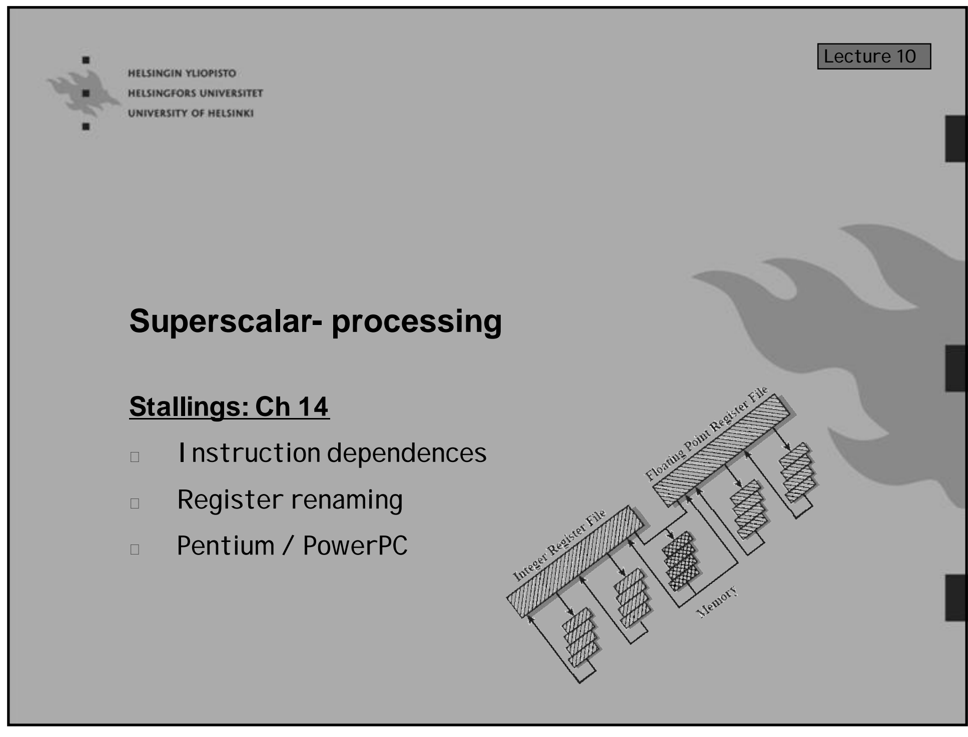
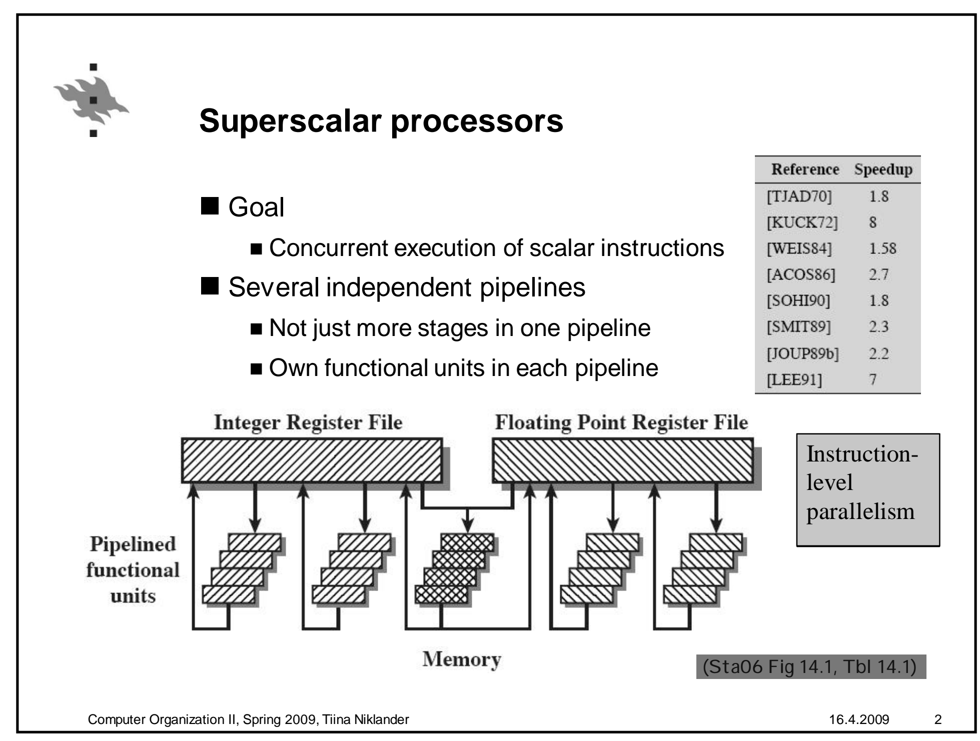
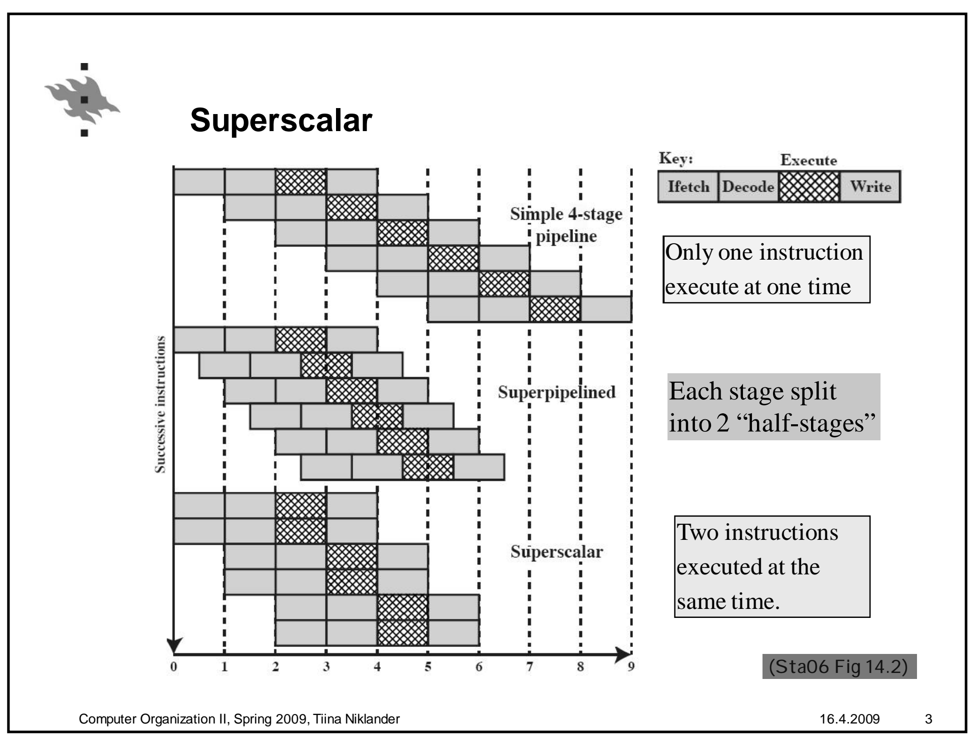
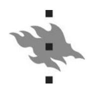




Reference	Speedup
[TJAD70]	1.8
[KUCK72]	8
[WEIS84]	1.58
[ACOS86]	2.7
[SOHI90]	1.8
[SMIT89]	2.3
[JOUP89b]	2.2
[LEE91]	7

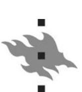




Superscalar processor

- Efficient memory usage
  - Fetch several instruction at once, prefetching (*ennaltanouto*)
  - Data fetch and store (read and write)
  - Concurrency
- Several instructions of the same process executed concurrently on different pipelines
  - Select executable instruction from the prefetched one following a policy (in-order issue/out-of-order issue)
- Finish more than one instruction during each cycle
  - Instructions may complete in different order than started (out-of-order completion)
- When can an instruction finish before the preceeding ones?

