Numeration systems
Float and Double representation
CISC and RISC

1

Important quote!

- There are 10 types of people in the world: Those who understand binary and those who don't.

2

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^10	1024=1K
2^16	64K
2^20	1M
2^24	16M
2^30	1G
2^32	4G
2^36	64G
2^40	1T

To know all values

3

The basics

- Humans: typically use the decimal system for calculations (although other systems exist)
- Computers: typically use the binary system for calculations

➔ Thus, a conversion is necessary

- Additionally, computer systems use other representations such as octal or hexadecimal for the more compact representation of larger binary numbers

➔ Therefore, it is important to understand some mathematical foundations of and relations between the different numbering systems

4

Number Systems

- Hexadecimal system: we typically use the letters A to F to represent the digits with values 10 to 15
- Binary system: most important system inside a computer
- Octal and Hexadecimal systems: very simple to convert into the binary system, easier to read.

Base (b)	Number system	Alphabet
2	Binary system	0,1
8	Octal system	0,1,2,3,4,5,6,7
10	Decimal system	0,1,2,3,4,5,6,7,8,9
16	Hexadecimal system	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

5

Different numerations systems (4 bits)

Binary	Decimal (unsigned)	Signal + Module	2 complements	Hex decimal
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	0	-8	8
1001	9	-1	-7	9
1010	10	-2	-6	A
1011	11	-3	-5	B
1100	12	-4	-4	C
1101	13	-5	-3	D
1110	14	-6	-2	E
1111	15	-7	-1	F

6

Conversion between bases

Binary for decimal

Sum the powers of 2 from all digits with 1

7 6 5 43 2 1 0
01100010 = 2^6+2^5+2^1=64+32+2=98

7 6 5 43 2 1 0
01111101 = 2^6+2^5+2^4+2^3+2^2+2^0=64+32+16+8+4+1=125

Decimal to binary

40 = 32+8 = 00101000
60 = 64-4 = 63-3 = 00111111 - 00000011 = 00111100
120 = 128-8 = 127-7 = 01111111 - 00000111 = 01111000

Conversion between bases

Binary to hex

1º Join groups of 4 bits from the right to de left

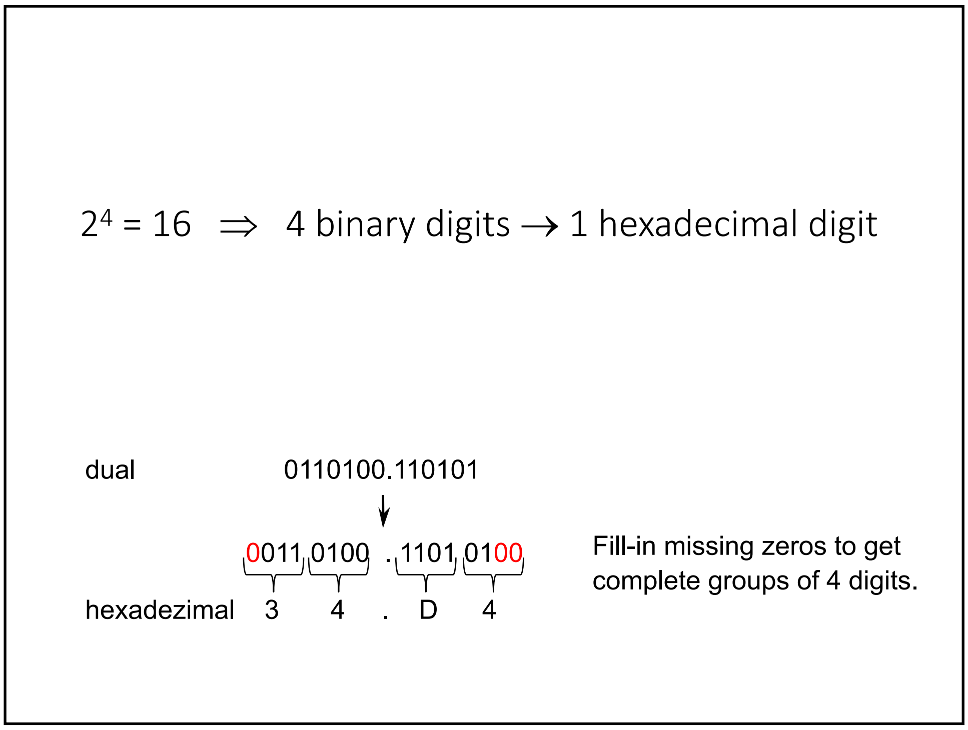
2º Convert each group of 4 bits in one single hex symbol

$\underbrace{1110}_{E}\underbrace{1110}_{E}\underbrace{0101}_{5}\underbrace{0001}_{1}\underbrace{0111}_{7}\underbrace{0101}_{5}\underbrace{1001}_{9}\underbrace{1111}_{F}$

Hex to binary

Converter each hex symbol in 4 bits

Binary	Hex decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F


$2^4 = 16 \quad \Rightarrow \quad$ 4 binary digits $\rightarrow$ 1 hexadecimal digit

