Instruction-Level Parallelism: Limits

Politecnico di Milano

v1

Outline

- *Review*
  - ILP Definition
  - Superscalar Architectures, Static and Dynamic Schedulers
- Limits to ILP
  - Ideal machine
  - Limits
  - Examples of real architectures

Definition of ILP

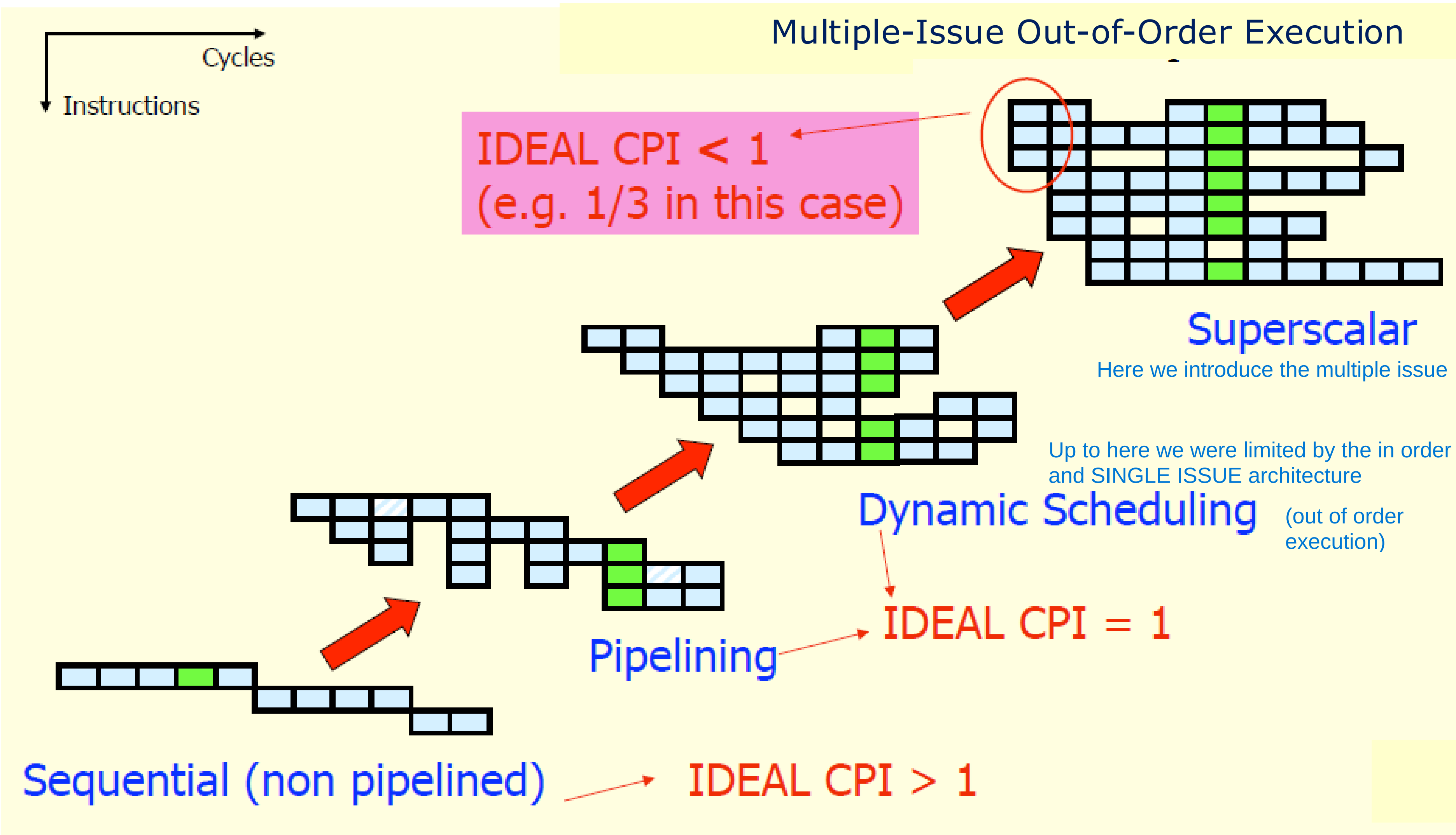
- ILP = Potential overlap of execution among unrelated instructions

- Overlapping possible if:
  - No Structural Hazards
  - No RAW, WAR of WAW Stalls
  - No Control Stalls

Try to find the largest number of instruction that we can execute in parallel. It can take a lot of time to check at runtime the conditions about the possibility that some instruction can be parallelized on the entire set of instruction.
That is, could be a good idea but difficult to apply to the entire set of instructions.

In Ideal conditions, the number of instructions we can execute in parallel is hight but not too impressive, so we don't actually need infinite instruction capability issue stage or infinite number of FU.

Several steps towards exploiting more ILP



Superscalar execution

- Why not more than one instruction beginning execution at each clock cycle?

Superscalar execution

- Why not more than one instruction beginning execution at each clock cycle?
- Key requirements:

Superscalar execution

- Why not more than one instruction beginning execution at each clock cycle?
- Key requirements:
  - Fetching more instructions per clock cycle (Fetch Unit): no major problem provided the instruction cache can sustain the bandwidth and can manage more requests at the same time

Superscalar execution

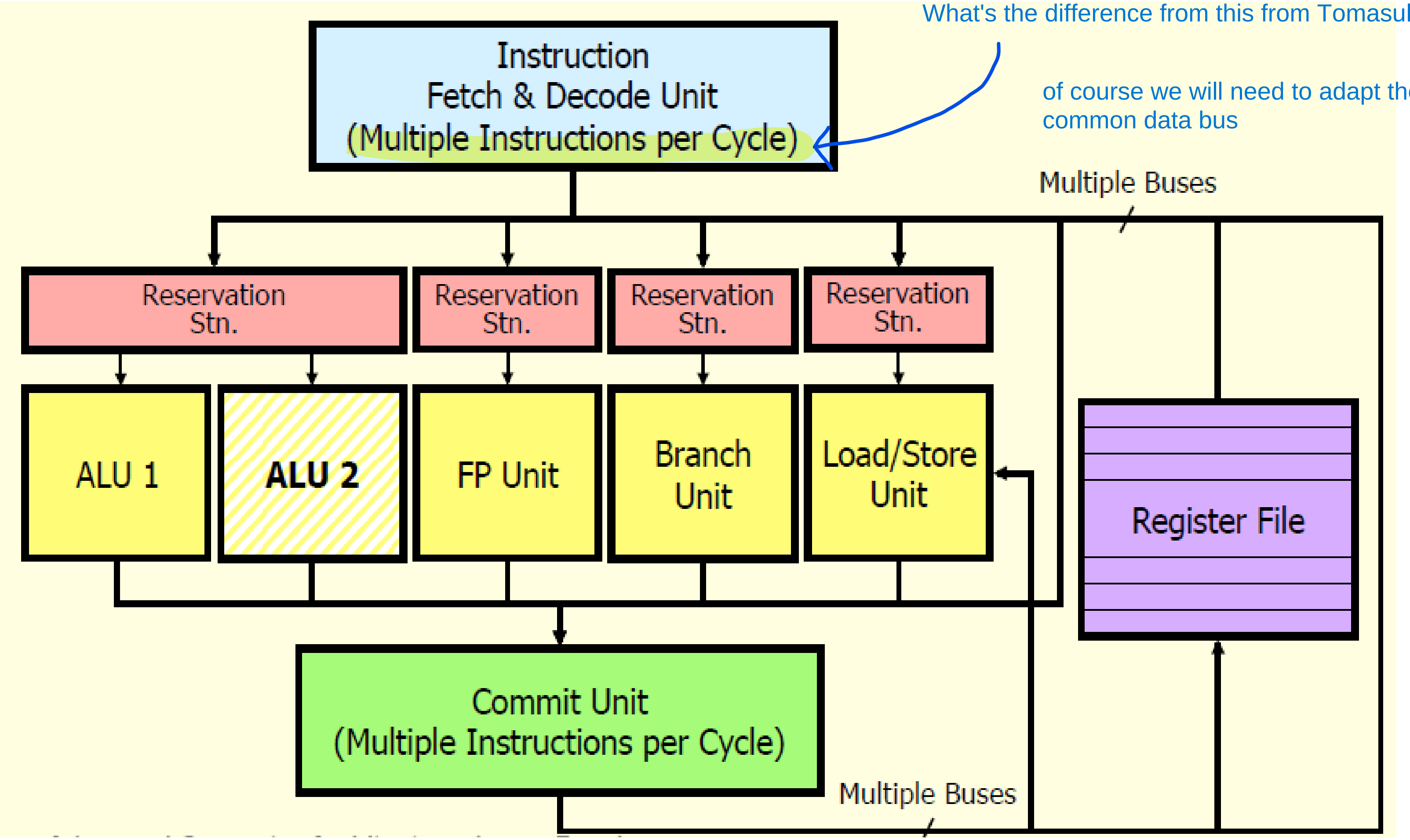
- Why not more than one instruction beginning execution at each clock cycle?
- Key requirements:
  - Fetching more instructions per clock cycle (Fetch Unit): no major problem provided the instruction cache can sustain the bandwidth and can manage more requests at the same time
  - Decide on data and control dependencies: dynamic scheduling and dynamic branch prediction

