Computer Architecture

R. Poss
Computer Systems Architecture group (UvA)
e-mail: r.c.poss@uva.nl

What is "computer architecture"?

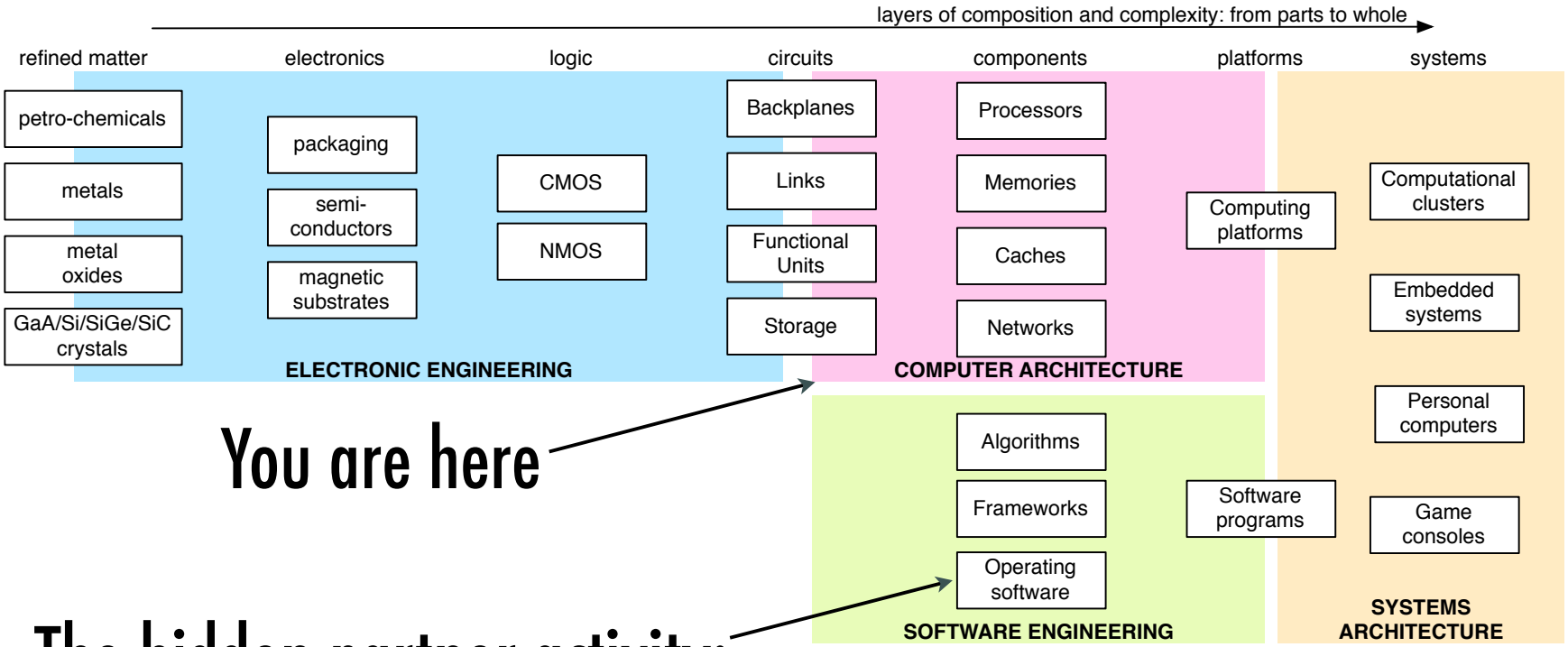
Your ideas and expectations

- What is part of “computer architecture”, what is not?
- Who are “computer architects”, what is their job?
- What is the role of “computer architecture” in science?
- Why do you need to know about “computer architecture”?

Vocabulary

- List words/concepts that you want explained during this course:
  - About the hardware/software interface
  - About hardware components
  - About choices in design
  - About how to be a better programmer
  - etc. - anything you think is relevant

