Lecture 12

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

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**

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/

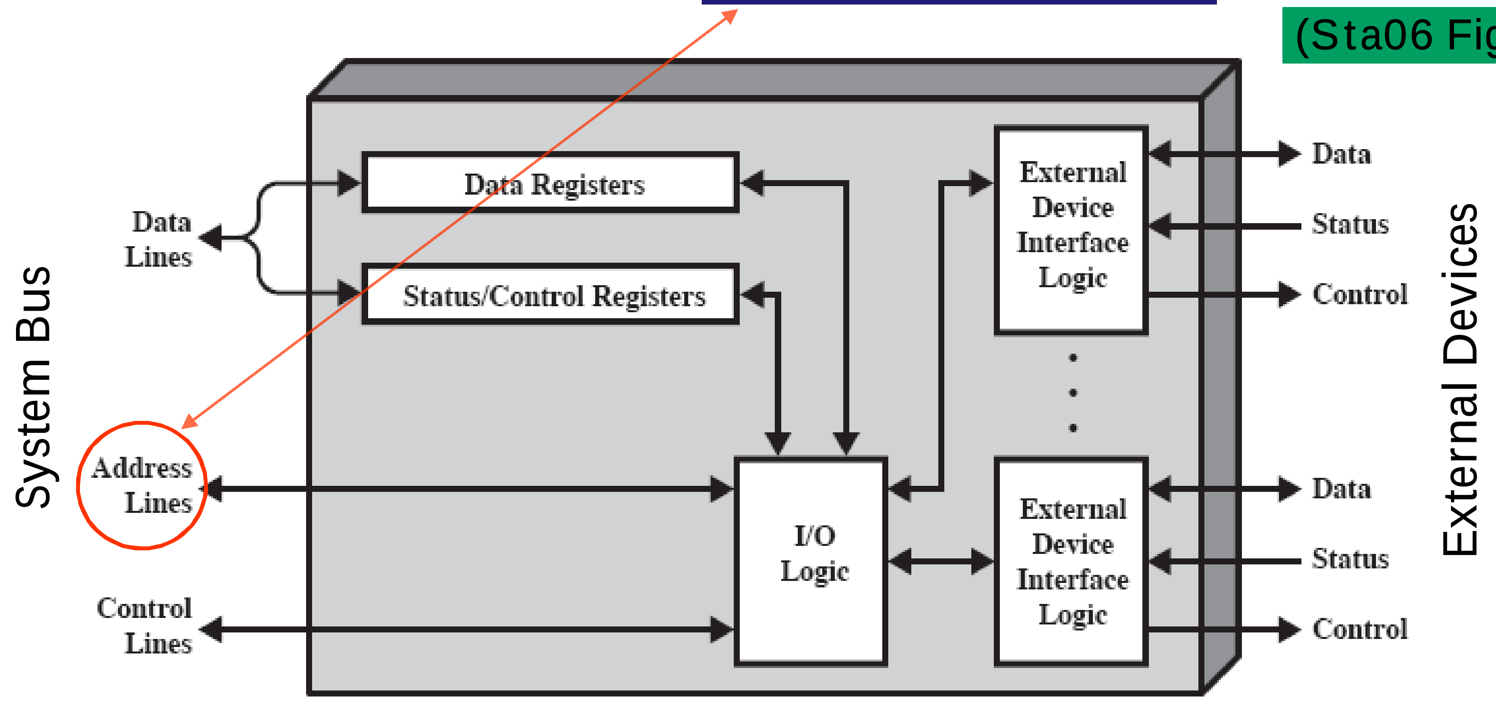
Lecture 1: Part 1 Overview (Ch 1-2 + 1-8) and Chapter 20 Digital logic

- 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

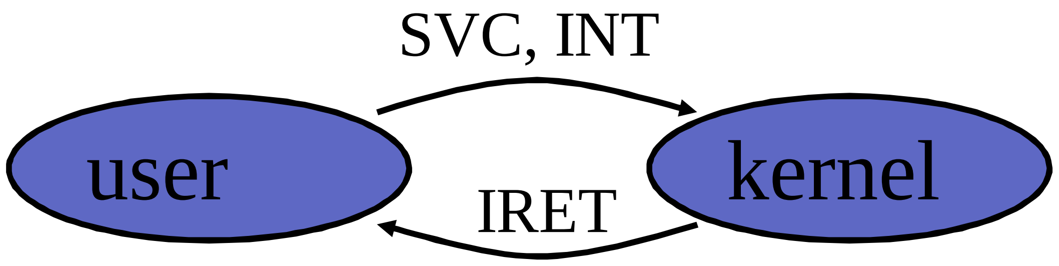
I/O controller and memory-mapped I/O

(Sta06 Fig 7.3)



- Device driver (*ajuri*) 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

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 instuction, for example IRET (return from interrupt)
  - Returns the cotnrol and mode as they were before the mode change
    - Very similar with return from a subroutine

Describing the Circuit

Boolean equations

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

Truth table

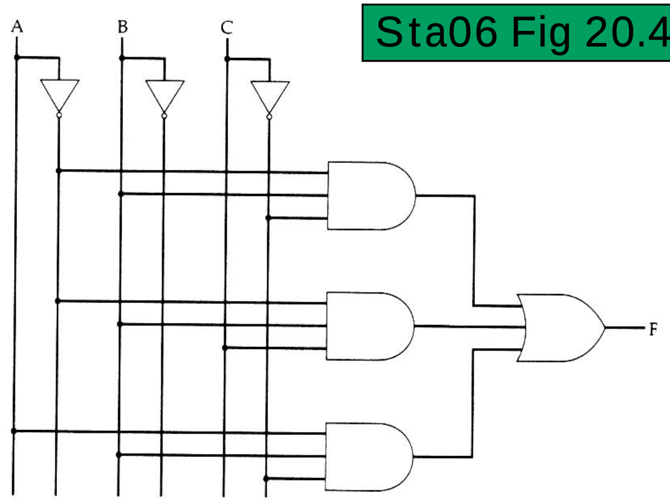
<------------- inputs -----------> <- output ->

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)

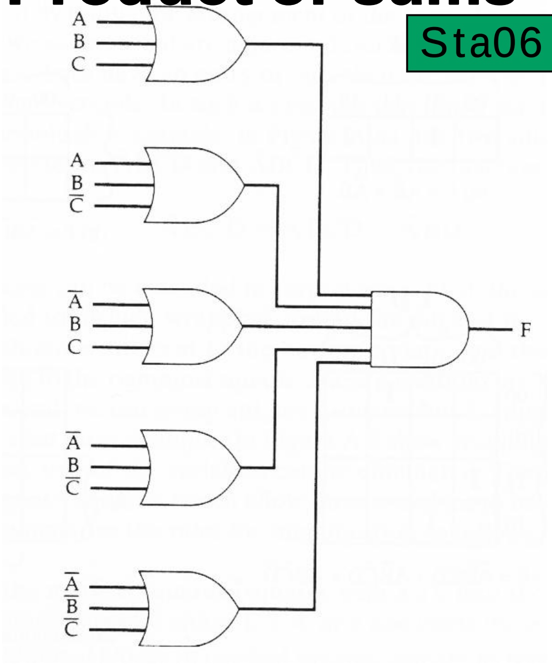
Sum of products

Sta06 Fig 20.4

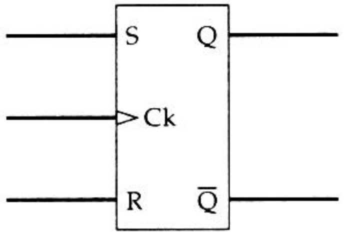
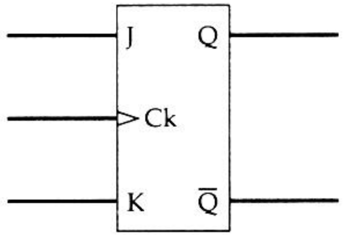
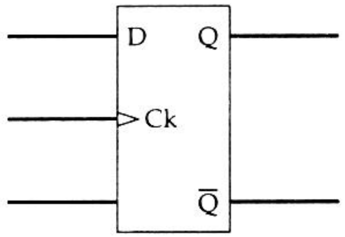


Product of sums

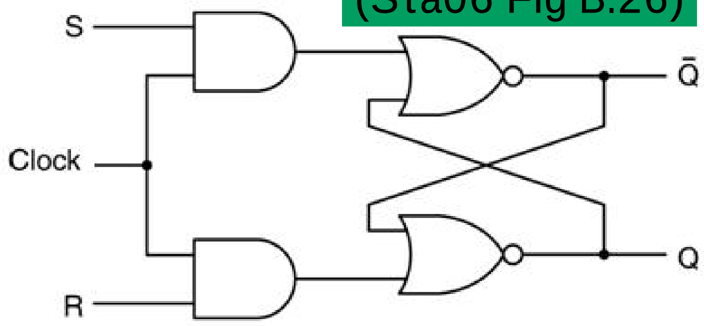
Sta06 Fig 20.5


A B C
A B $\overline{C}$
$\overline{A}$ B C
$\overline{A}$ B $\overline{C}$
$\overline{A}$ $\overline{B}$ $\overline{C}$
F

Clocked Flip-Flops

Name	Graphic Symbol	Characteristic Table
S–R		S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J–K		J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\bar{Q}_n$
D		D Q_{n+1} 0 0 1 1

(Sta06 Fig B.26)