Beyond CPI = 1

- Superscalar:
  - Issue multiple instructions per clock-cycle
  - varying no. instructions/cycle (1 to 8), --> in reality we will have a number of issued instructions varying from 1 to 8
  - scheduled by compiler or by HW (Tomasulo)
  - e.g. IBM PowerPC, Sun UltraSparc, DEC Alpha, HP 8000, Pentium

- Anticipated success lead to use of Instructions Per Clock cycle (IPC) vs. CPI
- CPI_{ideal} = 1 / issue-width

Superscalar Processor



What's the difference from this from Tomasulo?

of course we will need to adapt the common data bus

Limits to ILP

Example of limiting factors to ILP

Assumptions for ideal/perfect machine to start:

1. Register renaming --> we introduced it because we were stalling for WAW and WAR. It was a technique to improve ILP actually, removing name dependencies that were a problem cause out-of-order, that we introduced to improve ILP
2. Branch prediction --> We want the best branch prediction
3. Jump prediction -->we didn't go really through a lot of this topic, but it's a problem of completing addresses pointed by different ILSs
4. Memory-address alias analysis
5. 1 cycle latency for all instructions

Limits to ILP

Assumptions for ideal/perfect machine to start:

1. Register renaming
   - infinite virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction
3. Jump prediction
4. Memory-address alias analysis
5. 1 cycle latency for all instructions

Limits to ILP

Assumptions for ideal/perfect machine to start:

1. Register renaming
   – infinite virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction
   – perfect; no mispredictions
3. Jump prediction
4. Memory-address alias analysis
5. 1 cycle latency for all instructions

Limits to ILP

Assumptions for ideal/perfect machine to start:

1. Register renaming
   – infinite virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction
   – perfect; no mispredictions
3. Jump prediction
   – all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. Memory-address alias analysis

5. 1 cycle latency for all instructions

Limits to ILP

Assumptions for ideal/perfect machine to start:

1. Register renaming
   – infinite virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction
   – perfect; no mispredictions
3. Jump prediction
   – all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. Memory-address alias analysis
   – addresses are known & a store can be moved before a load provided addresses not equal
5. 1 cycle latency for all instructions

Limits to ILP

In the ideal case we would have the following (ideal) assumption:

Assumptions for ideal/perfect machine to start:

1. Register renaming
   – infinite virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction
   – perfect; no mispredictions
3. Jump prediction
   – all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. Memory-address alias analysis
   – addresses are known & a store can be moved before a load provided addresses not equal
5. 1 cycle latency for all instructions
   – unlimited number of instructions issued per clock cycle

Initial assumptions

Initial assumptions

- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be

Initial assumptions

- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be executed in one cycle (including loads and stores);
- All functional unit latencies = 1; any sequence of

Initial assumptions

- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be executed in one cycle (including loads and stores);
- All functional unit latencies = 1; any sequence of depending instructions can issue on successive cycles;
- Perfect caches = all loads, stores execute in one cycle ⇨ only fundamental limits to ILP are taken into account.

Initial assumptions

- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be executed in one cycle (including loads and stores);
- All functional unit latencies = 1; any sequence of depending instructions can issue on successive cycles;
- Perfect caches = all loads, stores execute in one cycle ⇨ only fundamental limits to ILP are taken into account.
- Obviously, results obtained are VERY optimistic! (no such CPU can be realized…);
- Benchmark programs used: six from SPEC92 (three FP

Initial assumptions

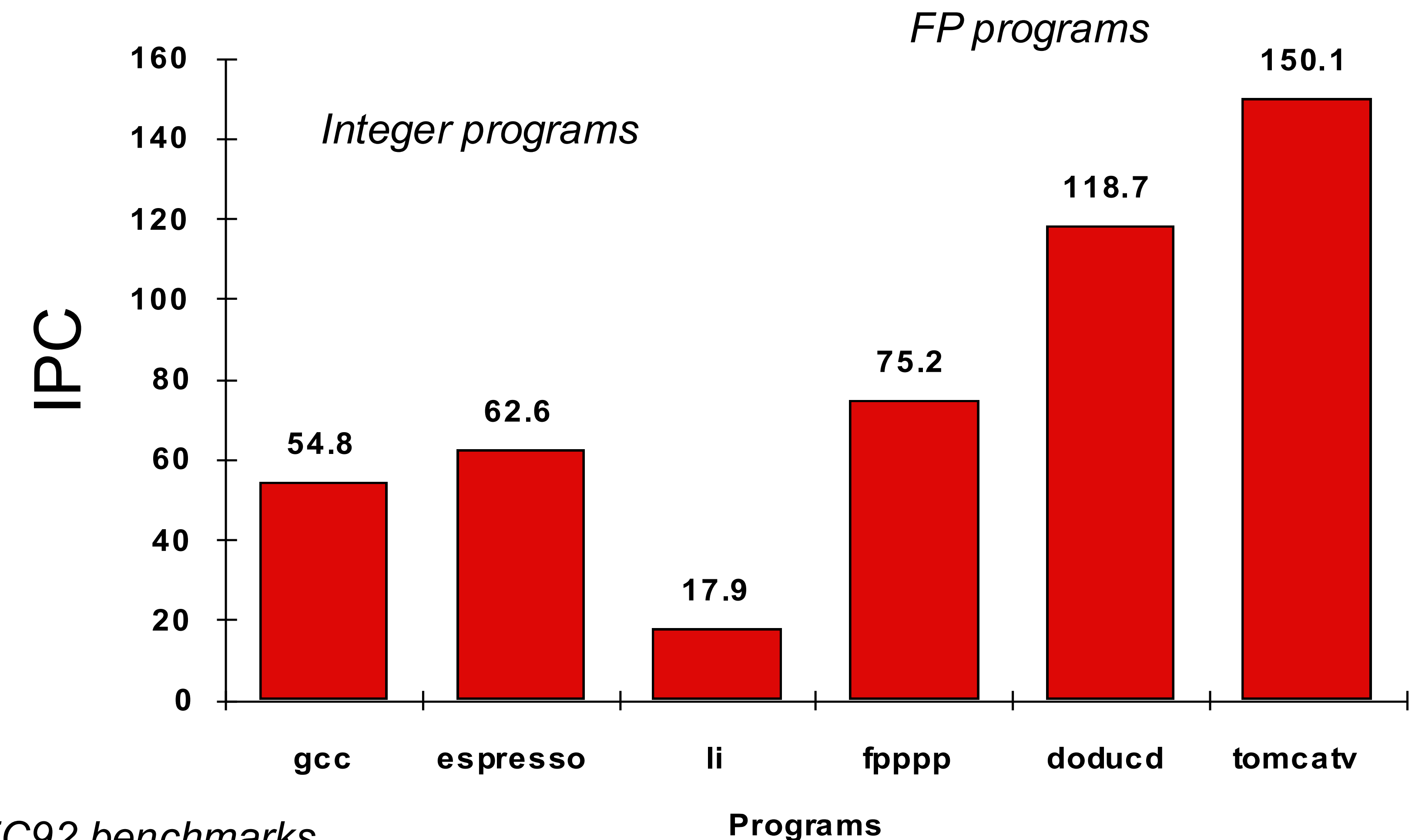
- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be executed in one cycle (including loads and stores);
- All functional unit latencies = 1; any sequence of depending instructions can issue on successive cycles;
- Perfect caches = all loads, stores execute in one cycle ⇨ only fundamental limits to ILP are taken into account.
- Obviously, results obtained are VERY optimistic! (no such CPU can be realized…);
- Benchmark programs used: six from SPEC92 (three FP-intensive ones, three integer ones).

