FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS

Parte 2 de 3

Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

1. **Defina atributos** de la computadora dependiendo de lo que se necesita
2. **Diseño**: diseñe de manera que se maximice desempeño y eficiencia en el uso de energía, dentro de las restricciones de costo, energía disponible, y estado tecnología. (Saber de compiladores, sistemas operativos, diseño lógico, ...)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

- Å almacenamiento operandos (RR, RM, Pila, Acumulador) Si es RR es RISC
- Å direccionamiento de memoria (por byte, alineada, tamaño dirección)
- Å modo direccionar (muchos)
- Å Tipo y tamaño de operandos
- Å Operaciones (transferencia datos, aritméticas, lógicas, de control, arit. de punto flotante)
- Å Codificación

b) organización funcional o **microarquitectura** (incluye aspectos de alto nivel del diseño, tal como el sistema de memoria, la interconexión de la memoria, y el diseño interno del procesador o CPU)

c) hardware (o implementación) debe tomar en cuenta también energía y enfriamiento

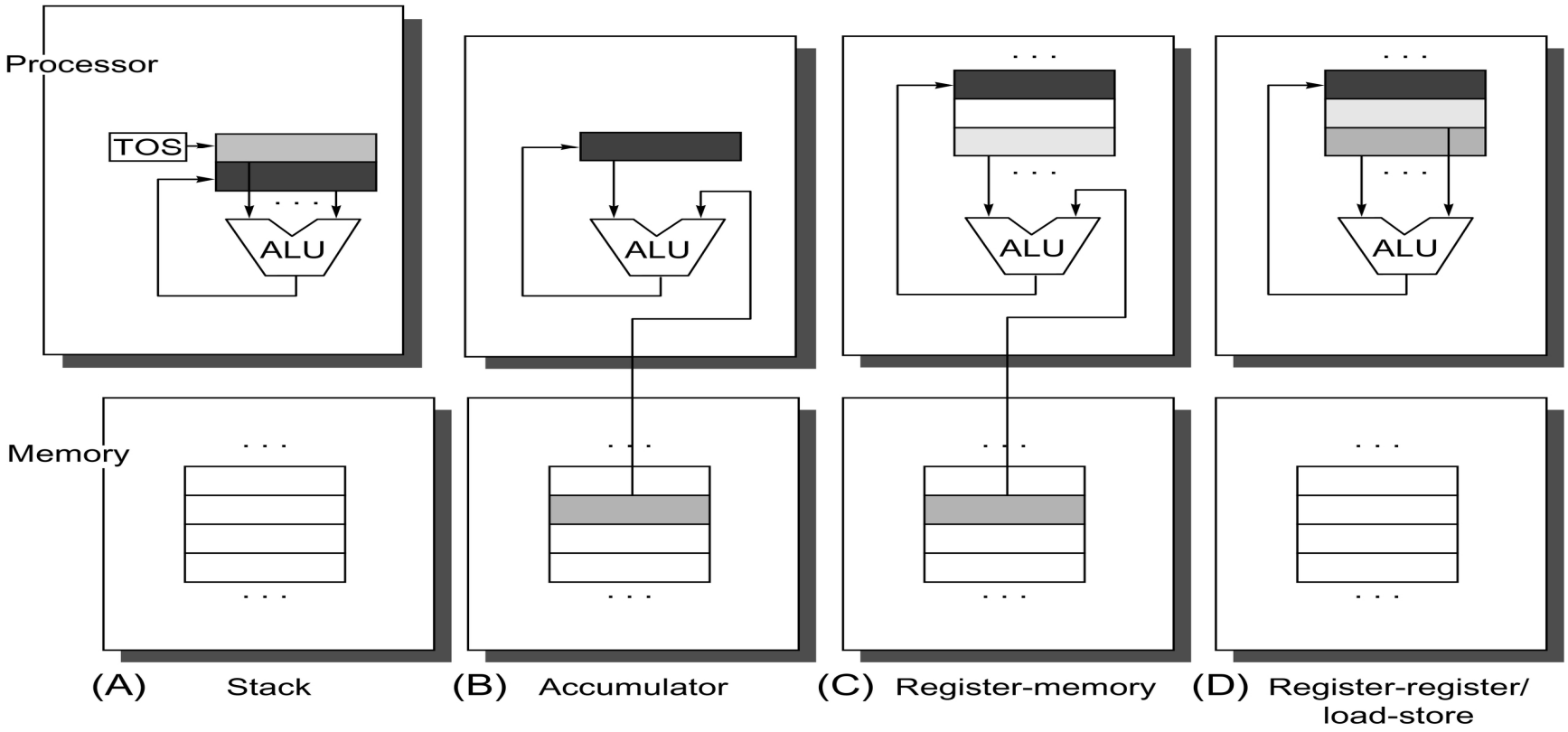
- Å diseño de circuitos (diseño lógico)
- Å empaque de circuitos

Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

Å almacenamiento operandos (RR, RM, Pila, Acumulador) Si es RR es RISC



Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

Å direccionamiento de memoria (por byte, alineada, tamaño dirección)

Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

Å modo direccionar (muchos)

En RISC-V : Registro, Inmediato y Desplazamiento

En 80 x86 : los 3 anteriores + 3 variaciones para el desplazamiento (absoluto, dos registros – base y desplazamientyo- , y dos registros en donde uno es multiplicado por el tamaño del operando en bytes. También existen Autoincremento, auto decremento

