Stallings 2012 edition

CS 320
Ch. 16 SuperScalar Machines

A superscalar machine is o**ne in which multiple instruction streams allow completion of more than one instruction per cycle.**

A superpipelined machine is o**ne in which a single pipeline is double clocked so that two instructions are executed per clock cycle.**

Instruction level parallelism **is a measure of the degree to which multiple instructions can be executed in parallel.**

Superscalar machines can execute instructions out of order. **Either the compiler or the CPU hardware recognizes instruction sequences, determines dependencies, and allows rearrangement of instructions.**

The author lists five limitations to instruction level parallelism.

True data dependency (RAW)
procedural dependency
resource conflicts
output dependency (WAW)
antidependency (WAR)

Here is an example of a true data dependency.

```
add r1, r2
mov r3, r1  ;can't do this until after the add
```

In the pipeline for a true data dependency a **NOP is added to allow one instruction to complete.**

A procedural dependency is one in which the i**nstruction following a branch forms a procedural dependency. Without executing the branch you don't know what instruction to execute next.**

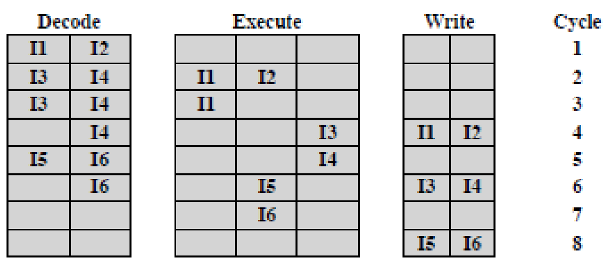
instruction level parallelism vs *machine level parallelism*
Instruction level parallelism exists when instructions have no dependencies and thus can be executed in parallel.
Machine level parallelism is the ability of a machine to exploit instruction level parallelism.

Instructions come out of memory in proper order. The processor can then reorder these instructions and issue them to the pipeline in a different order to achieve some parallelism and efficiency. This process is called the *instruction issue policy.* There are three instruction issue policies in use.
In-order issue with in-order completion
in-order issue with out-of-order completion
out-of-order issue with out-of-order completion.

Stallings 2012 edition

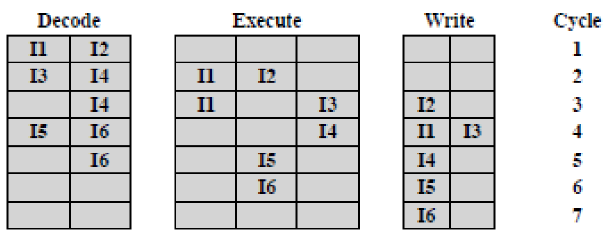
The following figure illustrates issue policy.

A) In-order issue with in-order completion
B) in-order issue with out-of-order completion
C) out-of-order issue with out-of-order completion.



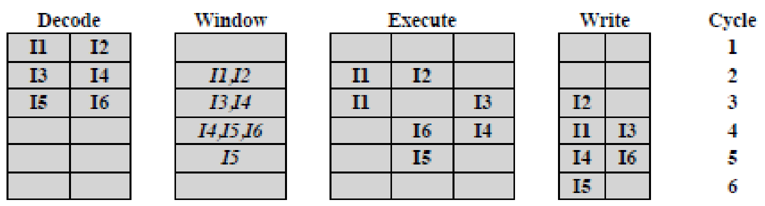
Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

(a) In-order issue and in-order completion



Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

(b) In-order issue and out-of-order completion



Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1,I2*	I1	I2				2
I5	I6	*I3,I4*	I1		I3	I2		3
		I4,I5,I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

(c) Out-of-order issue and out-of-order completion

Figure 14.4 Superscalar Instruction Issue and Completion Policies

I1 requires 2 cycles to execute
I3 and I4 conflict for the same functional unit
I5 depends on the value produced by I4
I5 and I6 conflict for a functional unit.

The register window that sits between the decode and the execute unit is a buffer that hold decoded instructions that are ready for execution. this buffer is significant because it is the thing that allows look ahead to determine what instructions can be done in parallel and in what order.