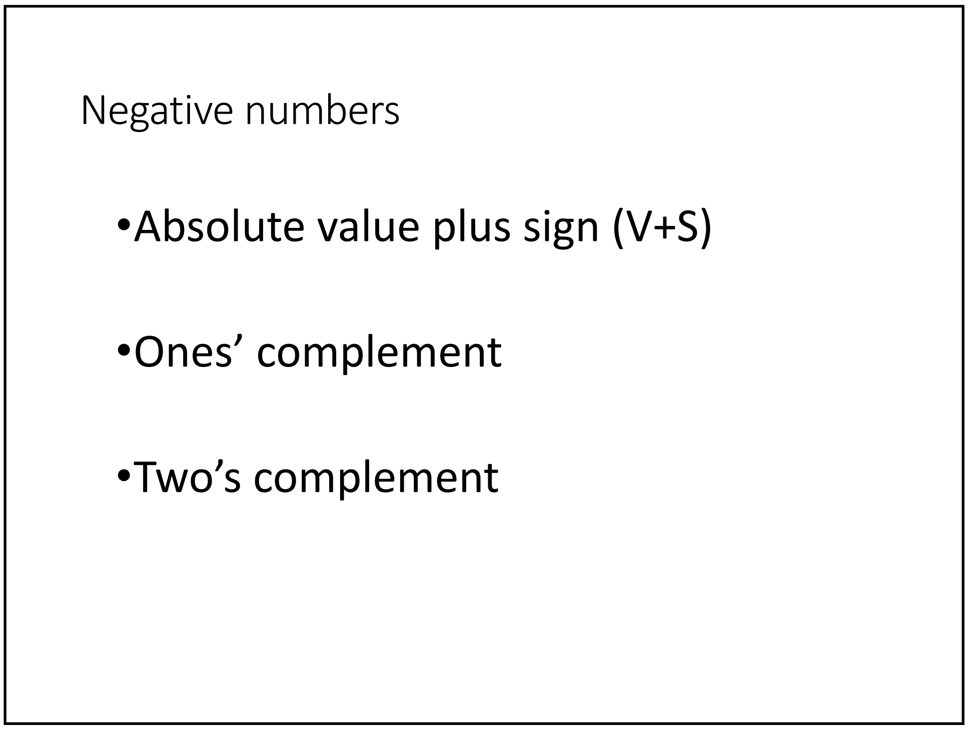
dual 0110100.110101

$$\downarrow$$

hexadezimal $\underbrace{0011}_{3}\,\underbrace{0100}_{4}\;.\;\underbrace{1101}_{D}\,\underbrace{0100}_{4}$

Fill-in missing zeros to get complete groups of 4 digits.

9


Negative numbers

- •Absolute value plus sign (V+S)
- •Ones' complement
- •Two's complement

10

Representation with absolute value plus sign (V+S)

- One digit represents the sign, typically the MSB
  - MSB = Most Significant Bit
- The leftmost bit represents the sign of a number (by convention)
  - MSB = 0 ➡ positive number
  - MSB = 1 ➡ negative number

- Example:
  - 0001 0010 = +18
  - 1001 0010 = -18

- Disadvantages:
  - Separate handling of the signs during addition and subtraction
  - There are two representations of the number 0
    - One with positive and one with negative sign (+0 and -0)

11

Ones' complement

- Flip all single bits of a binary number to get the number with a reversed sign.

- This is called a ones' complement
  - n is the number of digits, e.g. $n=4$ ➡ 4 bit numbers

- Example:

  $4_{10} = 0100_2$ ➡ $-4_{10} = 1011_{oc}$

  $-4_{10} = (2^4 - 1) - 4 = 11_{10} = 1011_2$

- Again, negative numbers have the MSB = 1

- Advantage (compared to absolute value plus sign)
  - No separate handling of the MSB during addition or subtraction
- Disadvantage
  - Still two representations of zero (0000 and 1111 for 4 bit numbers)

12

Two’s complement

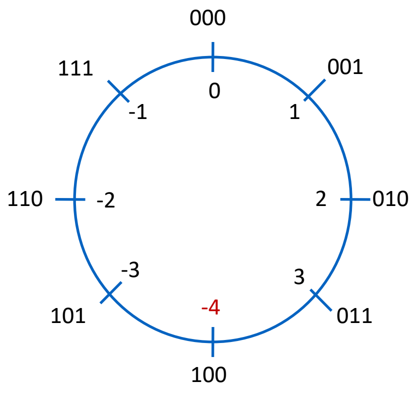
- Avoid the disadvantage by adding 1 after applying ones’ complement:
- This results in the two’s complement:
- Only one representation of the zero!

0 . . . 0 ➡ $1 \ldots 1_{oc}$
➡ $0 \ldots 0_{tc}$

13

Two’s complement

- Disadvantage:
  - Asymmetric interval of numbers that can be represented
  - The lowest number has a greater absolute value (by 1) than the highest number
- Example: 3 bit two’s complement numbers
- Again, negative numbers have the MSB = 1



14

Represent -77_{10} using 8 bits

$77_{10} = \texttt{0100 1101}_2$

Flip all the bits

Value plus sign: $-77 = \texttt{1100 1101}_2$

Ones' complement : $-77 = \texttt{1011 0010}_2$

Add 1

Two's complement : $-77 = \texttt{1011 0011}_2$

15

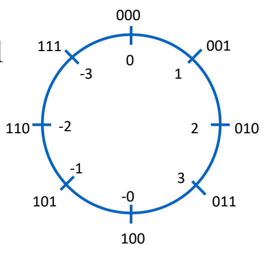
Offset binary / excess / biased representation

- Commonly used for the representation of exponents of floating point numbers (but also e.g. in signal processing as the converters are unipolar, i.e., they cannot handle negative values).
- This representation of an exponent is also called **characteristic**.
- The whole number range is shifted by adding a constant value (offset/excess/bias) so that the smallest number (largest negative value) gets the representation **0…0**.
- Assuming n digits: **Offset** = 2^{n-1}
  - Example: $n=8$ ➡ Offset 128
- The number range is **asymmetric**.

16

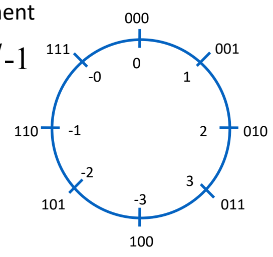
Value plus sign

$-(2^{n-1}-1) : 2^{n-1}-1$



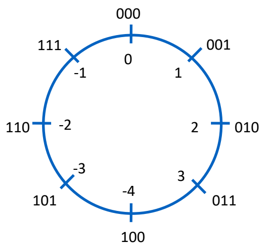
Ones' complement

$-(2^{n-1}-1) : 2^{n-1}-1$



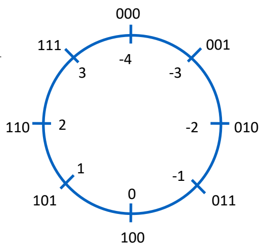
Two's complement

$-2^{n-1} : 2^{n-1}-1$



Offset

$-2^{n-1} : 2^{n-1}-1$



17

Size (bit)	Typical names	Sign	Number range (using two's complement)	
			min	max
8	char, octet, byte, modern: **int8_t** or **uint8_t**	signed	−128	127
		unsigned	0	255
16	Word, Short/short, Integer, modern: **int16_t** or **uint16_t**	signed	−32,768	32.767
		unsigned	0	65.535
32	DWord/Double Word, int, long (Windows on 16/32/64 bit systems; Unix/Linux on 16/32 bit systems), modern: **int32_t** or **uint32_t**	signed	−2,147,483,648	2,147,483,647
		unsigned	0	4,294,967,295
64	Int64, QWord/Quadword, long long, Long/long (Unix/Linux on 64 bit systems), modern: **int64_t** or **uint64_t**	signed	−9,223,372,036,854, 775,808	9,223,372,036,854,775,807
		unsigned	0	18,446,744,073,709,551,615
128	Int128, Octaword, Double Quadword	signed	$\approx -1.70141 \cdot 10^{38}$	$\approx 1.70141 \cdot 10^{38}$
		unsigned	0	$\approx 3.40282 \cdot 10^{38}$

