



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Lecture 12

Recapitulation (Kertaus)



Course Structure

- Week 1
 - Overview (Ch 1 – 8)
 - Bus (Ch 3)
 - Self-study: Digital logic
- Week 2
 - Memory, cache (Ch 4, 5)
 - Virtual memory (Ch 8.3-8.6)
- Week 3
 - Computer arithmetics (Ch 9)
 - Instruction set (Ch 10, 11)
- Week 4
 - CPU struc.& func. (Ch 12)
 - RISC-architecture (Ch 13)
- Week 5
 - Instruction-level parallelism, superscalar processor (Ch 14)
 - Control Unit (Ch 15-16)
- Week 6
 - Parallel Processing & Multicore (Ch 17-18)
 - **Recapitulation**

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 2




Exam Wed 3.3. at 16.00 in auditorium A111

- 2,5 hours – three or four questions
- You can write on all answers on the same paper using pencil or pen
- There is no need for a calculator, but a simple one is allowed
 - If there is math needed, you can just write the formula and you do not need to write the result number without a calculator



For the exam

- Go through the exercises
- Read the book and lecture slides
 - If there is nothing on the slides about the subsection, then there very probably is not a question in the exam
- The review questions in the slides are good hints!
- You can look for the collection of questions from the 2006 course. Teemu Kerola has collected several years of questions there
 - Direct link to the collection
<http://www.cs.helsinki.fi/u/kerola/tikra/kokeet/>




Lecture 1: Part 1 Overview (Ch 1-2 + 1-8) and Chapter 20 Digital logic

- Overview
 - No questions focusing only on this, but the content may be needed to understand future Chapters
- Chapter 7 I/O (7.1. – 7.5)
 - MUST KNOW: memory mapped I/O, interrupt-driven I/O, DMA
 - (covered in earlier course, but still valid)
- Digital logic
 - Boolean algebra, gates and flip-flops
 - No optimization, no Carnaugh maps
 - MUST KNOW:
 - from Boolean tables to gates
 - Flip-flops and basic circuits basic functionality

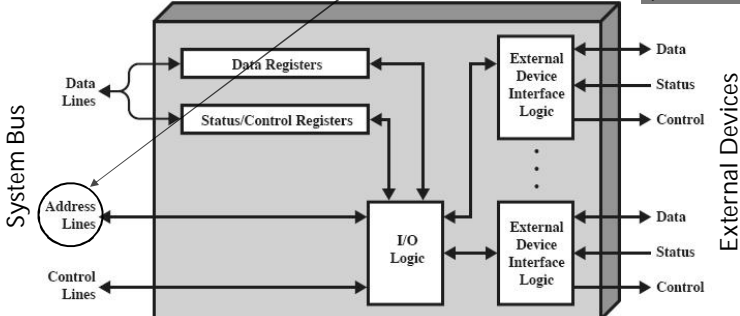
7th ed, 2006:
Appendix B:
Digital logic

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010
5




I/O controller and memory-mapped I/O

(Sta06 Fig 7.3)

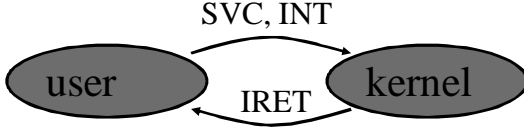


- Device driver (*ajur*) controls the device via controller's registers
- Driver refers to these registers as regular memory locations
 - Common memory references, like in load/store -instructions
 - Controller (*ohjain*) detects its own memory addresses on the bus
 - Device controller ~ 'intelligent' memory location

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010
6




User mode, kernel mode



- User mode, normal mode → kernel mode, privileged mode
 - Interrupt or special SVC instructions (service request)
 - Interrupt handler checks the right for mode change
- Kernel mode → User mode
 - Privileged instruction, for example IRET (return from interrupt)
 - Returns the control and mode as they were before the mode change
 - Very similar with return from a subroutine

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 7



Describing the Circuit

Boolean equations

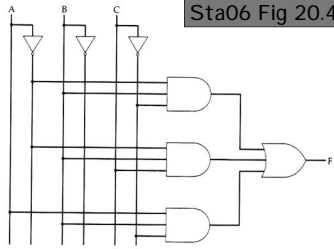
$$F = \overline{A}BC + A\overline{B}C + ABC$$

Truth table

inputs			output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

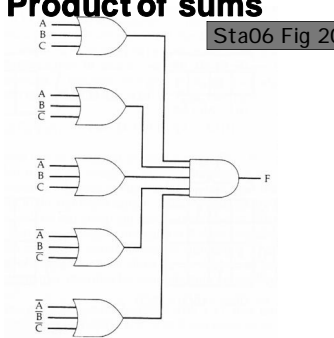
(Sta06 Table 20.3)

Sum of products




Sta06 Fig 20.4

Product of sums



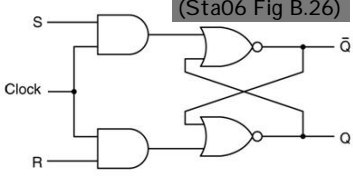
Sta06 Fig 20.5

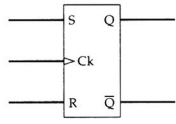
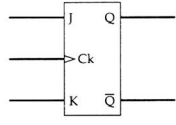
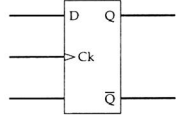
Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 8



Clocked Flip-Flops

(Sta06 Fig B.26)




Name	Graphic Symbol	Characteristic Table															
S-R		<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>S</th> <th>R</th> <th>Q_{n+1}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q_n</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>-</td> </tr> </tbody> </table>	S	R	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	-
S	R	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	-															
J-K		<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>J</th> <th>K</th> <th>Q_{n+1}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q_n</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>$\frac{1}{Q_n}$</td> </tr> <tr> <td>1</td> <td>1</td> <td>$\frac{1}{Q_n}$</td> </tr> </tbody> </table>	J	K	Q_{n+1}	0	0	Q_n	0	1	0	1	0	$\frac{1}{Q_n}$	1	1	$\frac{1}{Q_n}$
J	K	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	$\frac{1}{Q_n}$															
1	1	$\frac{1}{Q_n}$															
D		<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>D</th> <th>Q_{n+1}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	D	Q_{n+1}	0	0	1	1									
D	Q_{n+1}																
0	0																
1	1																

- State change can happen only when clock is 1
 - more control on state changes
- Clocked S-R Flip-Flop
- J-K Flip-Flop
 - Toggle Q when J=K=1
- D Flip-Flop
 - only one input D
 - D = 1 and CLOCK → write 1
 - D = 0 and CLOCK → write 0

Computer Organization II, Spring 2009, Tiina Niklander

25.2.2010 9




Lecture 2: Bus, Chapter 3

- Sections 3.1 – 3.3 part of lecture 1
 - Needed to understand the other sections
 - MUST KNOW: Instruction cycle, interrupts
- Sections 3.4 and 3.5: Bus and PCI
 - MUST KNOW: terms like speed, width, timing, signaling, arbitration
 - MUST KNOW: PCI read, PCI write sequences

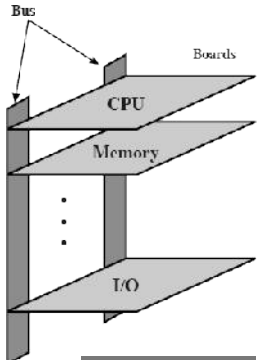
Computer Organization II, Spring 2010, Tiina Niklander

