Instruction-level Parallel Processors

How architectures exploit ILP

Tópico 3

Instruction-level Parallel Processing

Tópico 2 : Exploiting Instruction-level Parallelism (ILP) – An Overview.

Tópico 3 : *Instruction-level Parallel Processors How architectures exploit ILP.*

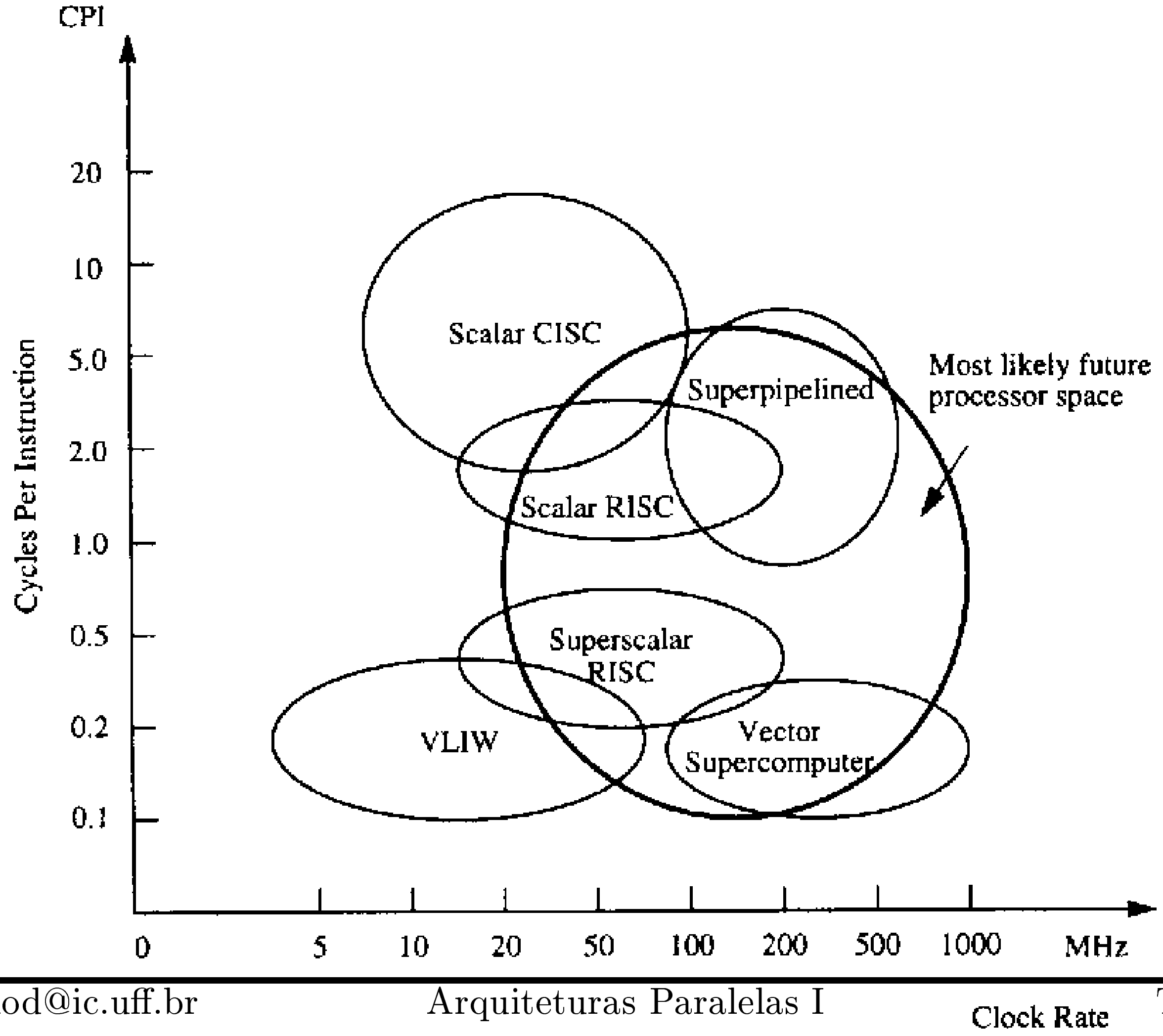
Tópico 4 : Processing Control Transfer Instructions.

Tópico 5 : Code Scheduling for Instruction-level Parallel Processors.

Outline for Tópico 3

- Very Long Instruction Word (VLIW) Processors
- Superscalar Processors
  - Parallel Decoding
  - Instruction Issue
  - Parallel Instruction Execution
  - Preserving the sequential consistency of execution and of exceptions.

