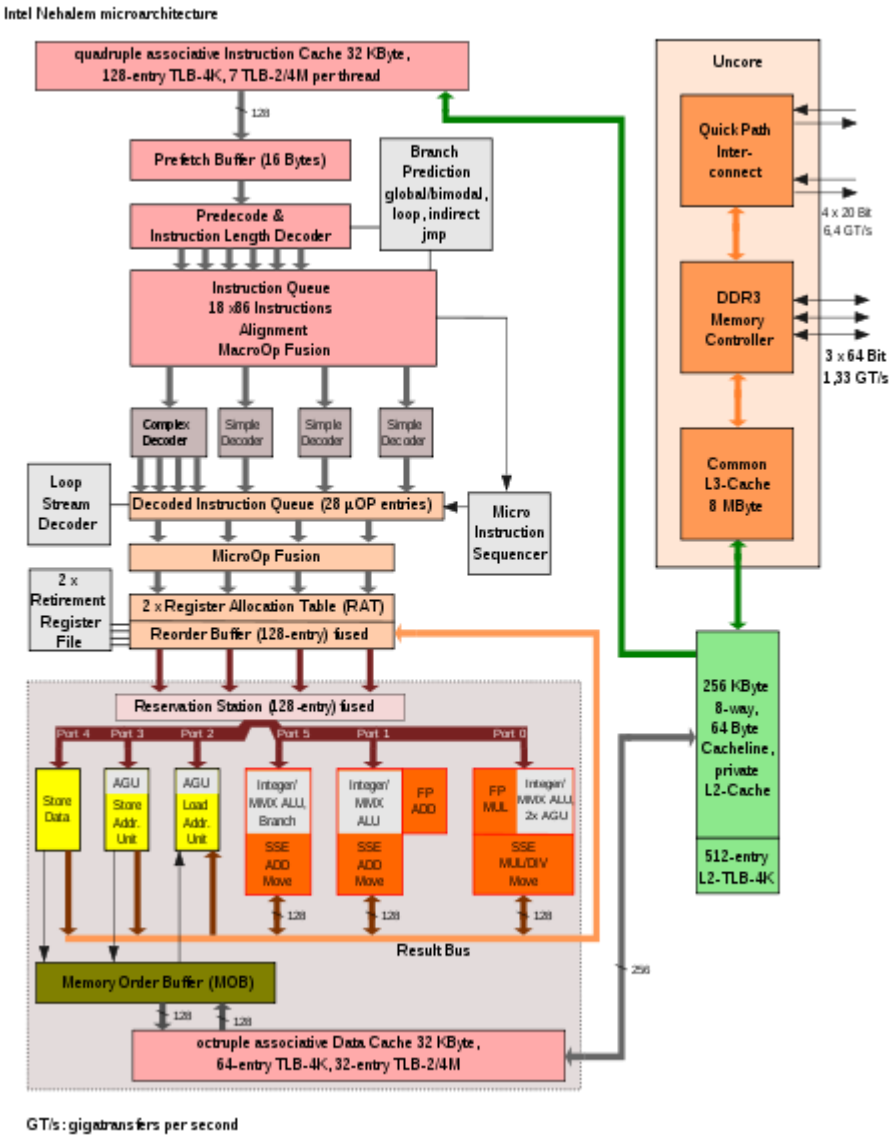


Conroe

Intel Core i7



Intel Core i7



Pentium 4 pipeline.

Explicació Bàsica de cada etapa. Indica com es processen les instruccions als processadors Pentium 4:

- **TC Nxt IP**: *Trace cache next instruction pointer*. Aquesta "etapa" mira al *Branch Target Buffer* (BTB) quina és la següent microinstrucció a executar. Aquest pas es fa en dues etapes del *pipeline*.
- **TC Fetch**: *Trace cache fetch*. Carrega de la microinstrucció de la *Trace Cache*. També són 2 etapes.
- **Drive**: Envia la microinstrucció a processar al *Resource Allocator* i al circuits de *Register Renaming*.
- **Alloc**: *Allocate*. Comprova quins recursos de laCPU necessitarà la microinstrucció.
- **Rename**: Si la instrucció fa servir un dels 8 registres estàndards x86 serà *Renamed* en 1 dels 128 registres interns que hi ha al Pentium 4. Això triga 2 etapes.
- **Que**: *Queue.* Les microinstruccions es fiquen a les cues en funció del seu tipus (per exemple, i*nteger* o *floating point*). Estaran a la cua fins que hi hagi un *slot* del mateix tipus obert al *Scheduler*.
- **Sch**: *Schedule*. Les microinstruccions són *Scheduled* per ser executades d'acord al seu tipus (*integer, floating point, etc*). Aquí es reordenen les instruccions*. Això triga 3etapes.
- **Disp**: *Dispatch*. Envia les microinstruccions a les seves corresponents UE. Això triga 2 etapes.
- **RF**: *Register file*. Es llegeixen els registres interns. Això triga 2 etapes..
- **Ex**: *Execute*. S'Executa la microinstrucció.
- **Flgs**: *Flags*. S'actualitzen els *flags* del microprocessador.
- **Br Ck**: *Branch check*. Comprova el salt pres pel programa és el mateix que el que es va predir.
- **Drive**: Envia els resultat d'aquesta comprovació al *Branch Target Buffer* (BTB) .

*Before arriving to this stage, all instructions are in order, i.e. on the same order they appear on the program. At this stage, the scheduler re-orders the instructions in order to keep all execution units full. For example, if there is one floating point unit going to be available, the scheduler will look for a floating point instruction to send it to this unit, even if the next instruction on the program is an integer one. The scheduler is the heart of the out-of-order engine of Intel 7th generation processors.