25.2.2010 10




Bus characteristics

- Width
 - ~ 50 – 100 lines (*johdinta*) – mother board, cable, connectors
- Bus type
 - Dedicated, non-multiplexed (*dedikoitu*)
 - Address and data – separate lines
 - Time multiplexed (*aikavuoroteltu*)
 - Address and data share lines
 - Address valid / data valid -line
- Arbitration (*käyttövuoron varaus*)
 - Centralized
 - One bus controller, arbiter (*väyläohjain*)
 - Distributed
 - Controllers have necessary logic



(Sta06 Fig 3.17)

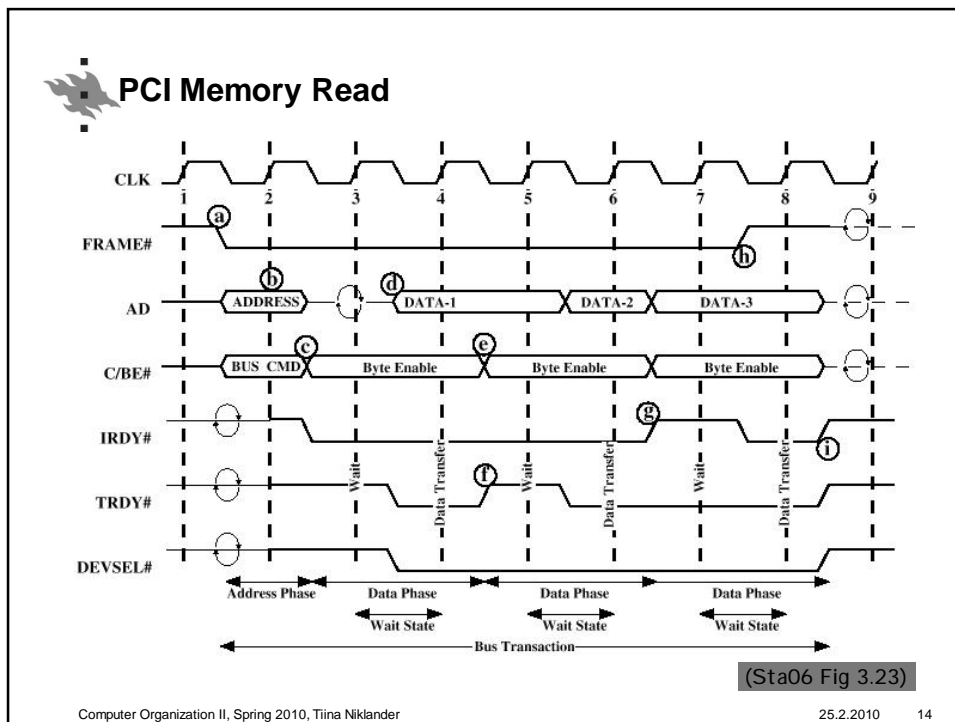
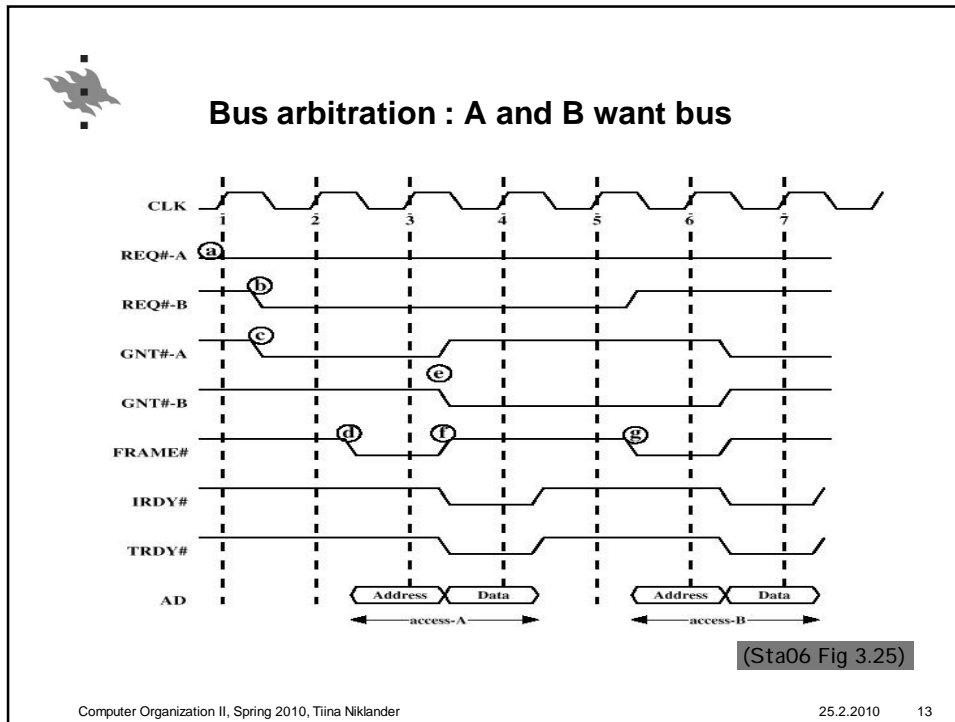
Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 11



Bus characteristics

- Timing (*ajoitus, tahdistus*)
 - Synchronous (*tahdistettu*)
 - Regular clock cycle (*kellopulssi*) – sequence of 0s and 1s
 - Asynchronous
 - Separate signals when needed
 - Shared traffic rules
 - everyone knows what is going to happen next
- Efficiency (*tehokkuus*)
 - Bandwidth (*kaistanleveys*)
 - How many bits per second

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 12






Packet-switched PCI Express (PCIe, PCI-E)

- PCI bus is too slow for some devices
- Replaces PCI bus (and possibly other I/O-bus)
 - Already available on new computers
- Hub on motherboard acting as a crossbar switch (*kytkin*)
- Based on point-to-point connections (*kaksipisteyhteys*)
 - Full-dublex, one lane has two lines (one send, one receive)
 - One device can use one or more (2,4,8,16,32) lanes
- Data stream (serial transfer)
 - Small packets (header + payload), bits in sequence
- No reservation, no control signals.
 - Each device may send at any time, when it wishes
 - Packet header contains the control information (like target)
- Data rate on one lane 250MB/s (future 3rd gen: 1GB/s)





Lecture 3: Cache and memory , Chapters 4 & 5

- Cache
 - MUST KNOW: all content, like cache organization, cache usage, access, write policies,
 - Mapping: Direct mapping, fully-associative, set-associative
- Memory
 - The most interesting part of memory section, 5.2. error correction, is NOT part of the course.
 - Not that important chapter
- Chapter 6 external memory - skip




Principle of locality

- In any given time period memory references occur only to a small subset of the whole address space
- Temporal locality (*ajallinen*) 
 - it is likely that a data item referenced a short time ago will be referenced again soon
- Spatial locality (*alueellinen*) 
 - it is likely that a data items close to the one referenced a short time ago will be referenced soon

MEM:

345	23	71	8	305	63	91	2
-----	----	----	---	-----	----	----	---

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 17



Cache Design

Cache Size	Write Policy
Mapping Function	Write through
Direct	Write back
Associative	Write once
Set Associative	Line Size
Replacement Algorithm	Number of caches
Least recently used (LRU)	Single or two level
First in first out (FIFO)	Unified or split
Least frequently used (LFU)	
Random	

- Cache Size & Line Size
 - Many blocks help for temporal locality
 - Large blocks help for spatial locality
 - Larger cache is slower
 - Multi-level cache