Processor Design Space



Very Long Instruction Word Processors

VLIW architectures are closely related to superscalar processors. Both:

- aim at speeding up computation by exploiting ILP;
- have nearly the same execution core, consisting basically of multiple execution units operating in parallel; and,
- employ either a unified register file for all data types or distinct (*split*) register files for fixed- and floating-point data.

But the two main differences between them are *how instructions are formulated* and *how instruction scheduling is carried out.*

Instructions for VLIW Processors

- VLIW architectures are controlled by long instruction words comprised of a control field for each of the execution units available.
- Thus, the length of the VLIW instructions depends on the number of available execution units and the code lengths required to control each of them.
- VLIW processors usually incorporate a considerable number of execution units (possibly 5-30). Each unit might require a control word of 16–32 bits.
- Trace 7/200, which is capable of executing 7 instructions per cycle, has a word size of 256 bits. The word length of the Trace 28/200 is 1 Kbits.

Instruction Scheduling for VLIWs

In general, superscalar architectures are assumed to be scheduled *dynamically*. In contrast, VLIW architectures are scheduled *statically*.

- Static scheduling moves the burden of instruction scheduling from the processor to the compiler.
- This reduces the complexity of VLIW architectures considerably. Compared to a superscalar processor, a large number of tasks are simpler or superfluous, such as instruction decoding, issuing and reordering.
- The lower complexity can in turn be exploited for boosting performance, by increasing either the *clock rate* or the *degree of parallelism*, or both.

[If RISC was better than CISC, is VLIW better than superscalar?]

Instruction Scheduling – A closer look

In static scheduling, the compiler takes full responsibility for the detection and removal of dependencies.

- Increases compiler complexity.
- In order to be able to schedule instructions, the architecture needs to be exposed to the compiler in considerable detail.
- As well as the semantics and syntax of instructions, the compiler has to be aware of the *technology dependent parameters* like latency and repetition rates of functional units.
- The consequence is that a given compiler cannot be used for subsequent models of a VLIW line.

Further Problems of VLIW Processors

- The compiler has to take into account worst-case delay values. Cache latency depends heavily on whether a cache access hits (perhaps 2 cycles) or misses (perhaps upto 5 cycles).
- Not all of the fields of an instruction may actually be used, resulting in wasted *memory space* and *memory bandwidth*. (E.g. Fortran object code is as much as three times larger for a trace processor than it is for a VAX architecture.)
- VLIW architecture is totally incompatible with that of any conventional general-purpose processor.

  A superscalar machine can be object-code compatible with a large family of nonparallel machines.

Further Problems of VLIWs$_{(cont)}$

- The limits on increasing parallelism are similar for both VLIWs and superscalar machines, e.g. limited number of ports on a register file.
- In order to exploit the fact that VLIWs have more functional units than superscalar ones, better high-performance parallelising compilers are required to extract sufficient parallelism from programs.
- Difficult to program by hand in assembly code.

The question is whether VLIWs can convert the benefits they gain from reduced complexity into a higher degree of utilised parallelism.

The Trace 200 Family

- A commercial product based on the results of research work done at Yale with VLIW machines and trace compilers in the early 1980s.

- Consists of three families, the *Trace 7/200*, *14/200* and *28/200*, capable of executing 7, 14 or 28 parallel operations respectively. The higher numbered models consisted of two or four linked *7/200*s.

- The 7/200 fetches 256-bit VLIW instructions. Each instruction word consists of 8 subwords to control the execution of seven operations (4 integer, 2 FP and a conditional branch).

The Trace 200 Family$_{(cont)}$

- Supports *multiway branching*. ALU operations can used as conditional branches. If more than one exists per instruction they are prioritised by the compiler.
- To save memory space, a 32bit mask is used to identify empty and non-empty 32bit subwords in 1Kbit VLIW word. Only non-empty subwords are actually stored in memory. This was at the expence of complex cache fill circuitry.
- Performance was quite impressive at its launch.

Machine	Issue Rate (nS)	Linpack (MFLOPS))
Trace 7/200	130	6
Trace 14/200	130	10
DEC 8700	45	0.97
Cray XMP	8	24

VLIWs – Where are they now?

- While the term "VLIW" was only coined in 1983, these architectures appeared as early as 1985. Floating Point Systems FPS-120B has two floating point units operating in parallel.
- ELI-512 together with Bulldog compiler based on trace scheduling designed at Yale (1983).
- Trace family developed by Multiflow (1987)
- CYDRA-5 from Cydrome (1990). Unfortunately, both Multiflow and Cydrome have gone bankrupt.
- While a number of academic machines exists (e.g. *i*Warp at CMU, StaCs), there are fewer commercial ones (TM-1 from Philips). HP, IBM and Intel have each revealed plans to develop a VLIW machine.

The Future of VLIW Processors

Although the idea is academically sound, the dependence on trace-scheduling compilation, code compaction, and the lack of compatibility with conventional hardware and software has prevented VLIW architectures from gaining acceptance in the commercial world.