Stallings 2012 edition

The following code is an example of an output dependency (WAW)

```
I1: R3 ← R3 op 5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

I2 cannot execute before I1 since it needs the result in R3. This is a data dependency.

I4 cannot execute before I3 for similar reasons.

But I1 and I3 have an output dependency. If I3 executes before I1 then the wrong value will be loaded into R3. This is an output dependency.

The following code illustrates an antidependency. (WAR)

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

I3 cannot complete before I2 since I2 uses the old value of R3 and not the value created by I3. This is like a data dependency only reversed in time.

Register renaming: **To avoid antidependencies and output dependencies, a register may be allocated in two forms: one form has its old value and one form gets the new value. This allows the old value to be used while the new value is calculated and changed. This renaming is done in hardware.**

The author cites simulation studies that indicate that without register renaming, the gains from added functional units are not significant.

****

PENTIUM 4

18. The Pentium 4 architecture is said to be a CISC shell around a RISC core. The **Inherited CISC instruction set is converted to microoperations that are more RISC like. These are fed into a superscalar pipeline.**

Stallings 2012 edition

The figure below shows the Pentium 4 block diagram. Answer the questions below related to this diagram.

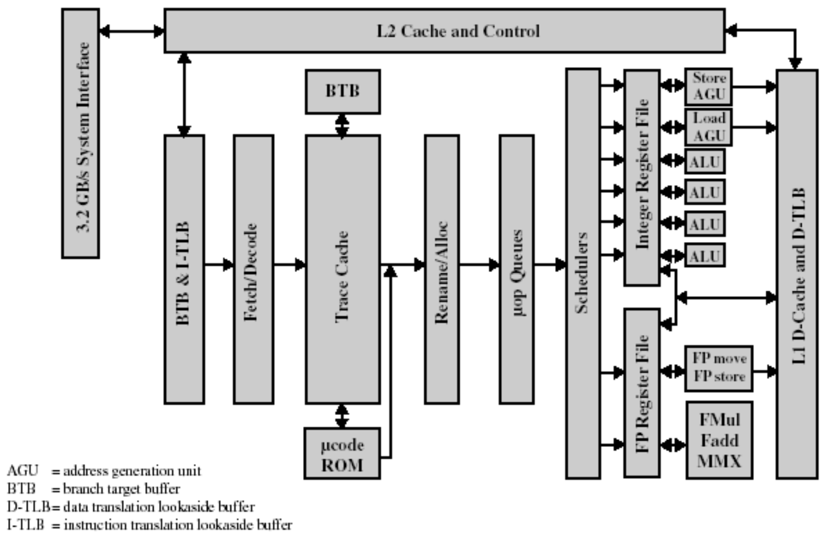


Figure 14.7 Pentium 4 Block Diagram

A) The P4 is said to have an in-order front end. **Instructions are fetched from memory in static order.**

B) The diagram shows three caches. **L2 cache holds data an instructions. The trace cache is the instruction L1 cache. The L1 D cache is the L1 data cache.**

