Limits to Speculation

Hardware based

Multiple instruction machines find branches an impediment to creating ILP

Branch prediction: fetches and issues instructions
Speculation: fetches, issues and executes instructions

Speculation involves
dynamic branch prediction, chooses which instructions to execute
speculation actually executes instructions before the branch is resolved
dynamic scheduling to deal with different combinations of basic blocks

Principle – predicted flow of data values determines when to execute instructions. *Data flow execution*. Operations proceed as soon as their operands are available.

Before we split execution into issue and execute. Here we split execution into execute and commit. We insist on *in-order* commit.

Commit

Principle is instructions are issued and execute and pass their results onto further instructions.

But not to make irreversible updates. Such as raising an exception.

To hold these results, computed but not committed a further set of buffers is required.

Re-order buffer ROB
Holds result between complete and commit.

ROB thus must provide operands to subsequent instructions, until commit

Rob Entry Instruction type	destination field	value field	ready field
branch Store load or ALU	register # memory loc	value until commit	yes if complete

Execution with Speculation

I*ssue*
get an instruction from the instruction queue
Issue if there is space in the *reservation station* (holds instructions between issue and execute) and space in ROB (holds between execution and commit). Update to show buffers in use.
A ROB entry is allocated for the result, this tags the instruction - its value is sent to the reservation station.
else stall

There must be somewhere to hold the output state during the next clock cycle. Where ever data must be stable during a cycle there must be a buffer. Execute/commit break means we provide another buffer.

Execute
Wait for any outstanding operands to become available. Check for RAW hazards

Execution with Speculation

Write result
Broadcast the result with the ROB tag and write to ROB plus any reservation stations waiting on result. Reservation station for this instruction is released.
If the command is a store and available write to the value field. Not available, then wait for broadcasted value

Commit
ROB entry reaches head of the queue
Is the instruction a store, an incorrect branch, other instruction.
If result is present in the buffer update the target register with the result . For a store update the memory. Incorrect branch, ROB flushed and execution restarted at correct point.

Head of queue implies time for in order commit

ROB slot reclaimed
If the ROB fills execution issue stalls.

Reservation Station (beginning)
ROB (End)
Separate checks required.

Alternative to ROB

Register renaming

End of execution copy result to buffer.
Hold in buffer until make permanent, then copy to register.

Suppose bypass the buffer and copy straight to the register.
Same thing as long as the register is untouched until the commit stage is reached.
Better only one data copy.

Head of queue implies time for in order commit

But will rapidly run out of registers.
Increase the number of registers. 20-80
A register may be a *temporary* register or an *architecturally visible* register.

WAW & WAR handled by renaming.
Speculation because physical register is not architectural until commit.

When a physical register is mapped to an architectural register the physical register previously mapped is *freed*

Register renaming

When to free a physical register?
When no longer needed.

When another instruction that writes to the same architectural register commits.

Must be free or else the other register could not have arrived at the head of the speculation queue.

May be earlier – when no functional unit designates the physical register as a source and has been superseded as architectural register.

Note difference in allocation and write

Architectural register commit is later, but easier to implement.

Speculation disadvantages

Additional on chip resources. Space and power
Recovery takes resources.
Erroneous speculation takes expensive resources which are pointless – such as second level cache misses.

Processors may stall speculative events when they require expensive resources until not speculative.

Which is better speed or clever ILP (instruction level parallelism

Speed comes from
1.Underlying material physics: Si, Ge
2.Material processing: feature size. Smaller size faster processing
3.Power: higher power faster switching
4.Complexity: high complexity slower.

It depends !

Driving feature size were the foundaries, this was fixed when SUN/HP/IBM/Dec …. Were competing. Intel (AMD) still have to decide where to work

Note difference in allocation and write

Power. Is it better to run two low powered processors or one high powered. Performance requirement / application specific

At given power and feature size, should we go for simpler and faster architecture.
Or clever algorithms with much more hardware.
Still it depends – affects clock rate. CPI v GHz.
8 cores v. 10 cores
Release in March v September
Yield of 50% v 60%?

X-box v. Playstation 3

Speculation

Limits : Ideal machine

What is a perfect ILP machine.