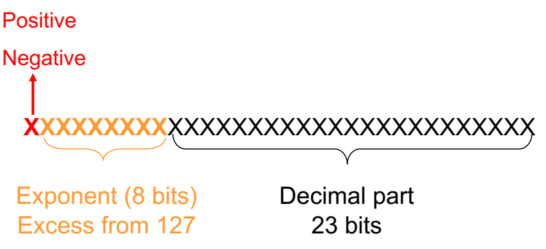
18

Floating point type	Memory requirement	Range
Float	4 bytes	±3.40282347E+38F i.e. 6-7 significant digits
Double	8 bytes	±1.79769313486231570E+308 i.e. 15-16 significant digits

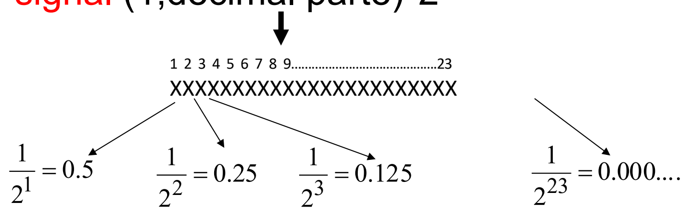
19

Floating numbers representation

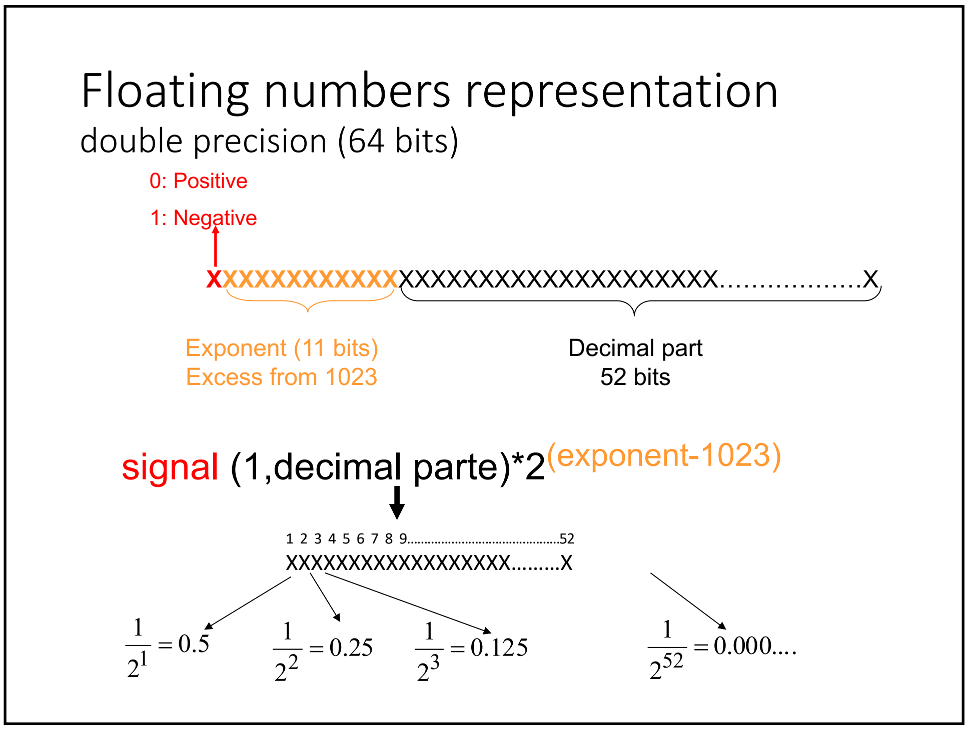
simple precision (**32 bits**)



signal (1,decimal parte)*2$^{(exponent-127)}$



20

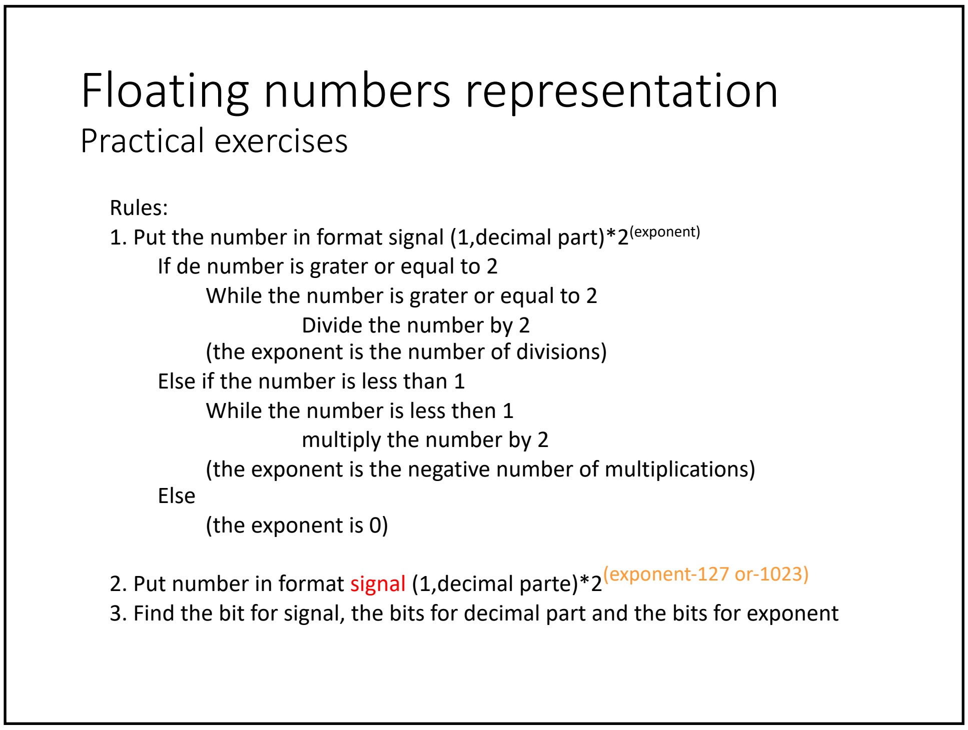


Floating numbers representation
double precision (64 bits)

signal (1,decimal parte)*$2^{(exponent-1023)}$

$\frac{1}{2^1} = 0.5$ $\frac{1}{2^2} = 0.25$ $\frac{1}{2^3} = 0.125$ $\frac{1}{2^{52}} = 0.000....$