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 4



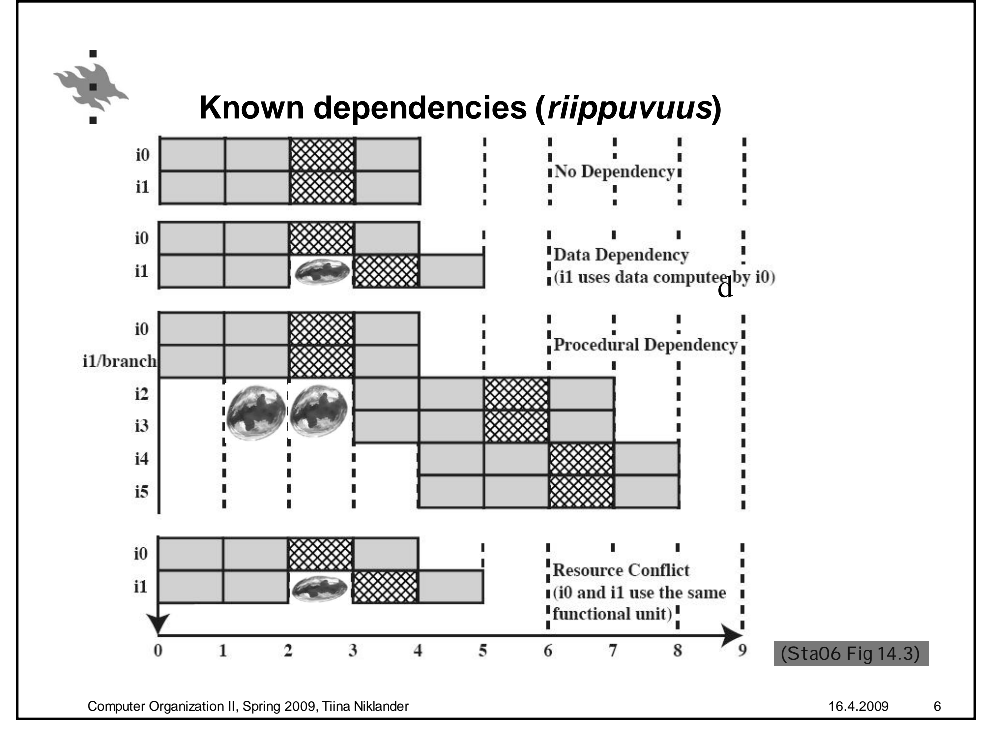
Known dependencies (*riippuvuus*)

- True Data/Flow Dependency (*datariippuvuus*)
  - write-read (Read after Write, RaW)
  - The latter instruction needs data from former instruction
- Procedural/Control Dependency (*kontrolliriippuvuus*)
  - Instruction after the jump executed only, when jump does not happen

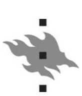
    JNZ R2, 100
    ADD R1, =1
  - Superscalar pipeline has more instructions to waste
  - Variable-length instructions: some additional parts known only during execution
- Resource Conflict (*Resurssiriippuvuus*)
  - One or more pipelilne stage needs the same resource
  - Memory buffer, ALU, access to register file, ...

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 5

Known dependencies (*riippuvuus*)



Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 6



New dependencies

```
load r1,X
add r2,r1,r3
add r1,r4,r5
```

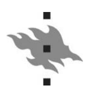
- Output Dependency (*Kirjoitusriippuvuus*)
  - write-after-write (WaW)
  - Two instructions alter the same register or memory location, the latter in the original code must stay
- Antidependency, Read-write dependency (*Antiriippuvuus*)
  - Write-after-read (WaR)
  - The former instruction must be able to fetch the register content, before the latter stores new value there

```
move r2,r1
add r1,r4,r5
```

- Alias?
  - Two registers use indirect references to the same memory location?
  - Different virtual address, same physical address?
  - What is visible on instruction level (before MMU)?

```
store R5, 40(R1)
?
load R6, 0(R2)
```

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 7



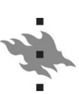
Dependencies

Instructions	Dependency
i: "write" R1 ……… j: "read" R1	data dependency
i: "read" R1 …….. j: "write" R1	antidependency
i: "write" R1 ……. j: "write" R1	output dependency

In data dependency instruction j cannot be executed before instr. i!

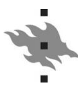
Anti- and output dependency allow change in execution order for instructions i and j, but afterwards must be checked that the right value and result remains

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 8



How to handle dependencies?

- Starting point
  - All dependences must be handled one way or other
- Simple solution (as before)
  - **Special hardware** detects dependency and force the pipeline to wait (bubble)
- Alternative solution
  - **Compiler** generates instructions in such a way that there will be NO dependencies
  - No special hardware
    - simpler CPU that need not detect dependencies
  - Compiler must have very detailed and specific information about the target processor's functionality

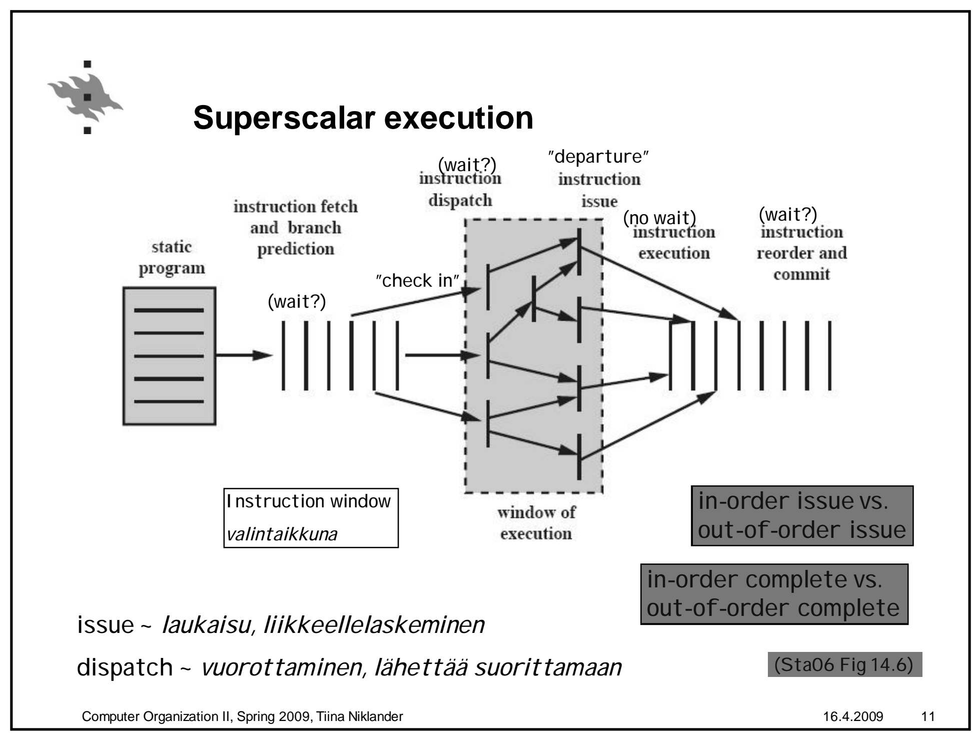


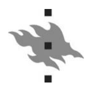
Parallelism (*rinnakkaisuus*)

```
load r1← r2
add r3 ← r3+1
add r4 ← r4, r2
```

```
add r3← r3+1
add r4 ← r3, r2
load r0 ← r4
```

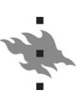
- Instruction-level parallelism (*käskytason rinnakkaisuus*)
  - Independent instructions of a sequence can be executed in parallel by overlapping
  - Theoretical upper limit for parallel execution of instructions
    - Depends on the code itself
- Machine parallelism (*konetason rinnakkaisuus*)
  - Ability of the processor to execute instructions parallel
  - How many instructions can be fetched and executed at the same time?
  - ~ How many pipelines can be used
  - Always smaller than instruction-level parallelism
    - Cannot exceed what instructions allow, but can limit the true parallelism
    - Dependences, bad optimization?





Superscalar execution

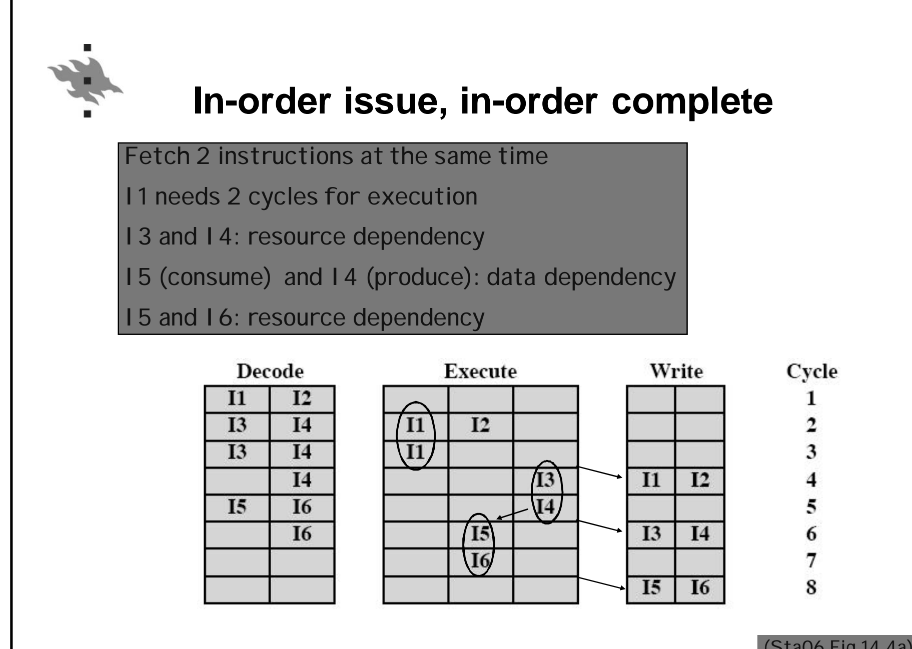
- Instruction fetch (*käskyjen nouto*)
  - Branch prediction (*hyppyjen ennustus*)
    → prefetch (*ennaltanouto*) from memory to CPU
  - Instruction window (*valintaikkuna*)
    ~ set of fetched instructions
- Instruction dispatch/issue (*käskyn päästäminen hihnalle*)
  - Check (and remove) data, control and resource dependencies
  - Reorder; dispatch the suitable instructions to pipelines
  - Pipelines proceed without waits
  - If no suitable instruction, wait here
- Instruction complete, retire (*suoritus valmistuu*)
  - Commit or abort (*hyväksy tai hylkää*)
  - Check and remove write and antidependencies
    → wait / reorder (*järjestä uudelleen*)