C) The P4 instructions are of variable length (inherited all the way back to the 8080). The trace cache gets 64 bytes at a time from the L2 cache. The fetch/decode unit scans the bytes and finds instruction boundaries. **Each instruction is broken up into from one to four micro-ops each of which is 118-bits long. These RISC like instructions are then used in the instruction pipeline.**

D) The 118 bit **micro-ops are needed to accommodate the CISC instructions on the P4.**

E) The micro-ops go i**nto the L1 instruction cache (trace cache) after being generated.**

F) The branch history is d**one at the trace cache and the branch target buffer logic (BTB). There are 4 bits for branch history since the pipe is 20 stages long, accurate branch prediction is essential. If an instruction has no history then static branch prediction is used.**

G) For static branch prediction t**here are three rules for prediction conditional branches with no history: 1) If not IP relative predict taken if the branch is a return, otherwise not taken. 2) If IP relative backward predict taken. 3) If IP relative forward predict not taken.**

H) The micro-ops are executed o**ut-of-order. The order depends on dependencies.**

I) Some instructions may require the execution of hundreds of microinstructions. **For instructions which require more than four microops, the microop is sent to a microprogram ROM which in turn, generates hundreds of microinstructions if necessary.**

J) The P4 does register renaming. **It remaps 16 architectural registers into a set of 128 physical registers. It thereby removes antidependencies and output dependencies.**

K) Note that register renaming does not eliminate data dependencies. This is n**ot possible.**

L) If a branch prediction is wrong m**icro-ops are flushed from the pipe and a correction is made to the branch predictor logic.**

M) It is critical that the P4 pipeline do a good job of branch prediction because t**he pipe is so long that it is expensive to flush.**