An engineering domain



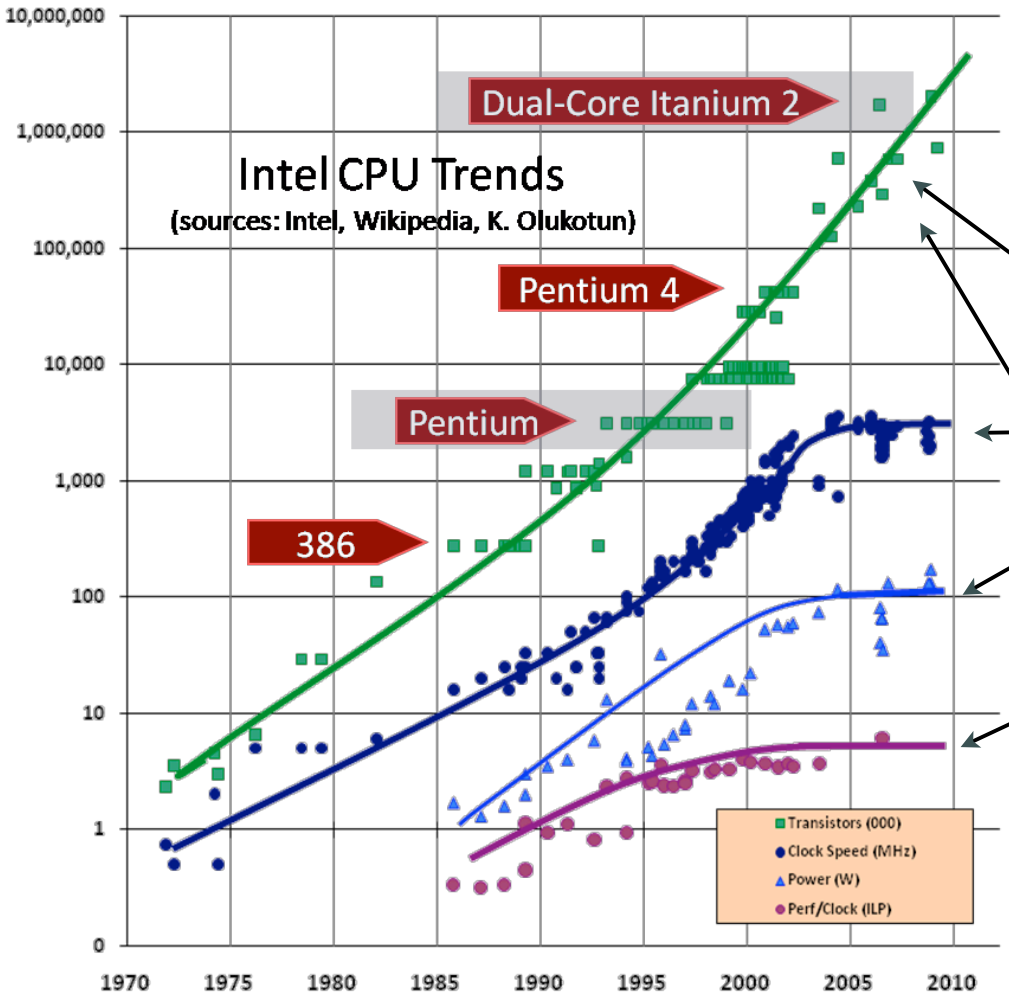
You are here

The hidden partner activity:
compilers, operating systems

Current state of affairs

- Power vs chip area:
  - before: power free, transistors expensive
  - now: power expensive, transistors cheap
- Storage vs computation:
  - before: computation slow, storage fast
  - now: storage slow, computation fast
- Computation vs storage cost:
  - before: small storage, ok to compute more to save space (eg compression)
  - now: large storage, expensive to compute more

Trends - "the free lunch is over"



This is the story of uniprocessor performance

This is the power wall

This is the memory wall + seq. performance wall

http://www.gotw.ca/publications/concurrency-ddj.htm

Current state of affairs (cont)

- A dramatic change in processor chips:
  - Memory wall: processors much faster than memories
  - Power wall: can't power all transistors lest the chip will fry
  - Sequential performance wall: more transistors don't help sequential performance any more
- This course will suggest how we got here, why these problems are happening and what we can do about it

Example questions

- You are in charge of selecting a processor chip for a new web server. You have a choice between two chips, a 1-core running at 2GHz and a 2-core running at 1 GHz. They use the same core micro-architecture, have nearly the same price. How do you choose? Why?
- The web server will support encrypted SSL connections. You can choose a 4-core processor, 4MB of cache on chip with a cryptographic accelerator with 4 channels. Or you can choose a 4-core processor with 8MB of cache on chip but no accelerator. How do you choose?