- 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

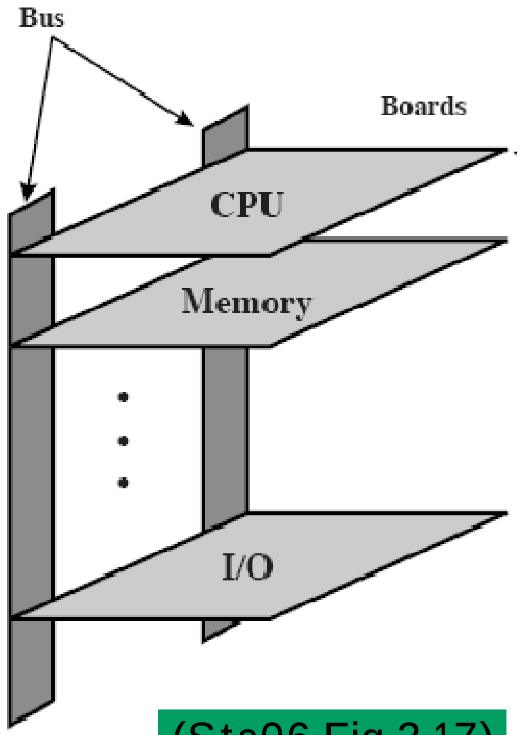
Lecture 2: Bus, Chapter 3

- 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

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



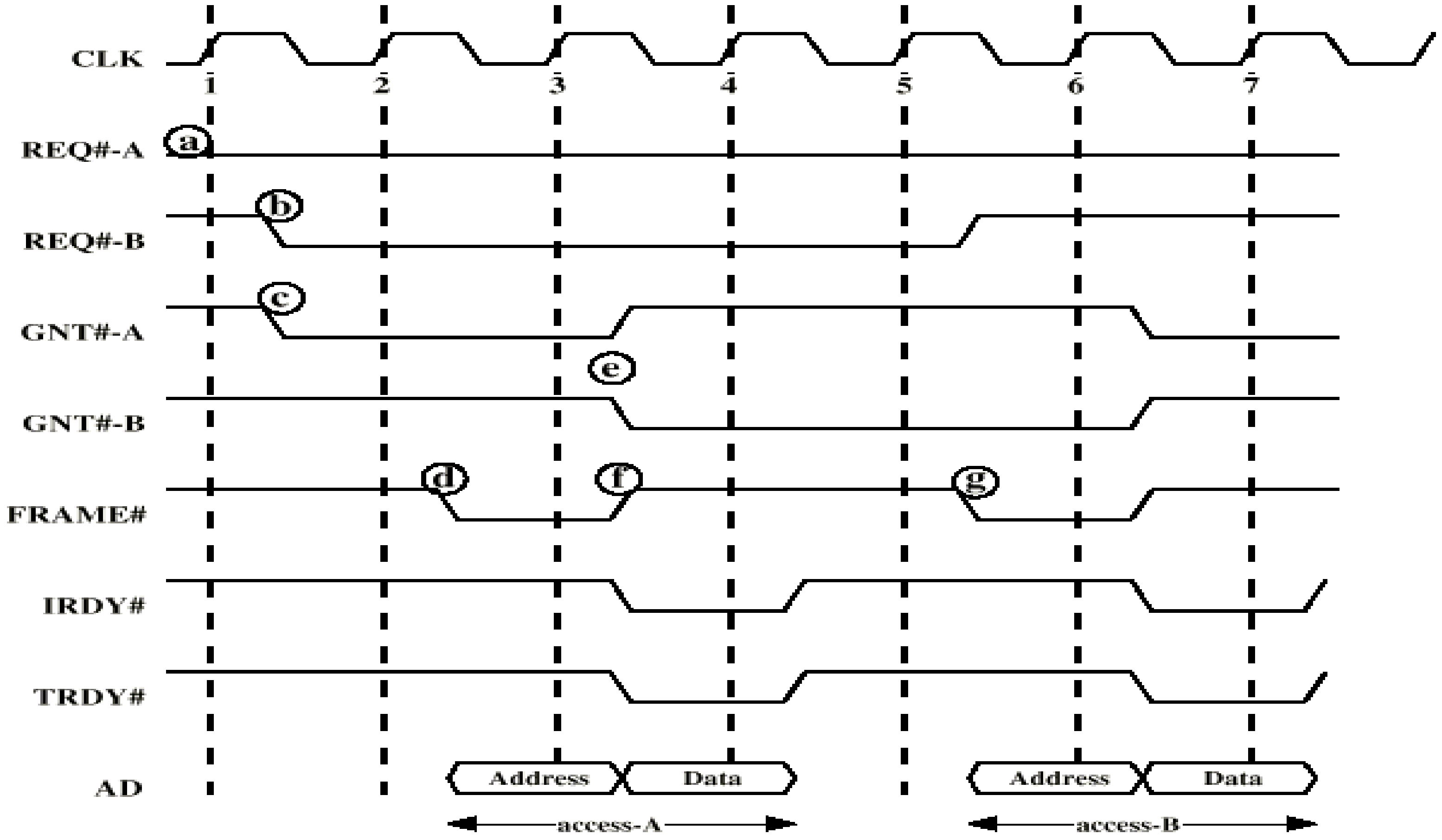
(Sta06 Fig 3.17)

Bus characteristics

- Timing (*ajoitus*, tahdistus)
  - Synchronous (*tahdistettu)*
    - Regular clock cycle (*kellopulssi)* – sequence of 0s and 1s
  - Asynchronous
    - Separate signals when needed
  - Shared traffic rules

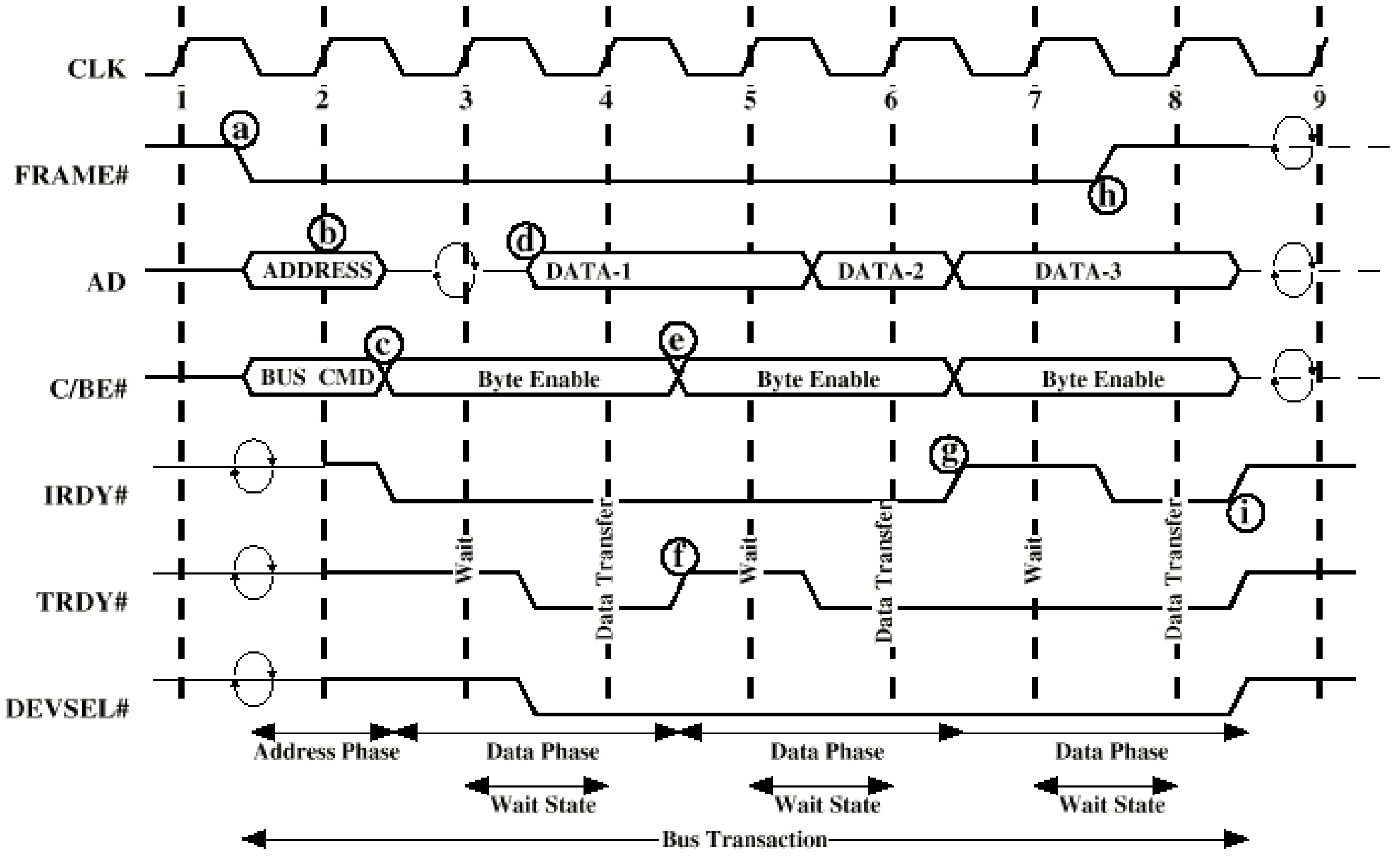
    everyone knows what is going to happen next
- Efficiency (*tehokkuus*)
  - Bandwidth (*kaistanleveys*)
    - How many bits per second

Bus arbitration : A and B want bus



(Sta06 Fig 3.25)

PCI Memory Read



(Sta06 Fig 3.23)

