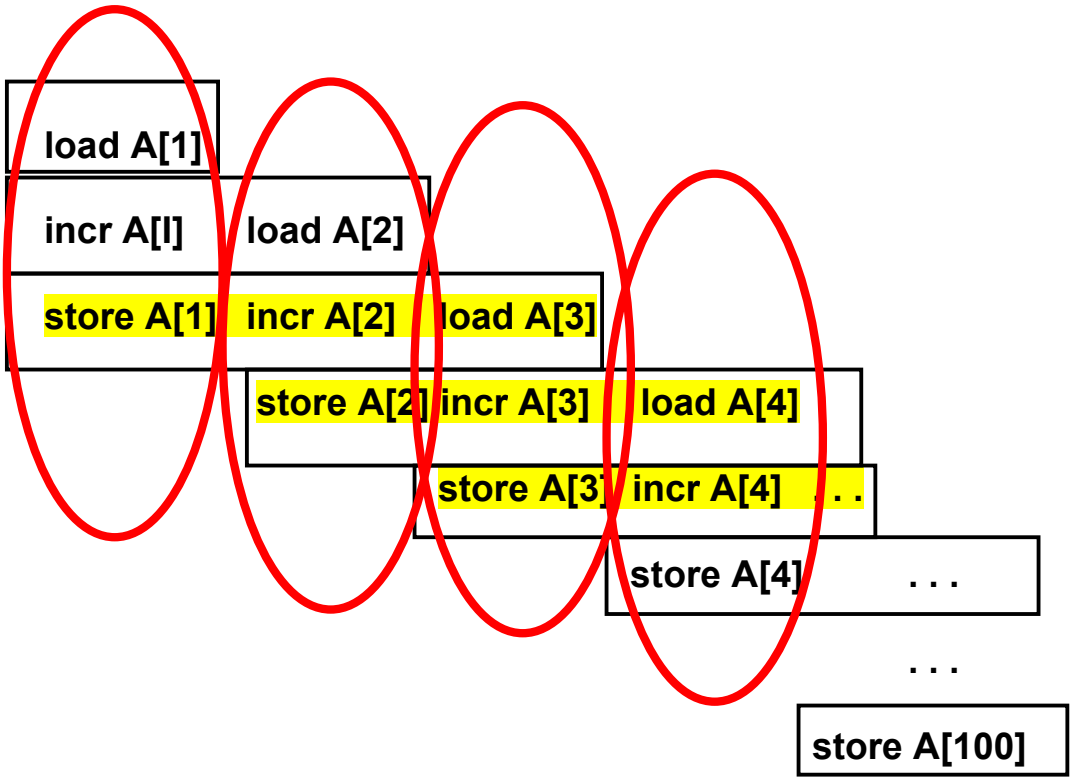
Software Pipelining

```
for (i=1, i<100, i++) {
    x := A[i];
    x := x+1;
    A[i] := x
    }
```

An alternative method of reorganizing loops to increase Instruction level Parallelism.



Now group these instructions horizontally as they are shown in the rectangular boxes. After the first three instructions, the loop can be restructured as

```
for (i=1, i<98, i++) {
        store A[i]
        incr A[i+1]
        load A[i+2]
}
```

These can be executed in parallel.

No data dependency exists

This is followed by the last three instructions.

More examples of compiler support to ILP

Problems with loop –carried dependence

Example 1 . The following type of computation is found in many recurrence relations:

```
for (i=2; i <= 100; i=i+1) {
        y(i) = y(i-1) + y(i)
}
```

Now unroll it, and notice the loop-carried dependencies.

```
y(2) = y(1) + y(2)
y(3) = y(2) + y(3)      } ILP suffers
y(4) = y(3) + y(4)
```

Loop unrolling does not help.

Example 2 Consider the following program

```
for (i=6; i <= 100; i=i+1) {
    y(i) = y(i-5) + y(i)
}
```

```
y(6) = y(1) + y(6)
y(7) = y(2) + y(7)
y(8) = y(3) + y(8)
```

} ILP improves!