MODOS DE DIRECCIONAMIENTO Fig A.6

Addressing mode	Example instruction	Meaning	When used
Register	`Add R4,R3`	`Regs[R4]←Regs[R4] + Regs[R3]`	When a value is in a register
Immediate	`Add R4,3`	`Regs[R4]←Regs[R4]+3`	For constants
Displacement	`Add R4,100(R1)`	`Regs[R4]←Regs[R4] + Mem[100+Regs[R1]]`	Accessing local variables (+ simulates register indirect, direct addressing modes)
Register indirect	`Add R4,(R1)`	`Regs[R4]←Regs[R4] + Mem[Regs[R1]]`	Accessing using a pointer or a computed address
Indexed	`Add R3,(R1+R2)`	`Regs[R3]←Regs[R3] + Mem[Regs[R1]+Regs[R2]]`	Sometimes useful in array addressing: `R1` = base of array; `R2` = index amount
Direct or absolute	`Add R1,(1001)`	`Regs[R1]←Regs[R1] + Mem[1001]`	Sometimes useful for accessing static data; address constant may need to be large
Memory indirect	`Add R1,@(R3)`	`Regs[R1]←Regs[R1] + Mem[Mem[Regs[R3]]]`	If `R3` is the address of a pointer p, then mode yields $*p$
Autoincrement	`Add R1,(R2)+`	`Regs[R1]←Regs[R1] + Mem[Regs[R2]]` `Regs[R2]←Regs[R2]+d`	Useful for stepping through arrays within a loop. `R2` points to start of array; each reference increments `R2` by size of an element, d
Autodecrement	`Add R1, -(R2)`	`Regs[R2]←Regs[R2] - d` `Regs[R1]←Regs[R1] + Mem[Regs[R2]]`	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	`Add R1,100(R2)[R3]`	`Regs[R1]←Regs[R1] + Mem[100+Regs[R2] + Regs[R3] * d]`	Used to index arrays. May be applied to any indexed addressing mode in some computers

The load and store instructions in RISC-V

ld x1,80(x2)	Load doubleword	Regs[x1] = Mem[80+Regs[x2]]
lw x1,60(x2)	**Load word**	Regs[x1] <-$_{64}$ Mem[60+Regs[x2]]$_0$)^{32} ## Mem[60+Regs[x2]]
lwu x1,60(x2)	Load word unsigned	Regs[x1] <-$_{64}$ 0^{32} ## Mem[60+Regs[x2]]
lb x1,40(x3)	Load byte	Regs[x1] <-$_{64}$ (Mem[40+Regs[x3]]$_0$)^{56} ## Mem[40+Regs[x3]]
lbu x1,40(x3)	Load byte unsigned	Regs[x1] <-$_{64}$ 0^{56} ## Mem[40+Regs[x3]]
lh x1,40(x3)	Load half word	Regs[x1]<-$_{64}$ (Mem[40+Regs[x3]]$_0$)^{48} ## Mem[40+Regs[x3]]
flw f0,50(x3)	**Load FP single**	Regs[f0]<-$_{64}$ Mem[50+Regs[x3]] ## 0^{32}
fld f0,50(x2)	Load FP double	Regs[f0]<-$_{64}$ Mem[50+Regs[x2]]
sd x2,400(x3)	Store double	Mem[400+Regs[x3]] <-$_{64}$ Regs[x2]
sw x3,500(x4)	**Store word**	Mem[500+Regs[x4]]<-$_{32}$ Regs[x3]$_{32..63}$
fsw f0,40(x3)	**Store FP single**	Mem[40+Regs[x3]] <-$_{32}$ Regs[f0]$_{0..31}$
fsd f0,40(x3)	Store FP double	Mem[40+Regs[x3]] <- 64 Regs[f0]
sh x3,502(x2)	Store half	Mem[502+Regs[x2]] <-$_{16}$ Regs[x3]$_{48..63}$
sb x2,41(x3)	Store byte	Mem[41+Regs[x3]] <-$_{8}$ Regs[x2]$_{56..63}$

Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

Å Tipo y tamaño de operandos

ARMv8 (Advance RISC Machine), 80x86, y RISC-V soportan:

8 bits (caracter ASCII)

16 bits (carácter Unicode o media palabra)

32 bits (entero o palabra)

64 bits (doble palabra, o entero "long")

Para punto flotante (estándar 754 de la IEEE) de **32 bits (precisión simple) y 64 bits (doble)**

80x86 también soporta punto flotante de doble precisión extendida de 80 bits

Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

- Operaciones (transferencia datos, aritméticas, lógicas, de control, arit. de punto flotante)

TIPOS DE OPERACIONES

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiply, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

Figura A.12

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 12-bit displacement + contents of a GPR*
`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)
`flw, fld, fsw, fsd`	Load SP float, load DP float, store SP float, store DP float
`fmv._._x, fmv._x._`	Copy from/to integer register to/from floating-point register; "__" = S for single-precision, D for double-precision
`csrrw, csrrwi, csrrs, csrrsi, csrrc, csrrci`	Read counters and write status registers, which include counters: clock cycles, time, instructions retired
Arithmetic/logical	*Operations on integer or logical data in GPRs*
`add, addi, addw, addiw`	Add, add immediate (all immediates are 12 bits), add 32-bits only & sign-extend to 64 bits, add immediate 32-bits only
`sub, subw`	Subtract, subtract 32-bits only
`mul, mulw, mulh, mulhsu, mulhu`	Multiply, multiply 32-bits only, multiply upper half, multiply upper half signed-unsigned, multiply upper half unsigned
`div, divu, rem, remu`	Divide, divide unsigned, remainder, remainder unsigned
`divw, divuw, remw, remuw`	Divide and remainder: as previously, but divide only lower 32-bits, producing 32-bit sign-extended result
`and, andi`	And, and immediate
`or, ori, xor, xori`	Or, or immediate, exclusive or, exclusive or immediate
`lui`	Load upper immediate; loads bits 31-12 of register with immediate, then sign-extends
`auipc`	Adds immediate in bits 31–12 with zeros in lower bits to PC; used with `JALR` to transfer control to any 32-bit address
`sll, slli, srl, srli, sra, srai`	Shifts: shift left logical, right logical, right arithmetic; both variable and immediate forms
`sllw, slliw, srlw, srliw, sraw, sraiw`	Shifts: as previously, but shift lower 32-bits, producing 32-bit sign-extended result
`slt, slti, sltu, sltiu`	Set less than, set less than immediate, signed and unsigned
Control	*Conditional branches and jumps; PC-relative or through register*
`beq, bne, blt, bge, bltu, bgeu`	Branch GPR equal/not equal; less than; greater than or equal, signed and unsigned
`jal, jalr`	Jump and link: save PC + 4, target is PC-relative (`JAL`) or a register (`JALR`); if specify `x0` as destination register, then acts as a simple jump
`ecall`	Make a request to the supporting execution environment, which is usually an OS
`ebreak`	Debuggers used to cause control to be transferred back to a debugging environment
`fence, fence.i`	Synchronize threads to guarantee ordering of memory accesses; synchronize instructions and data for stores to instruction memory