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

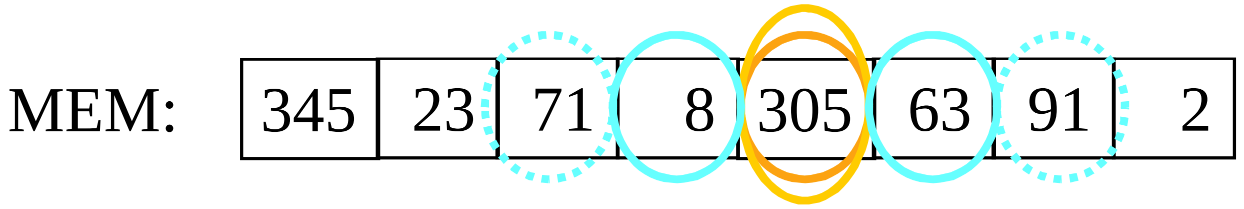
- PCI bus is too slow for some devices
- Replaces PCI bus (and possibly other I/O-bus)
    - Already available on new computers
- Hub on motherboard acting as a crossbar switch (*kytkin*)
- Based on point-to-point connections (*kaksipisteyhteys*)
    - Full-dublex, one lane has two lines (one send, one receive)
    - One device can used 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

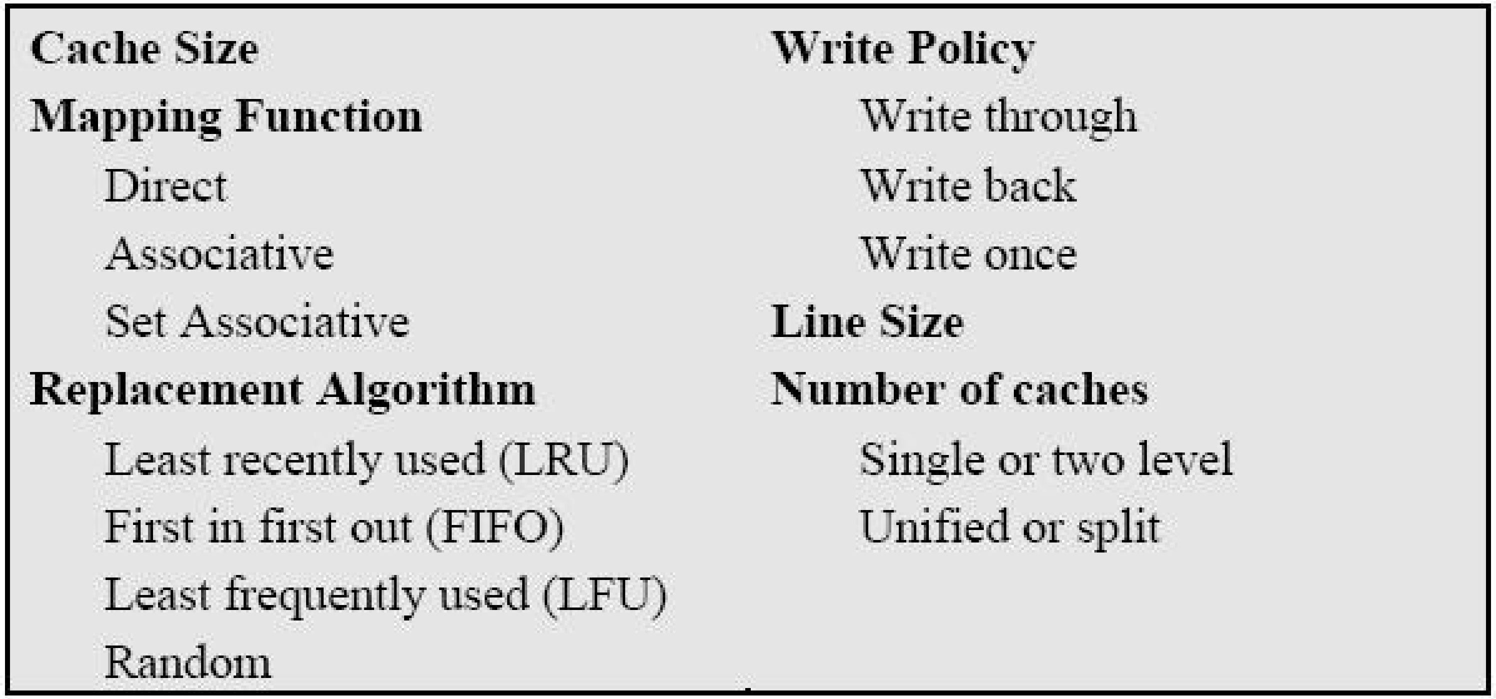
- Cache
  - MUST KNOW: all content, like cache organization, cache usage, access, write policies,
  - Mapping: Direct mapping, fully-associative, set-associative
- Memory
  - The most interesting part of memory section, 5.2. error correction, is NOT part of the course.
  - Not that important chapter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



Cache Design



Cache Size

Mapping Function

- Direct
- Associative
- Set Associative

Replacement Algorithm

- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy

- Write through
- Write back
- Write once

Line Size

Number of caches

- Single or two level
- Unified or split

- 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

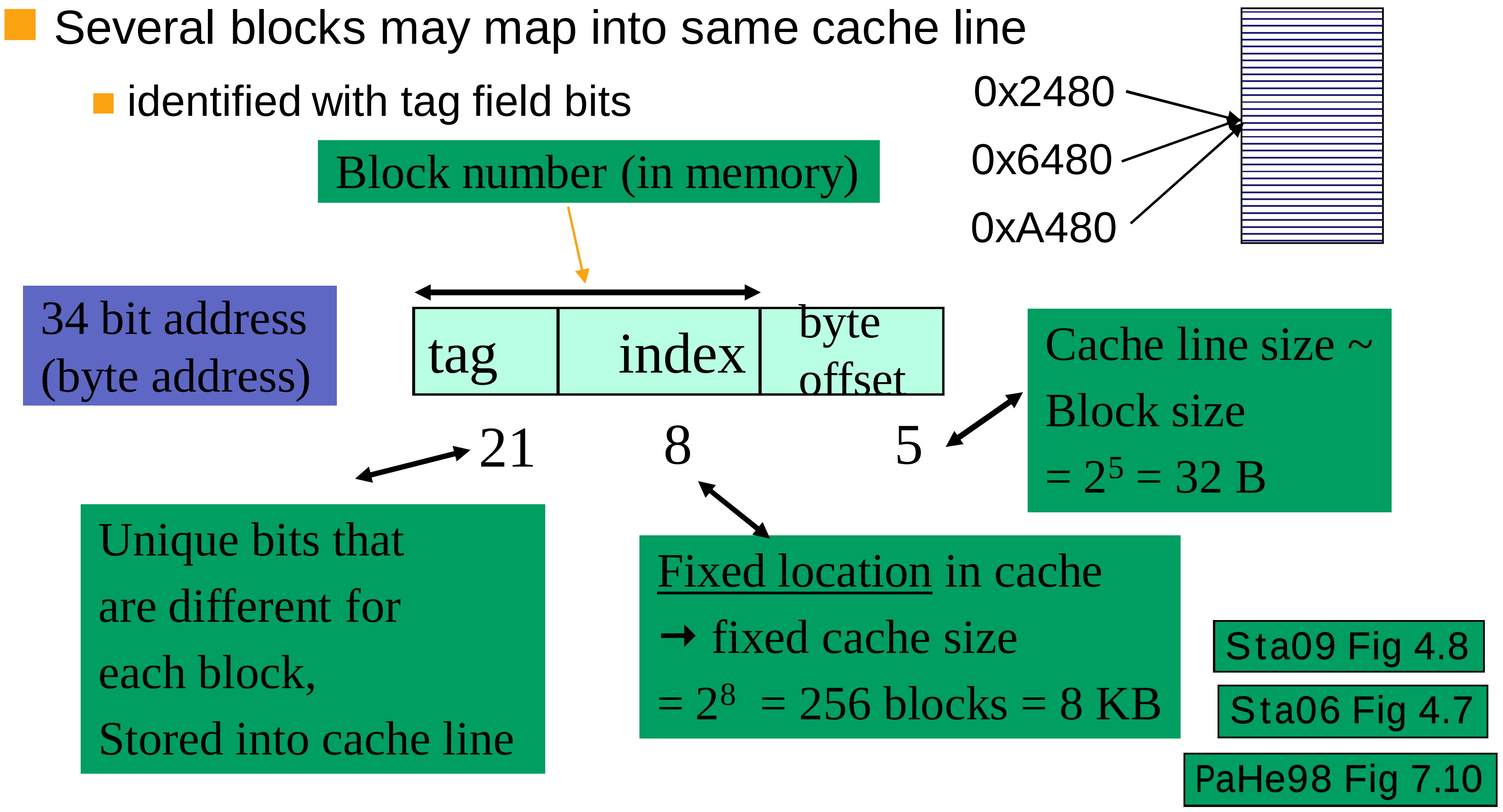
L2: 256 KB - 8 MB

L3: 2 MB - 48 MB

(Sta09Table 4.3)