Loop unrolling helps!

Pentium Micro-architecture

Pentium 3 > Pentium Pro > Pentium 4

Instruction length 1-17 bytes, awkward for pipelining

All decode IA-32 instructions into micro-operations (MIPS like instructions) since it makes pipelining easier. Complex instructions requiring many cycles are executed by standard micro-programmed control.

Up to four micro-operations scheduled per clock

Speculative pipeline

Multiple functional units (7 in Pentium vs 5 in Pro)

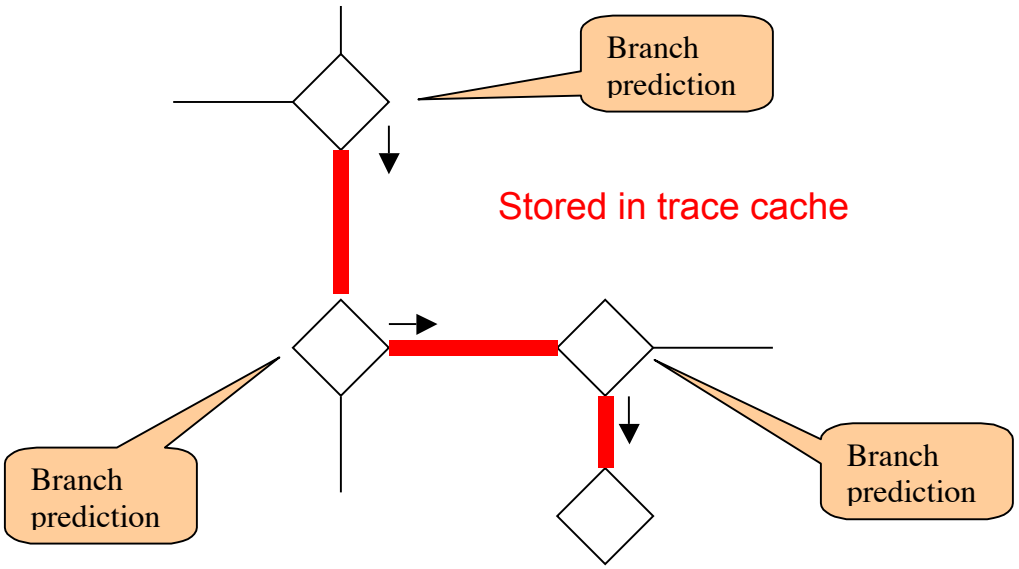
Pipeline depth = 20 (vs. 10 in Pentium Pro)

Uses trace cache (it has its own 512 entry BTB)

Better Branch Prediction (4K size in Pentium 4 vs. 512 in Pro)