Fig 1.5 Algunas instrucciones de RISC-V

Instruction type/opcode	Instruction meaning
Floating point	*FP operations on DP and SP formats*
`fadd.d`, `fadd.s`	Add DP, SP numbers
`fsub.d`, `fsub.s`	Subtract DP, SP numbers
`fmul.d`, `fmul.s`	Multiply DP, SP floating point
`fmadd.d`, `fmadd.s`, `fnmadd.d`, `fnmadd.s`	Multiply-add DP, SP numbers; negative multiply-add DP, SP numbers
`fmsub.d`, `fmsub.s`, `fnmsub.d`, `fnmsub.s`	Multiply-sub DP, SP numbers; negative multiply-sub DP, SP numbers
`fdiv.d`, `fdiv.s`	Divide DP, SP floating point
`fsqrt.d`, `fsqrt.s`	Square root DP, SP floating point
`fmax.d`, `fmax.s`, `fmin.d`, `fmin.s`	Maximum and minimum DP, SP floating point
`fcvt._._`, `fcvt._._u`, `fcvt._u._`	Convert instructions: `FCVT.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). Integers can be unsigned (`U`)
`feq._`, `flt._`,`fle._`	Floating-point compare between floating-point registers and record the Boolean result in integer register; "__" = `S` for single-precision, `D` for double-precision
`fclass.d`, `fclass.s`	Writes to integer register a 10-bit mask that indicates the class of the floating-point number ($-\infty$, $+\infty$, -0, +0, NaN, …)
`fsgnj._`, `fsgnjn._`, `fsgnjx._`	Sign-injection instructions that changes only the sign bit: copy sign bit from other source, the oppositive of sign bit of other source, XOR of the 2 sign bits

Fig 1.6 Algunas operaciones para operandos de punto flotante de RISC-V

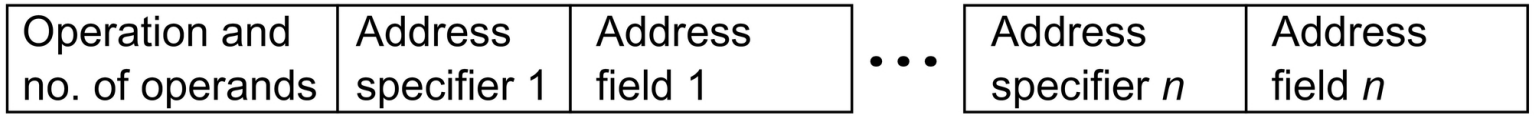
Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

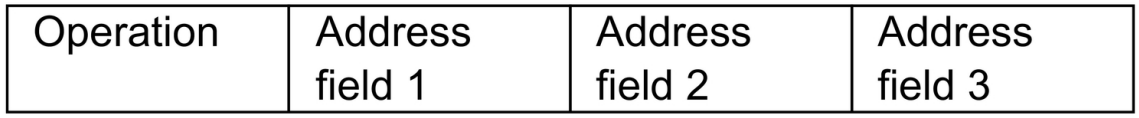
a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

- Codificación

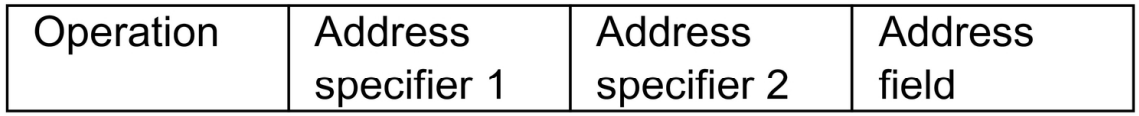
Figura A.18 ALGUNOS TIPOS DE CODIFICACIÓN

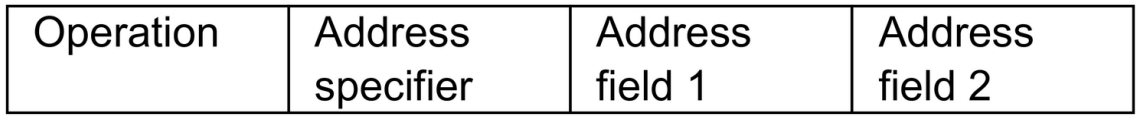


(A) Variable (e.g., Intel 80x86, VAX)



(B) Fixed (e.g., RISC V, ARM, MIPS, PowerPC, SPARC)





(C) Hybrid (e.g., RISC V Compressed (RV32IC), IBM 360/370, microMIPS, Arm Thumb2)