1. *Register renaming* – infinite virtual registers
2. => all register WAW & WAR hazards are avoided

2. *Branch prediction* – perfect; no bad predictions

3. *Jump prediction* – all jumps perfectly predicted (returns, case statements)

4. *Memory-address alias analysis* – addresses known & a load can be moved before a store provided addresses not equal

5. *Perfect caches* – 1 cycle latency

2&3 no control dependencies; perfect speculation & an unbounded buffer of instructions available

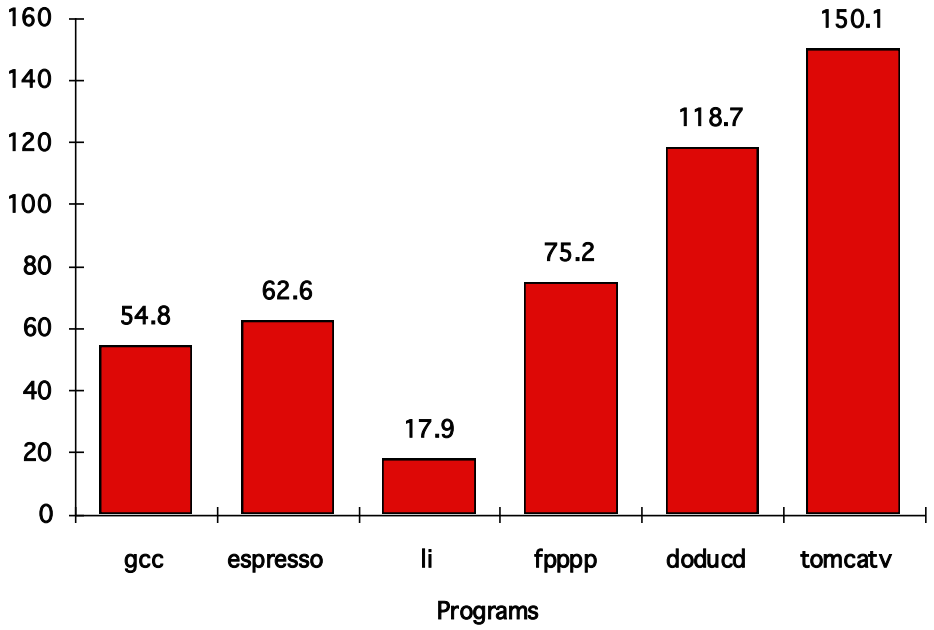
1&4 only true data dependencies

5 depends on ILP to hide cache misses. Possible but optimistic

A bit like a Carnot engine in thermodynamics

Speculation

Ideal machine performance



Numbers from Hennessy on Spec 95

Average ~ 70

Window size
Set of instructions examined for simultaneous execution

Perfect machine is not infinitely fast.

Instr per clock	Ideal ∞	actual 4
Instr Window size	Ideal ∞	actual 200
Rename registers	Ideal ∞	48 int, 40 FP
Branch prediction	Ideal 100%	94%-98%
Cache	perfect	Levels
Memory alias	perfect	??

Actual from '05. Instr per clock is 4 issue, upto 6 initiates.

200 instructions in flight!

How do the different imperfections affect the performance. How far are we from ideal

Window size and maximum issue

Perfect machine

1.Looks arbitrarily far ahead to find instructions with perfect branch prediction
2.Rename all registers to avoid WAR and WAW
3.Find data dependencies
4.Find memory dependencies
5.Have enough functional units

Each instruction in the window must be permanently in the cpu. Hard.
Worse we must check every instruction every cycle.
Number of checks is window size × operations in flight × operands per instruction.
For the example 6 × 200 × 2 = 2400.
Increasing window size has serious overheads.

12 How much possible

Best

Started by observing an average of 70 improvement possible.

Running at less than 2% efficiency.

Saw a couple of places where improves of 3 may be possible.

People are looking at various other ways of improving things.

Speculating along multiple branches. It seems likely that the system would run faster if we could examine all possible branches. Branches must be parallel, but of course the same variables may exist in more than one branch, which causes problems. The number of paths clearly grows exponentially, limiting its use in complicated.

Real but unnecessary dependences. Loop counter for instance.

Speculation

How much better can we do?

Feature size still shrinking.
natural speed up
Moore's Law still good

So more transistors means more functional units, more parallel execution – more wasted effort

Faster – but green issues

Smaller units – ubiquitous computing

Latest AMD chip coming out this year. Duplication of functional units looks very like the CDC6600.

Not exactly revolutionary.

Computers are faster – but designers are no smarter!