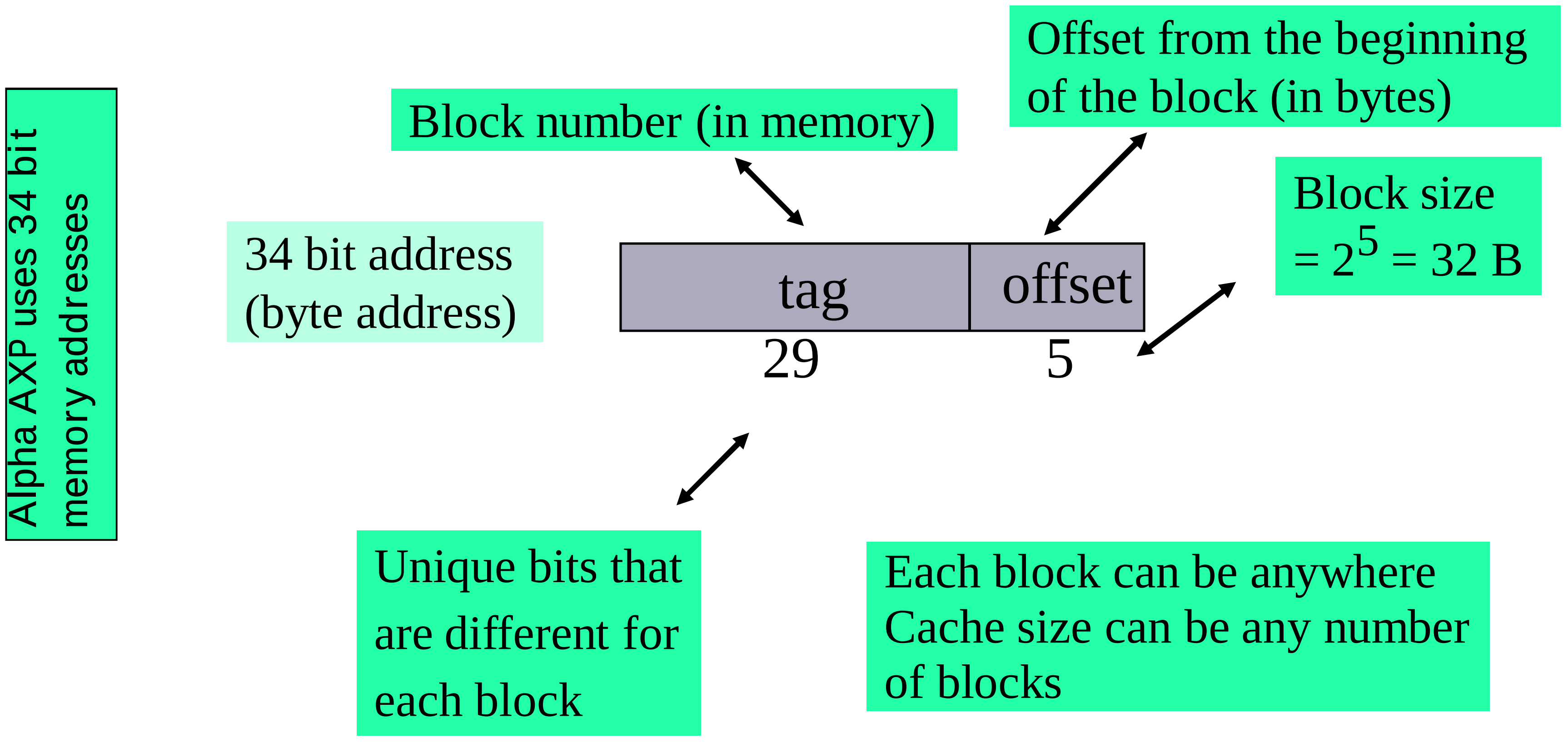
Direct Mapping

- 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


0x2480
0x6480
0xA480
Block number (in memory)
34 bit address (byte address)
tag
index
byte offset
21
8
5
Cache line size ~ Block size = 2^5 = 32 B
Unique bits that are different for each block, Stored into cache line
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

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
34 bit address (byte address)
Block number (in memory)
Offset from the beginning of the block (in bytes)
Block size = 2^5 = 32 B
tag
offset
29
5
Unique bits that are different for each block
Each block can be anywhere
Cache size can be any number of blocks

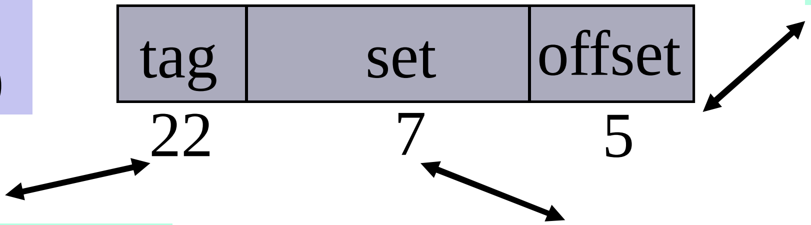
Sta06 Fig 4.9

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

Block size = 2^5 = 32 B

34 bit address (byte address)



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!)

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

A bit set

  - 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)

Coherence problems:

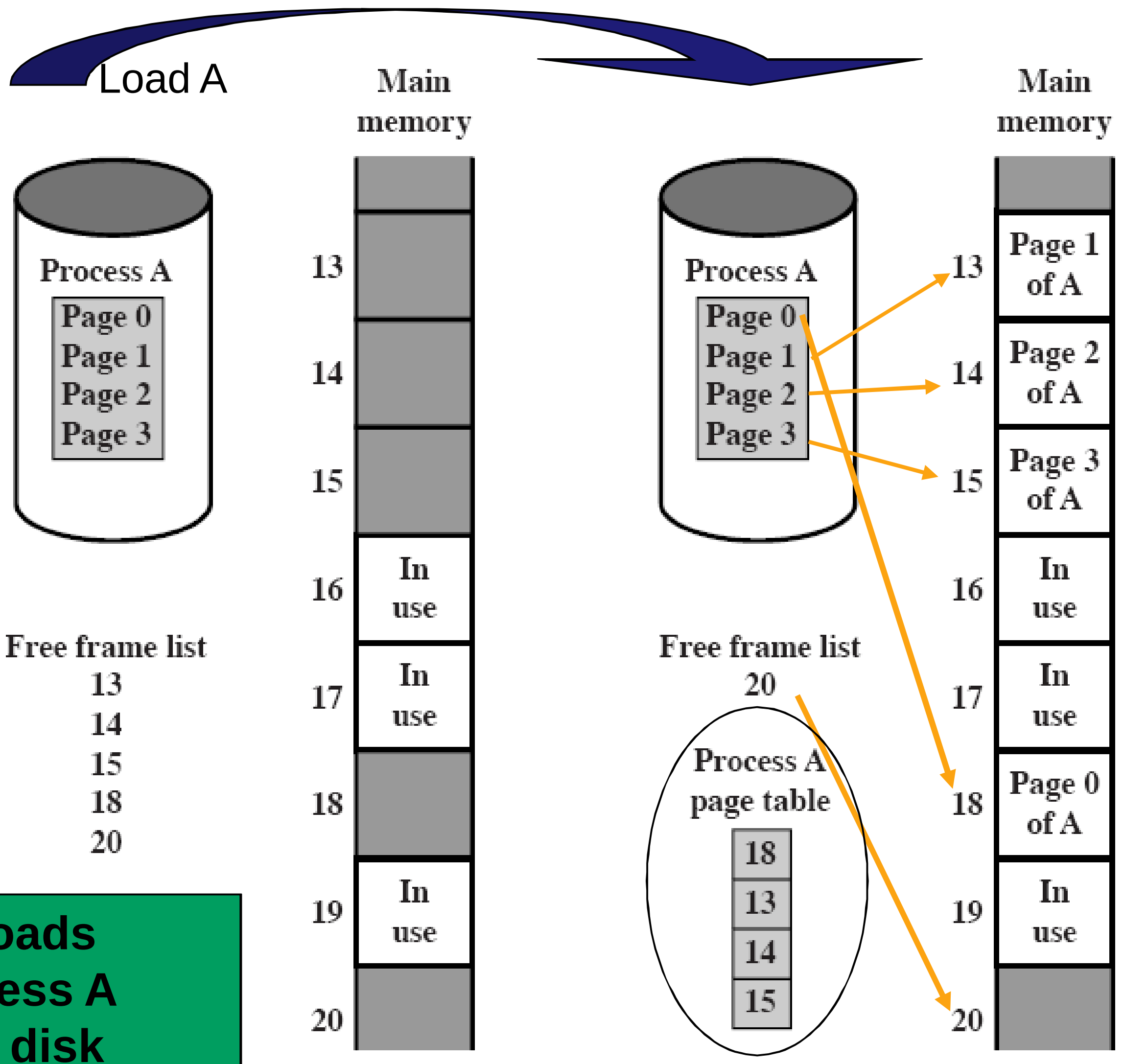
- More users of the same data: memory valid? cache valid?
- multiple processors with own caches

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)

Virtual Memory: Paging (*sivutus*)