Initial assumptions

- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be executed in one cycle (including loads and stores);
- All functional unit latencies = 1; any sequence of depending instructions can issue on successive cycles;
- Perfect caches = all loads, stores execute in one cycle ⇨ only fundamental limits to ILP are taken into account.
- Obviously, results obtained are VERY optimistic! (no such CPU can be realized…);
- Benchmark programs used: six from SPEC92 (three FP-intensive ones, three integer ones).

Upper Limit to ILP: Ideal Machine

With the ideal asumption above, those are the performance we cannot go further



SPEC92 benchmarks

Limits on window size

Said that there are some performance limits that we cannot go beyond of, we want to find something that can be physically implemented, the right trade off.
The news that in ideal case we are limited it's a good new since it means that in any case we are bounded.

Limits on window size

The following are the things we are going to play with, in order to achieve the best implementable framework

- Dynamic analysis is necessary to approach perfect branch prediction (impossible at compile time!);
- A perfect dynamic-scheduled CPU should:

Said that there are some performance limits that we cannot go beyond of, we want to find something that can be physically implemented, the right trade off

Limits on window size

- Dynamic analysis is necessary to approach perfect branch prediction (impossible at compile time!);
- A perfect dynamic-scheduled CPU should:
  1. Look arbitrarily far ahead to find set of instructions to issue, predict all branches perfectly;
  2. Rename all registers uses (⇨ no WAW, WAR hazards);

Limits on window size

- Dynamic analysis is necessary to approach perfect branch prediction (impossible at compile time!);
- A perfect dynamic-scheduled CPU should:
  1. Look arbitrarily far ahead to find set of instructions to issue, predict all branches perfectly;
  2. Rename all registers uses (⇨ no WAW, WAR hazards);
  3. Determine whether there are data dependencies among

Limits on window size

- Dynamic analysis is necessary to approach perfect branch prediction (impossible at compile time!);
- A perfect dynamic-scheduled CPU should:
  1. Look arbitrarily far ahead to find set of instructions to issue, predict all branches perfectly;
  2. Rename all registers uses (⇨ no WAW, WAR hazards);
  3. Determine whether there are data dependencies among instructions in the issue packet; rename if necessary;
  4. Determine if memory dependencies exist among issuing instructions, handle them;

Limits on window size

- Dynamic analysis is necessary to approach perfect branch prediction (impossible at compile time!);
- A perfect dynamic-scheduled CPU should:
  1. Look arbitrarily far ahead to find set of instructions to issue, predict all branches perfectly;
  2. Rename all registers uses (⇨ no WAW, WAR hazards);
  3. Determine whether there are data dependencies among instructions in the issue packet; rename if necessary;
  4. Determine if memory dependencies exist among issuing instructions, handle them;
  5. Provide enough replicated functional units to allow all ready instructions to issue.

Limits on window size

- Dynamic analysis is necessary to approach perfect branch prediction (impossible at compile time!);
- A perfect dynamic-scheduled CPU should:
  1. Look arbitrarily far ahead to find set of instructions to issue, predict all branches perfectly;
  2. Rename all registers uses (⇨ no WAW, WAR hazards);
  3. Determine whether there are data dependencies among instructions in the issue packet; rename if necessary;
  4. Determine if memory dependencies exist among issuing instructions, handle them;
  5. Provide enough replicated functional units to allow all ready instructions to issue.

Limits on instruction windows

- Size affects the number of comparisons necessary to determine RAW dependences (we need to find the size of the window in which we will perform comparisons between instruction in order to find a subset of instructions which can be issued together)
- Example: # comparisons to evaluate data dependences among n register-to-register instructions in the issue phase (with an infinite # of regs) =

$$2n-2+2n-4+...+2=2\sum_{i=1}^{n-1} i = 2\frac{(n-1)n}{2}=n^2-n$$

  - Window size = 2000 ↳ almost 4 Million comparisons!
  - Issue window of 50 instructions requires 2450 comparisons!

It'd be great to have that window as big as possible, but the biggest is going to imply higher number in terms of comparison that we have to do

- Today's CPUs: constraints deriving from the limited number of registers + search for dependent instructions + in-order issue

Limits on window size, maximum issue count

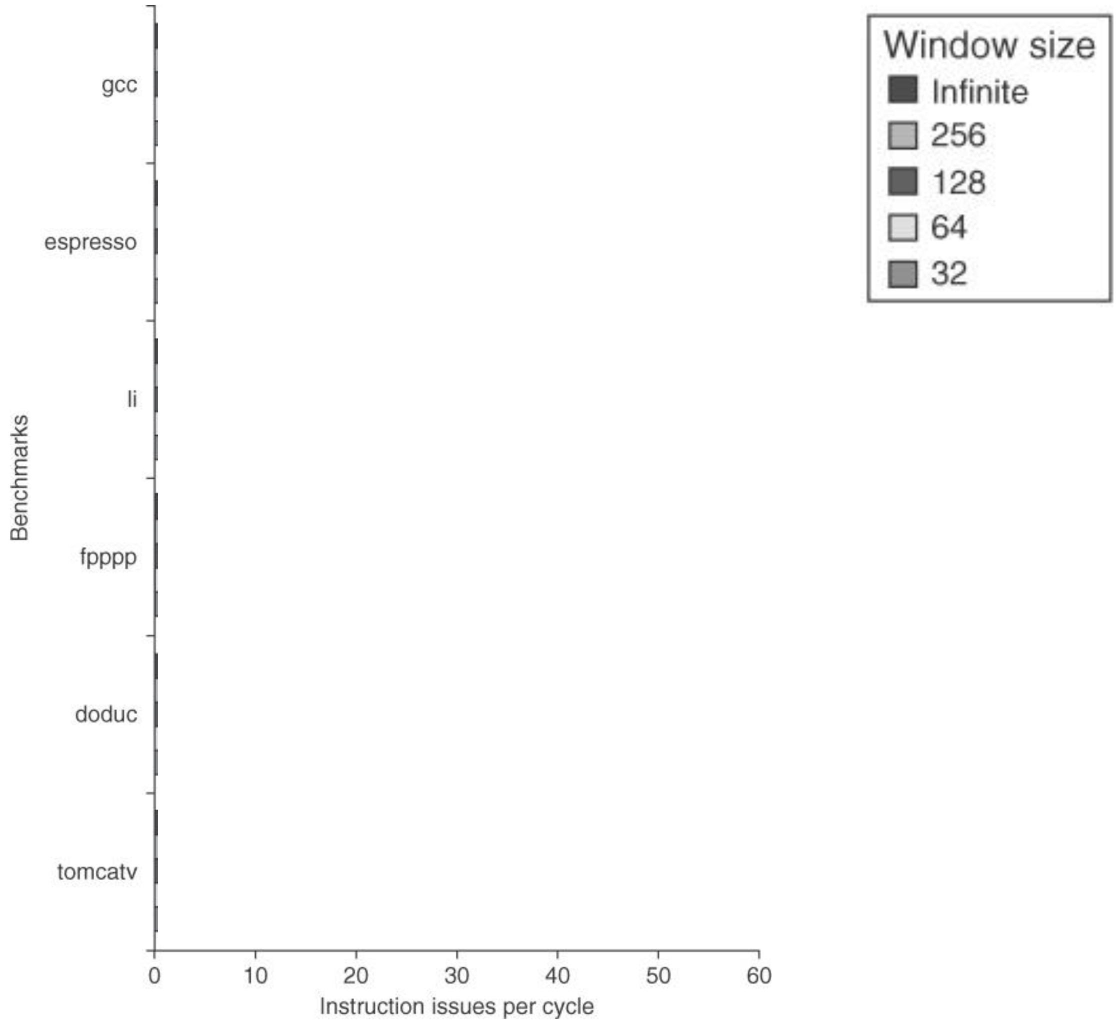
All instructions in the window must be kept in the processor

number of comparisons required at each cycle =
maximum completion rate x
window size x
number of operands per instruction ⇨
total window size limited by storage + comparisons + limited issue rate

(today: window size 32-200 ⇨ up to over 2400 comparisons!)