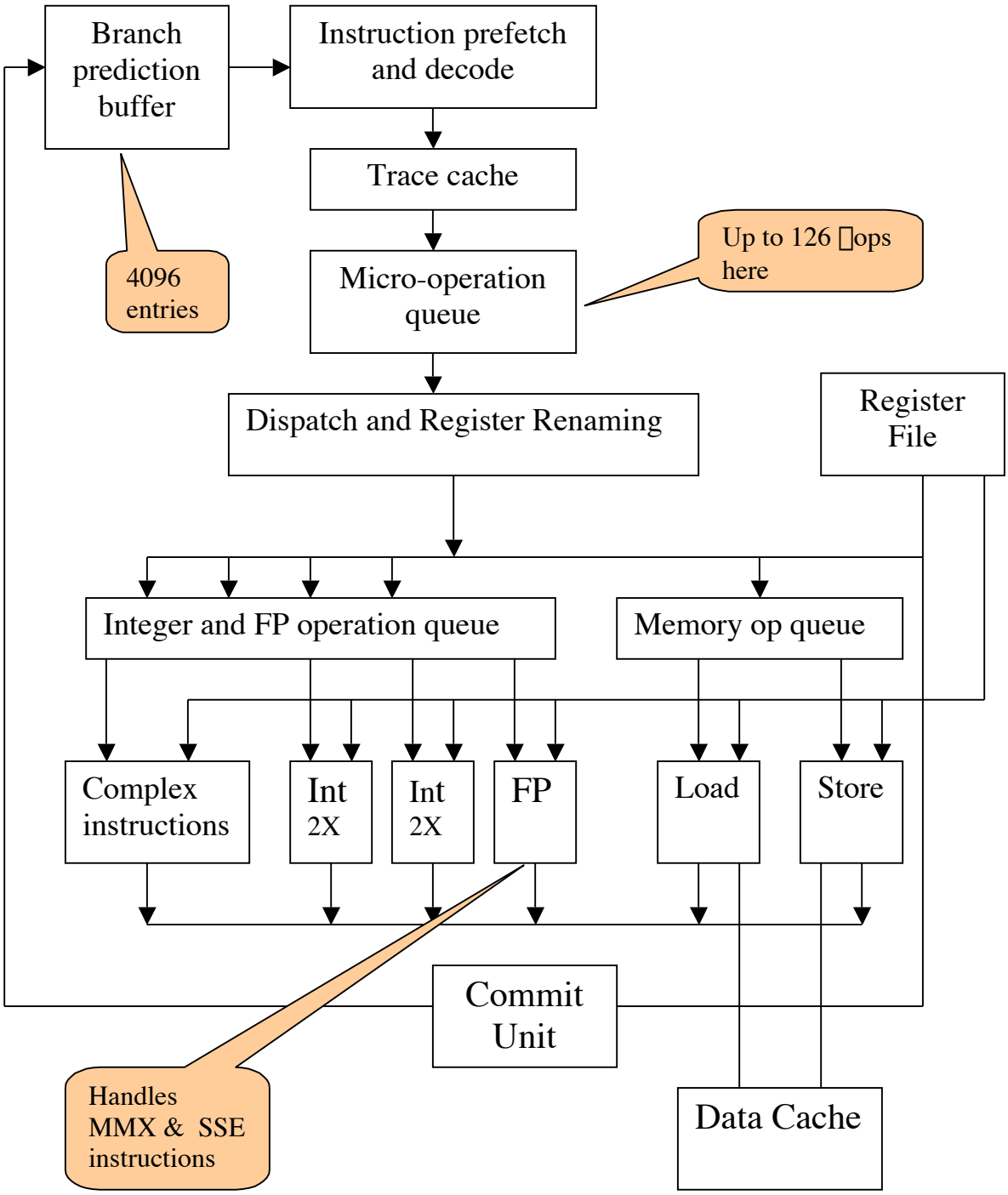
Trace cache

A **trace cache** is a mechanism for increasing the instruction fetch bandwidth by storing traces of instructions that have already been fetched. The mechanism was first proposed by Eric Rotenberg, Steve Bennett, and Jim Smith in their 1996 paper *"Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching."*