Currently, the comercial future of VLIWs is not yet clear! Also their fiercest competitor, superscalar processors, have achieved tremendous progress in increasing performance.

Superscalar Processors

- Currently dominate the processor market.
- The idea was first proposed as early as 1970.
- During the 1980s a number of architectural research proposals and prototype machines appeared. IBM was the first with the *Cheetah* and *America projects* which later spawned the Power1. The term "superscalar" first appeared in an internal IBM technical report in 1987.
- The first commerical superscalar processor was introduced by Intel in 1989 (Intel 960).

Superscalar Processors$_{(cont)}$

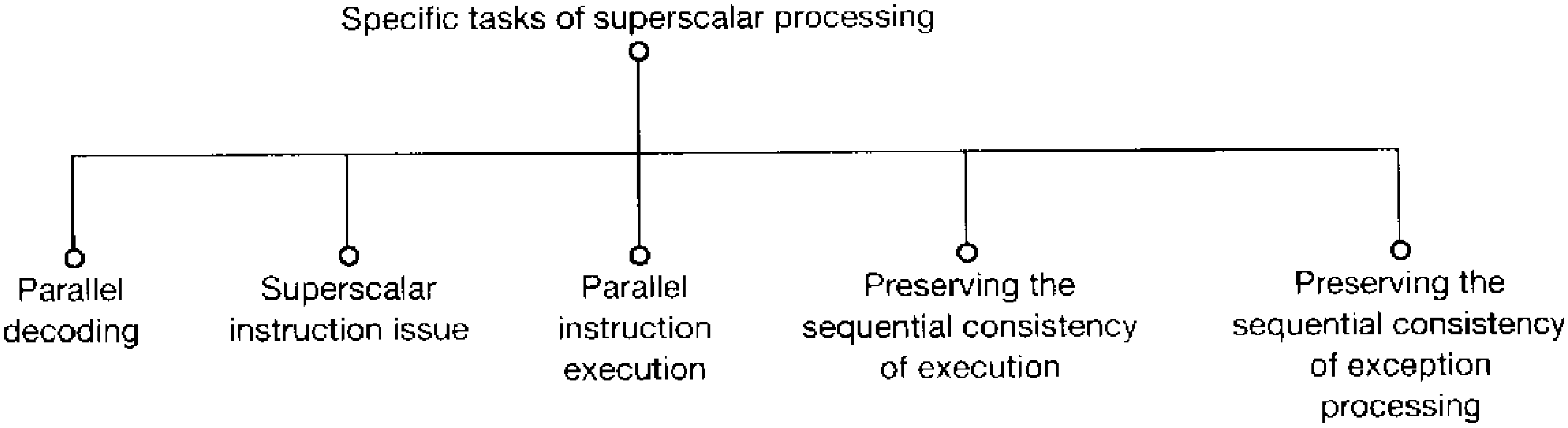
- Superscalar RISC machines appeared as a result of two different approaches:

    1. converting an existing scalar RISC line into a superscalar one, e.g. MC88000, MIPS; or

    2. conceiving a new architecture, e.g. Power1 (RS/6000), DEC Alpha, PowerPC.

- Superscalar CISC machines appeared much later (1993) owing to their higher complexity, e.g. Pentium, MC68060.

Superscalar Processing



Parallel Decoding

- Considerably more complex than in the case of scalar processors and becomes even more sophisticated as the issue rate increases.
- Higher issue rates can lengthen the decoding cycle or give rise to multiple decoding cycles unless decoding is enchanced.
- A scalar processor has to decode only a single instruction in each cycle, checking for dependencies in order to decide whether this instruction can be issued or not.

Parallel Decoding$_{(cont)}$

- A superscalar processor has to decode multiple instructions in a single clock cycle. It also has to check for dependencies from two perspectives:
  1. with respect to all of the instruction currently executing; and
  2. among the instructions which are candidates for the next issue.
- Since the processor has more execute units than a scalar one, the number of instructions in execution will be higher. Therefore, more comparisons have to be performed, making the *decode-issue path* a much more critical issue in achieving a high clock rate.

[Superscalar processors tend to require two or more pipeline cycles for decoding and issuing.]

Parallel Decoding$_{(cont)}$

- *Predecoding* shifts part of the decoding task to the loading phase of the on-chip instruction cache.
- While the I-cache is being loaded, a *predecode unit* performs partial decoding – appending the number of decode bits to each instruction.
- Typically, between 4 and 7 bits are attached indicating the instruction class, or type of resources required or even that the branch target addresses have been calculated.
- Predecoding either shortens the overall cycle time or reduces the number of cycles needed for decoding and issue.
- Most commercial processor now use predecoding.