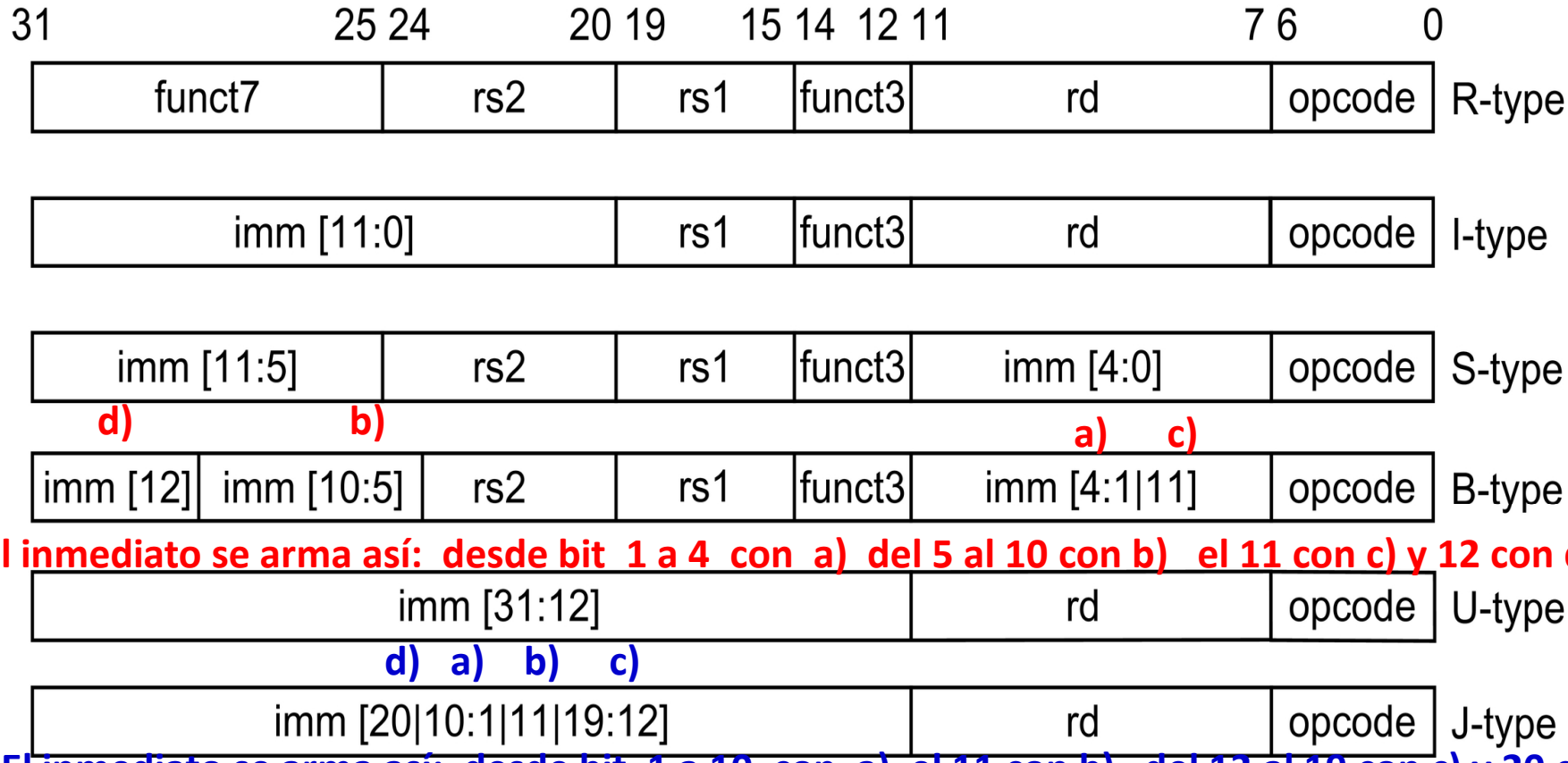


Figura 1.7 Formatos de codificación base para RISC-V Todas son de 32 bits .

Tipo **R**: Para operaciones de enteros R-R
Tipo **I:** Para Loads y operaciones con inmediato.
Tipo **B:** Para Branches
Tipo **J:** Para Jumps and link
Tipo **S:** Para Stores (eso para que los especificadores de los 3 registros estén siempre en la misma posición - (rd, rs1, rs2)
Tipo **U:** Para instrucciones con “inmediato ancho” (LUI, AUIPC)

Base Integer Subset - Base Instruction Formats

tomado de riscv-spec-v2.2

Figure 2.4 shows the immediates produced by each of the base instruction formats, and is labeled to show which instruction bit (inst[y]) produces each bit of the immediate value.