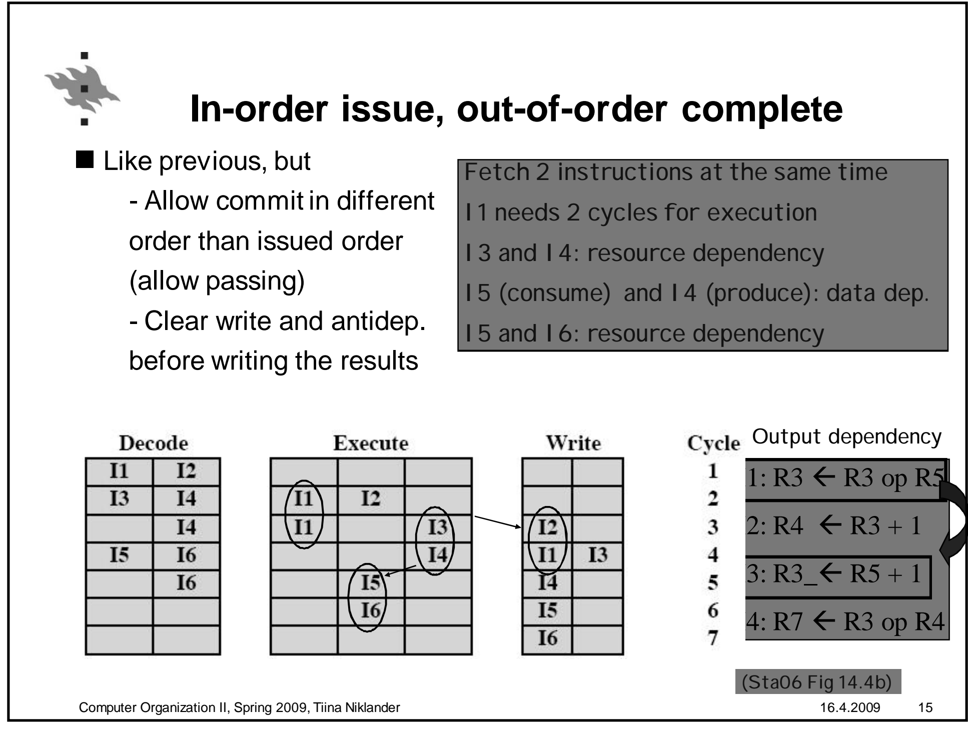
In-order issue, in-order complete

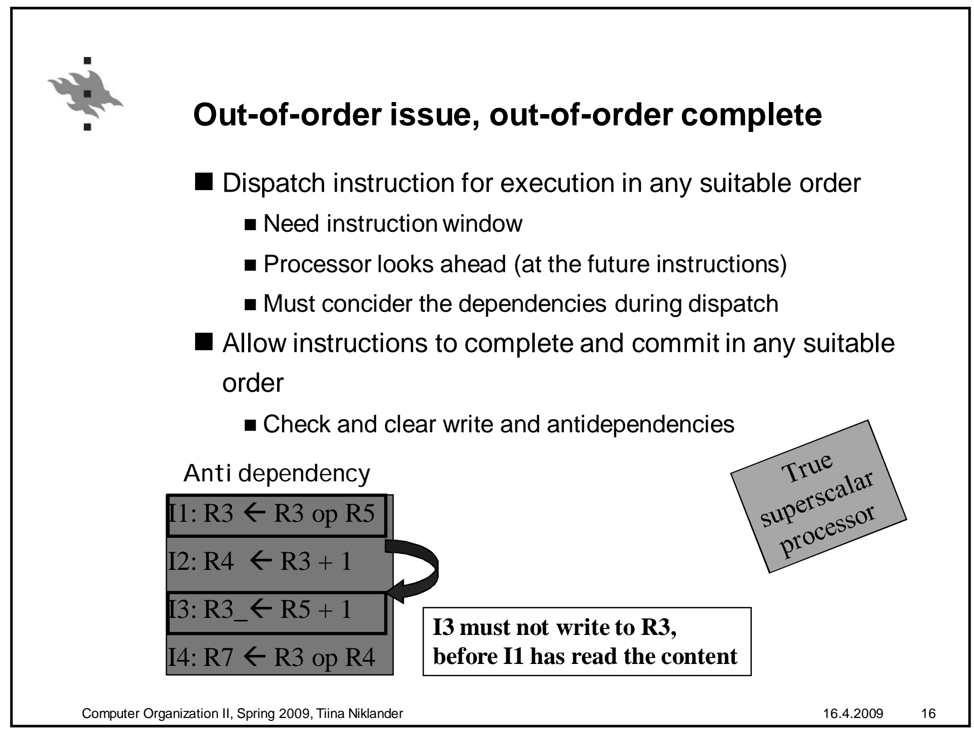
- Traditional sequencial execution order
- No need for instruction window
- Instructions dispatched to pipelines in original order
  - Compiler handles most of the dependencies
  - Still need to check dependencies, if needed add bubbles
  - Can allow overlapping on multiple pipelines
- Instructions complete and commit in original oder
  - Cannot pass, overtake (*ohittaa*) on other pipeline
  - Several instructions can complete at same time
  - Commit/Abort

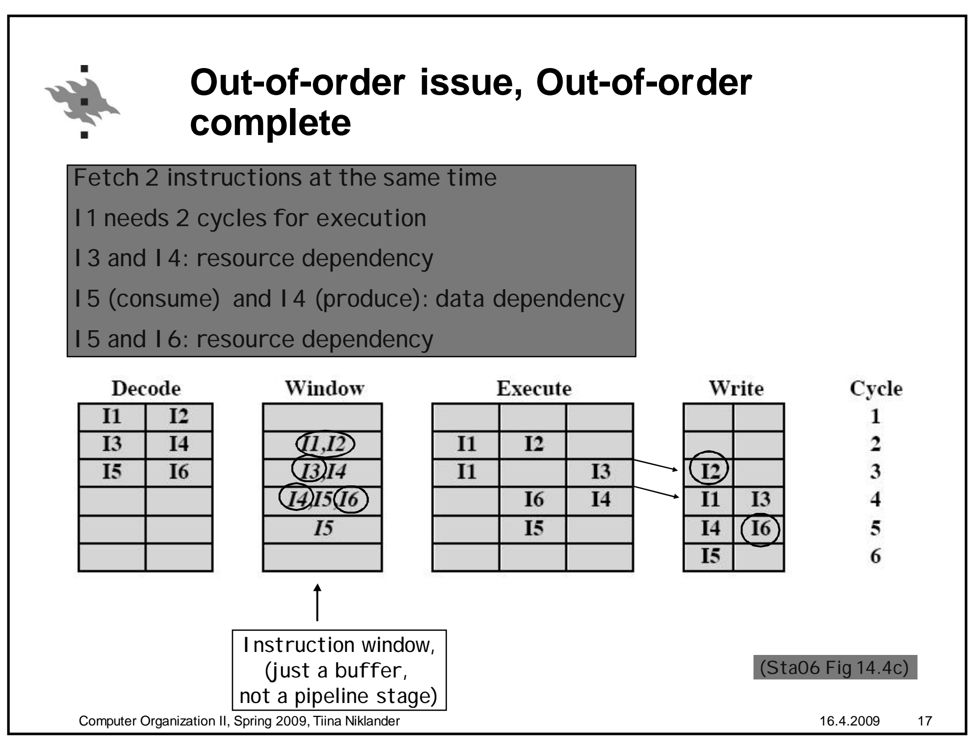
Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 13