OS loads process A from disk

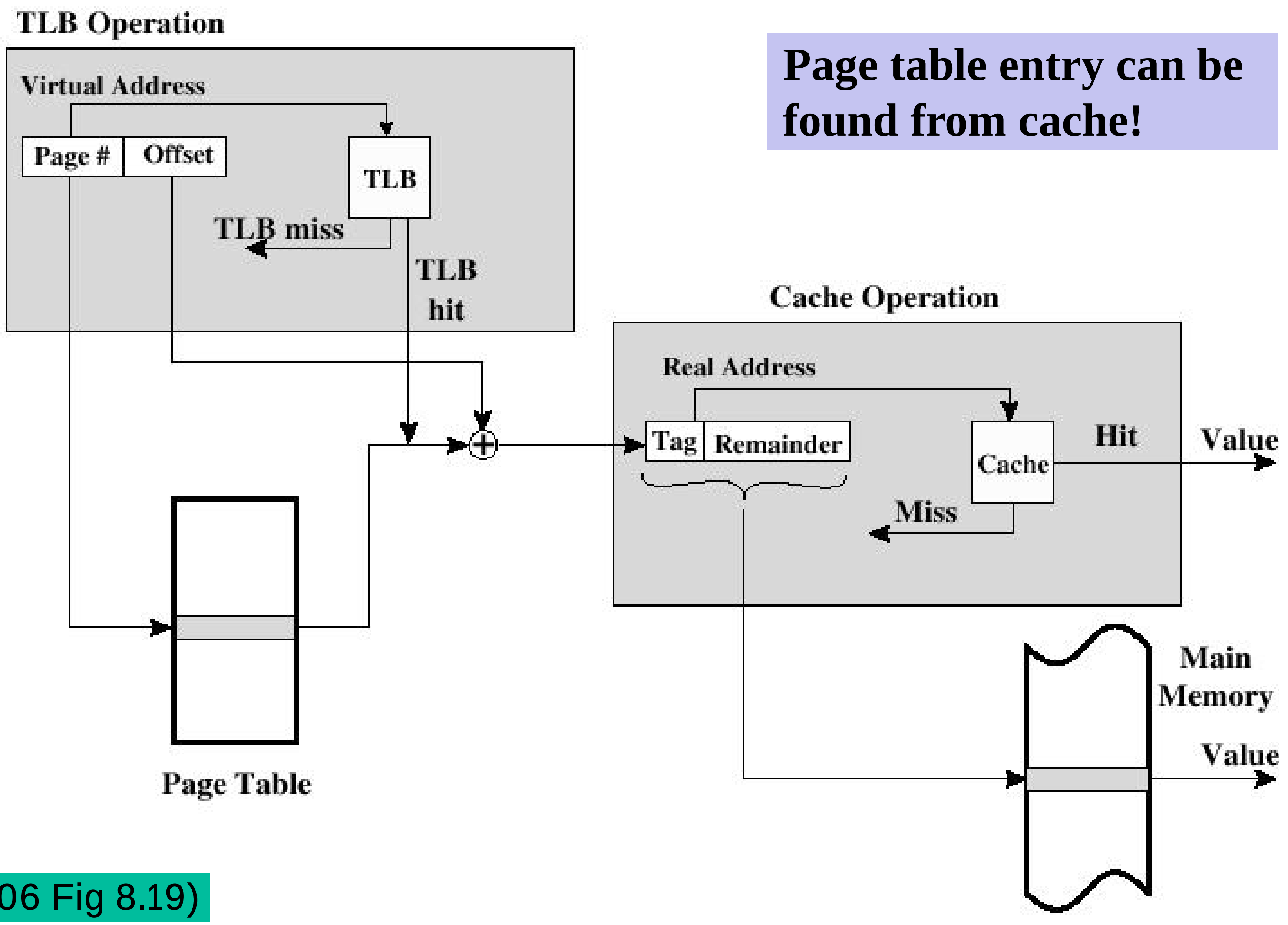
Program: pages

Memory: frames

Process A in main memory

(Sta06 Fig 8.15)

TLB and cache

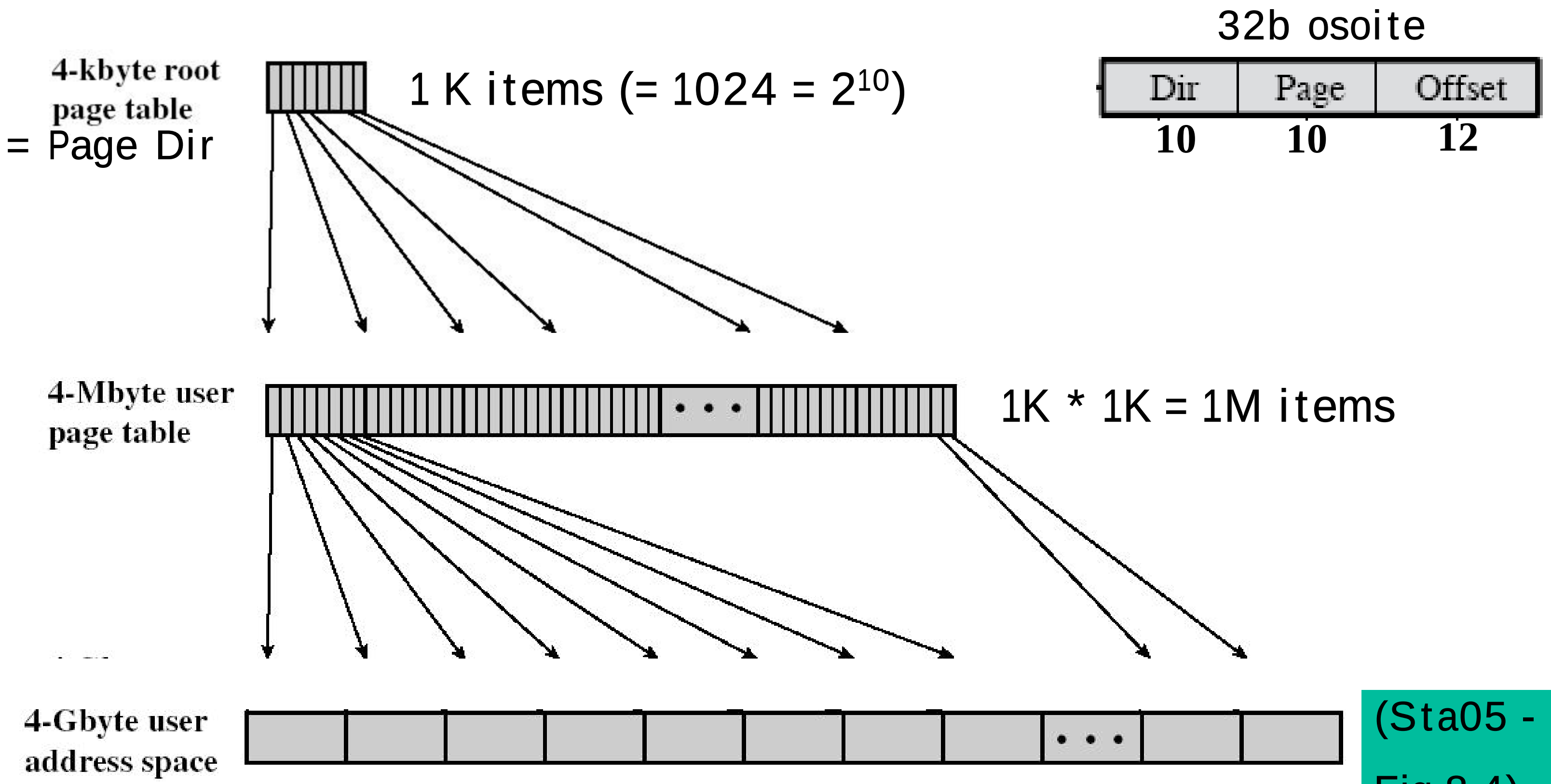


Page table entry can be found from cache!

(Sta06 Fig 8.19)

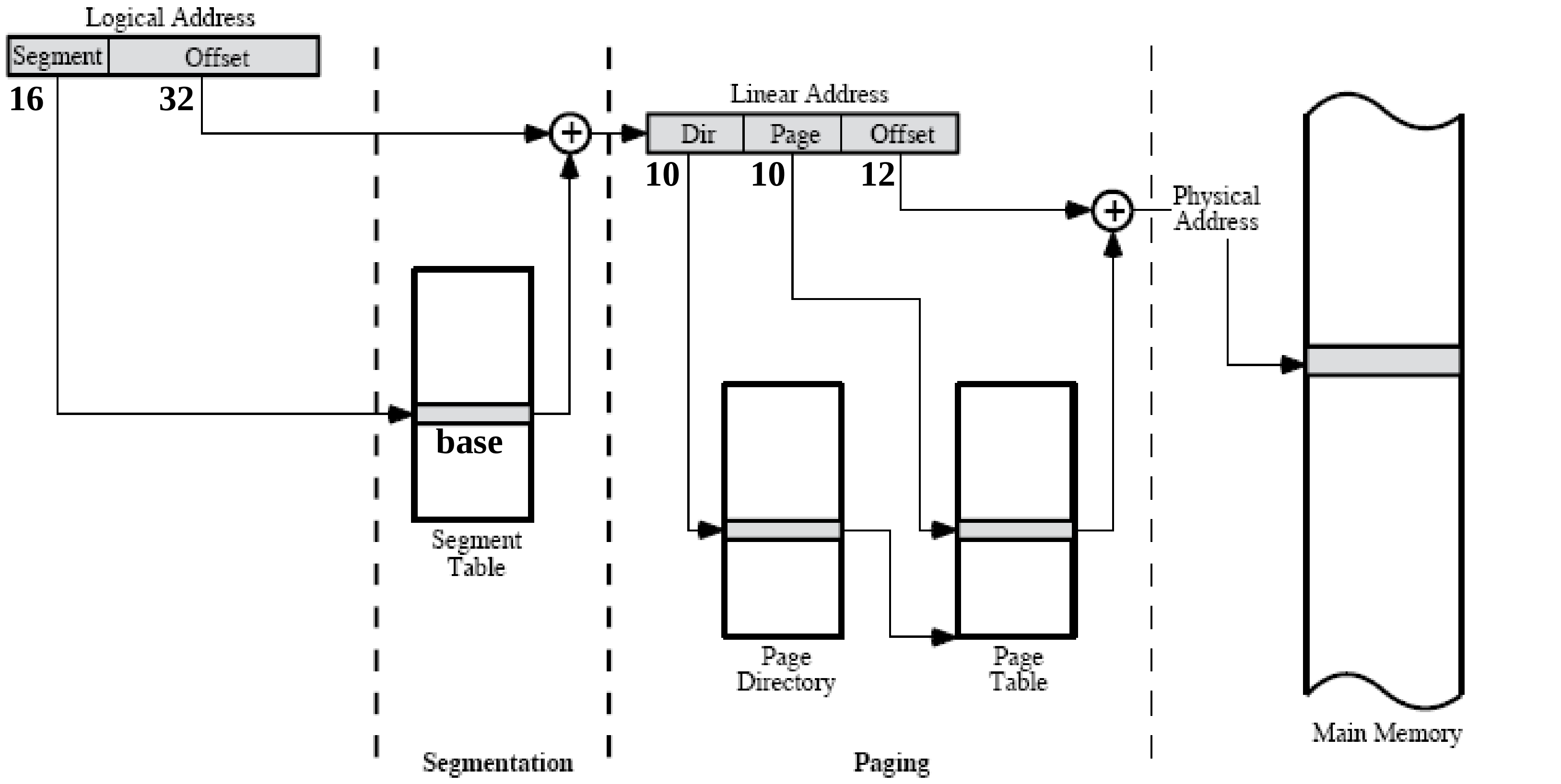
Hierarchical page table (*monitasoinen sivutaulu*)