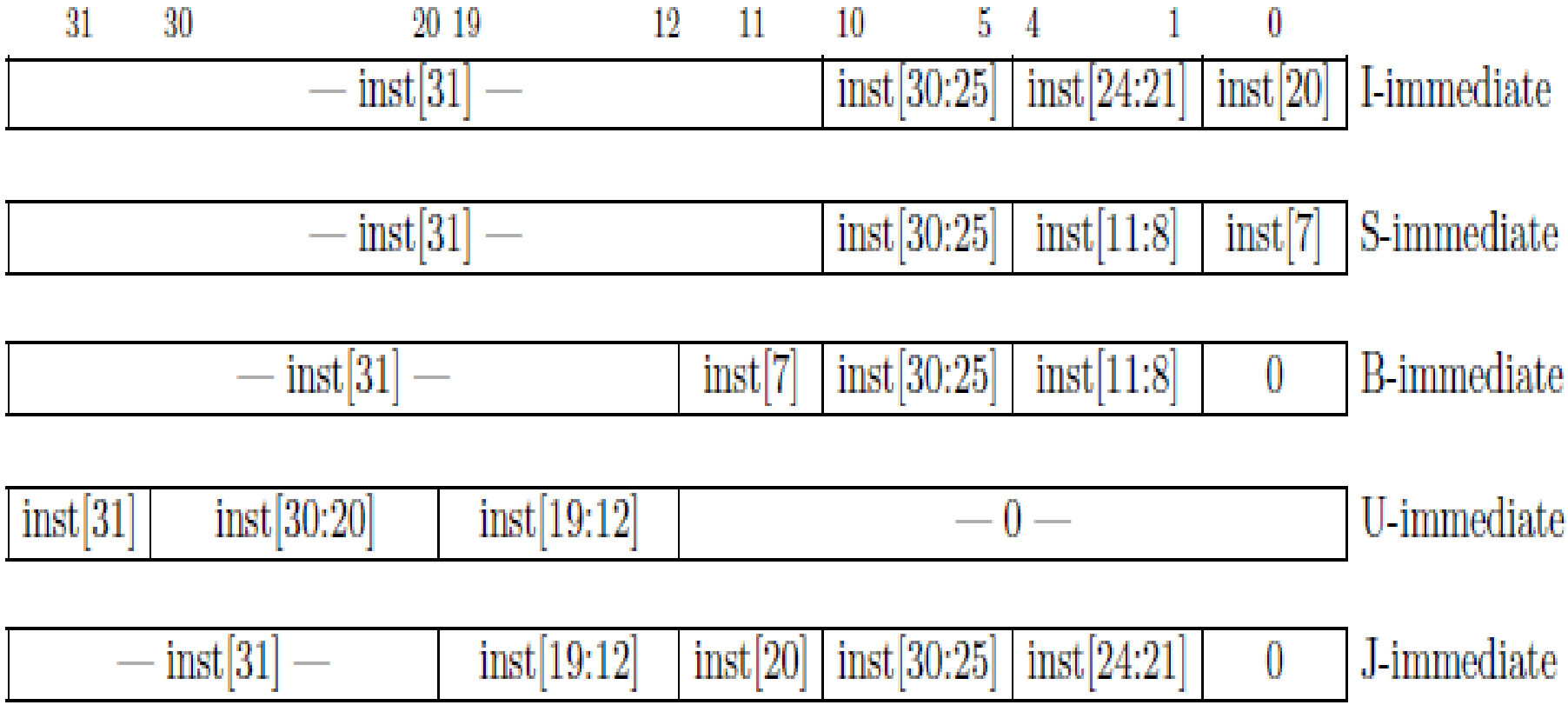


Figure 2.4: **Types of immediate produced by RISC-V instructions.** The fields are labeled with the instruction bits used to construct their value. Sign extension always uses inst[31].

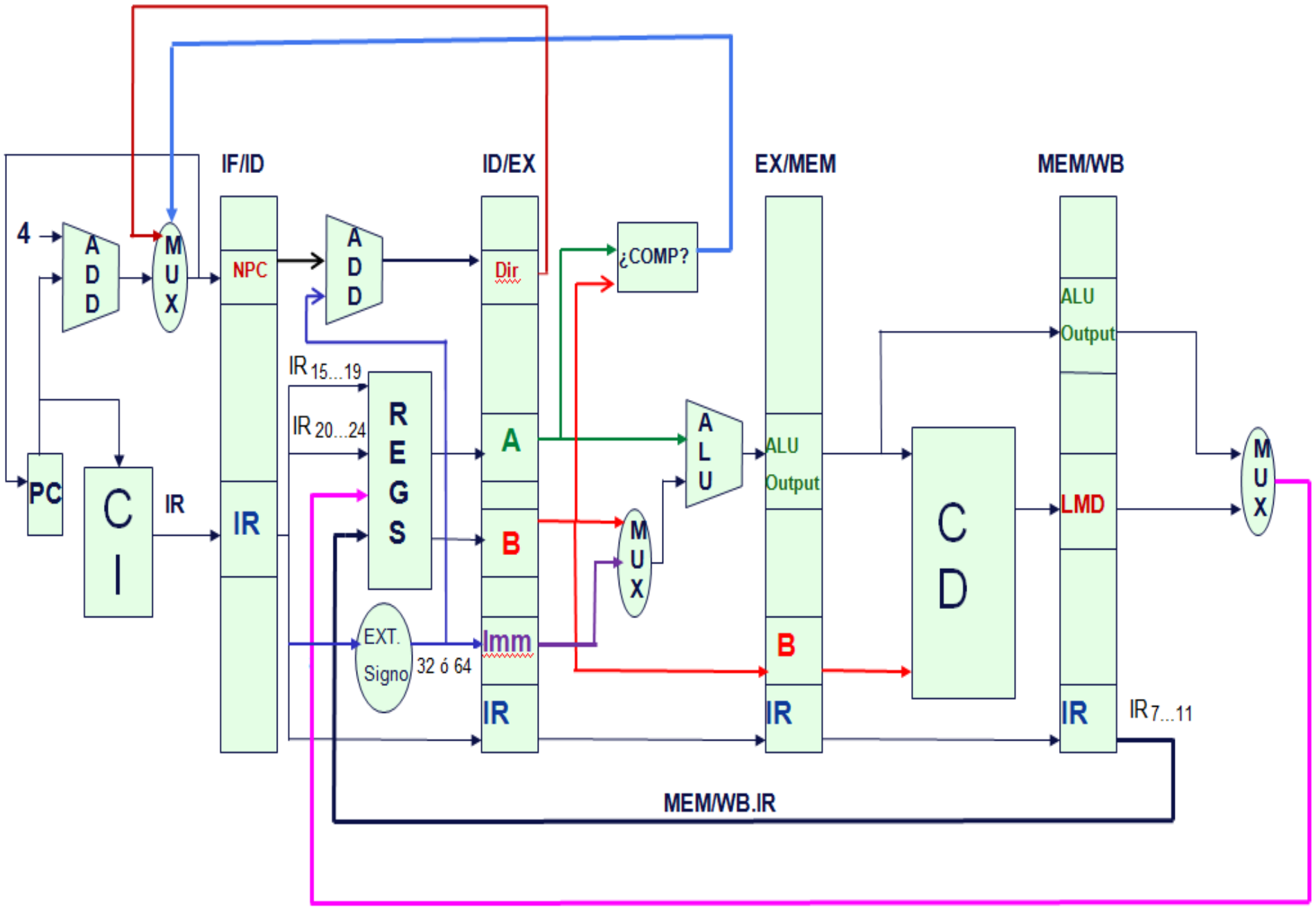
Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