Amount of parallelism vs window size

Up to 64 arbitrary issues
Per clock cycle



SPEC92 benchmarks

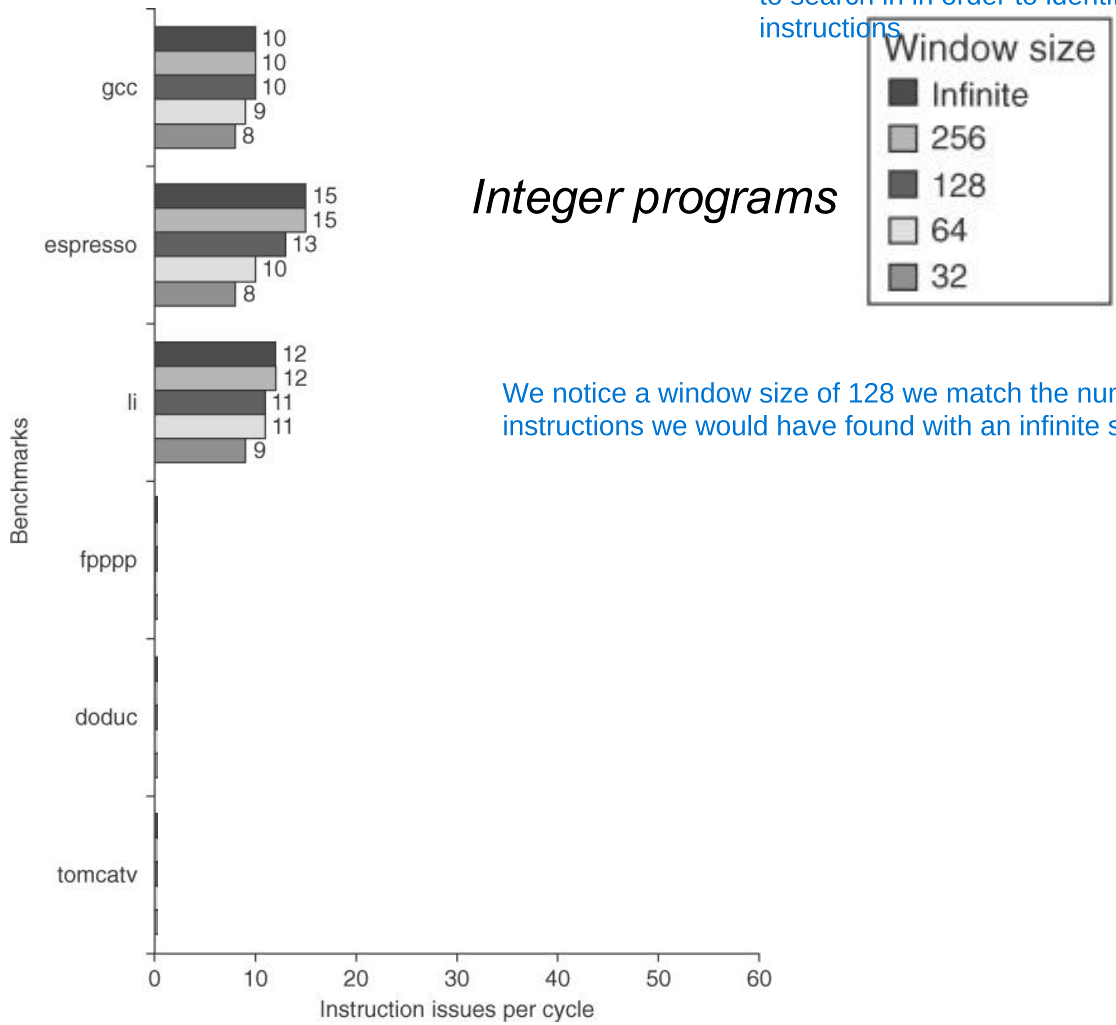
Amount of parallelism vs window size

--> that is how many instruction we are going to search in in order to identify the 64 instructions

Up to 64 arbitrary issues
Per clock cycle

Integer programs

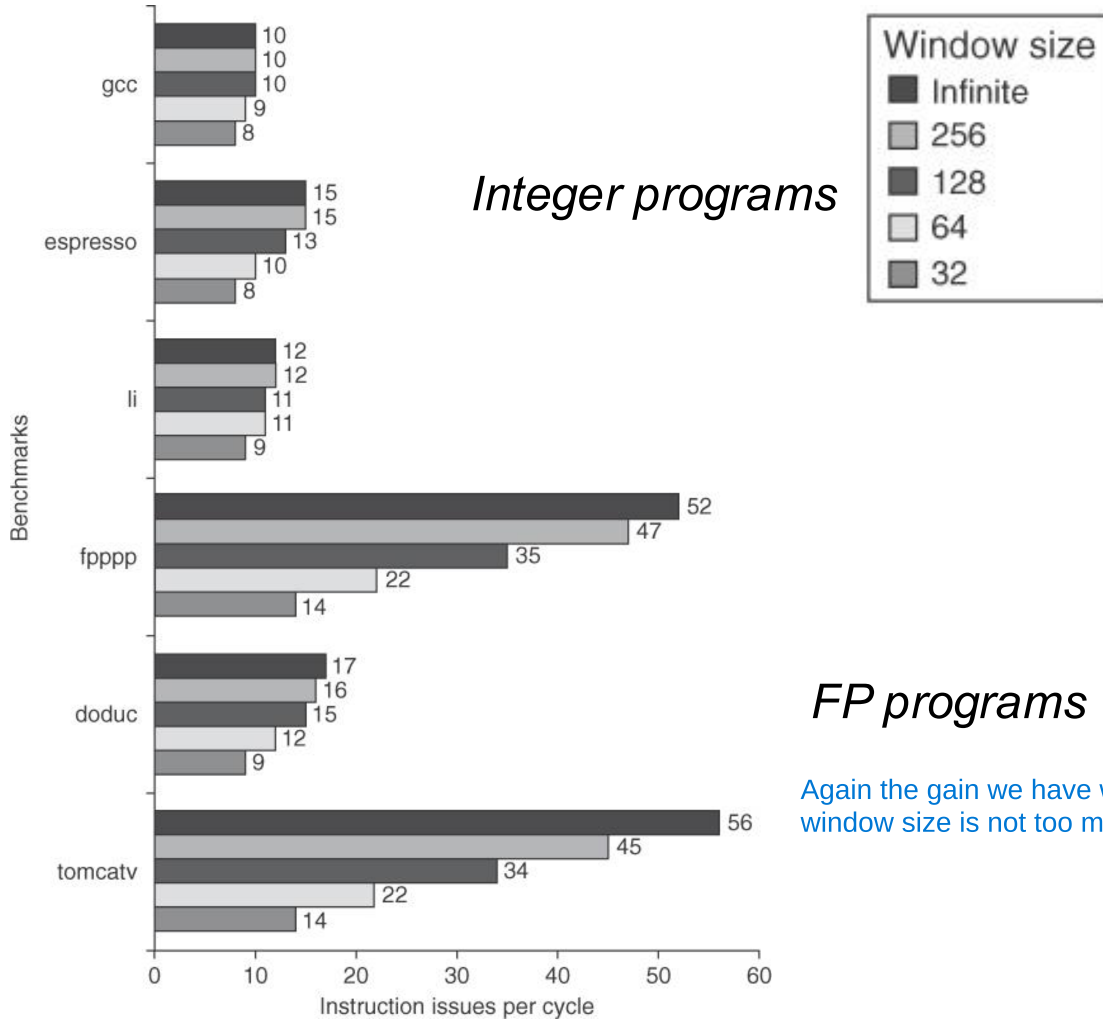
We notice a window size of 128 we match the number of instructions we would have found with an infinite size



SPEC92 benchmarks

Amount of parallelism vs window size

Up to 64 arbitrary issues
Per clock cycle



Integer programs

FP programs

Again the gain we have with a infinite window size is not too much

SPEC92 benchmarks

HW model comparison

	Ideal model	IBM Power 5 (2004-2006) Dual core @ 1.5 – 2.3 GHz
Instructions Issued per clock	Infinite	4
Instruction Window Size	Infinite	200
Renaming Registers	Infinite	48 integer + 40 FP
Branch Prediction	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	L1 (32KI+32KD)/core L2 1.875MB/core L3 36 MB/chip (off chip)
Memory Alias Analysis	Perfect	??