21



Floating numbers representation
Practical exercises

```
Rules:
1. Put the number in format signal (1,decimal part)*2^(exponent)
    If de number is grater or equal to 2
        While the number is grater or equal to 2
                Divide the number by 2
        (the exponent is the number of divisions)
    Else if the number is less than 1
        While the number is less then 1
                multiply the number by 2
        (the exponent is the negative number of multiplications)
    Else
        (the exponent is 0)
```

2. Put number in format signal (1,decimal parte)*$2^{(exponent-127 \text{ or} -1023)}$
3. Find the bit for signal, the bits for decimal part and the bits for exponent

22

Floating numbers representation
Practical exercises

Represent in single precision 3,5

Since the number is greater or equal to 2, we must divide the number until we get a number less then 2.

3,5/2=1,75

3,5= 1,75*2^1

3,5=1,75*$2^{(\mathbf{128}-127)}$

Signal 0

Decimal part 0,75=0,5+0,25. Bit sequence: 11000000000000000000000

Exponent=128. Bit sequence:10000000

Final result: 0 10000000 11000000000000000000000

Final result in hex: 40600000H

23

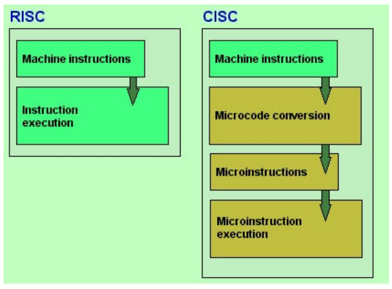
Exercices

Please represente in **Float** e **Double** formats the follwing decimal numbers. Clearly identify the signal bit, exponent and decimal part.

a) 2
b) -2
c) 4
d) 6.5
e) 10

24

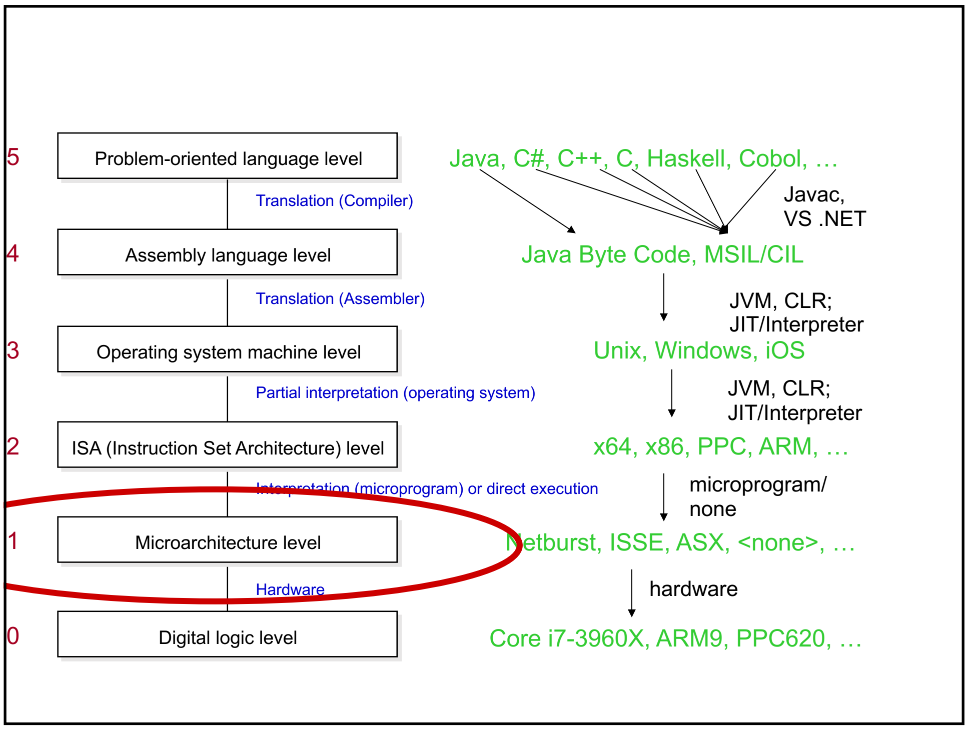
CISC (Complex Instruction Set Computer) vs *RISC (Reduced Instruction Set Computer)*.



25

RISC	CISC
1. RISC stands for Reduced Instruction Set Computer.	1. CISC stands for Complex Instruction Set Computer.
2. RISC processors have simple instructions taking about one clock cycle.	2. CSIC processor has complex instructions that take up multiple clocks for execution.
3. Performance is optimized with more focus on software	3. Performance is optimized with more focus on hardware.
4. It has no memory unit and uses a separate hardware to implement instructions..	4. It has a memory unit to implement complex instructions.
5. The instruction set is reduced i.e. it has only a few instructions in the instruction set. Many of these instructions are very primitive.	5. The instruction set has a variety of different instructions that can be used for complex operations.
6. Complex addressing modes are synthesized using the software.	6. CISC already supports complex addressing modes
7. Multiple register sets are present	7. Only has a single register set
8. Execution time is very less	8. Execution time is very high
9. Decoding of instructions is simple.	9. Decoding of instructions is complex
10. It does not require external memory for calculations	10. It requires external memory for calculations
11. The most common RISC microprocessors are Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.	11. Examples of CISC processors are the System/360, VAX, PDP-11, Motorola 68000 family, AMD and Intel x86 CPUs.
12. RISC architecture is used in high-end applications such as video processing, telecommunications and image processing.	12. CISC architecture is used in low-end applications such as security systems, home automation, etc.

