High Performance Co$^{mputing}_{ding}$ (HPC)

Alberto Dassatti - 2020

An instruction set, or instruction set architecture (ISA), is the part of the computer architecture related to programming, including the native data types, instructions, **registers**, addressing modes, memory architecture, interrupt and **exception handling**, and external I/O.

Instruction set - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Instruction_set**

ISA

All information needed by the prgrammer for programming an architecture

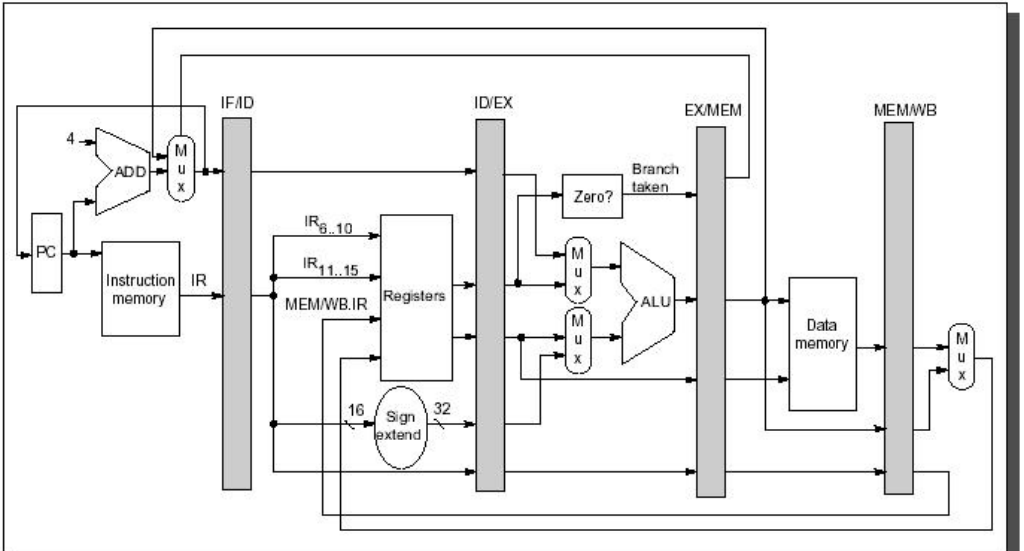
Microarchitecture

The real hardware, CPU, memory and so on, giving (an) implementation(s) of an ISA

In computer engineering, **microarchitecture** (sometime abbreviated to µarch or uarch) is a description of the electrical circuitry of a computer, central processing unit, or digital signal processor that is sufficient for completely describing the operation of the hardware.

Microarchitecture - Simple English Wikipedia, the free ...
https://simple.wikipedia.org/wiki/**Microarchitecture**

DLX


IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
M
u
x
PC
Instruction memory
IR
IR6..10
IR11..15
MEM/WB.IR
Registers
16
Sign extend
32
Zero?
Branch taken
ALU
Data memory

Complex instruction set computing (**CISC** /ˈsɪsk/) is a processor design, where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.

Complex instruction set computing - Wikipedia
https://en.wikipedia.org/wiki/Complex_instruction_set_computing

CISC

PDP-11, VAX, Motorola 68k, and your desktop PCs on intel's x86 architecture.

RISC

RISC families include DEC Alpha, AMD Am29000, ARC, ARM, Atmel AVR, Blackfin, Intel i860 and i960, MIPS, Motorola 88000, PA-RISC, Power (including PowerPC), RISC-V, etc...

Reduced instruction set computing, or **RISC** (pronounced 'risk', /ɹɪsk/), is a CPU design strategy based on the insight that a simplified instruction set provides higher performance when combined with a microprocessor architecture capable of executing those instructions using fewer microprocessor cycles per instruction.

Reduced instruction set computing - Wikipedia
https://en.wikipedia.org/wiki/Reduced_instruction_set_computing

Q&A

1. Why RISC has been so succesful?

Q&A

1. Why RISC has been so succesful?
2. What kind of ISA is X86?

Q&A

1. Why RISC has been so succesful?
2. What kind of ISA is X86?
3. Why Intel was so succesful?

Q&A

1. Why RISC has been so succesful?
2. What kind of ISA is X86?
3. Why Intel was so succesful?
4. How a CISC ISA is implemented today?