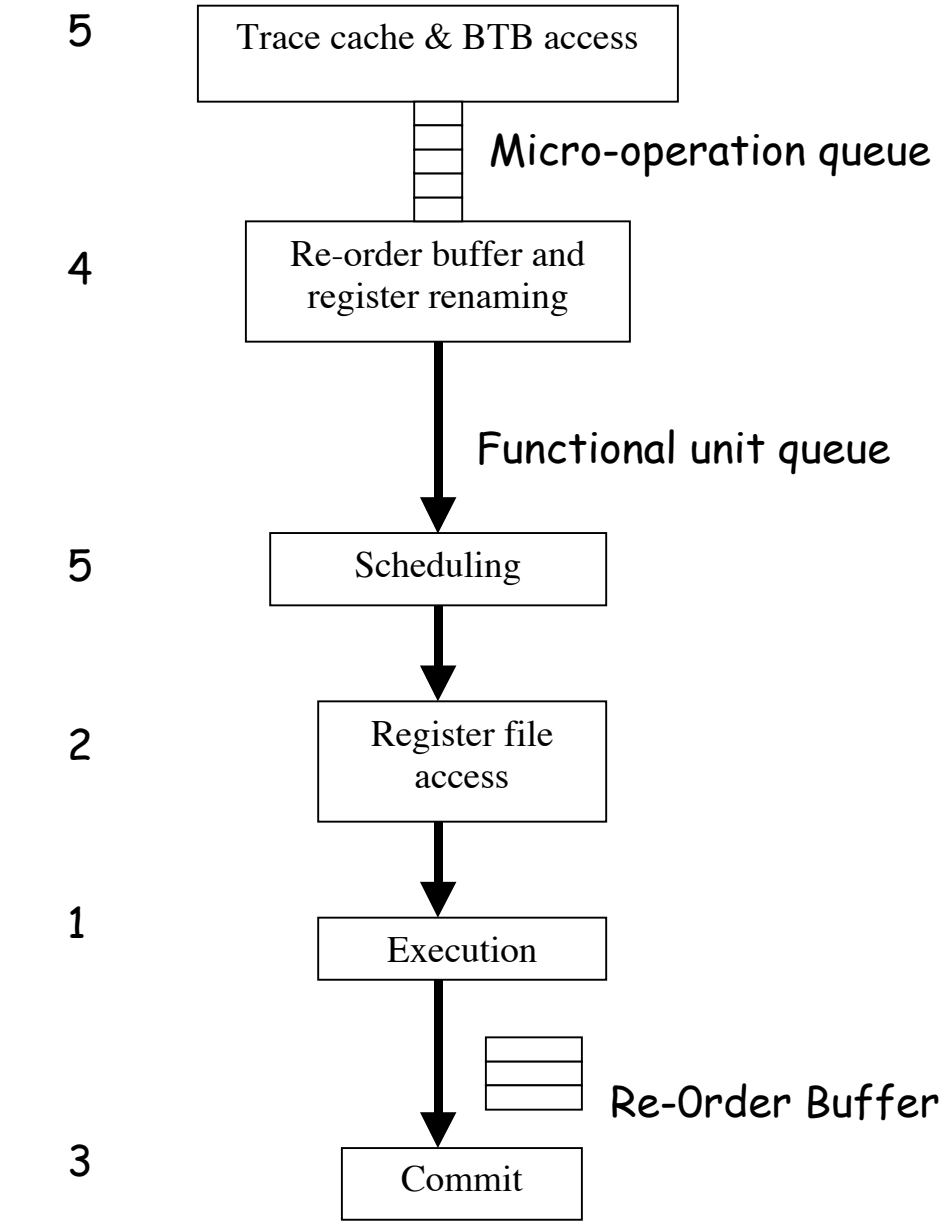


Instructions are added to trace caches in groups representing either individual basic blocks or dynamic instruction traces. A basic block consists of a group of non-branch instructions ending with a branch. A dynamic trace ("trace path") contains only instructions whose results are actually used, and eliminates instructions following taken branches (since they are not executed); a dynamic trace can be a concatenation of multiple of basic blocks.