Typical sizes:
 L1: 8 KB - 64 KB
 L2: 256 KB - 8 MB
 L3: 2 MB - 48 MB

(Sta09Table 4.3)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 18

Direct Mapping

- Each block has only one possible location (line) in cache
 - determined by index field bits
- Several blocks may map into same cache line
 - identified with tag field bits

Block number (in memory)

34 bit address (byte address)

tag	index	byte offset
-----	-------	-------------

21 8 5

Unique bits that are different for each block, Stored into cache line

Cache line size ~ Block size = $2^5 = 32$ B

Fixed location in cache → fixed cache size = $2^8 = 256$ blocks = 8 KB

Sta09 Fig 4.8

Sta06 Fig 4.7

PaHe98 Fig 7.10

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 19

Fully Associative Mapping (6)

- Each block can be in any cache line
 - tag must be complete block number

Alpha AXP uses 34 bit memory addresses

Block number (in memory)

34 bit address (byte address)

tag	offset
-----	--------

29 5

Unique bits that are different for each block


Offset from the beginning of the block (in bytes)

Block size = $2^5 = 32$ B

Each block can be anywhere
Cache size can be any number of blocks

Sta06 Fig 4.9

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 20



Set Associative Mapping

- With set size $k=2$, each cache entry contain 2 blocks
 - Use set (set index) field to find the cache entry
 - Use tag to determine if the block belongs to the set
 - Use offset to find the proper byte in the block

34 bit address
(byte address)

tag	set	offset
22	7	5


Block size
 $= 2^5 = 32$ B

Unique bits that are different for each block, stored with block

Nr of sets = $v = 2^7 = 128$ blocks = 4 KB

Total cache size = $k*v = 2*4$ KB = 8 KB (without tag bits!)

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 21



Cache Write Policy – memory writes?

- Write through (*läpikirjoittava*)
 - Each write goes always to cache and memory
 - Each write is a cache miss!
- Write back (*lopuksi/takaisin kirjoittava*)
 - Each write goes only to cache
 - Write cache block back to memory
 - only when it is replaced in cache
 - Memory may have stale (old) data
 - cache coherence problem (*eheys, yhdenmukaisuus, yhtäpitävyys*)
- Write once (*"vain kerran kirjoittava?"*)
 - Write-invalidate Snoopy-cache coherence protocol for multiprocessors
 - Write invalidates data in other caches
 - Write to memory at replacement time, or when some other cache needs it (has read/write miss)

A bit set

Coherence problems:
 - More users of the same data: memory valid? cache valid?
 - multiple processors with own caches

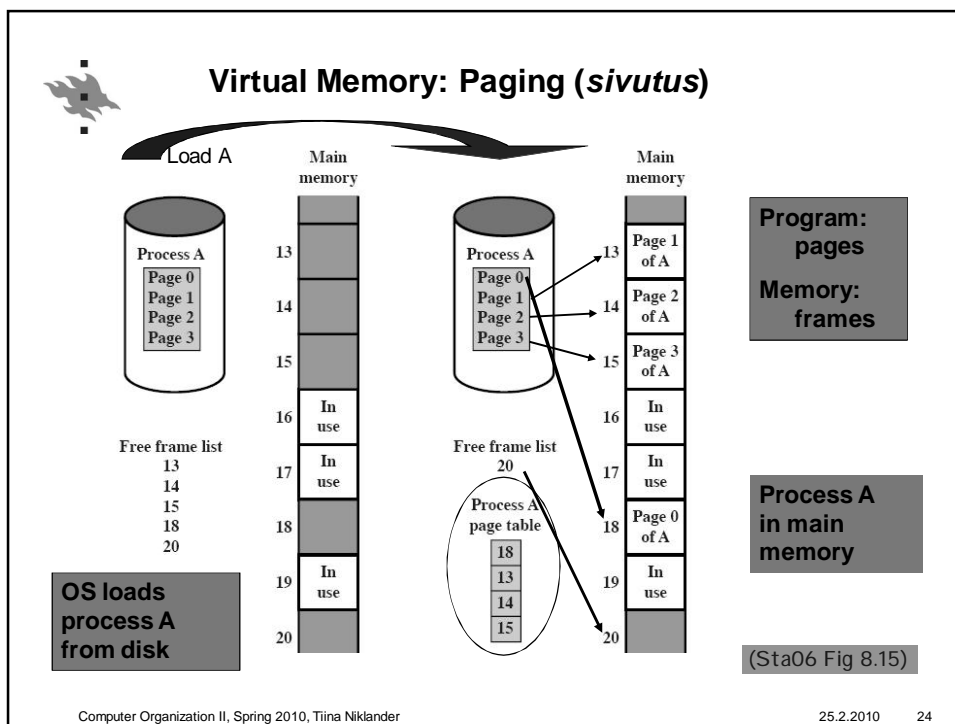
Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 22

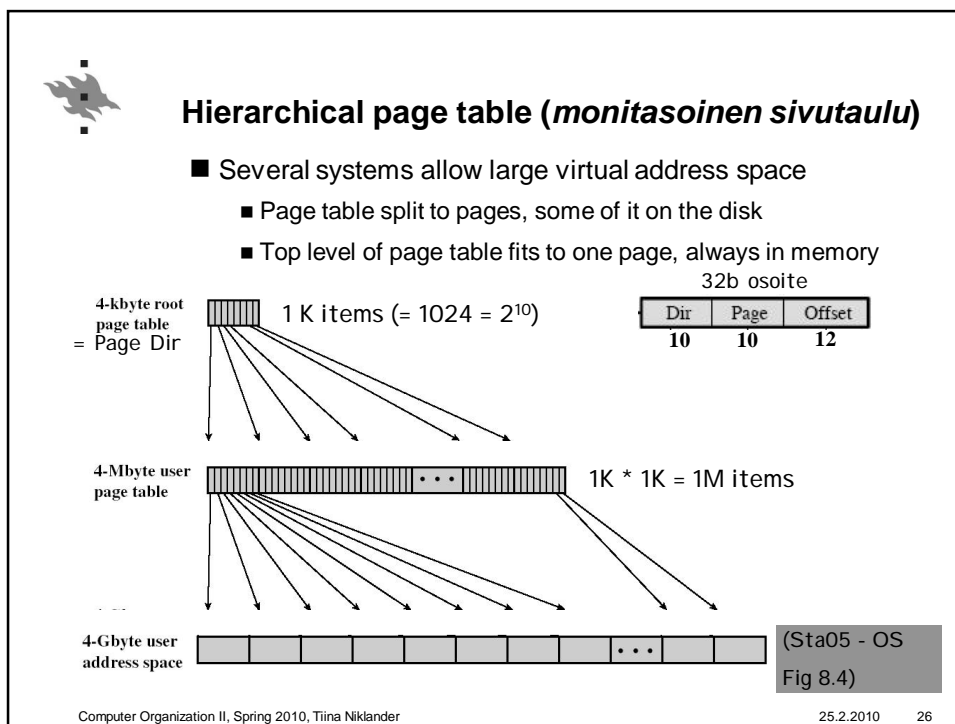
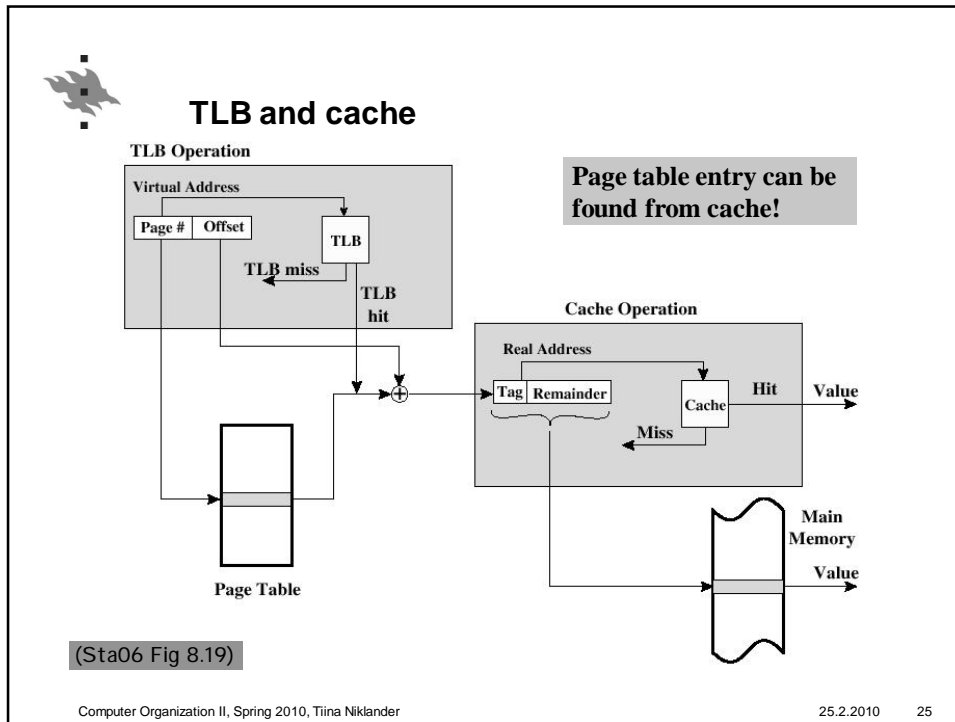
Lecture 4 Memory management, Chapters 8.3 – 8.6