Other limits of today’s CPUs

Other limits of today's CPUs

- N. of functional units
  - For instance: not more than 2 memory references per cycle

Other limits of today's CPUs

- N. of functional units
  - For instance: not more than 2 memory references per cycle
- N. of busses

Other limits of today's CPUs

- N. of functional units
  - For instance: not more than 2 memory references per cycle
- N. of busses
- N. of ports for the register file

Other limits of today's CPUs

- N. of functional units
  - For instance: not more than 2 memory references per cycle
- N. of busses
- N. of ports for the register file

- All these limitations define that the maximum number of instructions that can be issued, executed or committed in the same clock cycle is much smaller than the window size

Issue-width limited in practice

- Now, the maximum (rare) is 6, but no more exists.
  - The widths of current processors range from single-issue (ARM11, UltraSPARC-T1) through 2-issue (UltraSPARC-T2/T3, Cortex-A8 & A9, Atom, Bobcat) to 3-issue (Pentium-Pro/II/III/M, Athlon, Pentium-4, Athlon 64/Phenom, Cortex-A15) or 4-issue (UltraSPARC-III/IV, PowerPC G4e, Core 2, Core i, Core i*2, Bulldozer) or 5-issue (PowerPC G5), or even 6-issue (Itanium, but it's a VLIW).
- Because it is too hard to decide which 8, or 16, instructions can

Issue-width limited in practice

- Now, the maximum (rare) is 6, but no more exists.
  - The widths of current processors range from single-issue (ARM11, UltraSPARC-T1) through 2-issue (UltraSPARC-T2/T3, Cortex-A8 & A9, Atom, Bobcat) to 3-issue (Pentium-Pro/II/III/M, Athlon, Pentium-4, Athlon 64/Phenom, Cortex-A15) or 4-issue (UltraSPARC-III/IV, PowerPC G4e, Core 2, Core i, Core i*2, Bulldozer) or 5-issue (PowerPC G5), or even 6-issue (Itanium, but it's a VLIW).
- Because it is too hard to decide which 8, or 16, instructions can execute every cycle (too many!)
  - It takes too long to compute
  - So the frequency of the processor would have to be decreased

Issue-width limited in practice

- Now, the maximum (rare) is 6, but no more exists.
  - The widths of current processors range from single-issue (ARM11, UltraSPARC-T1) through 2-issue (UltraSPARC-T2/T3, Cortex-A8 & A9, Atom, Bobcat) to 3-issue (Pentium-Pro/II/III/M, Athlon, Pentium-4, Athlon 64/Phenom, Cortex-A15) or 4-issue (UltraSPARC-III/IV, PowerPC G4e, Core 2, Core i*2, Bulldozer) or 5-issue (PowerPC G5), or even 6-issue (Itanium, but it's a VLIW).

TRADE-OFF

It id not that we cannot do more in term of performance, but these are the best trade off (es: with energy consumption)

- Because it is too hard to decide which 8, or 16, instructions can execute every cycle (too many!)
  - It takes too long to compute
  - So the frequency of the processor would have to be decreased

Intel P6 Family

Processor	First issue	Clock frequency	L1 cache	L2 cache
Pentium Pro	1995	100-200 MHz	8KB I + 8KB D	256-1025 KB
Pentium II	1998	233-450	16KB + 16KB	256-512
Pentium II Xeon	1999	400-450	16KB + 16KB	512-2 MB
Celeron	1999	500-900	16KB + 16KB	128
Pentium III	1999	450-1100	16KB + 16KB	256-512
Pentium III Xeon	2000	700-900	16KB + 16KB	1-2 MB

Example: Intel P6

Example: Intel P6

- P6 microarchitecture:
  - CPU with dynamic scheduling, translates every instruction IA-32 (IA aka Intel Architecture) in a sequence of micro-operations (uops) executed by the pipeline;

Example: Intel P6

- P6 microarchitecture:
  - CPU with dynamic scheduling, translates every instruction IA-32 (IA aka Intel Architectue) in a sequence of micro-operations (uops) executed by the pipeline;
- Micro-operations:
  - typical RISC instructions;

Example: Intel P6

- P6 microarchitecture:
  - CPU with dynamic scheduling, translates every instruction IA-32 (IA aka Intel Architectue) in a sequence of micro-operations (uops) executed by the pipeline;
- Micro-operations:
  - typical RISC instructions;

RISC???

RISC vs CISC Architectures

RISC vs CISC Architectures

- RISC
  - Reduced Instruction Set Computer
  - Examples: ARM, SPARC, MIPS, PowerPC, RISC-V

RISC vs CISC Architectures

- RISC --> smaller (es: embedded) CPUs went for RISCV
  - Reduced Instruction Set Computer
  - Examples: ARM, SPARC, MIPS, PowerPC, RISC-V

- CISC -> All big player went for this type of instruction in order to optimize better logic of CPU and get high performance operations
  - Complex Instruction Set Computer
  - Examples: x86, AMD

So at beginning CISC was for more performance CPUs while RISC was for embedded and energy efficient

RISC vs CISC Architectures

- RISC
  - Reduced Instruction Set Computer
  - Examples: ARM, SPARC, MIPS, PowerPC, RISC-V

- CISC
  - Complex Instruction Set Computer
  - Examples: x86, AMD

ISA wars: Understanding the relevance of ISA being RISC or CISC to performance, power, and energy on modern architectures

- Blem, Emily & Menon, Jaikrishnan & Vijayaraghavan, Thiruvengadam & Sankaralingam, Karthikeyan.
- ACM Transactions on Computer Systems, 33(1) 2015
- https://dl.acm.org/doi/10.1145/2699682

Back to the Intel P6 example

- P6 microarchitecture:
  - CPU with dynamic scheduling, translates every instruction IA-32 (IA aka Intel Architectue) in a sequence of micro-operations (uops) executed by the pipeline;
- Micro-operations:
  - typical RISC instructions;

Back to the Intel P6 example

- P6 microarchitecture:
  - CPU with dynamic scheduling, translates every instruction IA-32 (IA aka Intel Architectue) in a sequence of micro-operations (uops) executed by the pipeline;
- Micro-operations:
  - typical RISC instructions;
- In every clock cycle read, decode and translate up to three IA-32 instructions into micro-ops