- Several systems allow large virtual address space
  - Page table split to pages, some of it on the disk
  - Top level of page table fits to one page, always in memory



(Sta05 - OS Fig 8.4)

Pentium: Address translation

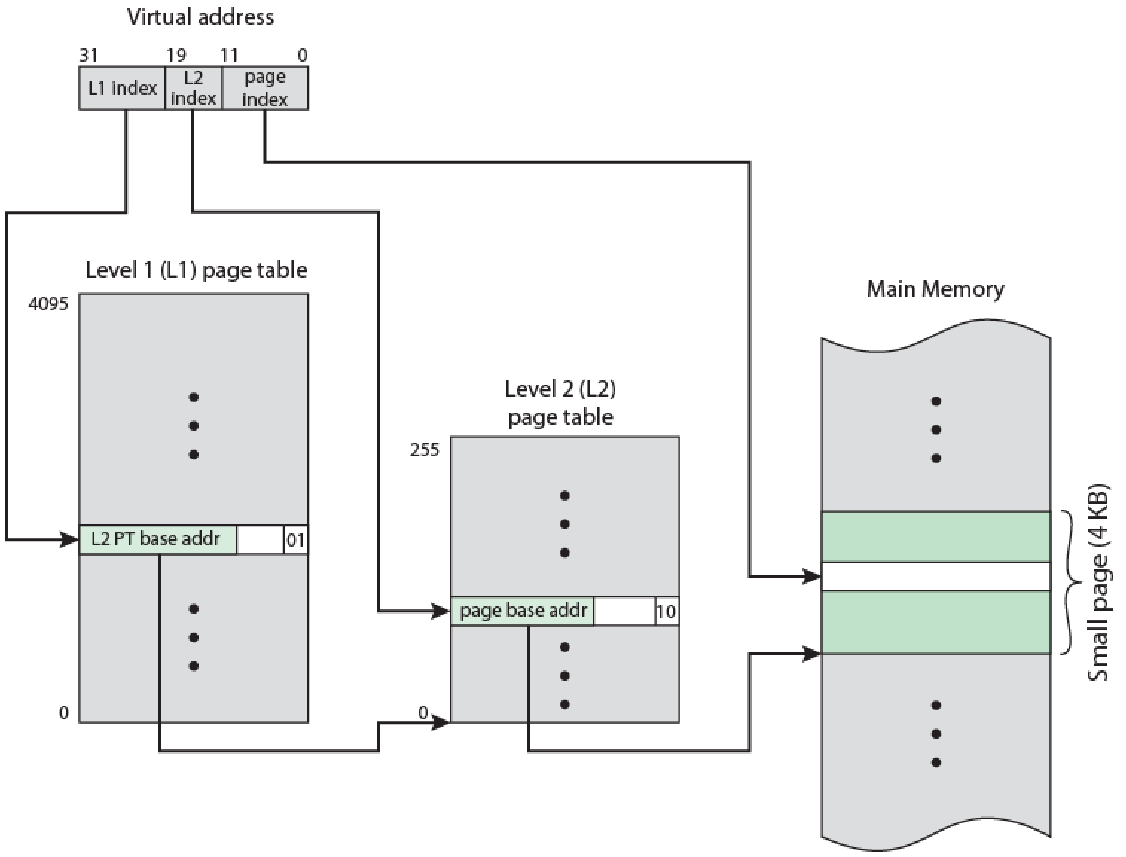


(Sta06 Fig 8.21)

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

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)

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)

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 = 1100 0111
      0011 1000
              1
      ---------
      0011 1001
= 57
```

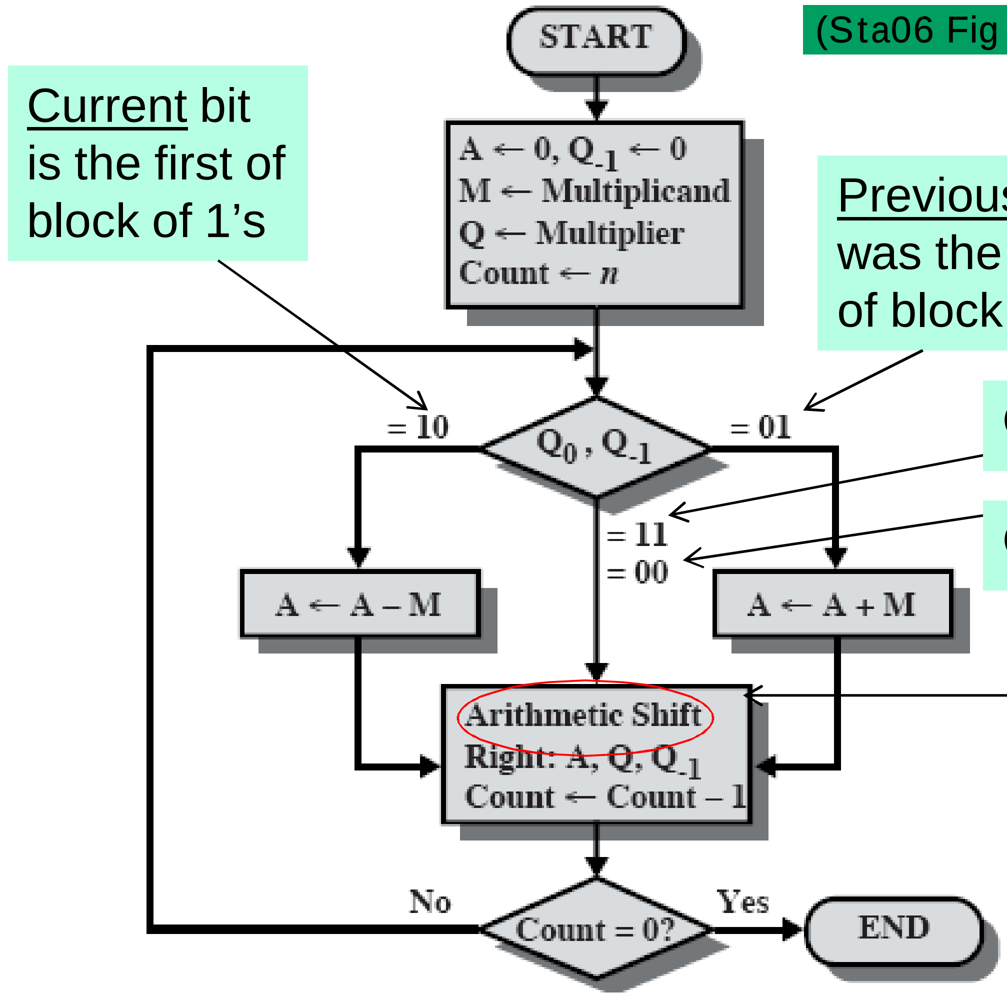
```
-128 = 1000 0000
       0111 1111
               1
       ---------
       1000 0000
```