- Memory management
 - MUST KNOW: virtual memory organization, page table, address translation, TLB, hierarchical page table like Pentium and ARM, combining paging, TLB and cache

7th ed, 2006:
8.4 PowerPC
(instead of ARM)

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 23





Pentium: Address translation

(Sta06 Fig 8.21)

- If *Paging=Enabled*, use page tables
else linear address = physical address (OS, f.ex. Device drivers?)
- Control registers (see further in the course book)


Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 27

ARM Virtual Memory Address Translation for Small Pages - Diagram

- Single L1 page table
 - 4K 32-bit entries
 - Each L1 entry points to L2 page table
- Each L2 page table
 - 256 32-bit entries
 - Each L2 entry points to 4-KB page in main memory
- 32-bit virtual address
 - 12 bit - L1
 - 8 bit - L2
 - 12 bit - offset (=page index)

(Sta09 Fig 8.23)


Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 28



Lecture 5: Computer arithmetic, Chapter 9

- Integer representation
 - MUST KNOW: sign-magnitude and twos complement, how to convert for different bit length
- Integer arithmetic
 - MUST KNOW: add, subtract, multiply, divide, Booth algorithm
- Floating-point representation
 - MUST KNOW: IEEE Standard,
- Floating-point arithmetic
 - MUST KNOW: over and under flow, general principles for calculations with floating points (not a detailed algorithm)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 29



Twos complement

- 1: invert all bits
- 2: add 1
- 3: Special cases
 - Ignore carry bit (*ylivuotobitti*)
 - Sign really changed?
 - Cannot negate smallest negative
 - Result in exception
- Simple hardware
- Easy to expand. As a 16-bit sequence

57 = 0011 1001 = 0000 0000 0011 1001

-57 = 1100 0111 = 1111 1111 1100 0111

-57 = 1100 0111

0011 1000

1

0011 1001

= 57

→ -128 = 1000 0000

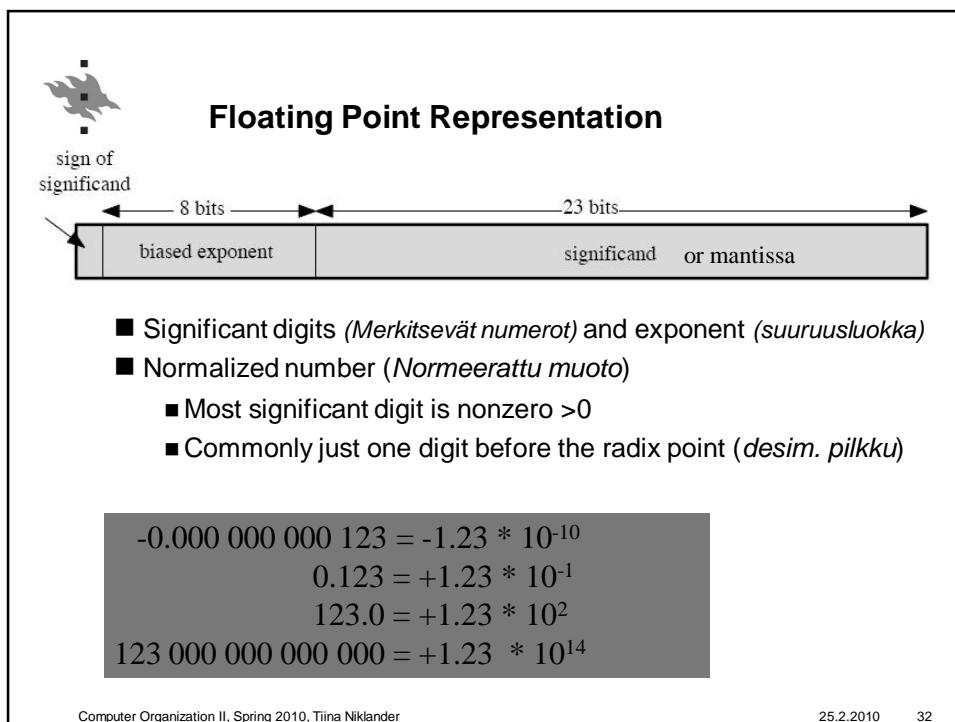
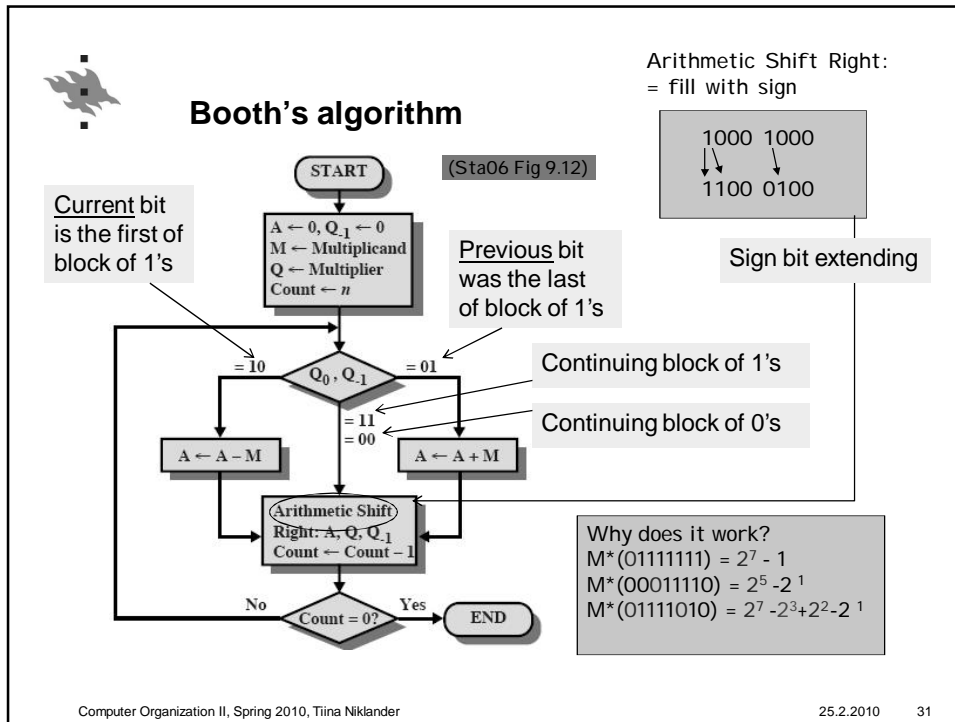
0111 1111