Maybe the entire instruction is not parallelizable but the sub-operations can be

The Intel P6 example

- 3 way superscalar
- Basic Idea, three engines

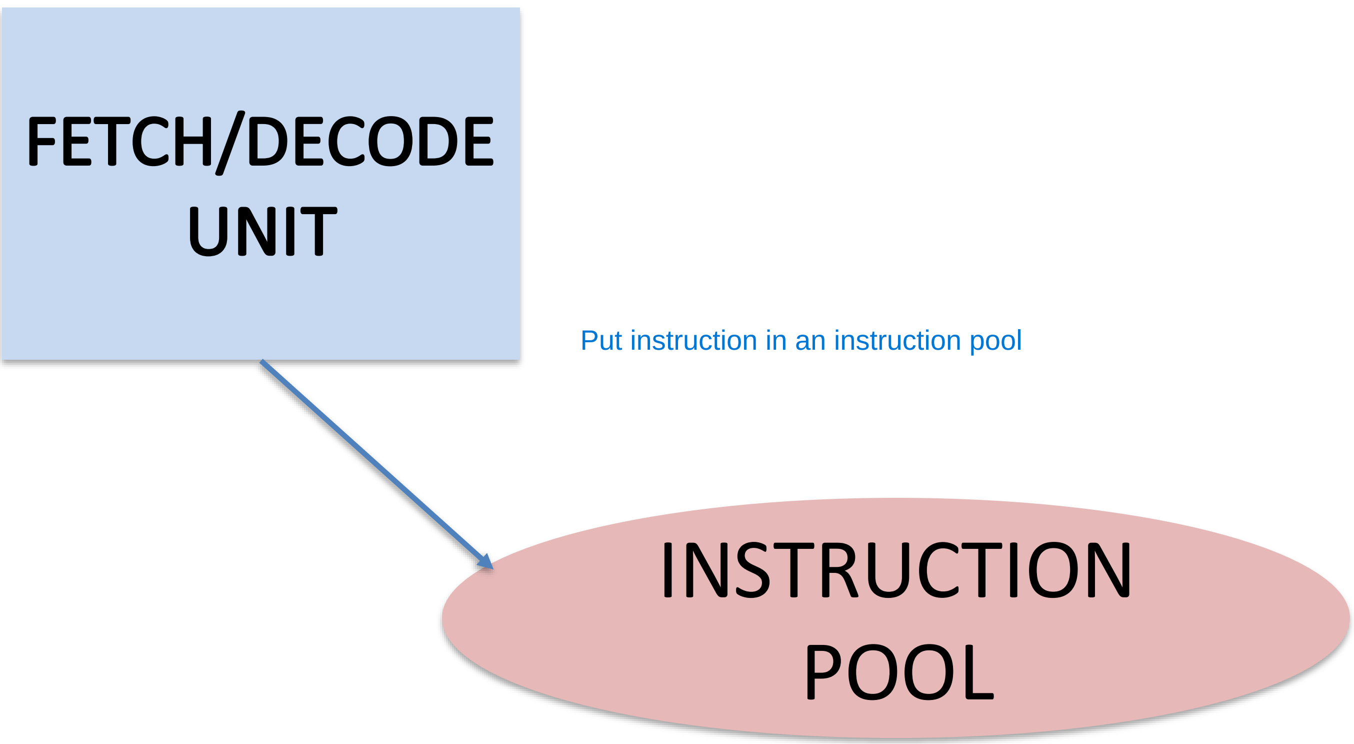
P6 Pipeline

- Fetch/Decode Unit: it decodes instructions and it puts them in the instruction pool in-order.
  - It converts the instructions in micro-ops that represent instruction code executed by the pipeline
    - The micro-ops are typical RISC instructions
    - In each clock cyle: fetch and decode up to 3 instructions

Micro-ops are the one that are actually executed by the pipeline

The Intel P6 example

- 3 way superscalar
- Basic Idea, three engines

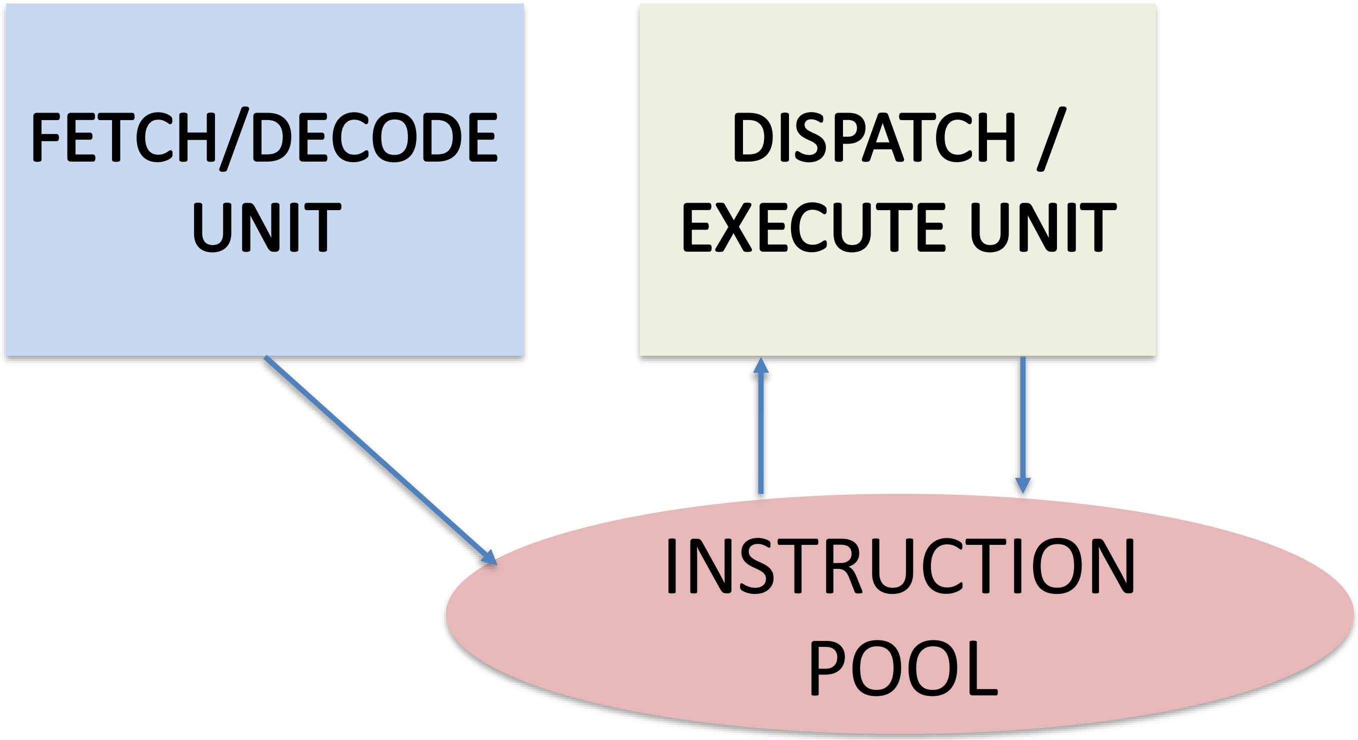


P6 Pipeline

- Fetch/Decode Unit: it decodes instructions and it puts them in the instruction pool in-order.
  - It converts the instructions in micro-ops that represent instruction code executed by the pipeline
    - The micro-ops are typical RISC instructions
    - In each clock cyle: fetch and decode up to 3 instructions
- Dispatch/Execute Unit: out-of-order issue from the instruction pool in a reservation station and out-of-order execution of micro-ops.