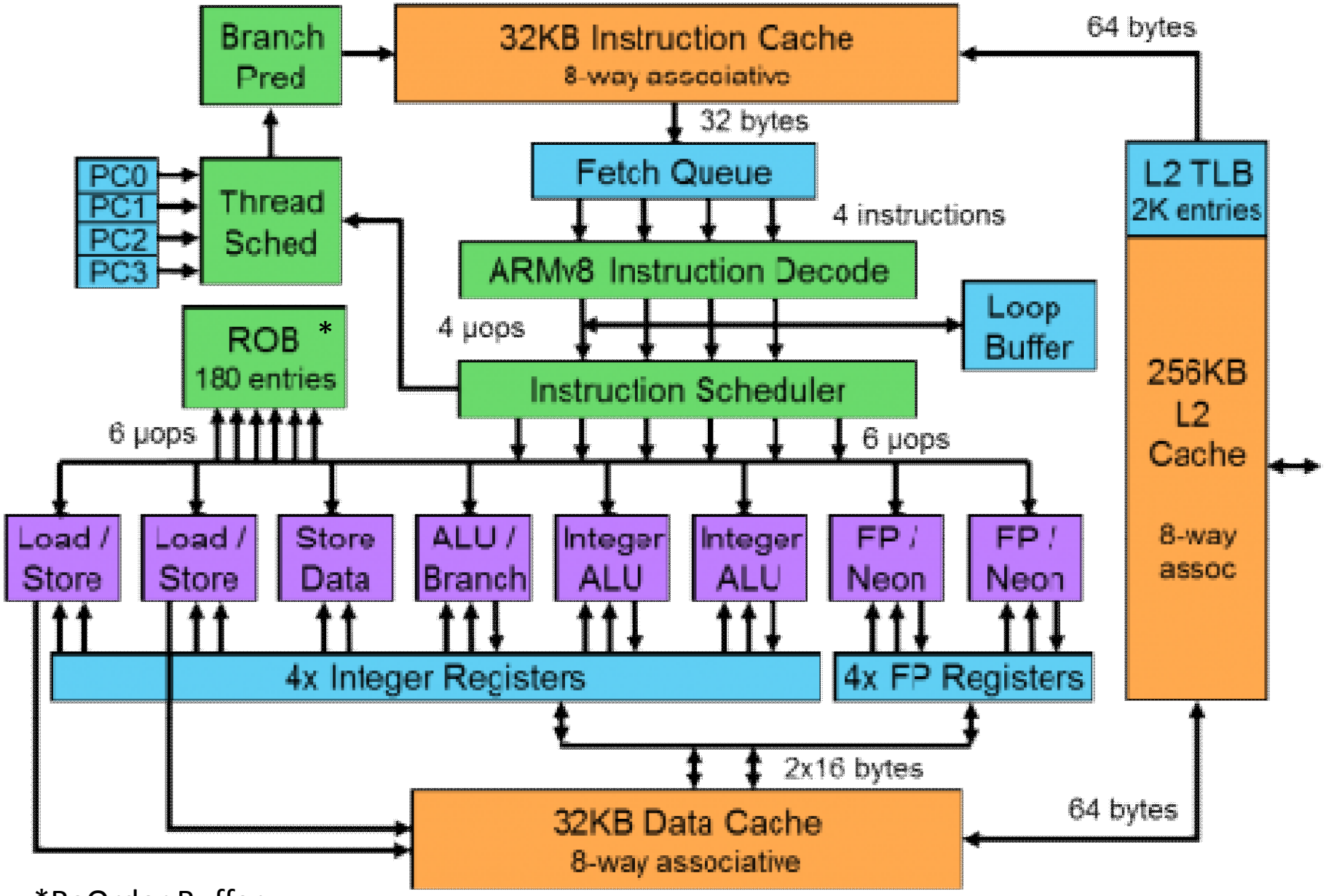
a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

b) organización funcional o **microarquitectura** (incluye aspectos de alto nivel del diseño, tal como el sistema de memoria, la interconexión de la memoria, y el diseño interno del procesador o CPU)