Pentium micro-architecture



Pentium 4 Pipeline schedule



20 cycles

Problems with IA-32

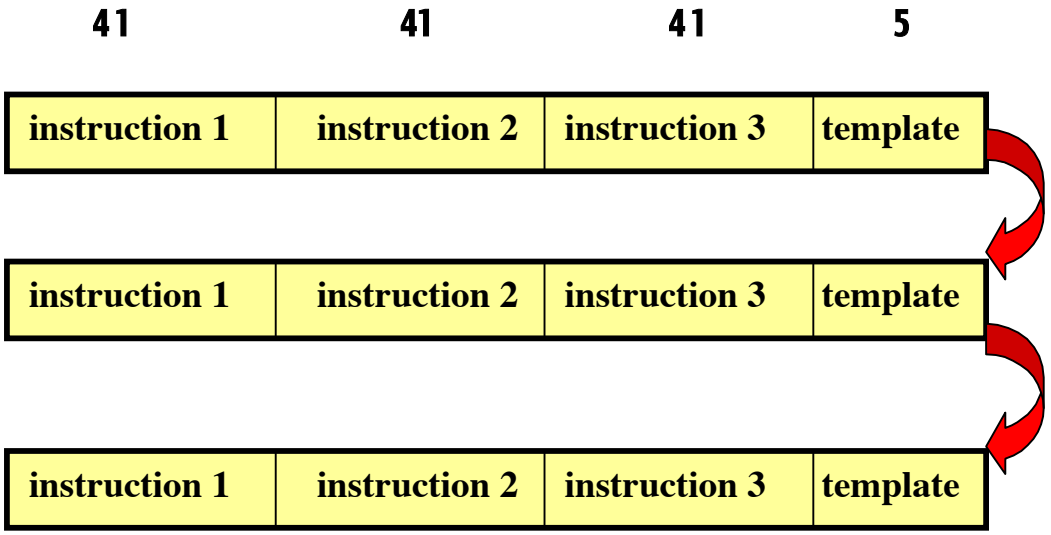
1. Two-address instruction set architecture.
2. Irregular register set and small number of registers.
3. Variable length instruction sizes and variable length opcodes make pipelining challenging.
4. The micro-architecture breaks the instructions into MIPS like micro-operations, but it complicates the design and wastes silicon.
5. Large number of pipeline stages (up to 20 in Pentium 4) increases branch penalty, unless the branch prediction is accurate.

The IA-64 model

Developed with HP using the model of PA-RISC.

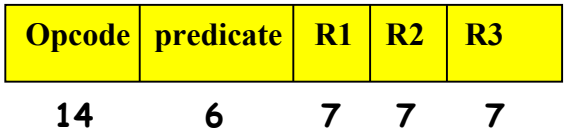
128 registers, each register is 64-bits long.

64 predicate registers



A bundle consists of 128 bits. Ordinarily the three instructions can be scheduled in parallel. Bundles can be chained, i.e. some instructions from different bundles can be scheduled in parallel.

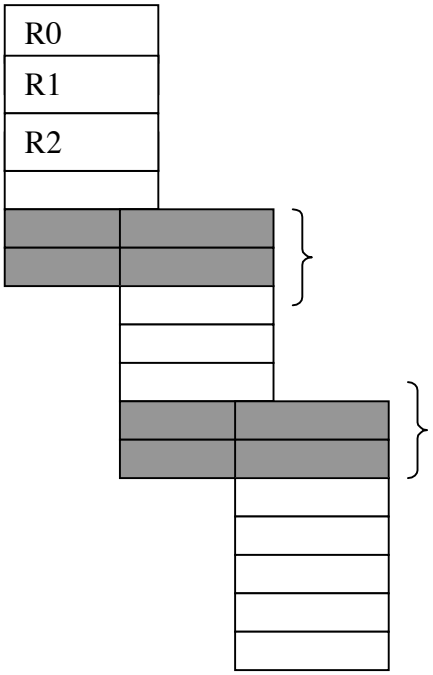
Instruction format



- All instructions are predicated.
- Templates link bundles consisting of instructions that can be scheduled in parallel.
- EPIC model, similar to VLIW.
- Supports speculative execution. One or more bundles can be scheduled every cycle.

Register rotation

Why 128 registers?



The called procedures use a different set of physical registers. Thus the architectural registers R0-R31 are essentially renamed. The overlapped sections simplify parameter passing.

Multiple iterations of a loop use multiple rotating registers. Emulates loop unrolling.

Compare operations

Play a key role in predication.

```
If (a == b) {          {
      c++                    cmp.eq p1, p2 = ra, rb
}     else {                 (p1)  add rc = rc, 1
      d++                    (p2) add rd = rd, 1
}                      }
```