The Intel P6 example

- 3 way superscalar
- Basic Idea, three engines



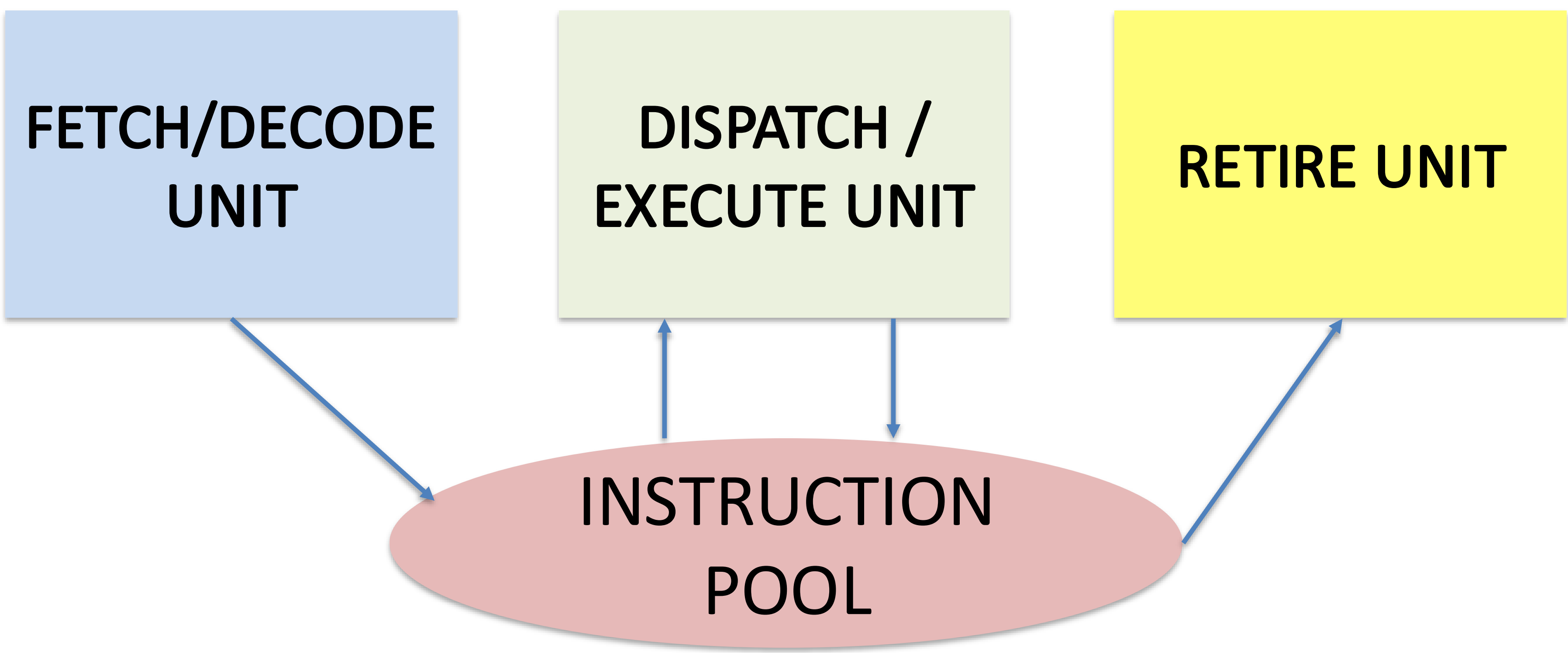
P6 Pipeline

- Fetch/Decode Unit: it decodes instructions and it puts them in the instruction pool in-order.
  - It converts the instructions in micro-ops that represent instruction code executed by the pipeline
    - The micro-ops are typical RISC instructions
    - In each clock cyle: fetch and decode up to 3 instructions
- Dispatch/Execute Unit: out-of-order issue from the instruction pool in a reservation station and out-of-order execution of micro-ops.
- Retire Unit: Reorders the instructions and commits speculative results to the architectural state.

The Intel P6 example

- 3 way superscalar
- Basic Idea, three engines

Here the interaction with register file is completed

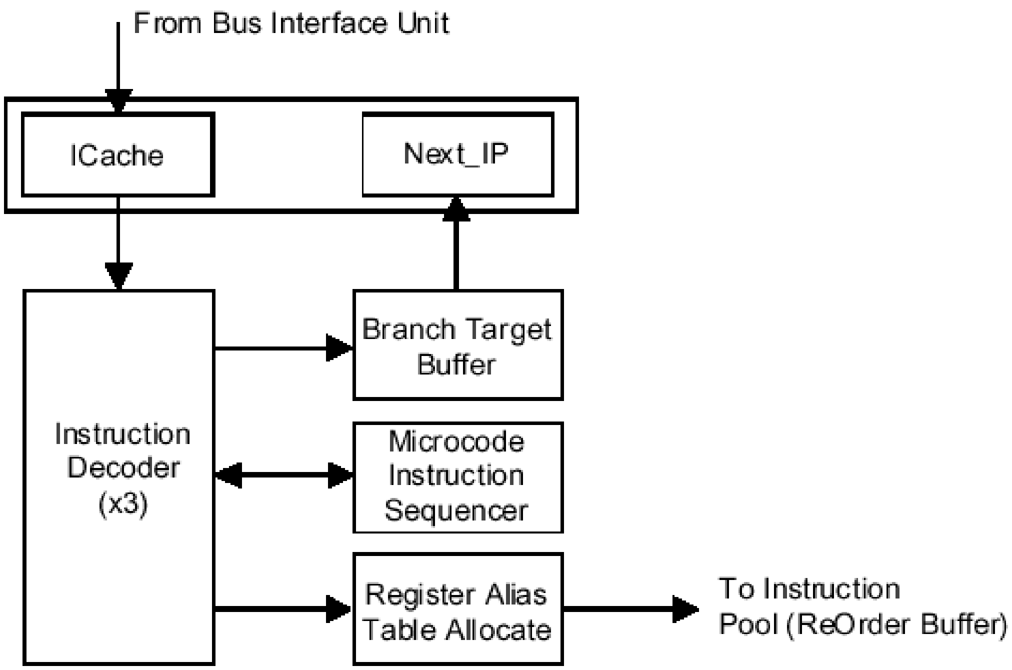


P6: 14 pipeline stages

P6: 14 pipeline stages

- 8 stages for the in-order issue, decode and dispatch
  - The next instruction is selected during the IF stage using a 2-levels branch predictor with 512 elements
  - Decode and issue include register renaming with 40 virtual registers
  - Dipatch to one of the 20 reservation stations and to one of the 40 positions of the Reorder Buffer

P6 Instruction Decode



8 pipeline stages

- The decoder fetches 16 bytes at each clock cycle from the cache
- 3 parallel decoders convert most of the instructions into one or more triadic micro-ops. Some instruction need microcode (several micro-ops) to be executed. Throughput=6 microops per clock cycle.
- Register Alias Table unit converts logical reg. ref. into virtual reg. ref. (40 registers).
- In-order Issue to reservation stations and reorder buffer

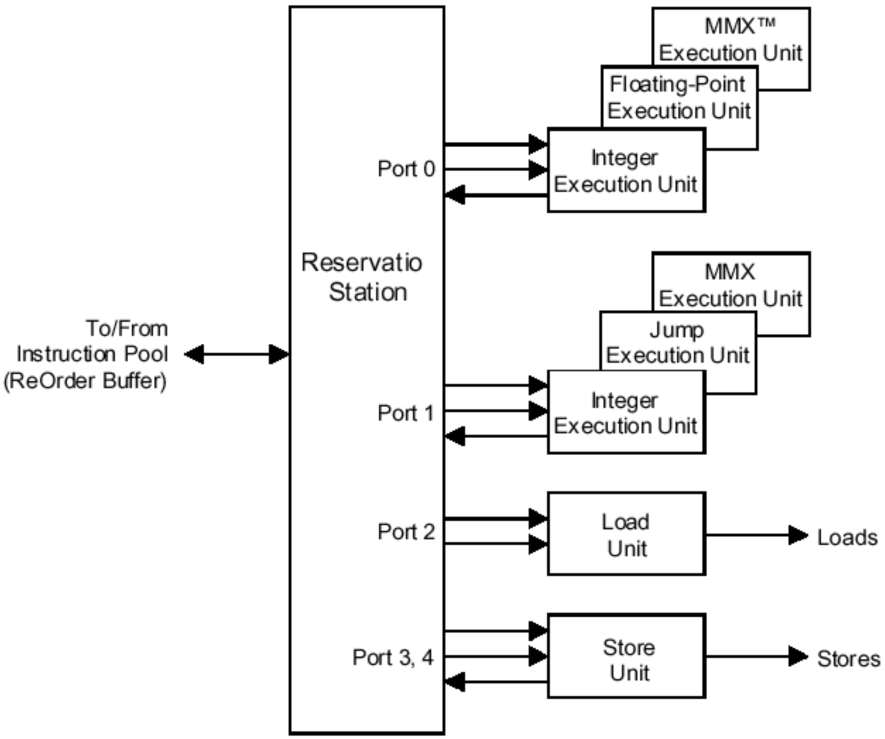
P6: 14 pipeline stages

- 8 stages for the in-order issue, decode and dispatch
  - The next instruction is selected during the IF stage using a 2-levels branch predictor with 512 elements
  - Decode and issue include register renaming with 40 virtual registers
  - Dipatch to one of the 20 reservation stations and to one of the 40 positions of the Reorder Buffer
- 3 stages for the out-of-order execution to one of the functional units (5 types: FX, FP, branches, memory addressing, memory access)
  - Execution pipeline: 1 clock cycle (simple ALU ops) up to 32 (FP division)

P6 Instruction Dispatch/Execute

20 entries RS

3 pipeline stages



- Out of order execution through the reservation station unit
- This happens when:
  - All the operands are ready
  - The resource needed is ready.
- Maximum throughput: 5 micro-ops/cycle.