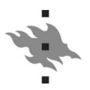


Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8


In-order issue, out-of-order complete
Like previous, but
- Allow commit in different order than issued order (allow passing)
- Clear write and antidep. before writing the results
Fetch 2 instructions at the same time
I1 needs 2 cycles for execution
I3 and I4: resource dependency
I5 (consume) and I4 (produce): data dep.
I5 and I6: resource dependency
Decode
Execute
Write
Cycle
Output dependency
1: R3 ← R3 op R5
2: R4 ← R3 + 1
3: R3_← R5 + 1
4: R7 ← R3 op R4
(Sta06 Fig 14.4b)
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009 15


Out-of-order issue, out-of-order complete
Dispatch instruction for execution in any suitable order
Need instruction window
Processor looks ahead (at the future instructions)
Must concider the dependencies during dispatch
Allow instructions to complete and commit in any suitable order
Check and clear write and antidependencies
Anti dependency
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3_← R5 + 1
I4: R7 ← R3 op R4
I3 must not write to R3, before I1 has read the content
True superscalar processor
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009 16



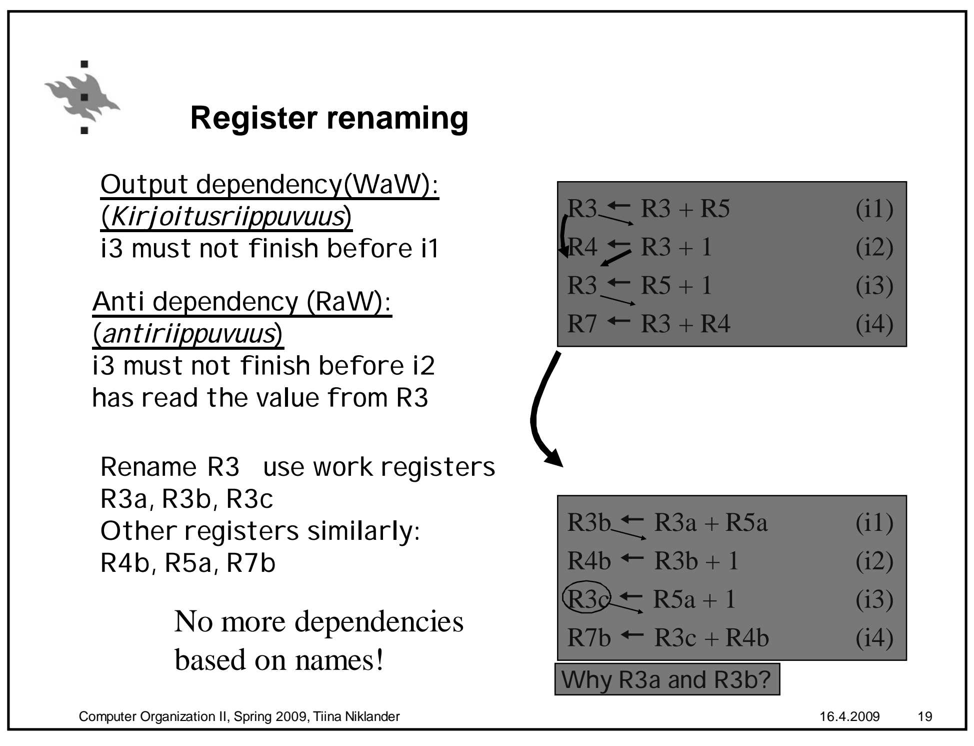


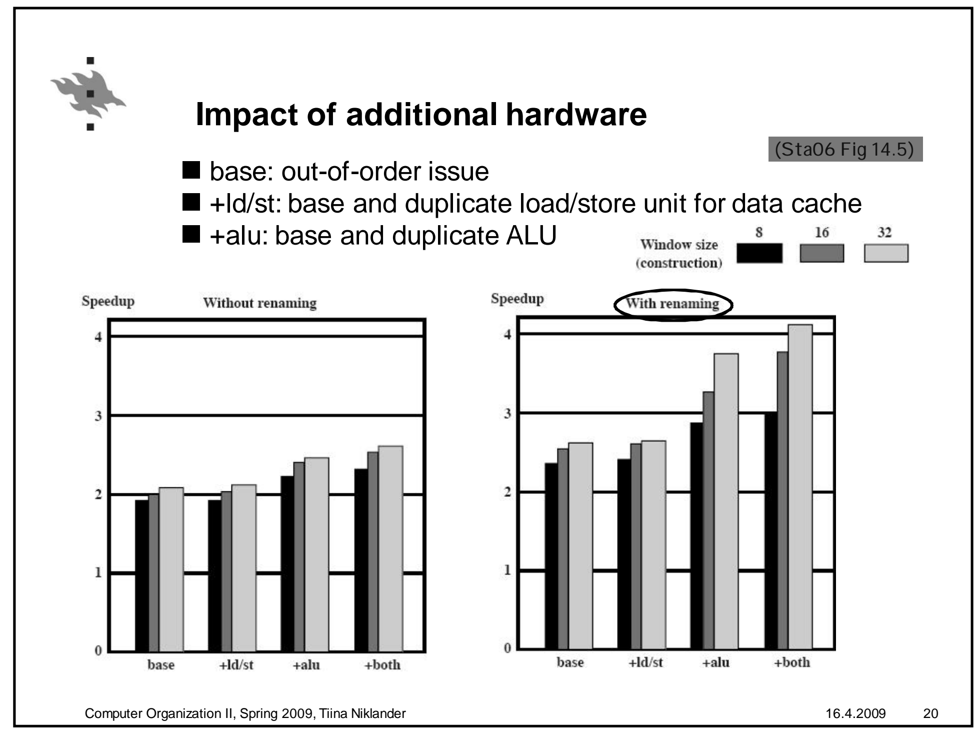
Register renaming (*rekistereiden uudelleennimeäminen*)

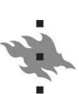
- One cause for some of the dependencies is the usage of names
  - The same name could be used for several independent elements
  - Thus, instructions have unneeded write and antidependencies
  - Causing unnecessary waits
- Solution: Register renaming
  - Hardware must have more registers (than visible to the programmer and compiler)
  - Hardware allocates new real registers during execution in order to avoid name-based dependencies (nimiriippuvuus)
- Need
  - More internal registers (register files, register set), e.g. Pentium II has 40 working registers
  - Hardware that is cabable of allocating and managing registers and performing the needed mapping

R3 ← R3 + R5
R4 ← R3 + 1
R3 ← R5 + 1
R7 ← R3 + R4

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 18


Register renaming
Output dependency(WaW):
(Kirjoitusriippuvuus)
i3 must not finish before i1
Anti dependency (RaW):
(antiriippuvuus)
i3 must not finish before i2
has read the value from R3
Rename R3 use work registers
R3a, R3b, R3c
Other registers similarly:
R4b, R5a, R7b
No more dependencies
based on names!
R3 ← R3 + R5 (i1)
R4 ← R3 + 1 (i2)
R3 ← R5 + 1 (i3)
R7 ← R3 + R4 (i4)
R3b ← R3a + R5a (i1)
R4b ← R3b + 1 (i2)
R3c ← R5a + 1 (i3)
R7b ← R3c + R4b (i4)
Why R3a and R3b?
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009 19

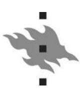

Impact of additional hardware
(Sta06 Fig 14.5)
base: out-of-order issue
+ld/st: base and duplicate load/store unit for data cache
+alu: base and duplicate ALU
Window size (construction)
8
16
32
Speedup
Without renaming
With renaming
0
1
2
3
4
base
+ld/st
+alu
+both
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009 20