1

1000 0000

sign extension

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 30



Accuracy (*tarkkuus*) (32b)

- Value range (*arvoalue*)
 - 8 b exponent → $2^{-126} \dots 2^{127} \sim -10^{38} \dots 10^{38}$
- Not exact value
 - 24 b mantissa → $2^{24} \sim 1.7 * 10^{-7} \sim 6$ decimals
- Balancing between range and precision

Numerical errors: Patriot Missile (1991), Ariane 5 (1996)
<http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html>

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 33


Floating point arithmetics

Floating Point Numbers	Arithmetic Operations
$X = X_s \times B^{X_E}$ $Y = Y_s \times B^{Y_E}$	$\left. \begin{aligned} X + Y &= (X_s \times B^{X_E - Y_E} + Y_s) \times B^{Y_E} \\ X - Y &= (X_s \times B^{X_E - Y_E} - Y_s) \times B^{Y_E} \end{aligned} \right\} X_E \leq Y_E$ $X \times Y = (X_s \times Y_s) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_s}{Y_s} \right) \times B^{X_E - Y_E}$

$X = 0.3 \times 10^2 = 30$
 $Y = 0.2 \times 10^3 = 200$
(Sta06 Table 9.5)

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$
 $X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$
 $X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$
 $X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 34




Lecture 6: Instruction sets, Chapters 10 & 11

- **MUST KNOW:** everything about the instruction structure, representation, data types, addressing, instruction formats, also Pentium and ARM
- Specific instruction functionalities covered in earlier course. You need to know enough to be able to handle example 'programs' as in the exercises.
 - So no need to memorize specific instruction types and their mnemonic representations!

7th ed, 2006: PowerPC instead of ARM
 You may need to read the IA-64 Predication from 15.3 for the conditional execution of instructions

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 35



Addressing modes (Sta06 Table 11.1)

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	<u>No memory reference</u>	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	Operand = (R)	<u>No memory reference</u>	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	<u>No memory reference</u>	Limited applicability

- EA = Effective Address
- (A) = content of memory location A
- (R) = content of register R
- One register for the top-most stack item's address
- Register (or two) for the top stack item (or two)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 36

Pentium: Addressing modes (osoitustavat)

x86 Addressing Mode	Algorithm	
Immediate	Operand = A	1, 2, 4, 8B
Register Operand	Operand = (R)	Registers: 1, 2, 4, 8B
Displacement	LA = (SR) + A	
Base	LA = (SR) + (B)	For indexing arrays For arrays in stack or for two dimensional arrays
Base with Displacement	LA = (SR) + (B) + A	
Scaled Index with Displacement	LA = (SR) + (I) × S + A	
Base with Index and Displacement	LA = (SR) + (B) + (I) + A	different element size
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A	
Relative	LA = (PC) + A	

LA = linear address R = register
 (X) = contents of X B = base register
 SR = segment register I = index register
 PC = program counter S = scaling factor
 A = contents of an address field in the instruction

(Sta06 Table 11.2)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 37

ARM Addressing modes

- Load/Store
- Indirect
 - base reg + offset
- Indexing alternatives
 - Offset
 - Address is base + offset
 - Preindex
 - Form address
 - Write address to base
 - Postindex
 - Use base as address
 - Calculate new address to base

STRB r0, [r1, #12]

(a) Offset

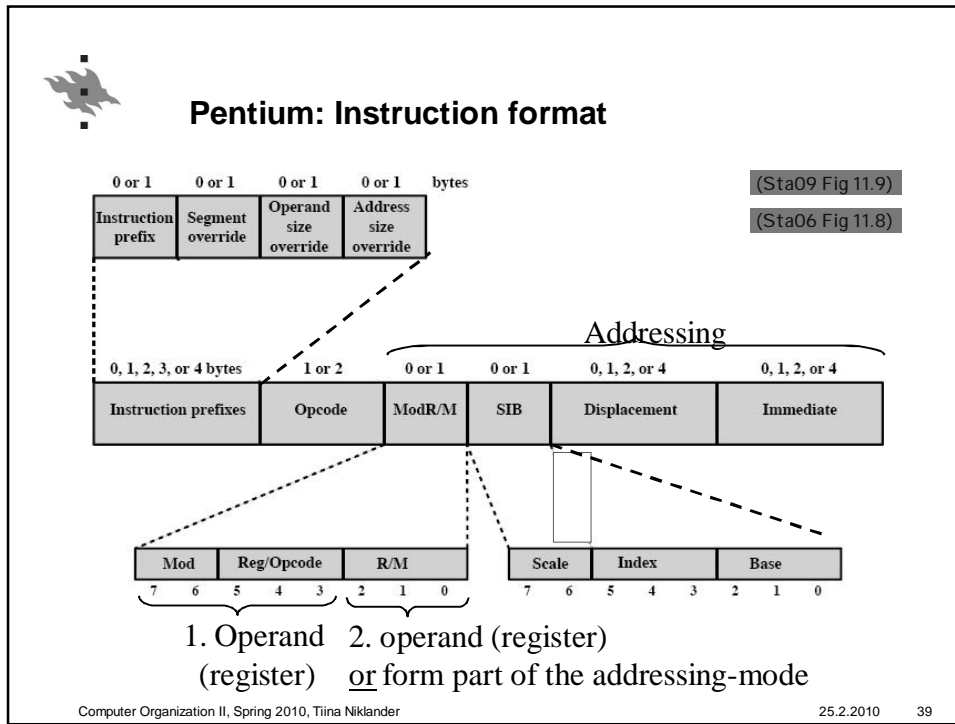
STRB r0, [r1, #12]!

(b) Preindex

STRBv r0, [r1], #12

(c) Postindex


Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 38



ARM Instruction Formats

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data processing immediate shift	cond	0	0	0	opcode			S	Rn			Rd			shift amount			shift	0	Rm												
data processing register shift	cond	0	0	0	opcode			S	Rn			Rd			Rs	0	shift	1	Rm													
data processing immediate	cond	0	0	1	opcode			S	Rn			Rd			rotate	immediate																
load/store immediate offset	cond	0	1	0	P	U	B	W	L	Rn			Rd			immediate																
load/store register offset	cond	0	1	1	P	U	B	W	L	Rn			Rd			shift amount			shift	0	Rm											
load/store multiple	cond	1	0	0	P	U	S	W	L	Rn			register list																			
branch/branch with link	cond	1	0	1	L	24-bit offset																										

- S = For data processing instructions, updates condition codes
- S = For load/store multiple instructions, execution restricted to supervisor mode
- P, U, W = distinguish between different types of addressing mode
- B = Unsigned byte (B==1) or word (B==0) access
- L = For load/store instructions, Load (L==1) or Store (L==0)
- L = For branch instructions, is return address stored in link register