If micro-ops are branches, their execution is compared with the predicted address (in the Fetch phase). If mispredicted the JEU changes the status of all the micro-ops behind the branch and removes them from the instruction pool.

P6: 14 pipeline stages

- 8 stages for the in-order issue, decode and dispatch
  - The next instruction is selected during the IF stage using a 2-levels branch predictor with 512 elements
  - Decode and issue include register renaming with 40 virtual registers
  - Dipatch to one of the 20 reservation stations and to one of the 40 positions of the Reorder Buffer
- 3 stages for the out-of-order execution to one of the functional units (5 types: FX, FP, branches, memory addressing, memory access)
  - Execution pipeline: 1 clock cycle (simple ALU ops) up to 32 (FP division)
- 3 stages for commit

P6 Instruction Retire

- The retire unit looks for micro-ops that have been executed and can be removed from the pool.
- The original architectural target of the micro-ops is written.
- This is done in-order by committing an instruction only if:
  - Previous instructions have been committed
  - The instruction has been executed.
- Up to 3 micro-ops can be retired at each clock cycle.

Current Superscalar & VLIW processors

- Dynamically-scheduled superscalar processors are the commercial state-of-the-art for general purpose: current implementations of Intel Core i, PowerPC, Alpha, MIPS, SPARC, etc. are all superscalar

Current Superscalar & VLIW processors

- Dynamically-scheduled superscalar processors are the commercial state-of-the-art for general purpose: current implementations of Intel Core i, PowerPC, Alpha, MIPS, SPARC, etc. are all superscalar
- VLIW processors are primarily successful as embedded media processors for consumer electronic devices (embedded):
  - TriMedia media processors by NXP
  - The C6000 DSP family by Texas Instruments
  - The ST200 family by STMicroelectronics
  - The SHARC DSP by Analog Devices
  - Itanium 2 is the only general purpose VLIW, a ‘hybrid’ VLIW (EPIC, Explicitly Parallel Instructions Computing)

Taxonomy of Multiple Issue Machines

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Limits to ILP

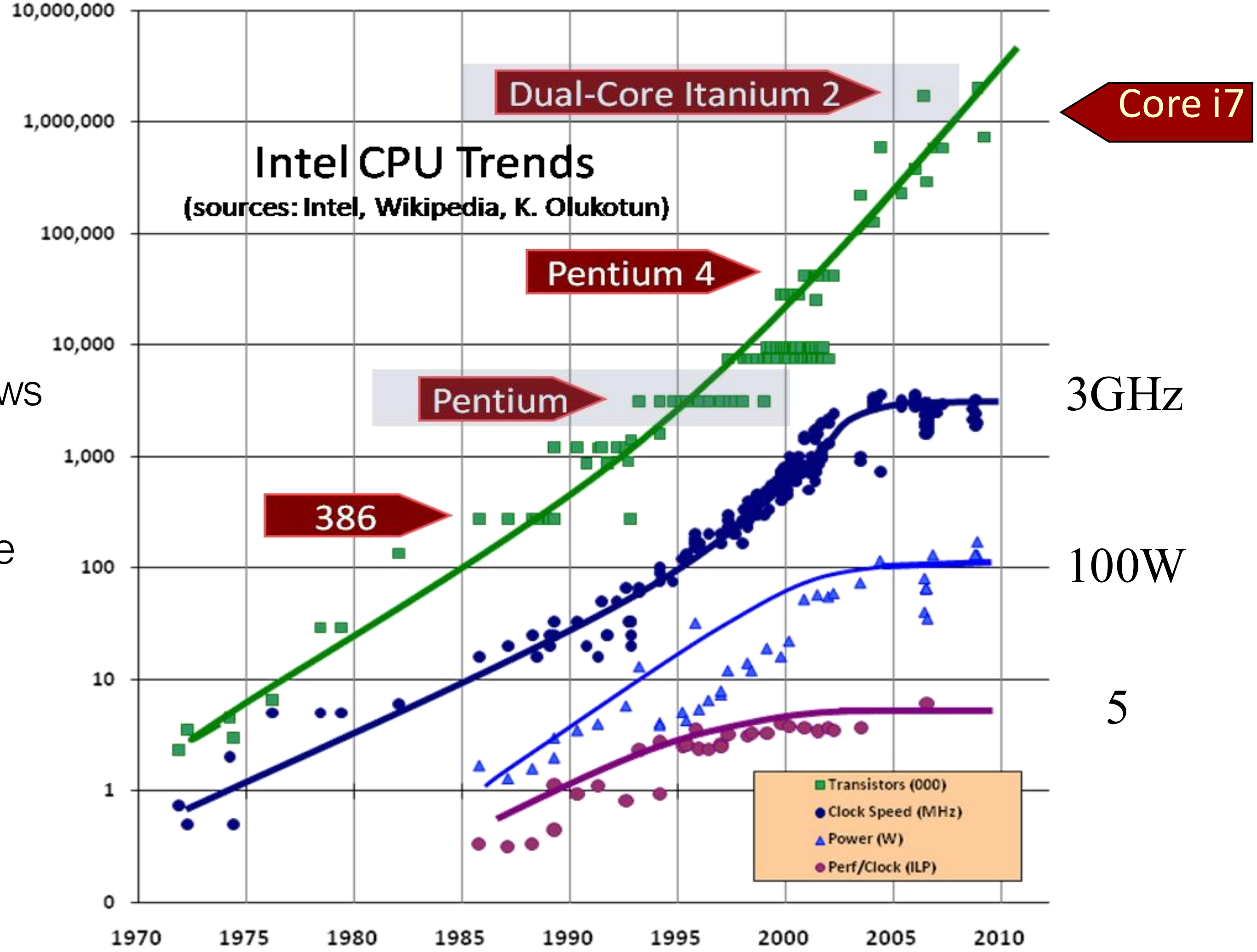
- Doubling issue rates above today's 3-6 instructions per clock, say to 6 to 12 instructions, probably requires a processor to
  - issue 3 or 4 data memory accesses per cycle,
  - resolve 2 or 3 branches per cycle,
  - rename and access more than 20 registers per cycle, and
  - fetch 12 to 24 instructions per cycle.
- The complexities of implementing these capabilities is likely to mean sacrifices in the maximum clock rate
  - E.g, widest issue processor is the Itanium 2, but it also has the slowest clock rate, despite the fact that it consumes the most power!

Limits to ILP

- Most techniques for increasing performance increase power consumption
- The key question is whether a technique is energy efficient
  - Does it increase power consumption faster than it increases performance?
- Multiple issue processors techniques all are energy inefficient:
  - Issuing multiple instructions incurs some overhead in logic that grows faster than the issue rate grows
  - Growing gap between peak issue rates and sustained performance
- Number of transistors switching = f(peak issue rate), and performance = f(sustained rate),
  growing gap between peak and sustained performance
  increasing energy per unit of performance

Next?

Shrinking and adding more components does not pay back anymore



Trends:

- #transistors follows Moore
- but not freq. and performance/core

Conclusions

- 1985-2002: >1000X performance (55% /year) for single processor cores
- Hennessy: industry has been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism and (real) Moore's Law to get 1.55X/year
  - Caches, (Super)Pipelining, Superscalar, Branch Prediction, Out-of-order execution, Trace cache
- **After 2002 slowdown** (about 20%/year increase)

Conclusions (cont'd)

- ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler/HW?

- Further problems:

Conclusions (cont'd)

- ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler/HW?

- Further problems:
  - Processor-memory performance gap
  - VLSI scaling problems (wiring)
  - Energy / leakage problems

Conclusions (cont'd)

- **ILP limits**: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler/HW?

- Further problems:
  - Processor-memory performance gap
  - VLSI scaling problems (wiring)
  - Energy / leakage problems

We can do better computer architectures designs

- However: other forms of parallelism come to rescue:
  - going Multi-Core
  - SIMD revival – Sub-word parallelism

Instruction-Level Parallelism: Limits

Politecnico di Milano

v1