Superscalar – conclusion

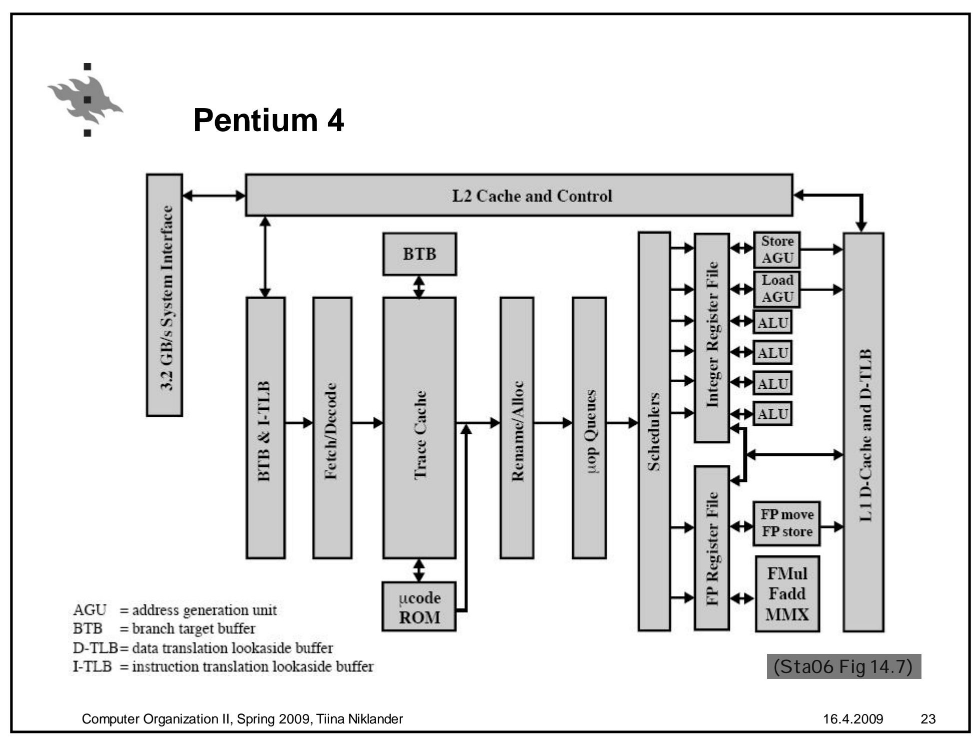
Sta06 Fig 14.6

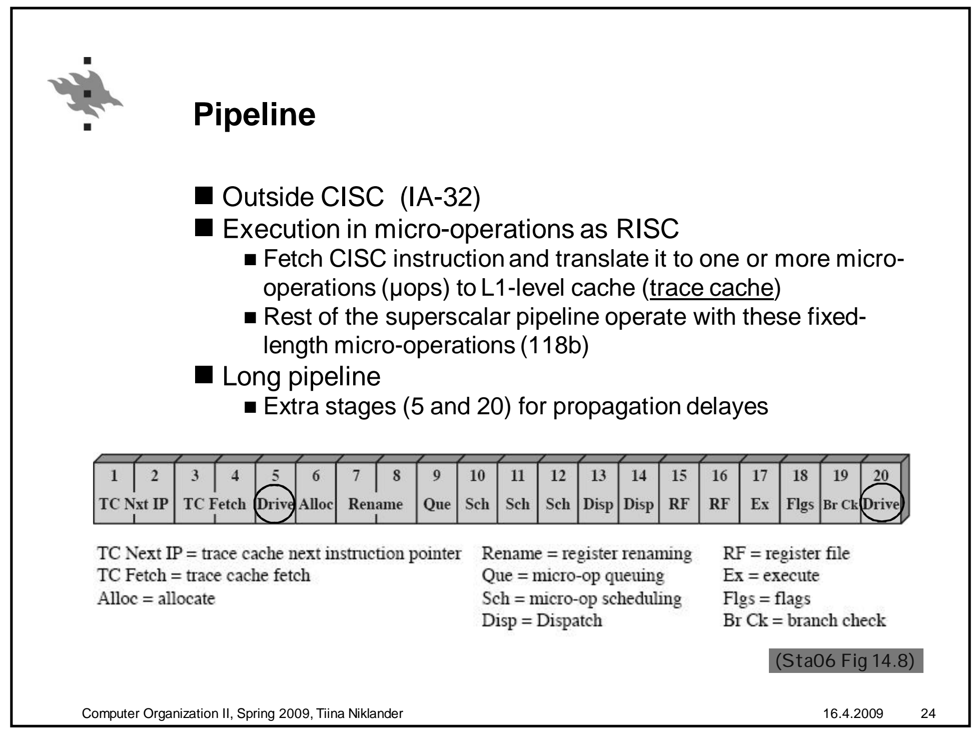
- Several functionally independent units
- Efficient use of memory hierarchy
  - Allows parallel memory fetch and store
- Instruction prefetch (*käskyjen ennaltanouto)*
  - Branch prediction (*hyppyjen ennustaminen*)
- Hardware-level logic for dependency detections
  - Circuits to pass information for other functional unit at the same time as storing to register or memory
- Hardware-level logic to dispatch several independent instructions
  - Dependencies → dispatching order
- Hardware-level logic to maintain correct completion order (*valmistumisjärjestys)*
  - Dependencies → commit-order

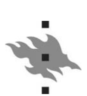


Computer Organization II

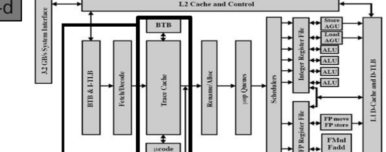
Pentium 4


Pentium 4
L2 Cache and Control
3.2 GB/s System Interface
BTB
BTB & I-TLB
Fetch/Decode
Trace Cache
μcode ROM
Rename/Alloc
μop Queues
Schedulers
Integer Register File
FP Register File
Store AGU
Load AGU
ALU
ALU
ALU
ALU
FP move FP store
FMul Fadd MMX
L1 D-Cache and D-TLB
AGU = address generation unit
BTB = branch target buffer
D-TLB = data translation lookaside buffer
I-TLB = instruction translation lookaside buffer
(Sta06 Fig 14.7)
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009 23


Pipeline
Outside CISC (IA-32)
Execution in micro-operations as RISC
Fetch CISC instruction and translate it to one or more micro-operations (μops) to L1-level cache (trace cache)
Rest of the superscalar pipeline operate with these fixed-length micro-operations (118b)
Long pipeline
Extra stages (5 and 20) for propagation delayes
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
TC Nxt IP TC Fetch Drive Alloc Rename Que Sch Sch Sch Disp Disp RF RF Ex Flgs Br Ck Drive
TC Next IP = trace cache next instruction pointer
TC Fetch = trace cache fetch
Alloc = allocate
Rename = register renaming
Que = micro-op queuing
Sch = micro-op scheduling
Disp = Dispatch
RF = register file
Ex = execute
Flgs = flags
Br Ck = branch check
(Sta06 Fig 14.8)
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009 24



Sta06 Fig 14.9a-d



Generation of µops

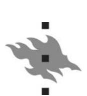
a) Fetch IA-32 instruction from L2 cache and generate µops to L1
- Uses Instruction Lookaside Buffer (I-TLB)
- and Branch Target Buffer (BTB)
  - four-way set-associative cache, 512 lines
- 1-4 µops (=118 bit RISC) per instruction (most cases), if more then stored to microcode ROM

b) Trace Cache Next Instruction Pointer - instruction selection
- Dynamic branch prediction based on history (4-bit)
- Static branch prediction, if no history information available
  - backward, predict "taken"
  - forward, predict "not taken"

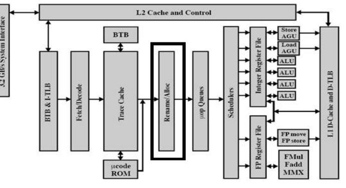
c) Fetch instruction from L1-level trace cache

d) Drive – wait (instruction from trace cache to rename/allocator)

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 25



Sta06 Fig 14.9e



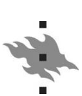
Resource allocation

e) Allocate resources
- 3 micro-operations per cycle
- Allocate an entry from Reorder Buffer (ROB) for the µops (126 entries available)
- Allocate one of the 128 internal work registers for the result
- And, possibly, one load (of 48) OR store (of 24) buffer

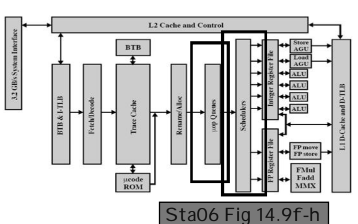
f) Register renaming
- Clear 'name dependencies' by remapping registers (16 architectural regs to 128 physical registers)
- If no free resource, wait (→ out-of-order)

- ROB-entry contains bookkeeping of the instruction progress
  - Micro-operation and the address of the original IA-32 instr.
  - State: scheduled, dispatched, completed, ready
  - Register Alias Table (RAT): which IA-32 register → which physical register

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 26



Window of Execution



g) Micro-Op Queueing

- 2 FIFO queus for μops
  - One for memory operations (load, store)
  - One for everything else
- No dependencies, proceed when room in scheduling

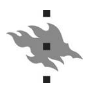
h) Micro-Op Scheduling

- Retrieve μops from queue and dispatch for execution
- Only when operands ready (check from ROB-entry)

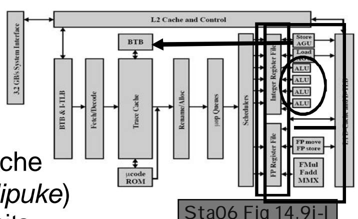
i) Dispatching

- Check the first instructions of FIFO-queues (their ROB-entries)
- If execution unit needed is free, dispatch to that unit
- Two queues → out-of-order issue
- max 6 micro-ops dispatched in one cycle
  - ALU and FPU can handle 2 per cycle
  - Load and store each can handle 1 per cycle

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 27



Integer and FP Units



j) Get data from register or L1 cache

k) Execute instruction, set flags (*lipuke*)