26



5 Problem-oriented language level — Java, C#, C++, C, Haskell, Cobol, …

Translation (Compiler) — Javac, VS .NET

4 Assembly language level — Java Byte Code, MSIL/CIL

Translation (Assembler) — JVM, CLR; JIT/Interpreter

3 Operating system machine level — Unix, Windows, iOS

Partial interpretation (operating system) — JVM, CLR; JIT/Interpreter

2 ISA (Instruction Set Architecture) level — x64, x86, PPC, ARM, …

Interpretation (microprogram) or direct execution — microprogram/ none

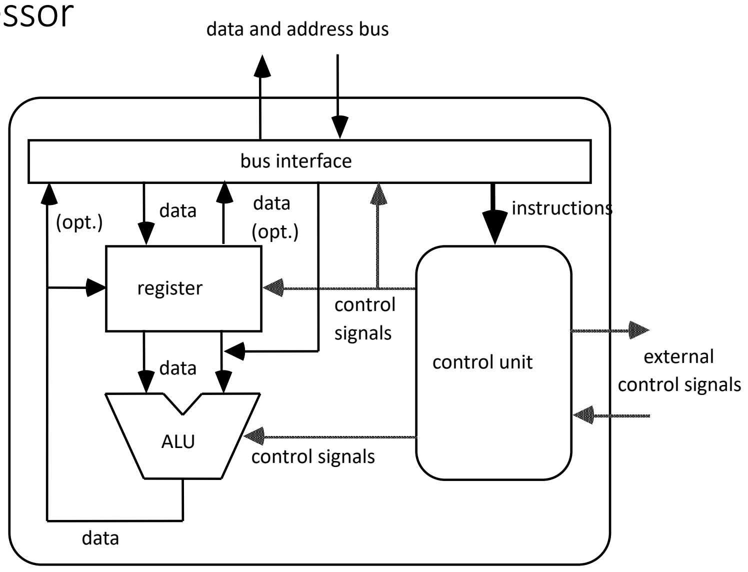
1 Microarchitecture level — Netburst, ISSE, ASX, <none>, …

Hardware — hardware

0 Digital logic level — Core i7-3960X, ARM9, PPC620, …

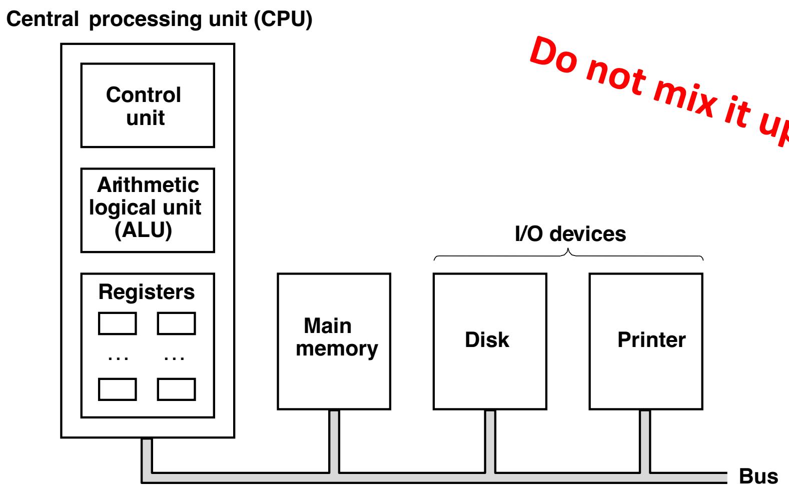
27

Basic architecture of a simple micro processor

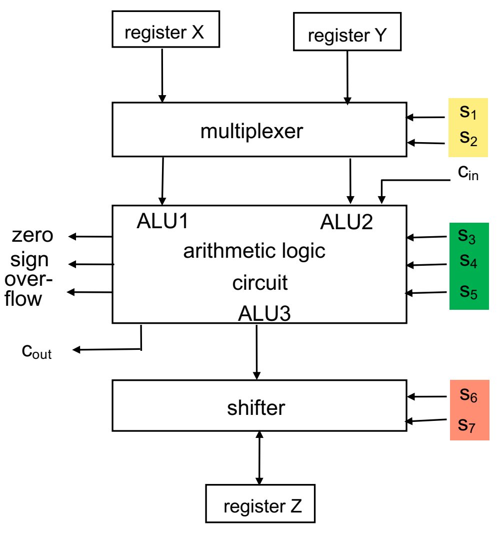


28

Single processor architecture



Do not mix it up!

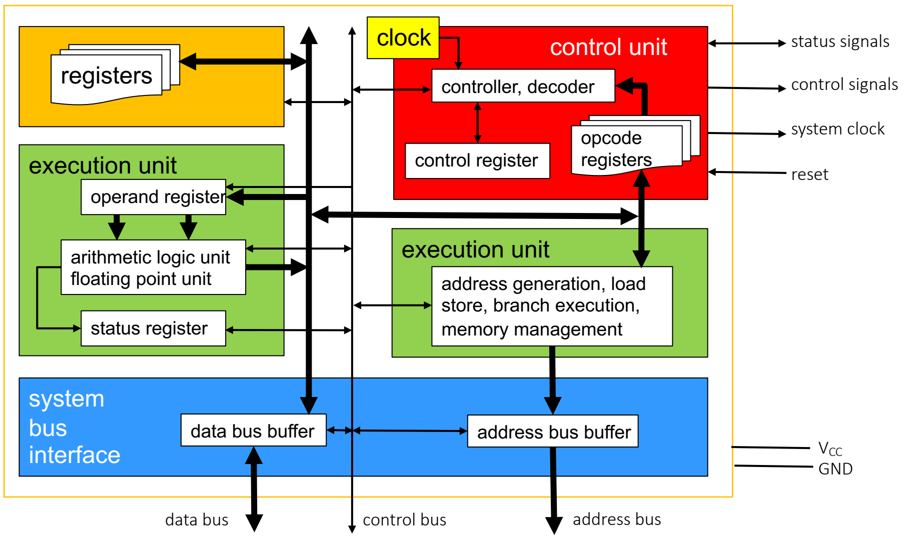


s_1	s_2	ALU1	ALU2
0	0	X	Y
0	1	X	0
1	0	Y	0
1	1	Y	X

s_3	s_4	s_5	ALU3
0	0	0	ALU1 + ALU2 +c_{in}
0	0	1	ALU1 – ALU2 – Not(c_{in})
0	1	0	ALU2 – ALU1 – Not(c_{in})
0	1	1	ALU1 $\vee$ ALU2
1	0	0	ALU1 $\wedge$ ALU2
1	0	1	Not(ALU1) $\wedge$ ALU2
1	1	0	ALU1 $\oplus$ ALU2
1	1	1	ALU1 $\leftrightarrow$ ALU2