Course & organization

Aims of this course

- The aims of this course are:
  - **to introduce the notion of hardware/software interface and variations in ISA design**
  - **to give a thorough understanding of modern microprocessor design and related issues**
  - **to introduce parallelism in computer architecture**
  - **to introduce simulators & architecture models**

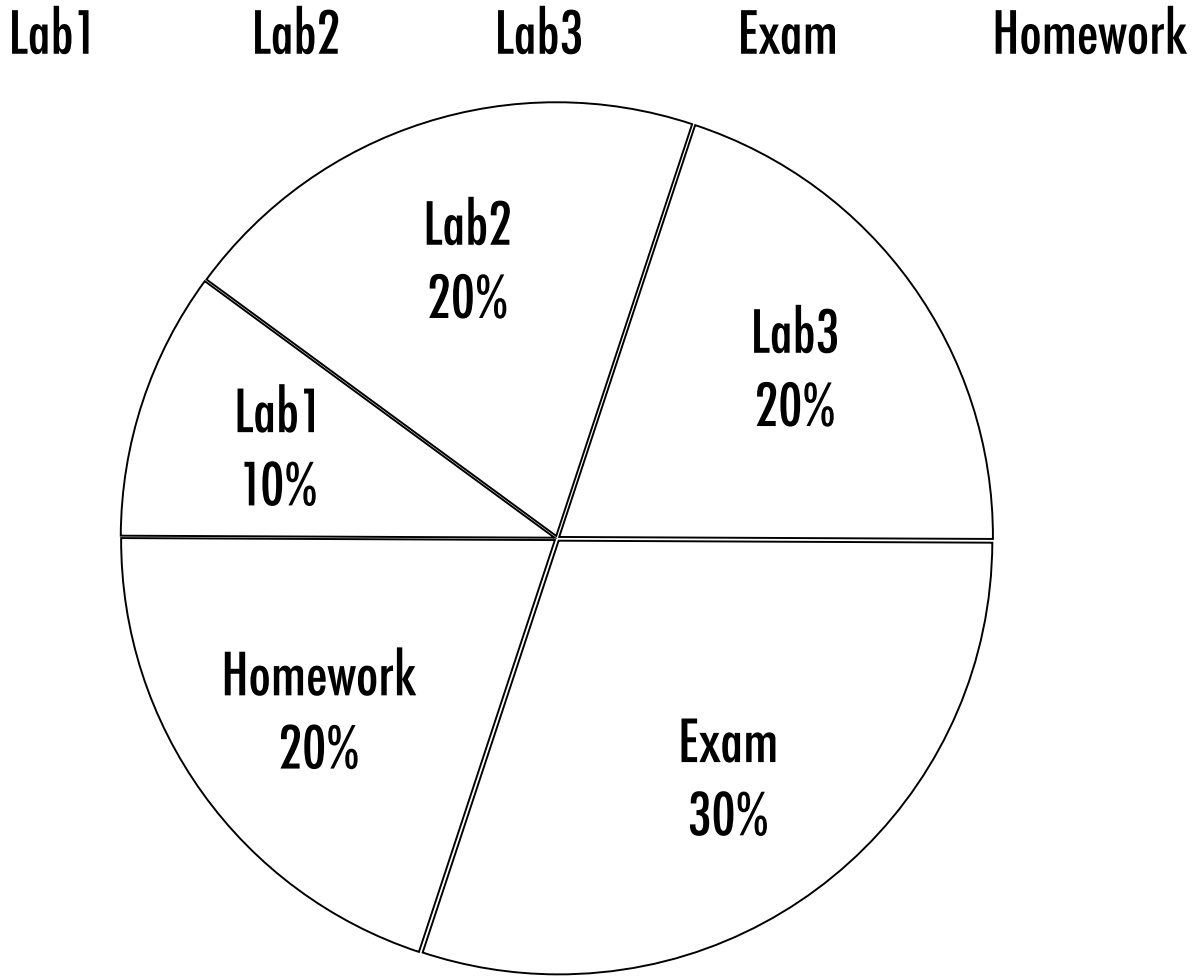
Bibliography

- The main course text is:
  - Computer architecture - a quantitative approach, Hennessy & Patterson, 4th Edition, ISBN 978-0-12-370490-0.
- Other useful texts are:
  - Processor Architecture, Silc, Robic and Ungerer, Springer, ISBN 13-540-64798-8
  - D. Sima, T. Fountain and P. Kacsuk, Advanced Computer Architecture a Design space approach (Addison-Wesley)
  - Your own search-fu – use Google Scholar!