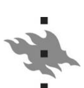
- Several pipelined execution units
  - 2 * Alu, 2 * FPU, 2 * load/store
  - E.g. fast ALU for simple ops, own ALU for multiplications
- Result storing: in-order complete
- Update ROB, allow next instruction to the unit

l) Branch check

- What happend in the jump /branch instruction
- Was the prediction correct?
- Abort in correct instruction from the pipeline (no result storing)

m) Drive – update BTB with the branch result

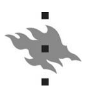
Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 28



Pentium 4 Hyperthreading

- One physical IA-32 CPU, but 2 logical CPUs
- OS sees as 2 CPU SMP (symmetric multiprocessing)
  - Processors execute different processes or threads
  - No code-level issues
  - OS must be cabable to handle more processors (like scheduling, locks)
- Uses CPU wait cycles
  - Cache miss, dependences, wrong branch prediction
- If one logical CPU uses FP unit the other one can use INT unit
  - Benefits depend on the applications

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 29



Pentium 4 Hyperthreading

- Duplicated (*kahdennettu*)
  - IP, EFLAGS and other control registers
  - Instruction TLB
  - Register renaming logic
- Split (*puolitettu*)
  - No monopoly, non-even split allowed
  - Reordering buffers (ROB)
  - Micro-op queues
  - Load/store buffers
- Shared (*jaettu*)
  - Register files (128 GPRs, 128 FPRs)
  - Caches: trace cache, L1, L2, L3
  - Registers needed during µops execution
  - Functional units: 2 ALU, 2 FPU, 2 ld/st-units

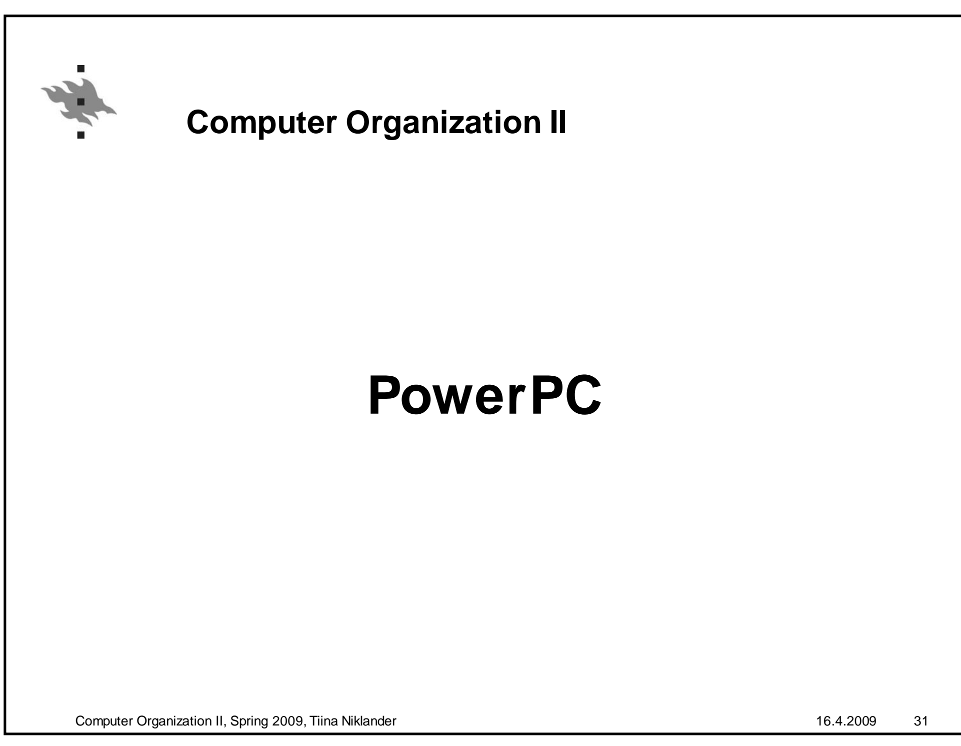
Sta06 Fig 14.7

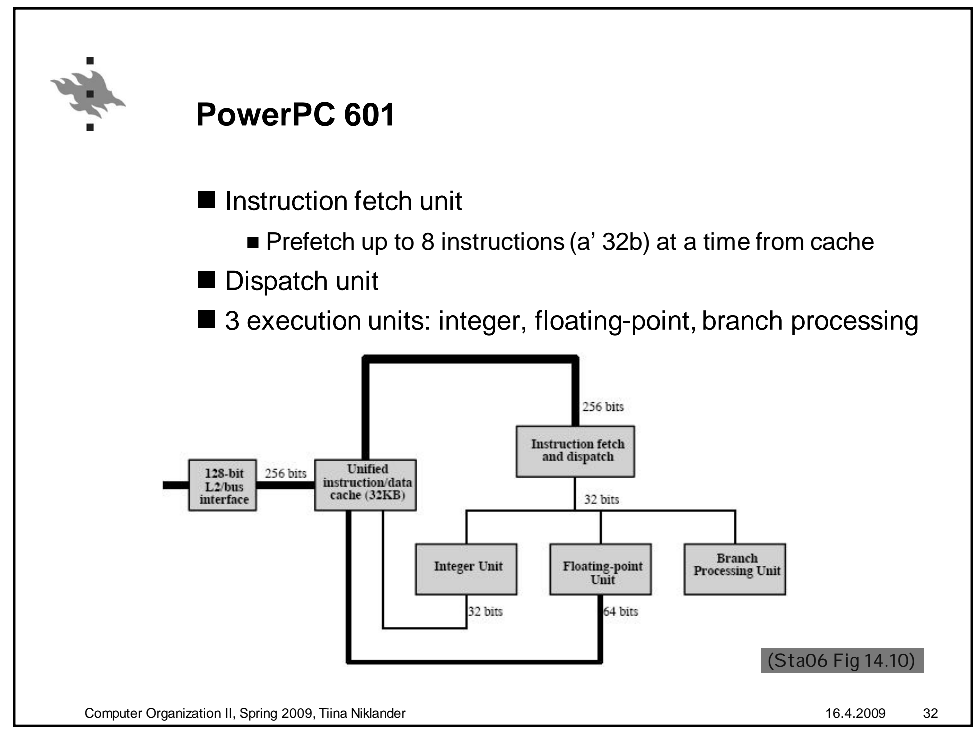
Sta06 Fig 14.8

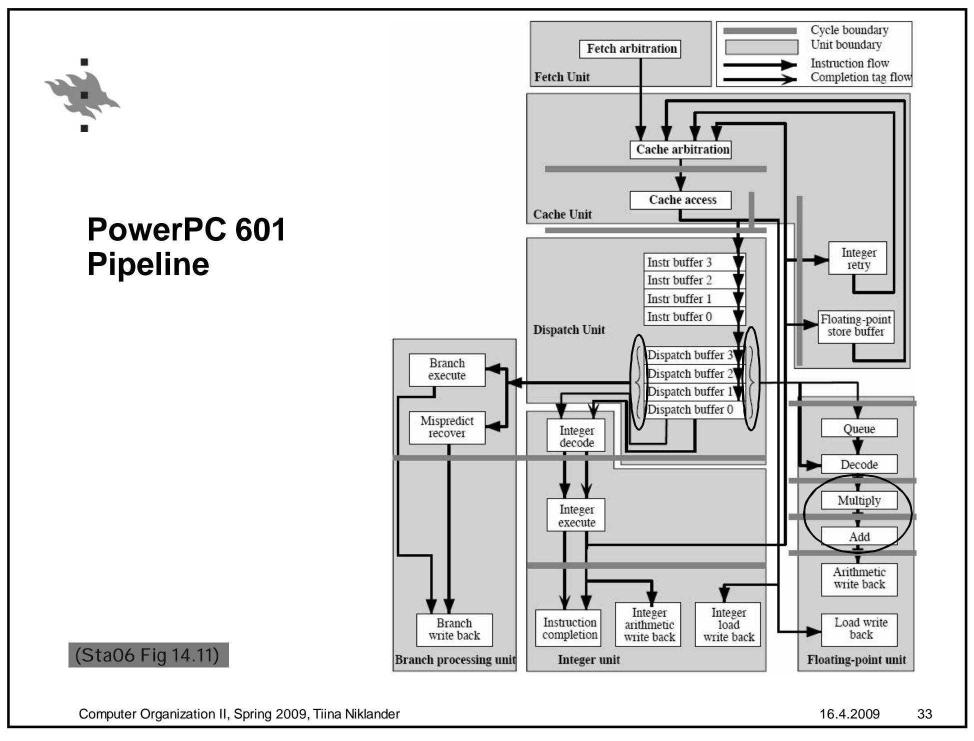
Intel Nehalem arch.: 8 cores on one chip, 1-16 threads (820 million transistors)