PIPELINE PARA ENTEROS RISC V MODIFICADO

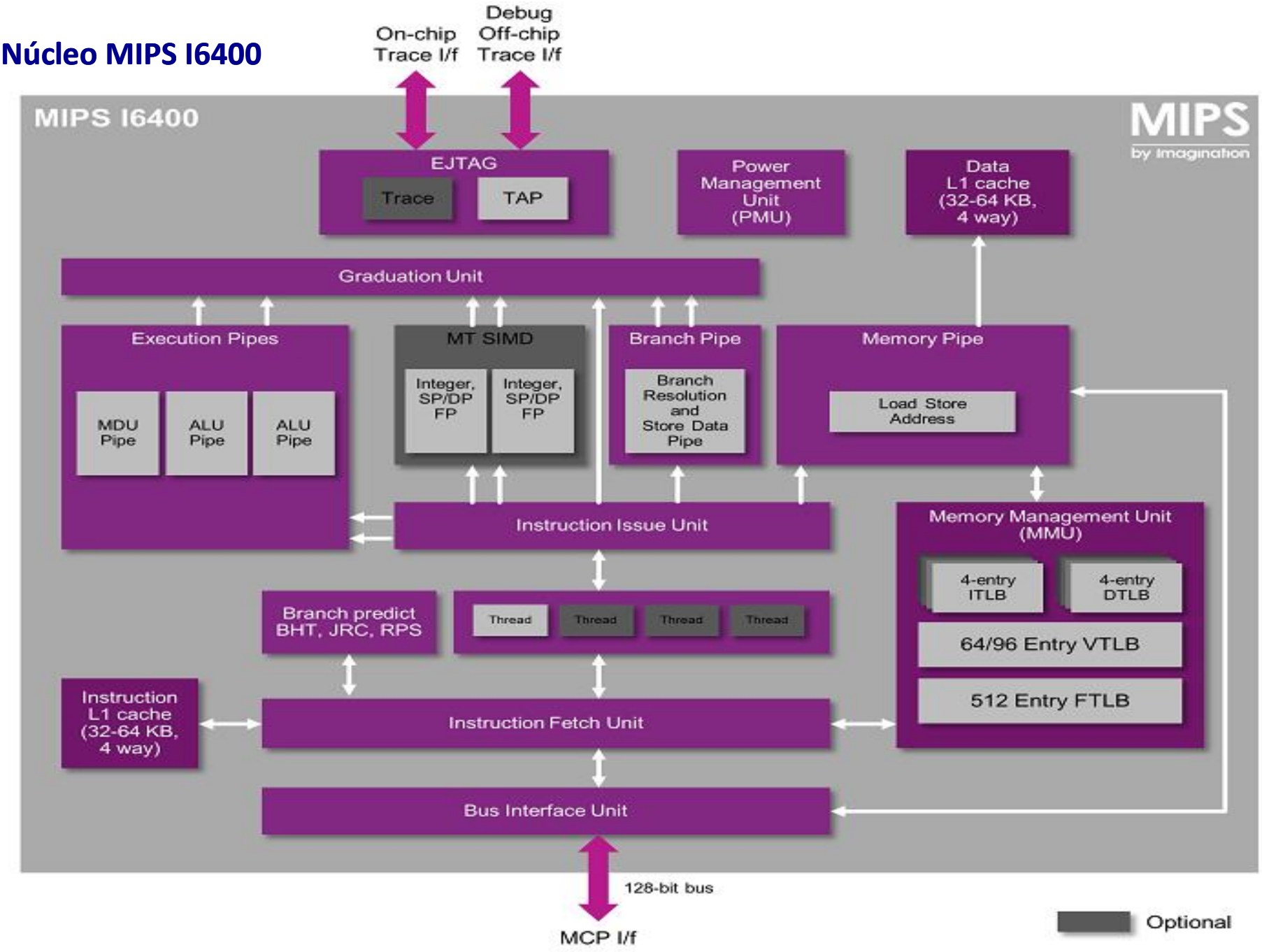


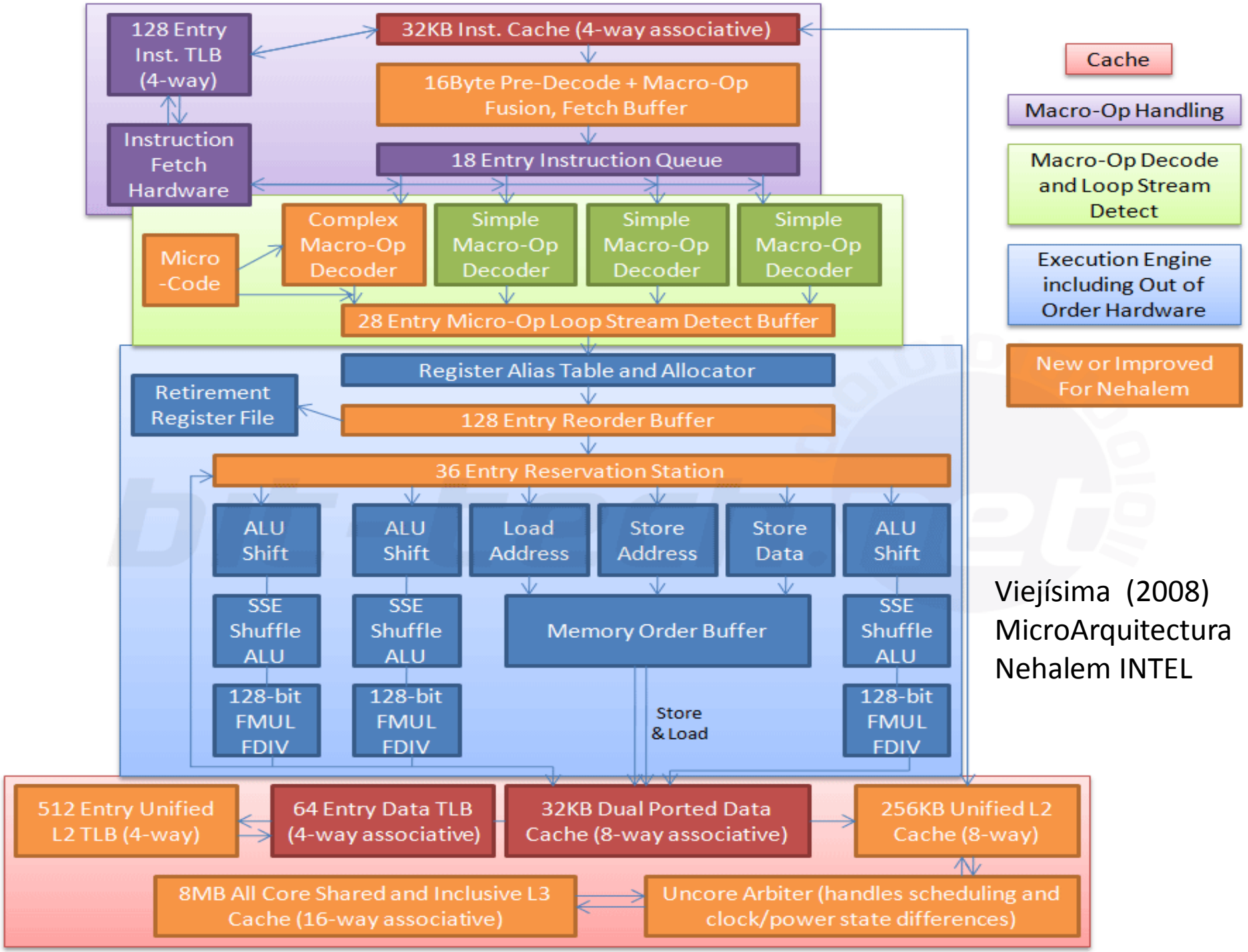
Microarquitectura de un núcleo del procesador ARM