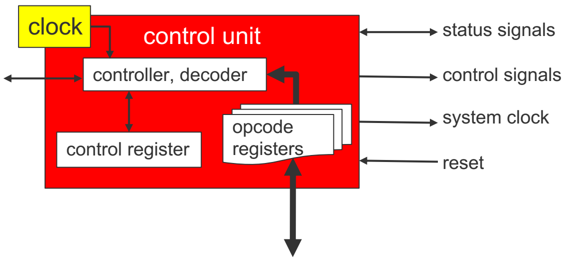
s_6	s_7	Z
0	0	ALU3
0	1	ALU3 $\div$ 2
1	0	ALU3 $\times$ 2
1	1	store Z

Internal architecture of a simple and simplified microprocessor



31

Control Unit



32

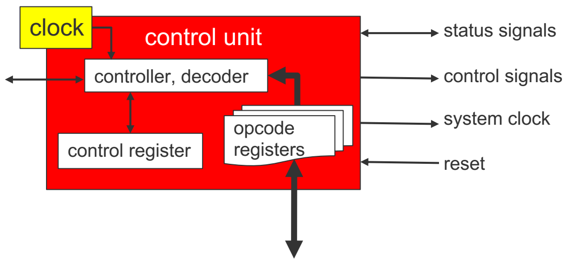
Control Unit

- The control unit controls all the components
- The **clock** generates the system clock for distribution to all components
- **Opcode registers** contain the portion of the instruction that specifies the currently executed operation to be performed (and maybe some additional opcodes)
- The **decoder** (often micro-programmable) generates all control signals for the components and uses status signals and opcode as input
- The **control register** stores the current status of the control unit

33

Clocking / synchronization

- Synchronous sequential circuit
  - Typically, CPUs use dynamic (clocked) logic
  - State is stored in gate capacitances
  - Static logic uses flip-flops instead
- Minimum clock-speed required
  - Otherwise, stored bits are lost due to leakage before the next clock-cycle
- Complex clock distribution network on-chip required



34

Micro programmable control unit

- The processor stores a microprogram for each instruction
  - Microprogram: sequence of micro instructions
  - Normal users cannot change the microprogram of a processor
  - However, manufacturers can update the microprogram
- Pure RISC processors typically do not use microprograms but a fixed sequential circuit.

35

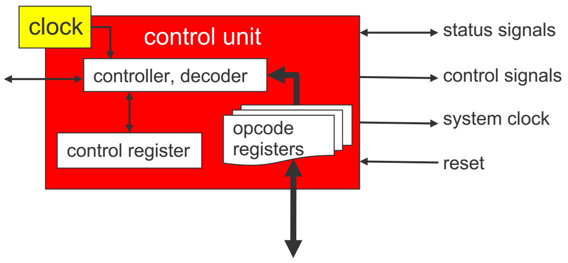
Phases of instruction execution

- Instruction fetch
  - Load the next instruction into the opcode register
- Instruction decode
  - Get the start address of the microprogram representing the instruction
- Execution
  - The microprogram controls the instruction execution by sending the appropriate signals to the other components and evaluating the returned signals

36

Opcode register

- The opcode register consists of several registers because
  - different instructions may have different sizes
    (1 byte, 2 bytes, 3 bytes …)
  - opcode prefetching may speed-up program execution
    - while decoding the current instruction the following instructions may be prefetched
    - this supports pipelining, branch prediction etc. (covered later)



37

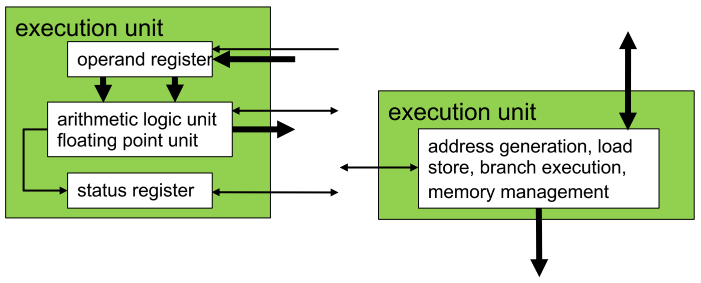
Control register

- The control register stores the current state of the control unit.
- This influences e.g. instruction decoding, operation mode.
- The meaning of the bits depend on the processor.

- Examples:
  - Interrupt enable bit
    - determines if the processor reacts to interrupts
  - Virtual machine extensions enable
    - enable hardware assisted virtualization on x86 CPUs
  - User mode instruction prevention
    - if set, certain instructions cannot be executed in user level

38

Execution unit



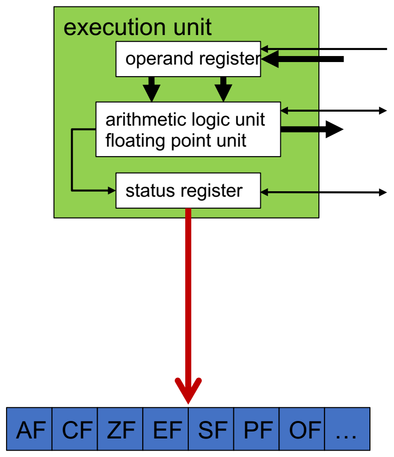
39

Execution unit

- The execution unit executes all logic and arithmetic operations controlled by the control unit.
- Examples:
  - Integer and float arithmetic operations
  - Logic operations, shifting, comparisons
  - All address related operations
  - Speculative operations (covered later)
  - Complex memory management, memory protection
  - ...
- Status register informs the control unit about the state of the processor after an operation
  - Examples: carry, overflow, zero, sign
- Operand registers, accumulators etc.: additional registers for temporary results, fetched operators etc.