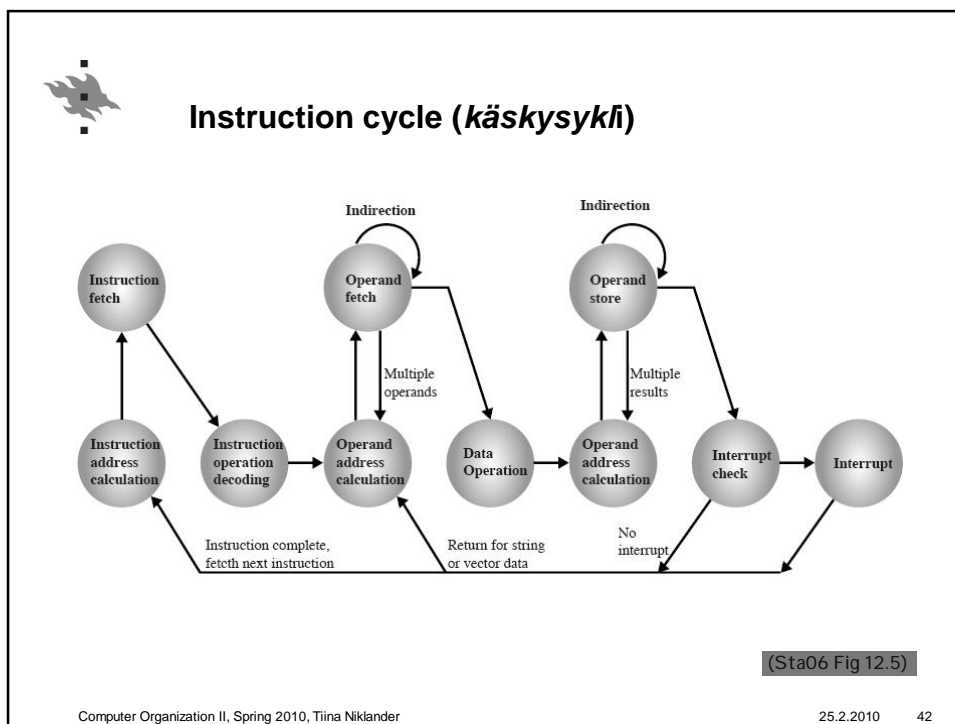


Lecture 7&8: Cpu structure and function, Chapter 12

- **MUST KNOW:** Everything, but not the tiny details of processors.
- **Most important issues:**
 - Instruction cycle details
 - Hazards, dependencies
 - Branching and pipelines
 - Register organization (different register types)
 - Typical program status word (PSW)

7th ed, 2006:
12.6 PowerPC
(instead of ARM)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 41



Instruction fetch (käskyn nouto)

- MAR ← PC
- MAR ← MMU(MAR)
- Control Bus ← Reserve
- Control Bus ← Read
- PC ← ALU(PC+1)
- MBR ← MEM[MAR]
- Control Bus ← Release
- IR ← MBR

MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Address Bus Data Bus Control Bus

Cache (välimuisti)!
 Prefetch (ennaltanouto)!

(Sta06 Fig 12.6)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 43

Operand fetch, Indirect addressing (Operandin nouto, epäsuora osoitus)

- MAR ← Address
- MAR ← MMU(MAR)
- Control Bus ← Reserve
- Control Bus ← Read
- MBR ← MEM[MAR]

- MAR ← MBR
- MAR ← MMU(MAR)
- Control Bus ← Read
- MBR ← MEM[MAR]
- Control Bus ← Release

ALU? Regs? ← MBR

Address Bus Data Bus Control Bus

Cache!

(Sta06 Fig 12.7)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 44

Data flow, interrupt cycle

- MAR ← SP
- MAR ← MMU(MAR)
- Control Bus ← Reserve
- MBR ← PC
- Control Bus ← Write
- MAR ← SP ← ALU(SP+1)
- MAR ← MMU(MAR)
- MBR ← PSW
- Control Bus ← Write
- SP ← ALU(SP+1)
- PSW ← privileged & disable
- MAR ← Interrupt number
- Control Bus ← Read
- PC ← MBR ← MEM[MAR] ← No address translation!
- Control Bus ← Release

SP = Stack Pointer (Sta06 Fig 12.8)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 45

Problems, design issues


- Structural dependency (*rakenteellinen riippuvuus*)
 - Several stages may need the same HW
 - Memory: FI, FO, WO
 - ALU: CO, EI
- Control dependency (*kontrolliriippuvuus*)
 - Jump destination of conditional branch known only after EI-stage
 - → Prefetched wrong instructions
- Data dependency (*datariippuvuus*)
 - Instruction needs the result of the previous non-finished instruction

STORE	R1, VarX
ADD	R2, R3, VarY
MUL	R3, R4, R5

ADD	R1, R7, R9
Jump	There
ADD	R2, R3, R4
MUL	R1, R4, R5

MUL	R1, R2, R3
LOAD	R6, Arr(R1)

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 46




Data dependency

- Read after Write (RAW) (a.k.a true or flow dependency)
 - Occurs if succeeding read takes place before the preceding write operation is complete
- Write after Read (WAR) (a.k.a antidependency)
 - Occurs if the succeeding write operation completes before the preceding read operation takes place
- Write after Write (WAW) (a.k.a output dependency)
 - Occurs when the two write operations take place in the reversed order of the intended sequence

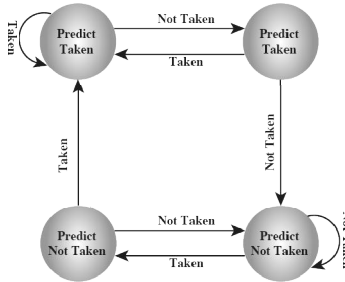
■ The WAR and WAW are possible only in architectures where the instructions can finish in different order

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 47




Dealing with branches

- Delayed branch
- Multiple instruction streams
 - Speculative execution
- Prefetch branch target
- Loop buffer
- **Branch prediction**
 - Static: always taken vs. never taken
 - Dynamic: based on Branch History Table




Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 48



Lecture 8: RISC, Chapter 13

- **MUST KNOW:** Everything, but 13.6 MIPS, from 13.7 Sparc only the register set is needed
- RISC vs CISC
- Load/Store architecture
- RISC pipelining
- Register windows, register optimization

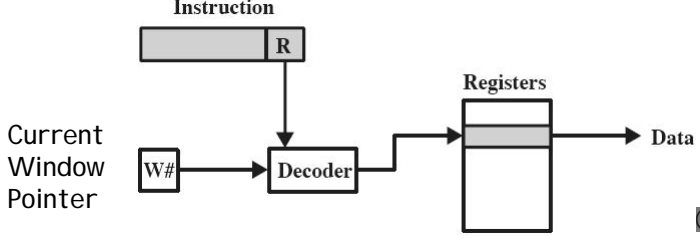
Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 49



Register storage (Register file)

- More registers than addressable in the instruction
 - E.g. SPARC has just 5 bits for register number → 0.. 31, but the processor has 40 to 540 registers
- Small subset of registers available for each instruction in **register window**
 - In the window references to register r0-r31
 - CPU maps them to actual (true) registers r0-r539

Instruction



(Sta06 Fig 13.3)

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 50

RISC-pipeline, Delayed Branch

100 LOAD X, rA

101 ADD I, rA

102 JUMP 105

103 ADD rA, rB

105 STORE rA, Z

1	2	3	4	5	6	7	8
I	E	D					
	I		E				
			I	E			
				I			
					I	E	D

Traditional

100 LOAD X, rA

101 ADD I, rA

102 JUMP 106

103 NOOP

106 STORE rA, Z

I	E	D					
	I	E					
		I	E				
			I	E			
				I	E	D	

RISC with inserted NOOP
Two port MEM

100 LOAD X, rA

101 JUMP 105

102 ADD I, rA

105 STORE rA, Z

I	E	D					
	I	E					
		I	E				
			I	E	D		

RISC with reversed instructions
(Sta06 Fig 13.7)

Cond. branch (*ehdollinen hypy*):
 JZERO 105, rA ??