Some more recent examples

Let's have a look at some modern microarchitectures:
Intel Nehalem Wikipedia, and more or
AMD Bulldozer and more

Nehalem

1. 6 Ops: 3 mem (load, store data and store address) + 3 ALU

Nehalem

1. 6 Ops: 3 mem (load, store data and store address) + 3 ALU
2. Out of Order dynamically scheduled

Nehalem

1. 6 Ops: 3 mem (load, store data and store address) + 3 ALU
2. Out of Order dynamically scheduled
3. Speculation: branch, pre-fetch (data-flow model)

Nehalem

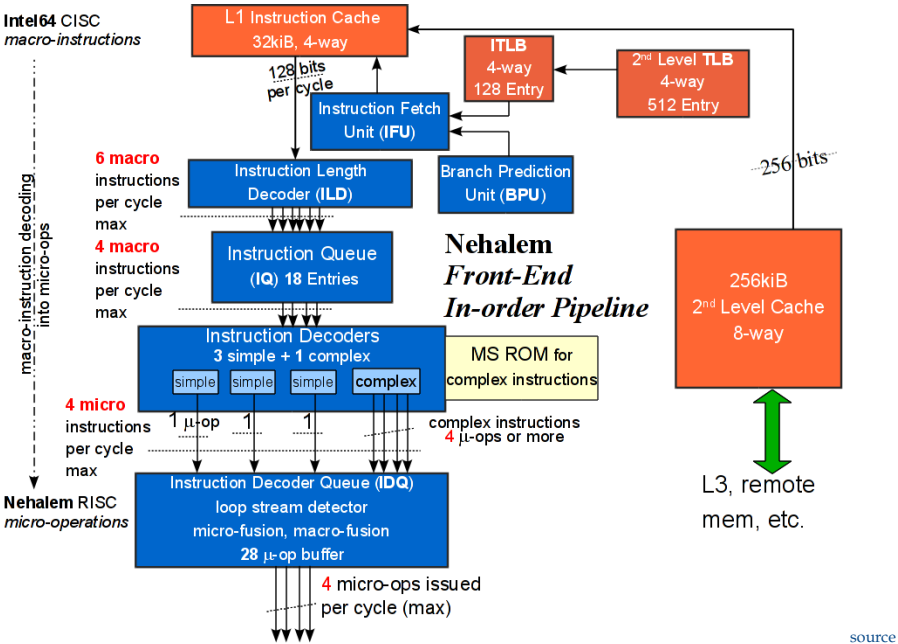
1. 6 Ops: 3 mem (load, store data and store address) + 3 ALU
2. Out of Order dynamically scheduled
3. Speculation: branch, pre-fetch (data-flow model)
4. $20 - 24$ stage pipeline

Nehalem

1. 6 Ops: 3 mem (load, store data and store address) + 3 ALU
2. Out of Order dynamically scheduled
3. Speculation: branch, pre-fetch (data-flow model)
4. $20 - 24$ stage pipeline
5. $128 \mu Ops$

Nehalem

1. 6 Ops: 3 mem (load, store data and store address) + 3 ALU
2. Out of Order dynamically scheduled
3. Speculation: branch, pre-fetch (data-flow model)
4. $20 - 24$ stage pipeline
5. $128 \mu Ops$
6. Very High and variable ILP


Intel64 CISC
macro-instructions
L1 Instruction Cache
32kiB, 4-way
128 bits
per cycle
ITLB
4-way
128 Entry
2nd Level TLB
4-way
512 Entry
Instruction Fetch
Unit (IFU)
6 macro
instructions
per cycle
max
Instruction Length
Decoder (ILD)
Branch Prediction
Unit (BPU)
256 bits
macro-instruction decoding
into micro-ops
4 macro
instructions
per cycle
max
Instruction Queue
(IQ) 18 Entries
Nehalem
Front-End
In-order Pipeline
256kiB
2nd Level Cache
8-way
Instruction Decoders
3 simple + 1 complex
simple
simple
simple
complex
MS ROM for
complex instructions
4 micro
instructions
per cycle
max
1 µ-op
1
1
complex instructions
4 µ-ops or more
L3, remote
mem, etc.
Nehalem RISC
micro-operations
Instruction Decoder Queue (IDQ)
loop stream detector
micro-fusion, macro-fusion
28 µ-op buffer
4 micro-ops issued
per cycle (max)