```
57 = 0011 1001 = 0000 0000 0011 1001
-57 = 1100 0111 = 1111 1111 1100 0111
```

sign extension

Booth's algorithm

(Sta06 Fig 9.12)



Current bit is the first of block of 1's

Previous bit was the last of block of 1's

Continuing block of 1's

Continuing block of 0's

Arithmetic Shift Right:
= fill with sign

1000 1000
1100 0100

Sign bit extending

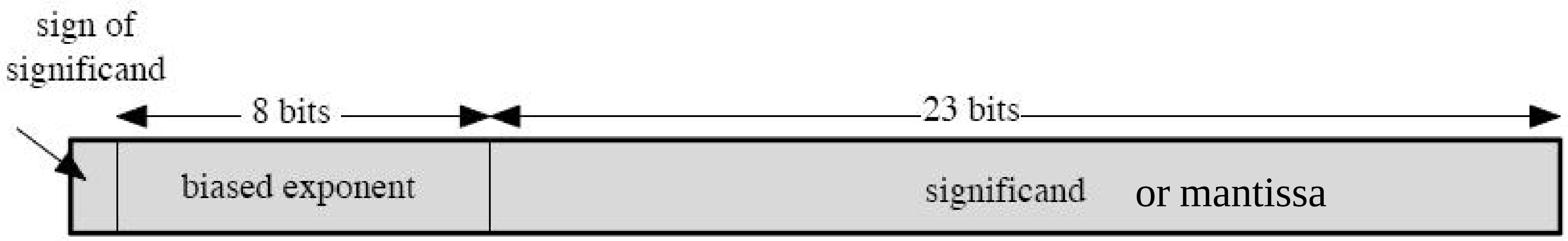
Why does it work?

M*(01111111) = 2^7 - 1

M*(00011110) = 2^5 -2^1

M*(01111010) = 2^7 -2^3+2^2-2^1

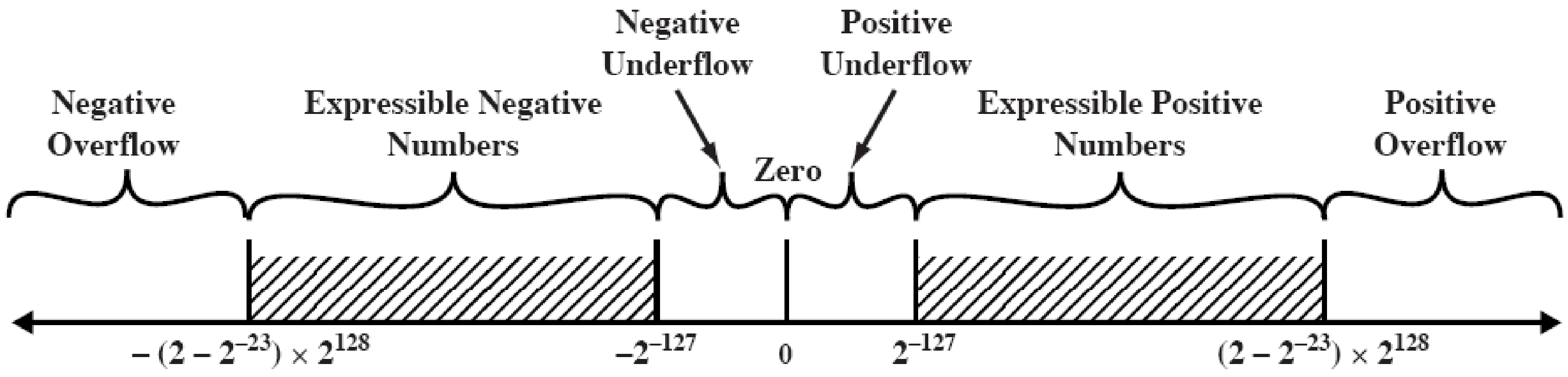
Floating Point Representation



- Significant digits *(Merkitsevät numerot)* and exponent *(suuruusluokka)*
- Normalized number (*Normeerattu muoto*)
  - Most significant digit is nonzero >0
  - Commonly just one digit before the radix point (*desim. pilkku*)

$$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$$
$$0.123 = +1.23 * 10^{-1}$$
$$123.0 = +1.23 * 10^{2}$$
$$123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$$

Accuracy (*tarkkuus*) (32b)


Negative Overflow
Expressible Negative Numbers
Negative Underflow
Zero
Positive Underflow
Expressible Positive Numbers
Positive Overflow
$-(2-2^{-23}) \times 2^{128}$
-2^{-127}
0
2^{-127}
$(2-2^{-23}) \times 2^{128}$

- Value range (*arvoalue*)
  - 8 b exponent→ 2^{-126} ... 2^{127} ~ -10^{-38} ... 10^{38}
- Not exact value
  - 24 b mantissa → 2^{24} ~ $1.7 * 10^{-7}$ ~ 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

Floating point arithmetics