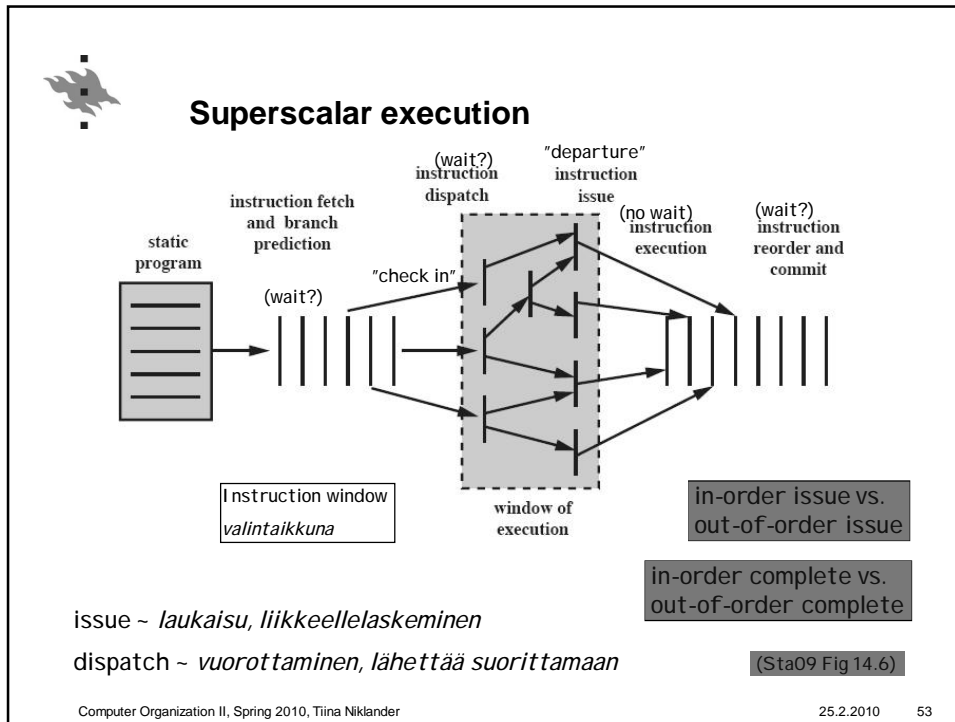
Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 51

Lecture 9: Superscalar, Chapter 14

- **MUST KNOW:** Everything, but the tiny details of the processors
- In-order / out-of-order issue / complete
- Instruction selection window
- Register renaming

7th ed, 2006:
 14.4 PowerPC
 (instead of ARM)

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 52

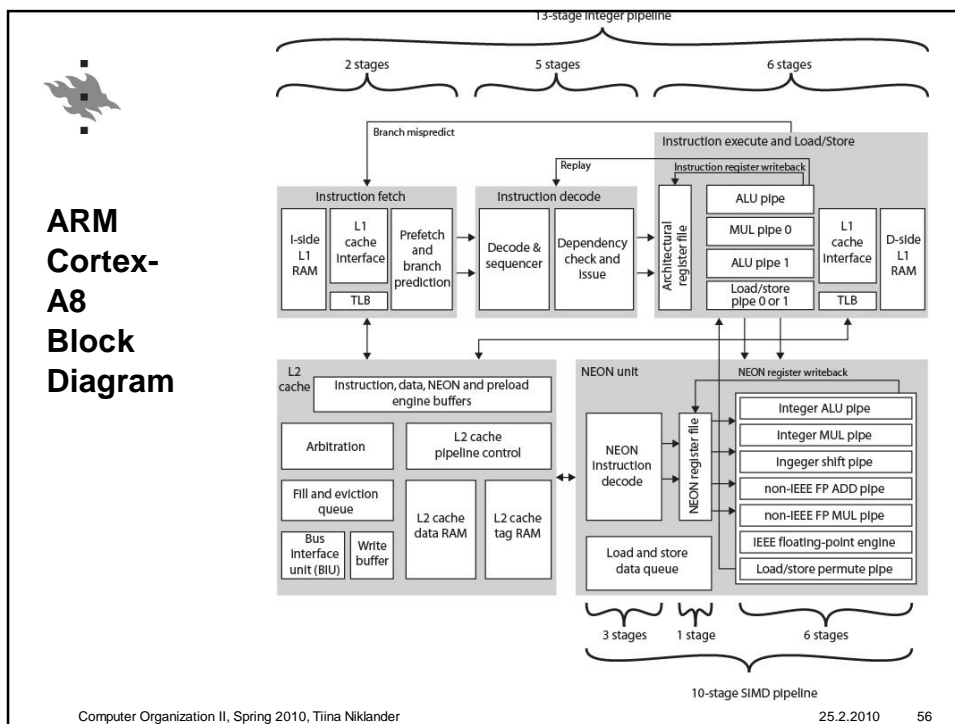
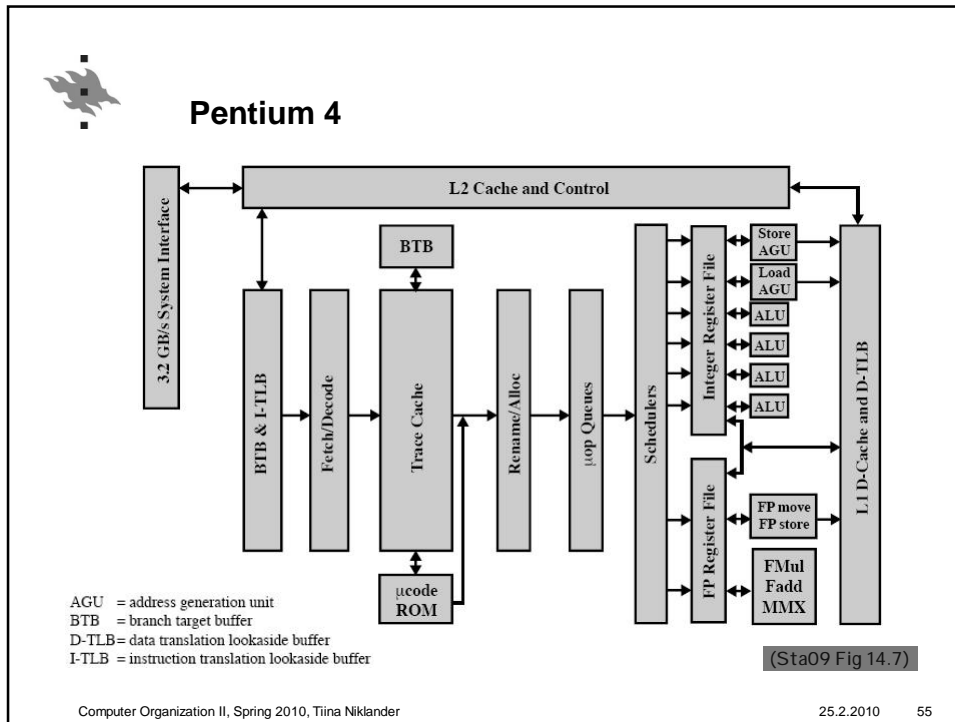


-
- Register renaming**
(*rekistereiden uudelleennimeäminen*)
- One cause for some of the dependencies is the usage of names
 - The same name could be used for several independent elements
 - Thus, instructions have unneeded write and antidependencies
 - Causing unnecessary waits
 - Solution: Register 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 capable of allocating and managing registers and performing the needed mapping
- $$R3 \leftarrow R3 + R5$$

$$R4 \leftarrow R3 + 1$$

$$R3 \leftarrow R5 + 1$$

$$R7 \leftarrow R3 + R4$$
- Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 54



Lecture 10: Control-Unit, Chapters 15 & 16

- Chapter 15 Everything but the tiny details of processors
- Chapter 16: 16.1 – 16.3

- Micro-operation sequences in different phases of the execution cycle
- Control signals

7th ed, 2006:
Chapters 16 & 17.1 -17.3

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 57

Instruction cycle flow chart (as state-machine?)