Assessment

- Assessment will be by coursework assignments and exam
- The following weighting will be used
  - Homework 20% Lab assignments 50% Exam 30%
- Assignment labs start on Sep 4th, supervised by J. Neuteboom & B. Hijmans
  - http://www.liacs.nl/ca
- Assignment assessment will be based on demonstration to the lab supervisors (60%) and a brief report (40%) to me on the observations of your results, deadline in details

Assessment


Lab1
Lab2
Lab3
Exam
Homework
Lab2
20%
Lab3
20%
Lab1
10%
Homework
20%
Exam
30%

Communication & contact

- Please use the mailing list
  https://list.uva.nl/mailman/listinfo/ca2012
- Prefer **group discussions** to one-on-one interactions
  - There are no stupid questions, don't be ashamed
  - However, try to find if your question has already been answered first

Course overview

The following topics will be covered in a bottom-up approach to the subject

1. **Hardware/software interface, ISAs**
2. **Processors & implicit concurrency**
   - Pipelined processors. superscalar microprocessors
3. **Memory, caches, interconnects & topology**
4. **Explicit concurrency & parallelism in systems**
   - Vector processors, VLIW, multi-cores, hardware multi-threading
5. **Co-processors and accelerators**

Getting started

Contribution of Comp. Arch.

- Quantitative principles of design
  - Take advantage of parallelism
  - Principle of locality
  - Focus on the common case
  - Amdahl's laws
- Careful, quantitative comparison: define, quantify, summarize
- Anticipating and exploiting advances in technology
- Well-defined interfaces, carefully implemented and thoroughly defined

Parallelism

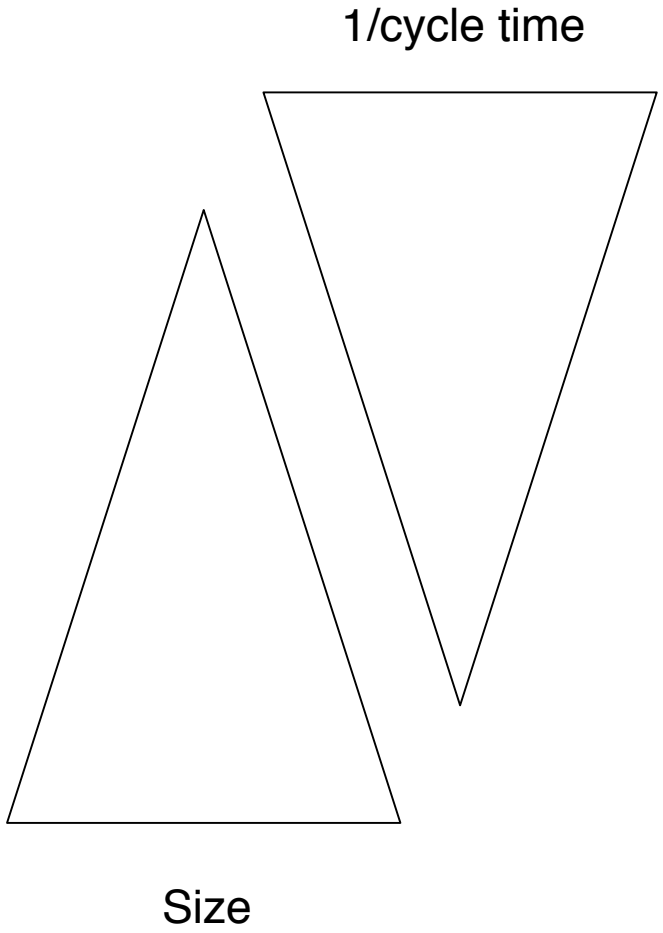
- Three main strategies:
  - Increase bandwidth and throughput by duplicating storage and data paths
  - Use pipelining, ie “assembly line”
  - Perform operations out of order, including simultaneously
- Fundamental limits:
  - pipeline hazards
  - time and data dependencies = mandatory order

Locality

- Principle: individual programs access a relatively small portion of memory in a small amount of time
- Two different types:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
- Caches are a fundamental mechanism to take advantage of locality

Memory hierarchy and latency

	100MHz clocks	GHz clocks
Registers - on-chip SRAM	< 1	< 1
L1 cache - on-chip SRAM	1-2	1-6
L2 cache - on-chip SRAM	2-6	≃10
off-chip SRAM	4-8	≥10
L3 cache - off-chip SRAM	≃10	≥100
Main memory - DRAM	≥10	≥1000
Distributed memory	≥100	≥10000