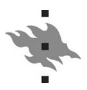
First lauched processor Core i7 (Nov 2008)

Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 30


Computer Organization II
PowerPC
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009
31


PowerPC 601
Instruction fetch unit
Prefetch up to 8 instructions (a' 32b) at a time from cache
Dispatch unit
3 execution units: integer, floating-point, branch processing
256 bits
Instruction fetch and dispatch
128-bit L2/bus interface
256 bits
Unified instruction/data cache (32KB)
32 bits
Integer Unit
Floating-point Unit
Branch Processing Unit
32 bits
64 bits
(Sta06 Fig 14.10)
Computer Organization II, Spring 2009, Tiina Niklander
16.4.2009
32

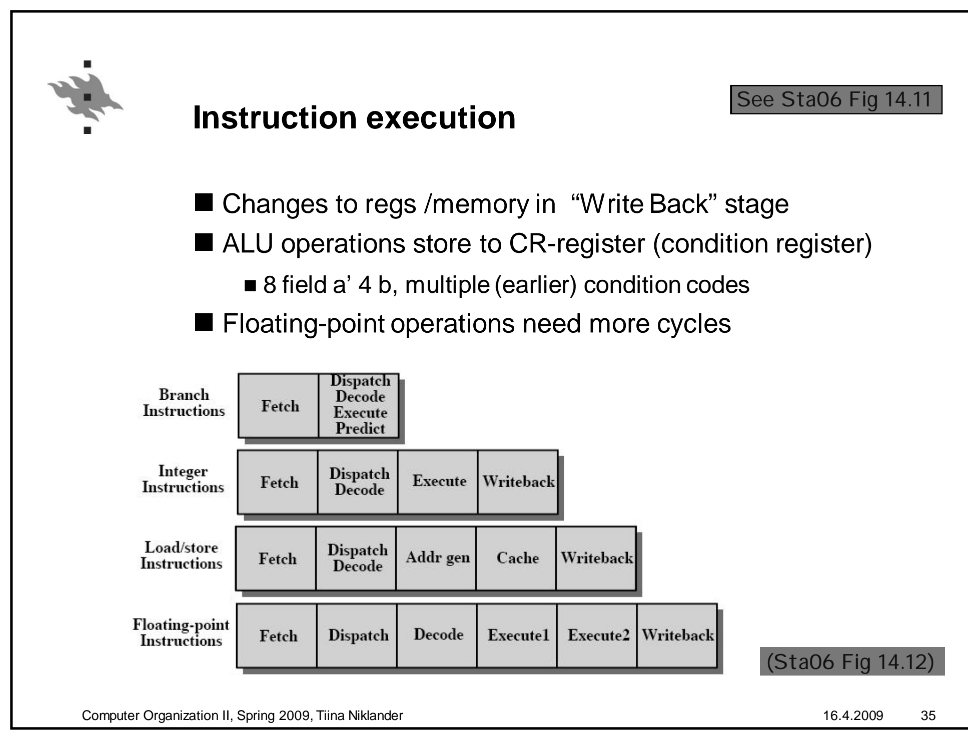


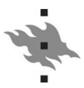


Dispatch unit

= (*Käskyjen suoritukseen valinta*)

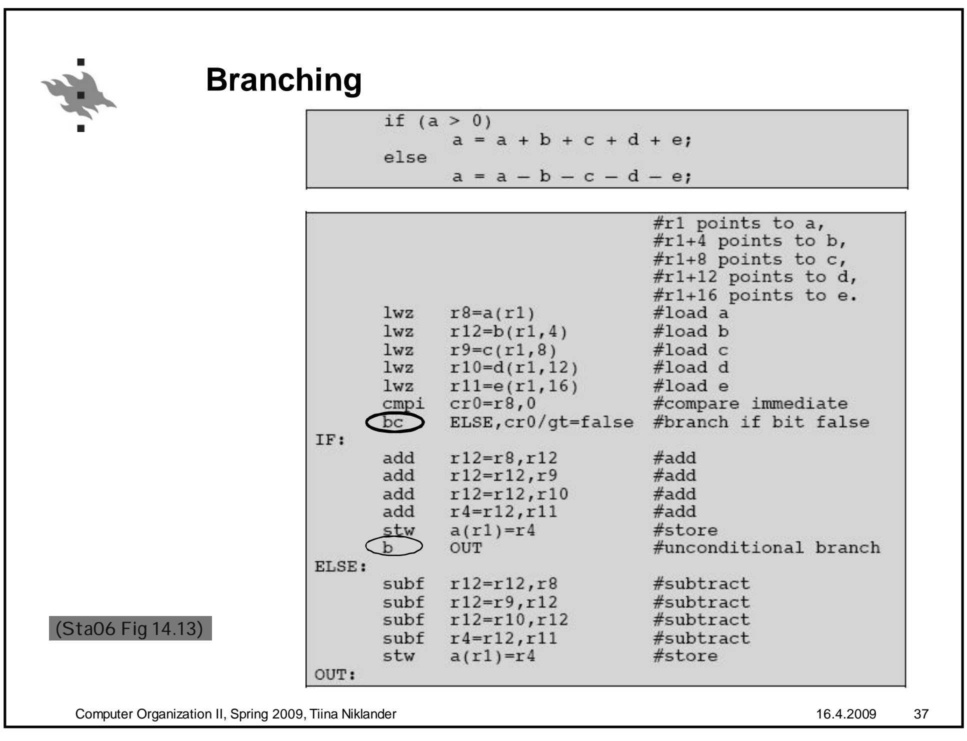
- 4 element Instruction buffer + 4 element dispatch buffer = window of execution (*valintaikkuna*)
- Dispatching instruction from buffer
  - To Integer unit only the top-most element
  - To other units, closest-to-top element
  - out-of-order issue
  - Superscalar degree is 3
- If dependency detected ->stall, bubble
- Hardware-level logic to calculate prefetch address
  - Instructions from jump destination can be fetched even before the instruction is executed