ARM Cortex A3

The figure below is an architectural diagram of the ARM Cortex M3 processor. In the center of the diagram is a Memory Protection Unit. This unit r**estricts access of data within a process so that two processes don't interfere with one another. This is used in multiprocess applications.**

This processor has a pipeline. **Fetch, Decode, Execute**

There is also a *Wake-up interrupt controller which* **allows processor to go to sleep in a very low-power mode and wake up in response to an external stimulus.**

This processor uses the Thumb-2 instruction set. This h**as mostly 16-bit Thumb1 instructions but has a few (5) 32-bit instructions.**

In the instruction *decode stage* of the pipeline the hardware d**ecodes the instruction , generates addresses for the load/store unit, and does a branch based on offset or link register.**

The link register **is a one level stack that is built into the registers.**

This processor does not do branch prediction. This is t**oo complicated for a microcontroller.**

To deal with branches the processor u**ses Branch Forwarding and Branch Speculation.**

In *Branch forwarding* s**ome branches are pushed ahead by one cycle in the pipeline to speed things up.**

In *Branch speculation* c**onditional branches the branch target is fetched speculatively before it is known that the branch will be taken.**

Stallings 2012 edition

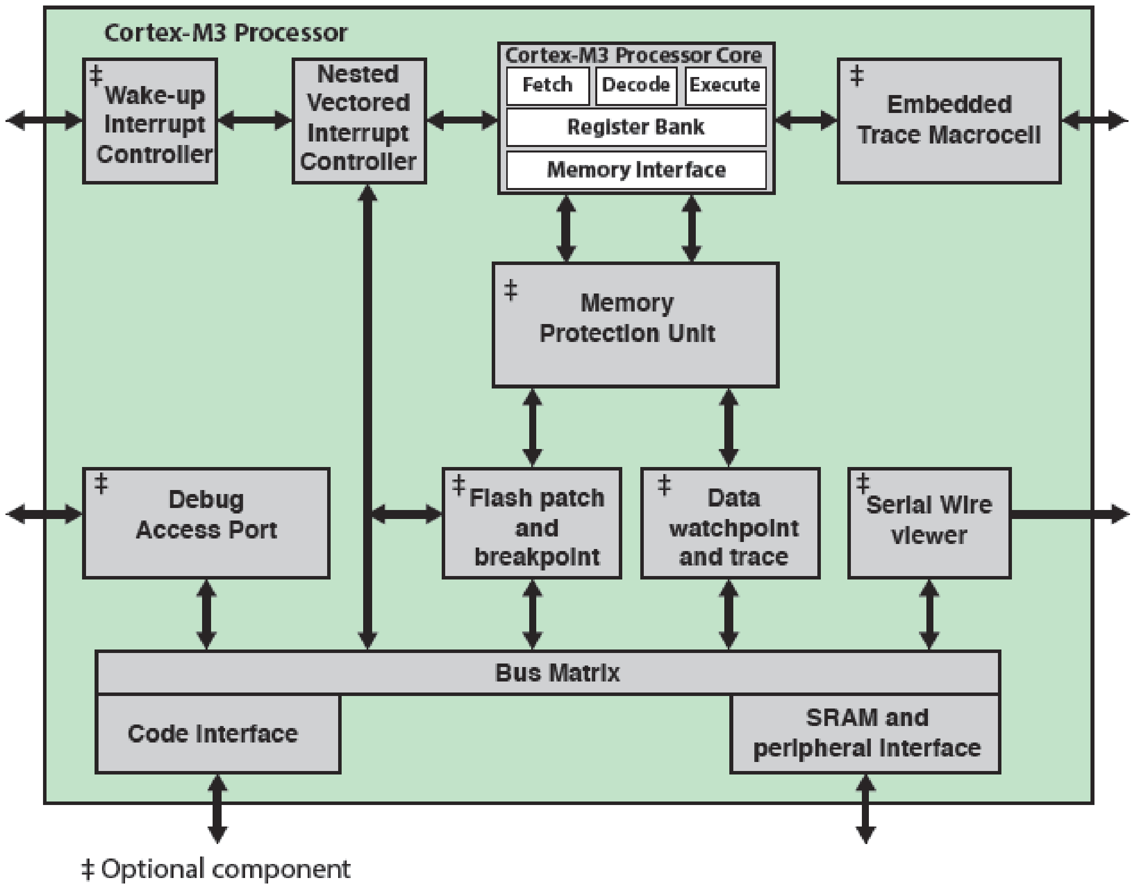
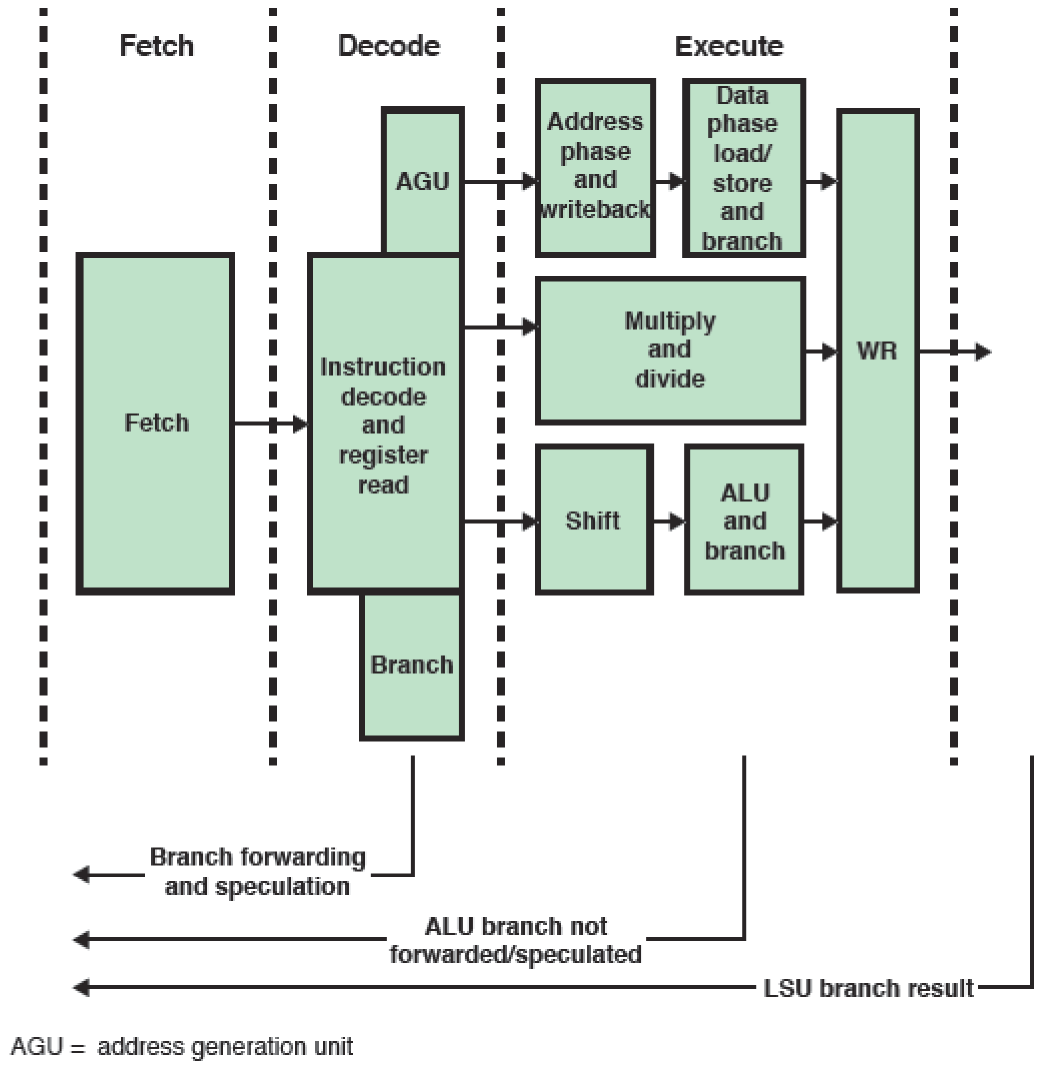


Figure 16.12 ARM Cortex-M3 Block Diagram

Stallings 2012 edition



Fetch
Decode
Execute
AGU
Address phase and writeback
Data phase load/ store and branch
Fetch
Instruction decode and register read
Multiply and divide
WR
Shift
ALU and branch
Branch
Branch forwarding and speculation
ALU branch not forwarded/speculated
LSU branch result
AGU = address generation unit

Figure 16.13 ARM Cortex-M3 Pipeline