Focus on the common case

- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

Amdahl's law on speedup

- Consider a computation P which contains two parts A and B in sequence
- A can be enhanced (eg more parallelism, more performance); B cannot
- $T(P) = T(A) + T(B)$ (T = time to complete)
- Imagine we can accelerate A infinitely so that T(A) becomes 0
- Intuitively: overall speedup is limited by T(B)
- If the complexity ratio between A and B is $P_{[A/B]}$ (proportion), and A can be accelerated by a factor S_A (speedup), Amdalh's law says:
  - $S_{overall} = 1 / ((1 - P_{[A/B]}) + (P_{[A/B]} / S_A))$

Amdahl's law example

- An algorithm contains a sequential section and a parallel section
- The parallel section contains 20% of the computation steps (P=0.2)
- The parallel section can be accelerated by a factor N by using N processors/cores
- Maximum speedup with N cores = 1 / ((1 - 0.2) + (0.2 / N))
- With N = 100, speedup = 1.24X (100 cores, yet only 24% perf increase!)
  - This is the fundamental limit to parallelism: to maximize performance gains, need to first increase the proportion of the parallel section.

Amdahl's law on design

- A balanced system design should provision 1 bit per second of external bandwidth for each potential instruction per second
  - Too little external bandwidth: "I/O bound"
  - Too little instructions/second: "compute-bound"
- "Desktop" computers are traditionally I/O bound
- Mainframes are the other way around
- Multi-cores require huge amount of bandwidth to stay balanced

Lab assignments

Lab assignments

- First series: **getting to know** the **hardware/software interface**, make your own MIPS code
- 2nd series: **implement your own** MIPS-like ISA in a simulator/emulator
- 3rd series: use your simulator/emulator with your own programs to **study the impact of different architecture parameters** on program execution
- Total: 50% of final grade;
  First series: 10% of final grade; 2nd: 20%, 3rd: 20%

How do programs run?

- General computer model:
  processor + memory + interconnect + I/O devices
- Software is just bits, so is data
- How does software translate into **behavior**?
  ie. communication, computation and control?
  - Your take here

Computers and interpreters

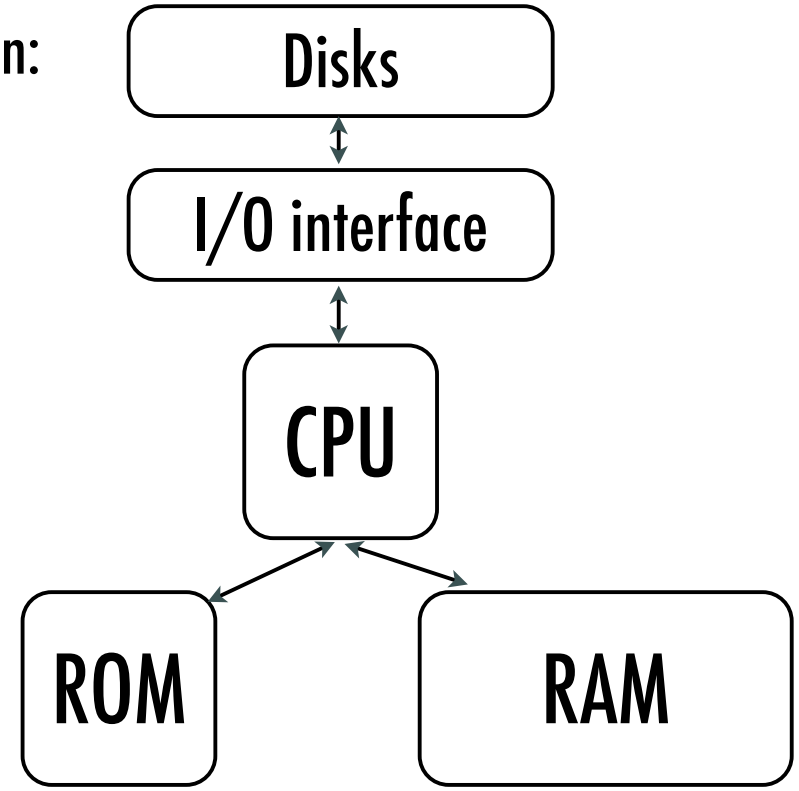
- A computer processes inputs and produces output
  Both inputs and outputs are just bits of data
- A universal computer also reads what to do (program) as data
  - it interprets the program code step by step as instructions
- Software = data = program code + input
  NB: The behavior of software comes from the machine that interprets it