source

Or another useful view for performance arch

Q&A

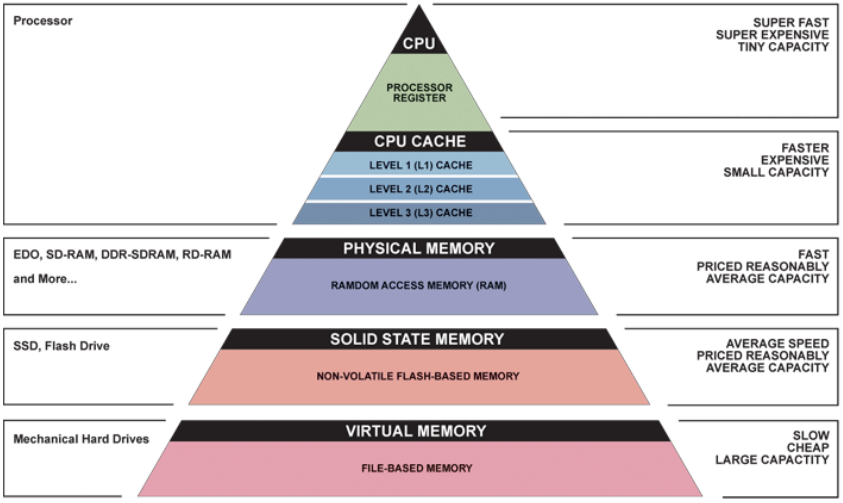
1. Why large μOps buffer is important?

Q&A

1. Why large μOps buffer is important?
2. Why caches are so important in this architectures?

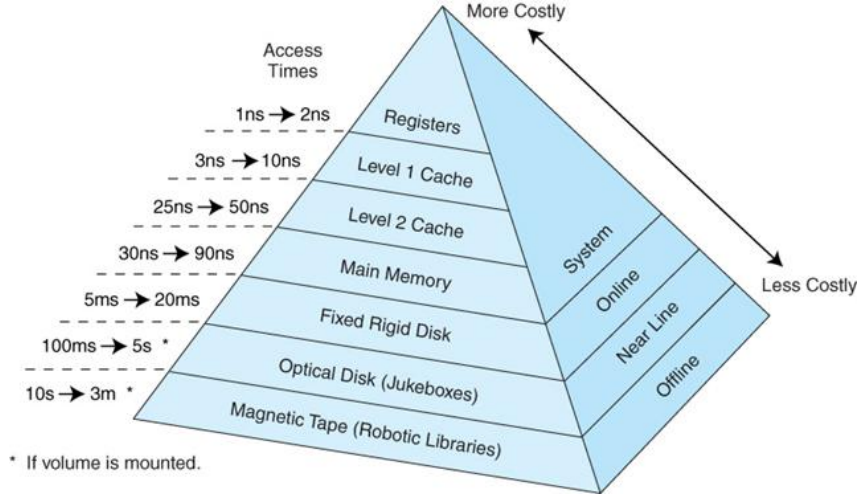
Q&A

1. Why large μOps buffer is important?
2. Why caches are so important in this architectures?
3. HiperThread technology is a compromise: when it makes sense?



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

source


More Costly
Access Times
1ns → 2ns
Registers
3ns → 10ns
Level 1 Cache
25ns → 50ns
Level 2 Cache
30ns → 90ns
Main Memory
5ms → 20ms
Fixed Rigid Disk
100ms → 5s *
Optical Disk (Jukeboxes)
10s → 3m *
Magnetic Tape (Robotic Libraries)
System
Online
Near Line
Offline
Less Costly
* If volume is mounted.

source

Cache Memory

If you need a gentle basic introduction to cache memories see this article

Cache Miss

- Coldness: first time the data is not there → Prefetching
- Capacity: too much data accessed → organize data and code differently
- Conflict: Multiple data mapped to the same location
- Sharing:
  - True: Thread in another processor wanted the data, it got moved to the other cache
  - False: data in the same cache line used by another core

Latency Numbers Every Programmer Should Know

latency.txt

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms

Notes
-----
1 ns = 10^-9 seconds
1 us = 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns

Credit
------
By Jeff Dean:               http://research.google.com/people/jeff/
Originally by Peter Norvig: http://norvig.com/21-days.html#answers
```

source

Q&A

1. When latency is critical?

Q&A

1. When latency is critical?
2. When throughput is more relevant?

Q&A

1. When latency is critical?
2. When throughput is more relevant?
3. What is the guiding principle to write performant code on these architectures?

Q&A

1. When latency is critical?
2. When throughput is more relevant?
3. What is the guiding principle to write performant code on these architectures?
4. Where are caches introduced with respect of MMU? Why?

Extra

If you are interested in the effects of branch predictors (and you should) have a look here. Cache effects are very complicated to study and isolate. Read this article to have a better idea.