■ ICC: I nstruction Cycle Code register 's state

```

graph TD
    Start(( )) --> ICC00[ICC = 00]
    ICC00 --> Setup[Setup interrupt]
    Setup --> ICC00
    
    ICC00 --> ICC10[ICC = 10]
    ICC10 --> Read[Read address]
    Read --> ICC10
    
    ICC10 --> ICC01[ICC = 01]
    ICC01 --> Fetch[Fetch instruction]
    Fetch --> Indirect{Indirect addressing?}
    Indirect -- No --> ICC10
    Indirect -- Yes --> ICC01
    
    ICC01 --> ICC00
    
    ICC00 --> ICC11[ICC = 11]
    ICC11 --> Interrupt{Interrupt for enabled interrupt?}
    Interrupt -- No --> ICC00
    Interrupt -- Yes --> ICC11
    
```

(Sta09 Fig 15.3)

(Sta06 Fig 16.3)

Computer Organization II, Spring 2010, Tiina Niklander
25.2.2010 58

Control signals and micro-operations

Micro-operations	Timing	Active Control Signals
Fetch:	$t_1: \text{MAR} \leftarrow (\text{PC})$	C_2
	$t_2: \text{MBR} \leftarrow \text{Memory}$	C_5, C_R
	$\text{PC} \leftarrow (\text{PC}) + 1$??
Indirect:	$t_1: \text{MAR} \leftarrow (\text{IR}(\text{Address}))$	C_8
	$t_2: \text{MBR} \leftarrow \text{Memory}$	C_5, C_R
	$t_3: \text{IR}(\text{Address}) \leftarrow (\text{MBR}(\text{Address}))$	C_4
Interrupt:	$t_1: \text{MBR} \leftarrow (\text{PC})$	C_1
	$t_2: \text{MAR} \leftarrow \text{Save-address}$ $\text{PC} \leftarrow \text{Routine-address}$??
	$t_3: \text{Memory} \leftarrow (\text{MBR})$	C_{12}, C_W

C_R = Read control signal to system bus.
 C_W = Write control signal to system bus.

(Sta09 Table 15.1) (Sta06 Table 16.1)

Sta09 Fig 15.5
Sta06 Fig 16.5

Vertical vs. Horizontal Microcode (3)

Simple register transfers

$0_1 0_1 0_1 0_1 0_1 0_1$ MDR \leftarrow Register

$0_1 0_1 0_1 0_1 0_1 0_1$ Register \leftarrow MDR

$0_1 0_1 0_1 0_1 1_0 0_1$ MAR \leftarrow Register

Register select

Memory operations

$0_1 0_1 0_1 0_1 0_1 0_1$ Read

$0_1 0_1 1_0 0_1 0_1 0_1$ Write

need 2 bits!
State 0: no mem-op

Special sequencing operations

$0_1 1_0 0_1 0_1 0_1 0_1$ CSAR \leftarrow Decoded MDR

$0_1 1_0 0_1 0_1 0_1 0_1$ CSAR \leftarrow Constant (in next byte)

$0_1 1_0 0_1 0_1 1_0 0_1$ Skip

ALU operations

$0_1 1_1 0_1 0_1 0_1 0_1$ ACC \leftarrow ACC + Register

$0_1 1_1 0_1 0_1 0_1 0_1$ ACC \leftarrow ACC - Register

$0_1 1_1 0_1 0_1 1_0 0_1$ ACC \leftarrow Register

$0_1 1_1 0_1 0_1 1_1 0_1$ Register \leftarrow ACC

$0_1 1_1 1_1 0_1 0_1 0_1$ ACC \leftarrow Register + 1

Register select

(a) Vertical microinstruction format (by resource)

(b) Horizontal microinstruction format

Vertical vs. Horizontal Microcode (3)

Next microinstruction address (CAR = CSAR)
Assumption: CAR=CAR+1

(Sta09 Fig 16.12)

(Sta06 Fig 17.12)



Next microinstruction?


- Selection normally based on flags
- Explicit: both addresses in the instruction
- Implicit: sequentially to next, but 'jump target' in instruction
- Variable format; separate jump instructions use the bits for address, signal instruction use the same bits for signals
- Address generation during execution:
 - Address combined directly from op-code and flags
- Subroutines and residual control: possibility to store one return address



Lecture 11: Parallel processing and multicore Chapters 17 & 18

- Chapters 17.1. – 17.6. in exam
- Chapter 18.3. multicore organization might be in exam
- Most important: cache coherence and MESI
- Other issues: SMP, NUMA and Clusters


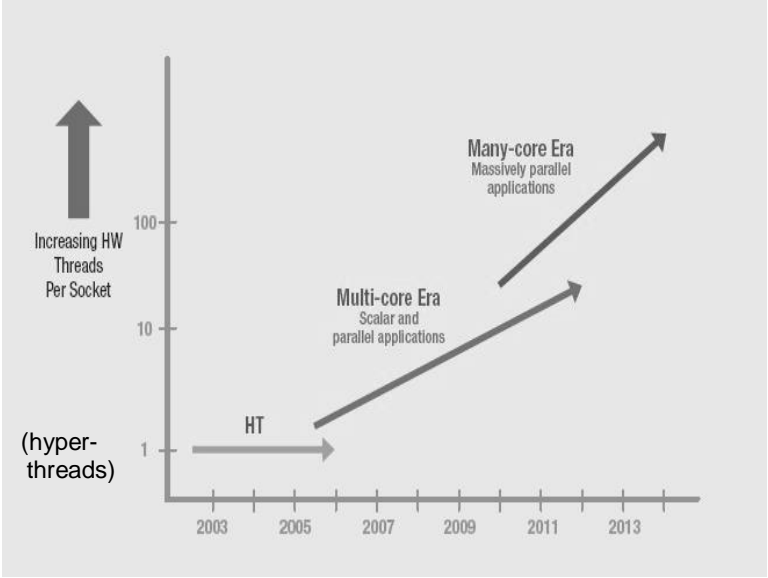
7th ed, 2006: Chapter 18 Parallel Processing,
Multicore organization not in the book



Example exam questions

- Available from 2006 course page:
- <http://www.cs.helsinki.fi/u/kerola/tikra/kokeet/>
- Page contains earlier exams, but a lot of them are only in Finnish because very few international students at that time.
- **Kk** is a course exam, **ek** separate exam,
- If the name has **e** or **en** in the end, the questions are in English

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 63

Borkar, Dubey, Kahn, et al. "Platform 2015." Intel White Paper, 2005. ([click](http://download.intel.com/technology/computing/archinnov/platform2015/download/Platform_2015.pdf))

http://download.intel.com/technology/computing/archinnov/platform2015/download/Platform_2015.pdf

Computer Organization II, Spring 2010, Tiina Niklander 25.2.2010 64

-- The End --

STI Cell Power processor element
(a) major units and
(b) pipeline