Multiple layers of interpreters

- Java bytecode and real hardware:
  - Java VM is an interpreter for Java bytecode
  - Hardware processor is an interpreter for Java VM program code
- Python bytecode and real hardware:
  - CPython is an interpreter for Python bytecode
  - Hardware processor is an interpreter for CPython
- System simulators/emulators and real hardware
  - MGSim is an interpreter for Alpha/SPARC/MIPS program code
  - Hardware processor is an interpreter for MGSim

System initialization

- What happens when you switch the computer on?
- Define/explain the relationships between:
  - Reset signal
  - Initial program counter
  - Boot ROM
  - Boot code
  - Operating system
  - Start-up storage



What does this code do?

```
        .ent sum
sum:
        mov $31,$0
        ble $16,L2
        mov $31,$1
L3:
        ldl $2,0($17)
        addl $2,$0,$0
        addl $1,1,$1
        lda $17,4($17)
        cmpeq $1,$16,$2
        beq $2,L3
L2:
        ret $31,($26),1
        .end sum
```

$31 is a special register "zero"

Alpha assembly uses left-to-right operands
- except for ldX and br/jsr/ret

Function arguments passed in $16-$21

"ble" = branch if lower or equal

"lda" = load address, a form of "add"

$26 is also called "ra" for "return address"

What does this code do?

```
        .ent sum
sum:
        mov $rz,$0
        ble $a0,L2
        mov $rz,$1
L3:
        ldl $2,0($a1)
        addl $2,$0,$0
        addl $1,1,$1
        lda $a1,4($a1)
        cmpeq $1,$a0,$2
        beq $2,L3
L2:
        ret $rz,($ra),1
        .end sum
```

"$a0" is an alias for the concrete register name "$16"
The assembler translates the former to the latter automatically

"ret A, (B)" means "place the current PC in A, then jump to the address in B"

How did we get this code?

- C source for sum.c:

```
int sum(int n, int x[])
{
  int s = 0;
  for (int i = 0; i < n; ++i)
    s += x[i];
  return s;
}
```

- Compile: `alpha-cc -S -o sum.s sum.c`
- Assemble: `alpha-as -o sum.o sum.c`
- Link: `alpha-ld -o demo sum.o ...`

Let's run this

- You probably don't have an Alpha processor at hand
  - First, let's try to compile/assemble/link/run natively
    - eg. using the x86 instruction set
  - But this course is about RISC - the example uses the Alpha instruction set so let's use an emulator instead!
- Emulator = program running on processor A that interprets program code made for another processor B
- Simulator = program that mimics the behavior of another system
- All emulators are simulators, but the reverse is not true

Why emulators? Why not native?

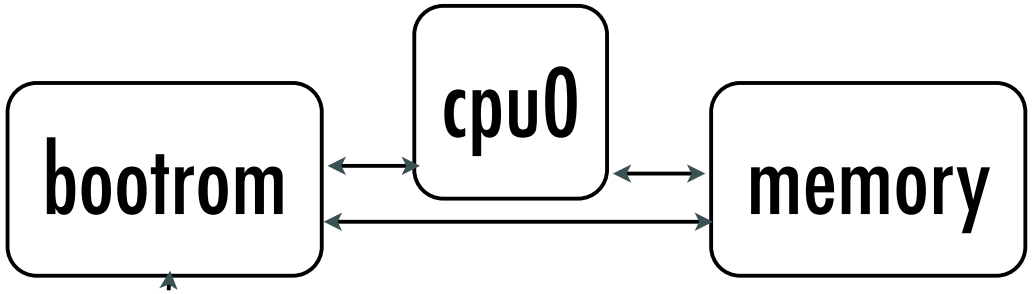
- This course will talk about the processor(s) in your desktops/ laptop machines
- But all x86 processors are really RISC "under the hood"
  - More useful to study RISC to understand the main problems
- Also: for your lab assignments you will modify an architecture and study its parameters
  - Easier with a simulator than real hardware!

Intro to MGSim

- Developed at the University of Amsterdam
- Purpose is to simulate multi-cores and do architecture research
- Simulates: cores, memory, interconnect, I/O devices
  - i.e. it is a full-system simulator
- It emulates the Alpha and SPARC instruction sets
  - you will add support for MIPS in your lab assignments

Intro to MGSim

- See the manual page mgsimdoc(7)
- System diagram for *minisim.ini*:



command to start:

```
mgsim -c minisim.ini yourprogram.bin
```

Add -i for an interactive prompt

Summary

- Seen today:
  - What is computer architecture and why it is important
  - Some general principles of architecture
  - Intro to the hardware/software interface, machine instructions and system initialization
  - How to compile/assemble/link/run a program in a simulated environment