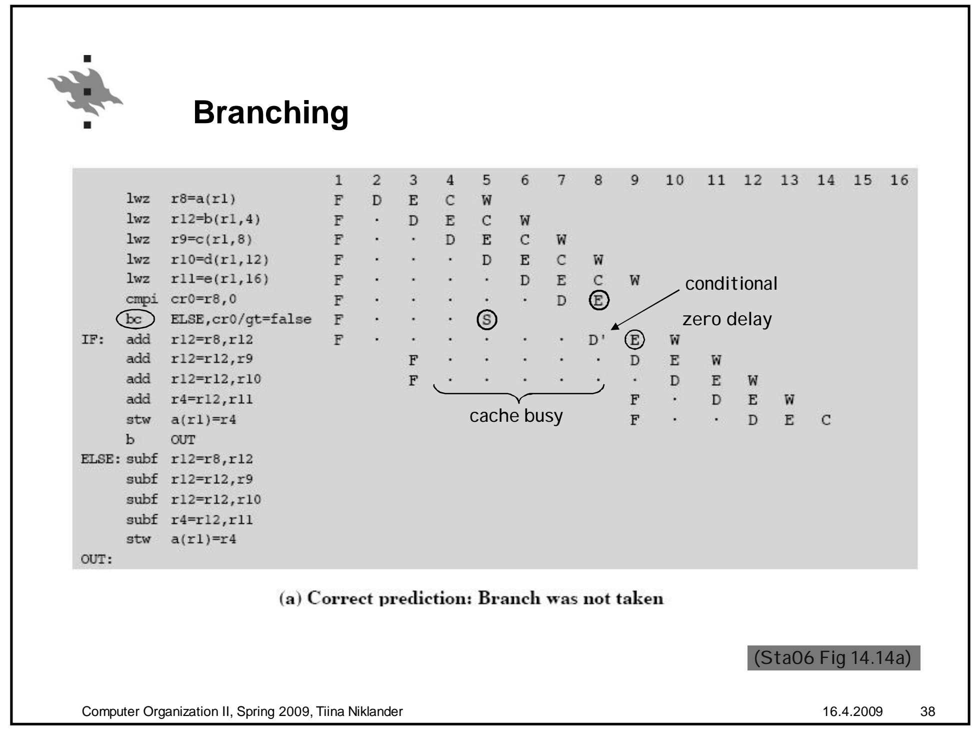




Branch processing

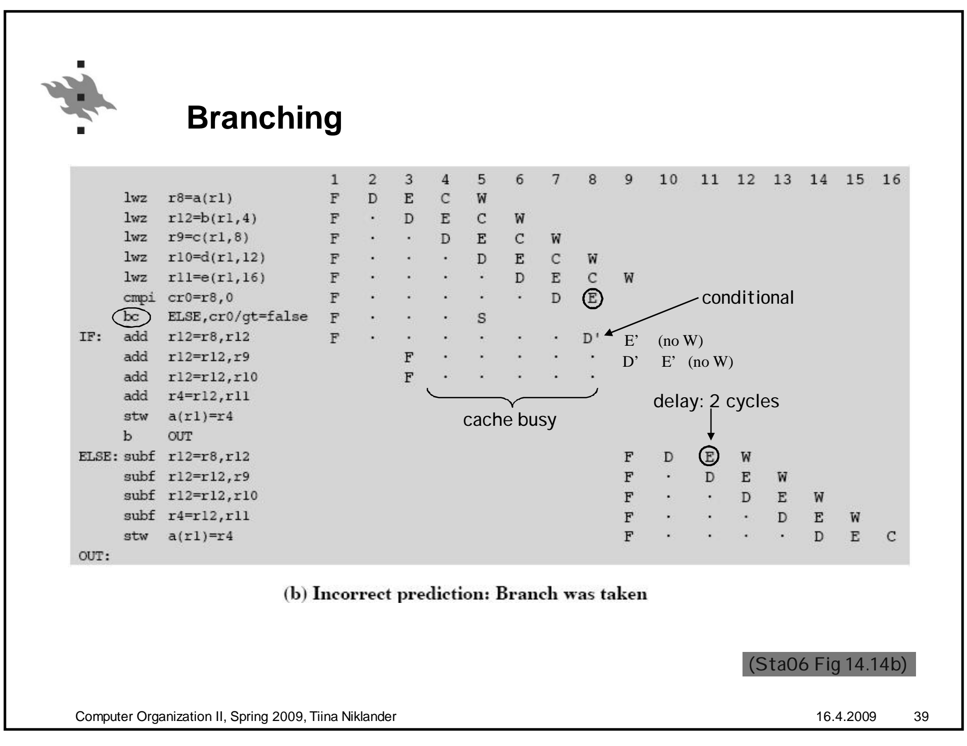
- GOAL: Zero cycle branches
  - No effect on the execution pace of other units
    - No need to clear pipeline or reject results!
- Branch target address generated already in the instruction buffer, before execution!
- Branching logic:
  - Unconditional branch, jump → taken (no choice)
  - Conditional branch and CR-register contains the result based on the result → taken / not taken
  - Unknown results: speculate backwards → taken, forwards → not taken
- Speculation failed: abort instruction before "writeback"

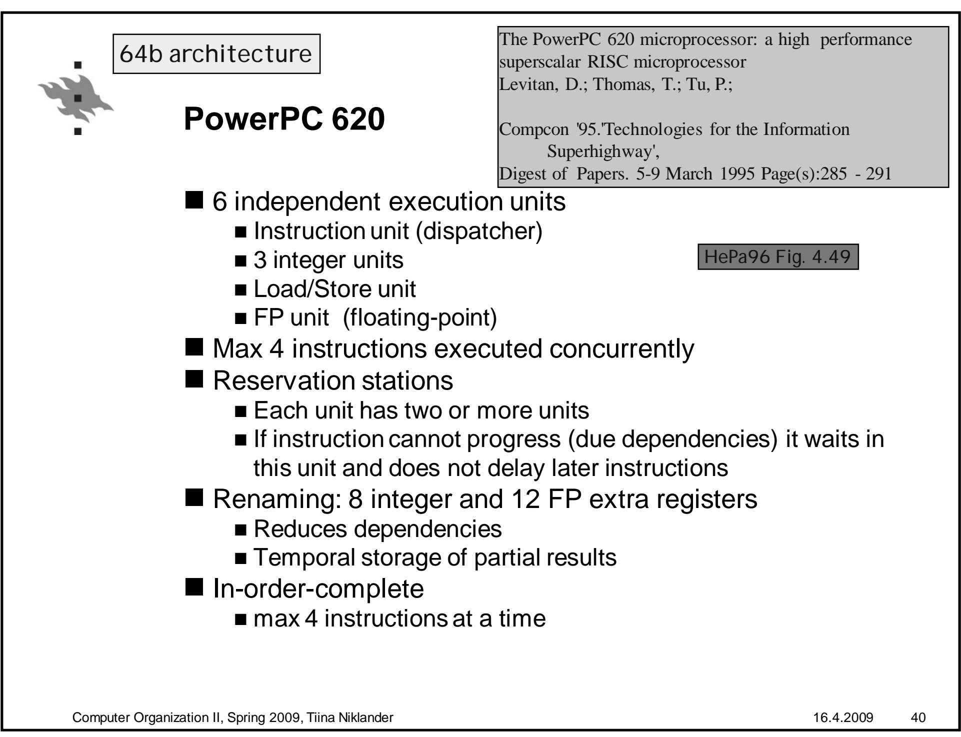


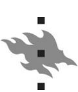


			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	lwz	r8=a(r1)	F	D	E	C	W											
	lwz	r12=b(r1,4)	F	·	D	E	C	W										
	lwz	r9=c(r1,8)	F	·	·	D	E	C	W									
	lwz	r10=d(r1,12)	F	·	·	·	D	E	C	W								
	lwz	r11=e(r1,16)	F	·	·	·	·	D	E	C	W							
	cmpi	cr0=r8,0	F	·	·	·	·	·	D	(E)								
	(bc)	ELSE,cr0/gt=false	F	·	·	·	(S)											
IF:	add	r12=r8,r12	F	·	·	·	·	·	·	D'	(E)	W						
	add	r12=r12,r9			F	·	·	·	·	·	D	E	W					
	add	r12=r12,r10			F	·	·	·	·	·	·	D	E	W				
	add	r4=r12,r11									F	·	D	E	W			
	stw	a(r1)=r4									F	·	·	D	E	C		
	b	OUT																
ELSE:	subf	r12=r8,r12																
	subf	r12=r12,r9																
	subf	r12=r12,r10																
	subf	r4=r12,r11																
	stw	a(r1)=r4																
OUT:																		

(a) Correct prediction: Branch was not taken

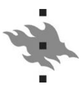

Branching
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
lwz r8=a(r1) F D E C W
lwz r12=b(r1,4) F · D E C W
lwz r9=c(r1,8) F · · D E C W
lwz r10=d(r1,12) F · · · D E C W
lwz r11=e(r1,16) F · · · · D E C W
cmpi cr0=r8,0 F · · · · · D E
bc ELSE,cr0/gt=false F · · · S
IF: add r12=r8,r12 F · · · · · · D' E' (no W)
add r12=r12,r9 F · · · · · D' E' (no W)
add r12=r12,r10 F · · · · · ·
add r4=r12,r11
stw a(r1)=r4
b OUT
ELSE: subf r12=r8,r12 F D E W
subf r12=r12,r9 F · D E W
subf r12=r12,r10 F · · D E W
subf r4=r12,r11 F · · · D E W
stw a(r1)=r4 F · · · · D E C
OUT:
conditional
cache busy
delay: 2 cycles
(b) Incorrect prediction: Branch was taken
(Sta06 Fig 14.14b)
Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 39


64b architecture
PowerPC 620
The PowerPC 620 microprocessor: a high performance superscalar RISC microprocessor
Levitan, D.; Thomas, T.; Tu, P.;
Compcon '95.'Technologies for the Information Superhighway',
Digest of Papers. 5-9 March 1995 Page(s):285 - 291
HePa96 Fig. 4.49
6 independent execution units
Instruction unit (dispatcher)
3 integer units
Load/Store unit
FP unit (floating-point)
Max 4 instructions executed concurrently
Reservation stations
Each unit has two or more units
If instruction cannot progress (due dependencies) it waits in this unit and does not delay later instructions
Renaming: 8 integer and 12 FP extra registers
Reduces dependencies
Temporal storage of partial results
In-order-complete
max 4 instructions at a time
Computer Organization II, Spring 2009, Tiina Niklander 16.4.2009 40



PowerPC 620

- Branching logic
  - 256 entries in branch target buffer (BTB)
    - Set-associative, set size 2
  - 2048 entries in branch history table
    - Used only if branch target is in BTB
- Speculative execution of max 4 unresolved branch instructions
- Results in the extra (renaming) registers
  - Commit: copy to actual target register
  - Abort: release register for other use



Review Questions / Kertauskysymyksiä

- Differences / similarities of superscalar and trad. pipeline?
- What new problems must be solved?
- How to solve those?
- What is register renaming and why it is used?

- Miten superskalaaritoteutus eroaa tavallisesta liukuhihnoitetusta toteutuksesta?
- Mitä uusia rakenteesta johtuvia ongelmia tulee ratkottavaksi?
- Miten niitä ongelmia ratkotaan?
- Mitä tarkoittaa rekistereiden uudelleennimeäminen ja mitä hyötyä siitä on?