40

Status register (flag register, Condition Code Register CCR)



41

Flags

- **Overflow Flag (OF)** – It indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.
- **Direction Flag (DF)** – It determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.
- **Interrupt Flag (IF)** – It determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.
- **Trap Flag (TF)** – It allows setting the operation of the processor in single-step mode. The DEBUG program we used sets the trap flag, so we could step through the execution one instruction at a time.

42

Flags

- **Sign Flag (SF)** − It shows the sign of the result of an arithmetic operation. This flag is set according to the sign of a data item following the arithmetic operation. The sign is indicated by the high-order of leftmost bit. A positive result clears the value of SF to 0 and negative result sets it to 1.
- **Zero Flag (ZF)** − It indicates the result of an arithmetic or comparison operation. A nonzero result clears the zero flag to 0, and a zero result sets it to 1.
- **Auxiliary Carry Flag (AF)** − It contains the carry from bit 3 to bit 4 following an arithmetic operation; used for specialized arithmetic. The AF is set when a 1-byte arithmetic operation causes a carry from bit 3 into bit 4.
- **Parity Flag (PF)** − It indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an odd number of 1-bits sets the parity flag to 1.
- **Carry Flag (CF)** − It contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a *shift* or *rotate* operation.

43

Program Status Word (PSW)

- Status register plus control register determine the current state of a processor
  - Result of an operation
  - Privilege level
  - …
- Together with the program counter (address of the current or next instruction) these registers determine the state of the processor at a certain instruction of a program (or process, task, …).
- The PSW combines the registers and program counter for simpler manipulation.
  - Pushed to stack before context switch (e.g. switch to another process)
  - Pulled from stack to continue execution of an interrupted process

44

Typical (simple) operations of an ALU

- Arithmetic
  - Addition with/without carry
  - Subtraction with/without carry
  - Increment/decrement
  - Multiplication with/without sign
  - Division with/without sign
  - Two's complement
- Logical
  - NOT
  - AND
  - OR
  - XOR
- Shift and rotation
  - Shift left
  - Shift right
  - Rotate right without carry
  - Rotate right with carry
  - Rotate left without carry
  - Rotate left with carry
- Memory
  - Transfer
  - Load, store

45

Complex Instruction Set Computer (CISC)

- Reasons for CISC
  - Execution of complex instructions faster than execution of equivalent programs with the same function
  - Micro programming allows for more complex instructions
  - More complex instructions lead to shorter programs thus faster loading (transfer-rate gap between CPU internally and CPU-main memory)
  - Bigger is better – more instructions sound more powerful…it's marketing!
  - Direct support of programming constructs of higher languages using more complex instructions (e.g. string compare)
  - Support of specialized powerful compilers
  - Compatibility (we can do everything like before plus xyz)
  - Support of special purpose applications (e.g. matrix operations)
- more transistors/chip, higher programming languages and special purpose applications favor "complex" instructions

46

Complex Instruction Set Computer (CISC)

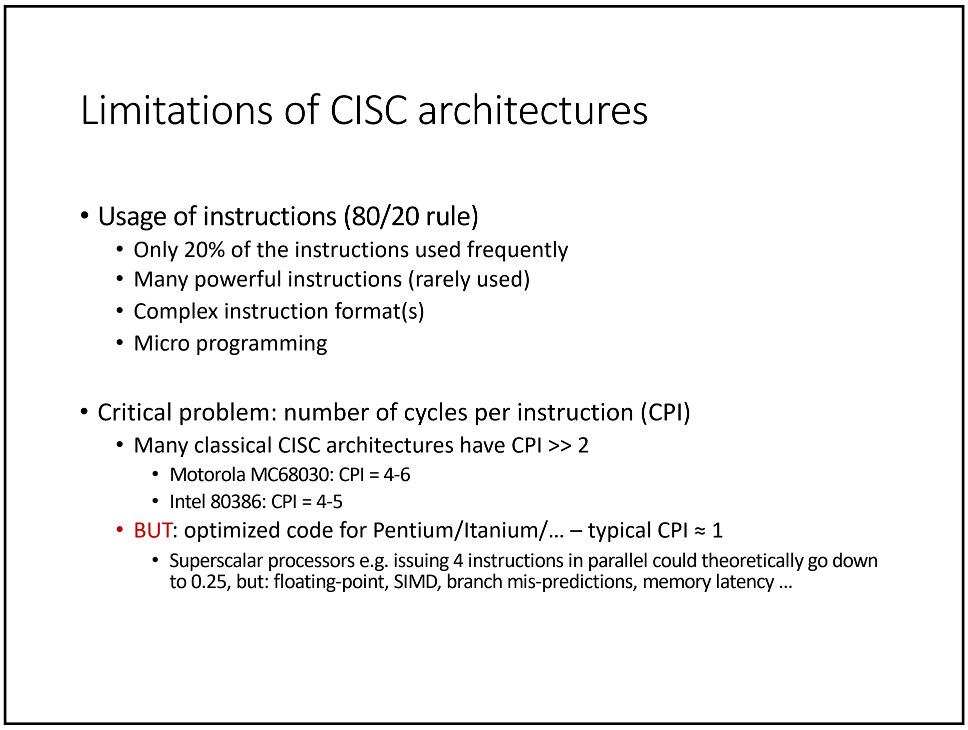
- Reasons against CISC
  - Much faster main memories (argument of the 80's, today again a problem) and the use of cache memory speed-up program execution
  - Micro programs are more and more complex (so where is the difference between programming and micro programming…)
  - Replacement of complex instructions using several simpler (much faster) instructions
  - Longer development cycles
  - Very complex control units
  - Large micro programs with (potentially with errors)
  - Real programs use only a small fraction of the large instruction set frequently!