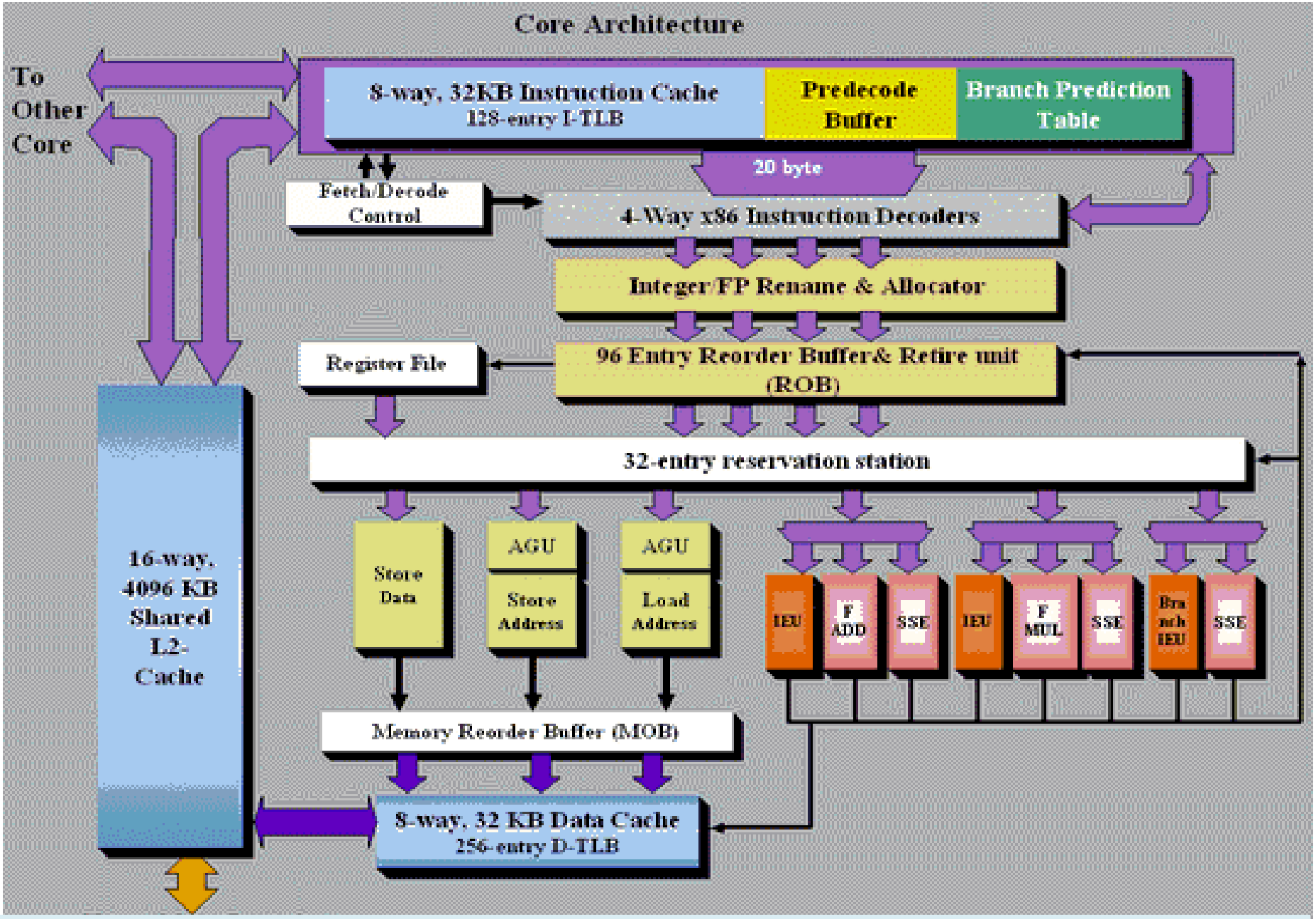
*ReOrder Buffer

Núcleo MIPS I6400





Viejísima (2008)
MicroArquitectura
Nehalem INTEL


Core Architecture
To Other Core
8-way, 32KB Instruction Cache
128-entry I-TLB
Predecode Buffer
Branch Prediction Table
20 byte
Fetch/Decode Control
4-Way x86 Instruction Decoders
Integer/FP Rename & Allocator
Register File
96 Entry Reorder Buffer& Retire unit (ROB)
32-entry reservation station
16-way, 4096 KB Shared L2-Cache
Store Data
AGU
Store Address
AGU
Load Address
IEU
F ADD
SSE
IEU
F MUL
SSE
Branch IEU
SSE
Memory Reorder Buffer (MOB)
8-way, 32 KB Data Cache
256-entry D-TLB

Definición de la Arquitectura de una Computadora (1.3)

(FUNDAMENTOS EN EL DISEÑO CUANTITATIVO DE COMPUTADORAS)

1. **Defina atributos** de la computadora dependiendo de lo que se necesita
2. **Diseño**: diseñe de manera que se maximice desempeño y eficiencia en el uso de energía, dentro de las restricciones de costo, energía disponible, y estado tecnología. (Saber de compiladores, sistemas operativos, diseño lógico, ...)

a) diseño del conjunto de instrucciones **(diseño de la arquitectura del conjunto de instrucciones: ISA)**

- Å almacenamiento operandos (RR, RM, Pila, Acumulador) Si es RR es RISC
- Å direccionamiento de memoria (por byte, alineada, tamaño dirección)
- Å modo direccionar (muchos)
- Å Tipo y tamaño de operandos
- Å Operaciones (transferencia datos, aritméticas, lógicas, de control, arit. de punto flotante)
- Å Codificación

b) organización funcional o **microarquitectura** (incluye aspectos de alto nivel del diseño, tal como el sistema de memoria, la interconexión de la memoria, y el diseño interno del procesador o CPU)

c) hardware (o implementación) debe tomar en cuenta también energía y enfriamiento

- Å diseño de circuitos (diseño lógico)
- Å empaque de circuitos