47

The 10 most used instructions in SPECint92 for Intel x86

Instruction	Percentage [%]
load	22
conditional branch	20
compare	16
store	12
add	8
and	6
sub	5
move register-register	4
call	1
return	1
Total	**95**

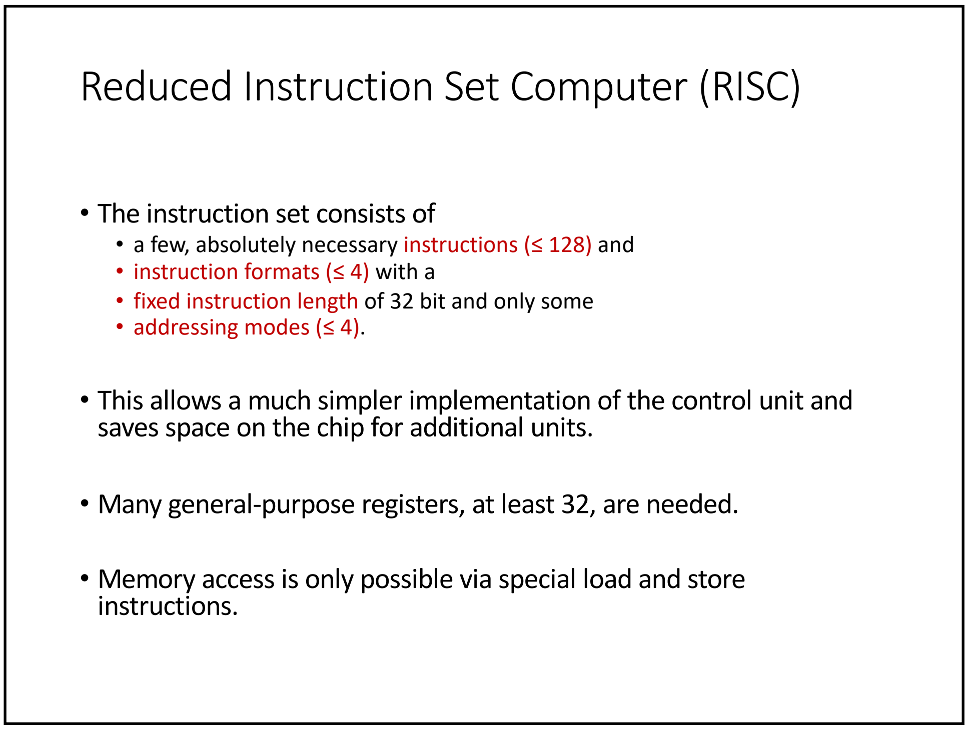
48



Limitations of CISC architectures

- Usage of instructions (80/20 rule)
  - Only 20% of the instructions used frequently
  - Many powerful instructions (rarely used)
  - Complex instruction format(s)
  - Micro programming
- Critical problem: number of cycles per instruction (CPI)
  - Many classical CISC architectures have CPI >> 2
    - Motorola MC68030: CPI = 4-6
    - Intel 80386: CPI = 4-5
  - BUT: optimized code for Pentium/Itanium/… – typical CPI ≈ 1
    - Superscalar processors e.g. issuing 4 instructions in parallel could theoretically go down to 0.25, but: floating-point, SIMD, branch mis-predictions, memory latency …

49



Reduced Instruction Set Computer (RISC)

- The instruction set consists of
  - a few, absolutely necessary instructions (≤ 128) and
  - instruction formats (≤ 4) with a
  - fixed instruction length of 32 bit and only some
  - addressing modes (≤ 4).
- This allows a much simpler implementation of the control unit and saves space on the chip for additional units.
- Many general-purpose registers, at least 32, are needed.
- Memory access is only possible via special load and store instructions.

50

Reduced Instruction Set Computer (RISC)

- Memory access is via load and store operations only.
- All other instructions work on the CPU registers only, e.g., arithmetic operations load operands from registers and store results in registers only.
- This basic principle is called
  - register/register architecture or
  - load/store architecture and is typical for many (original) RISC computers.

51

RISC

- If possible, all instructions should be implemented in a way that they finish within a single processor cycle.
- Consequence: pure RISC processors do not use micro programming
  - RISC processors introduced enhanced pipelining mechanisms (today, many processors use pipelining for the micro instructions, e.g., Pentium 4 and up).
- Furthermore, the early RISC processors had a software-controlled pipeline (compilers inserted delay NOPs, introduced delayed jumps etc.) instead of special hardware.
- Aside
  - PC processors like the Pentium 4 (and up) use micro programming, the internal micro architecture (netburst) is rather RISC, the ISA is CISC.

52

RISC

- Reasons for
  - Single-chip implementation (yes, today “everything” fits on a single chip)
  - Shorter development cycles
  - Higher clock rates, pipelining
  - Re-use of saved chip space for, e.g., cache
- Reasons against
  - Bottleneck in the memory interface, today again main memory is much slower compared to internal registers/cache
  - Space on a chip is not that critical anymore

53

CISC / RISC

- Pure RISC prefer the Harvard architecture
  - Separate memory for instructions and data (operands) and, thus, two address and two data busses

  ➡ parallel fetching of instruction(s) and operand(s) possible
- Simplified versions

1. Two separate bus systems up to the L1 caches, but only one main memory/unified L2/L3 cache (cheaper, standard with today’s systems)
2. Only a single, multiplexed bus system

54

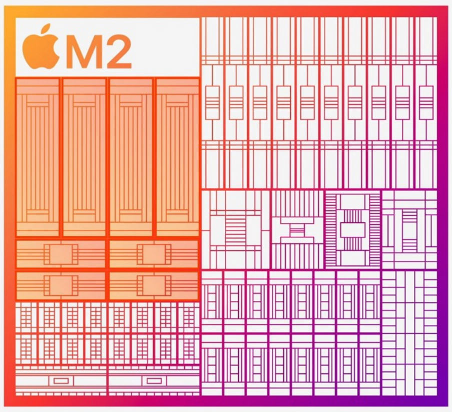
CISC / RISC

- Control unit
  - Hard-wired
  - Instruction register is a simple FIFO queue
  - Each pipeline stage has its own register
  - A simple combinational circuit can “interpret” the OpCodes in each stage directly
- Register file
  - Consists of a large number of (general purpose) registers
  - Supports the simultaneous selection of several registers
    - E.g. 4 port register file: simultaneous write in R0, R1 and read from R2, R3

55

4 high-performance cores
Ultrawide microarchitecture
192KB instruction cache
128KB data cache
Shared 16MB cache

4 high-efficiency cores
Wide microarchitecture
128KB instruction cache
64KB data cache
Shared 4MB cache



56