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = \left(X_S \times B^{X_E - Y_E} + Y_S\right) \times B^{Y_E} \\ X - Y = \left(X_S \times B^{X_E - Y_E} - Y_S\right) \times B^{Y_E} \end{array}\right\} X_E \leq Y_E$ $X \times Y = \left(X_S \times Y_S\right) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

(Sta06 Table 9.5)

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

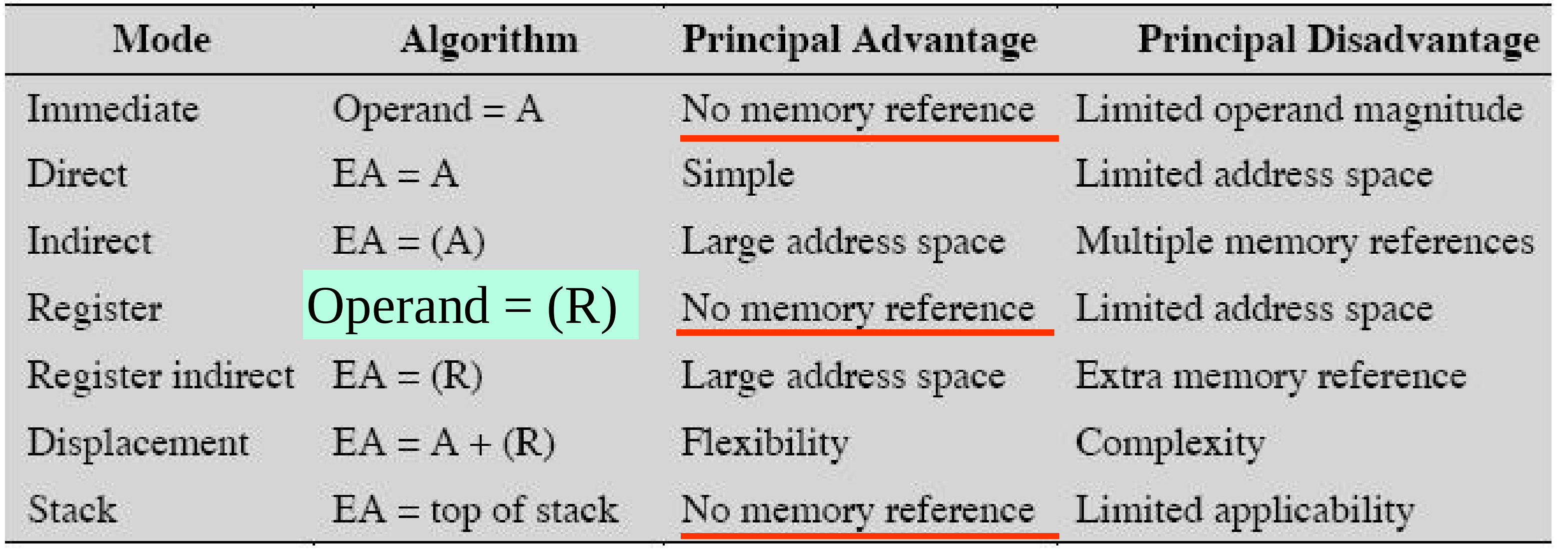
Lecture 6: Instruction sets, Chapters 10 & 11

- 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

Addressing modes

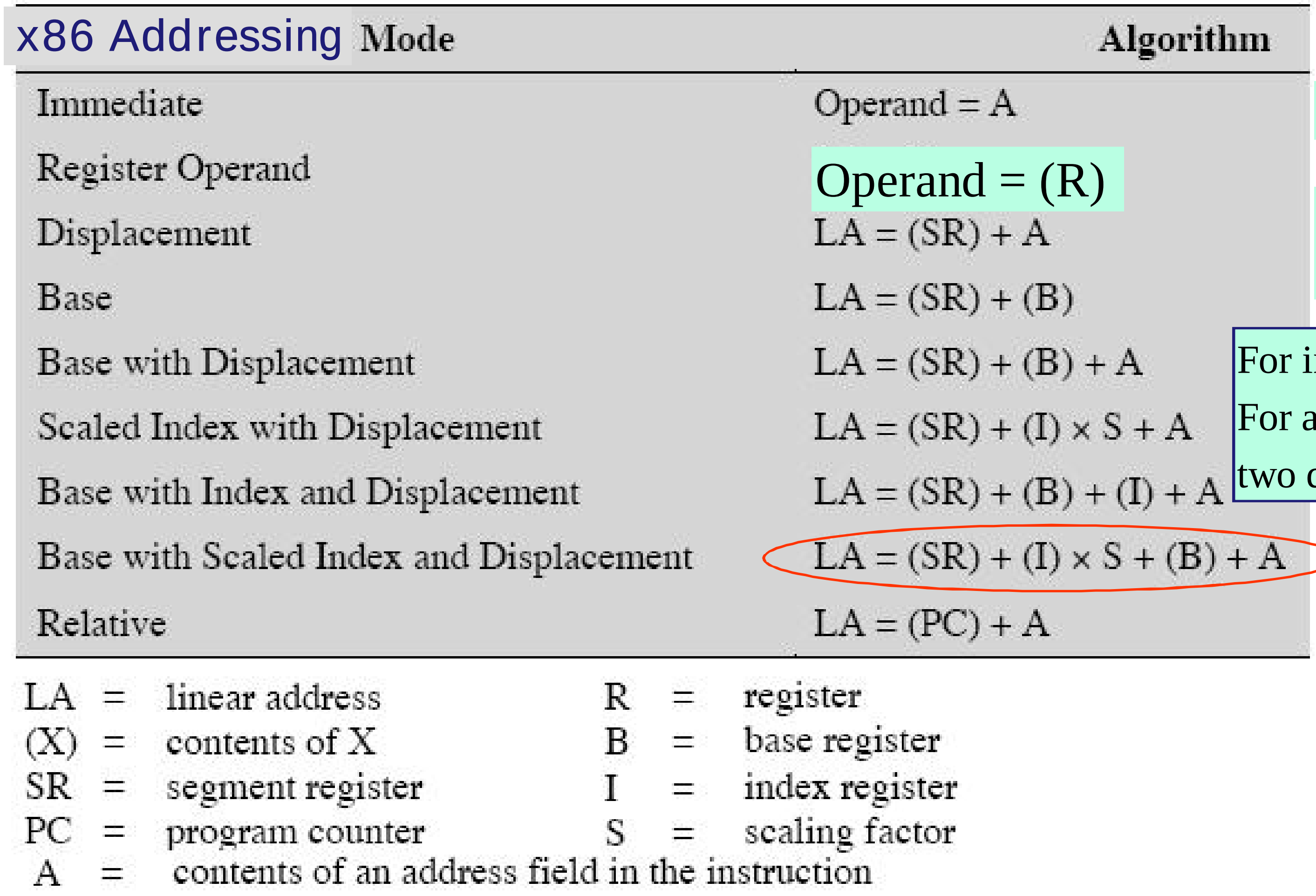
(Sta06 Table 11.1)



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

Pentium: Addressing modes (osoitustavat)



x86 Addressing Mode	Algorithm
Immediate	Operand = A
Register Operand	Operand = (R)
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
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
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			

1, 2, 4, 8B

Registers:
1, 2, 4, 8B

For indexing arrays

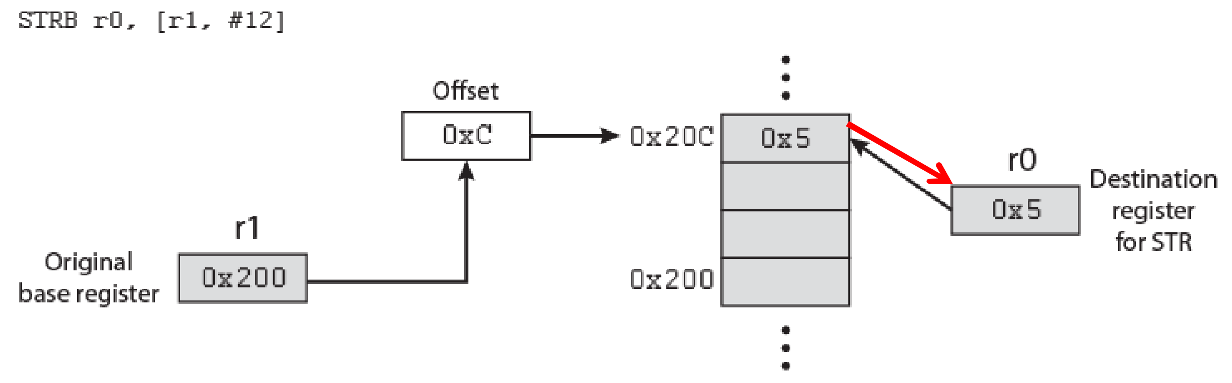
For arrays in stack or for two dimensional arrays

different element size

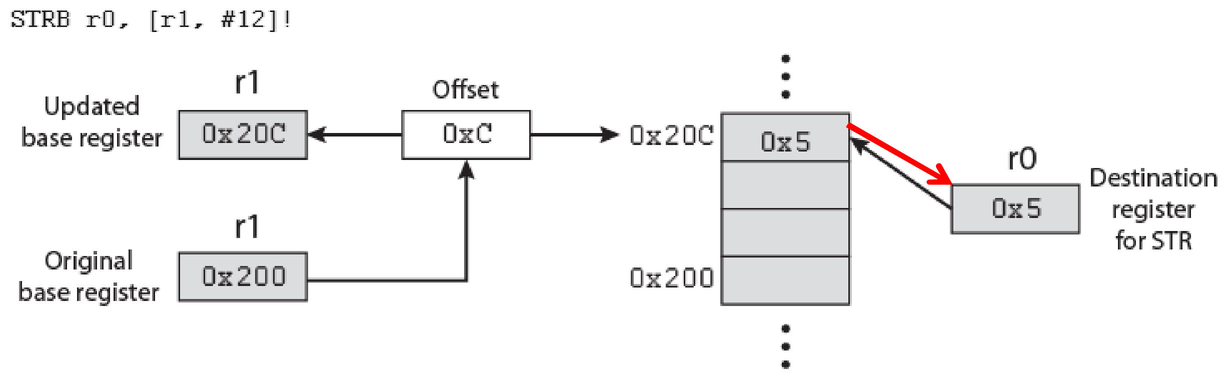
(Sta06 Table 11.2)

ARM Addressing modes

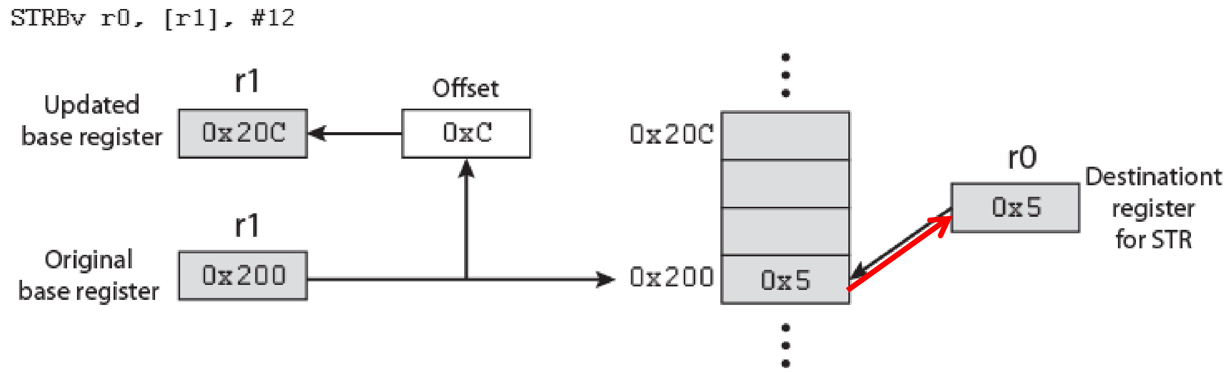
- **Load/Store**
- Indirect
  - base reg + offset
- Indexing alternatives
  - Offset
    - Address is base + offset
  - Preindex
    - Form address
    - Write address to base
  - Postindex
    - Use base as address
    - Calculate new address to base



(a) Offset



(b) Preindex

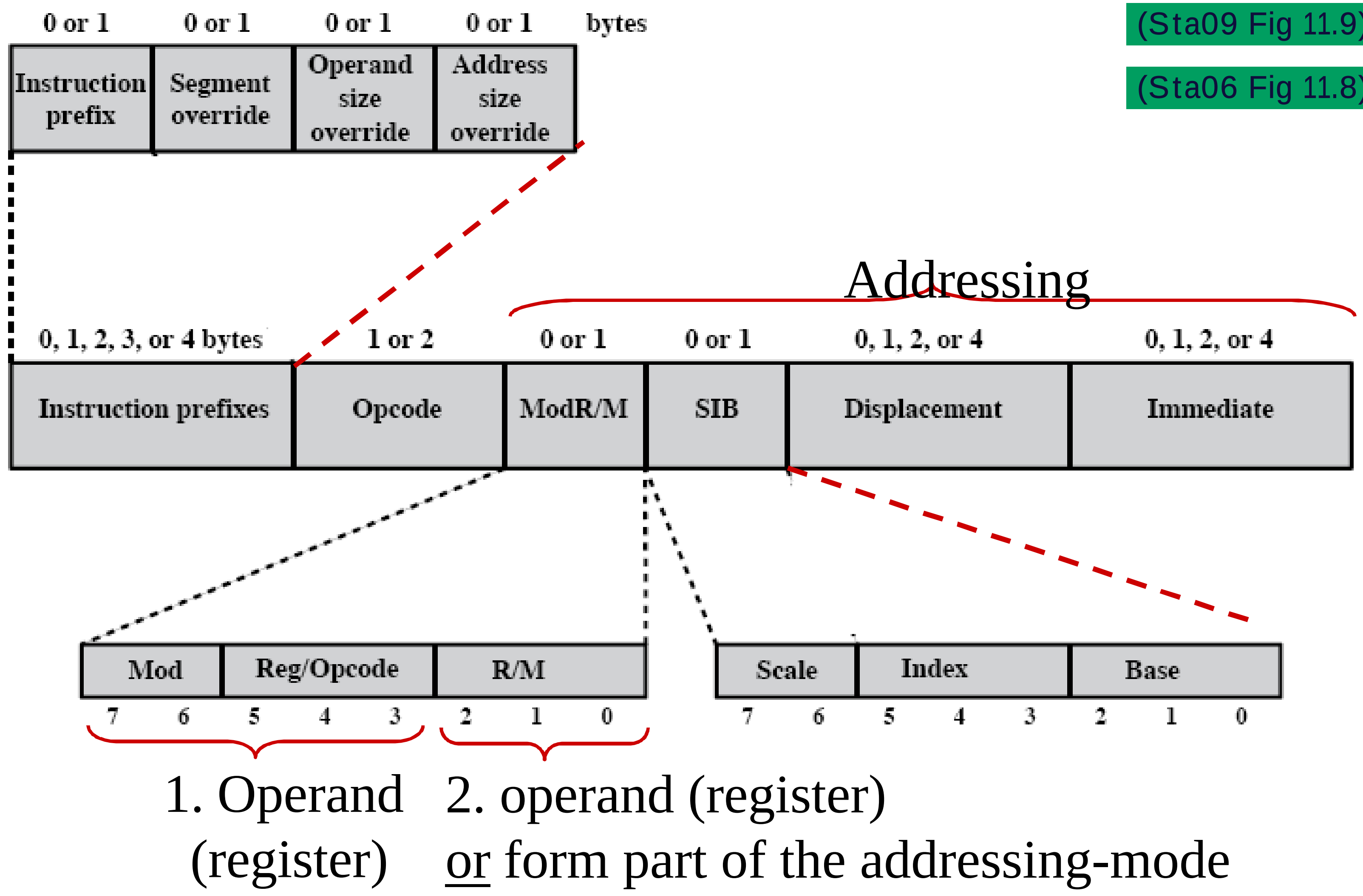


(c) Postindex

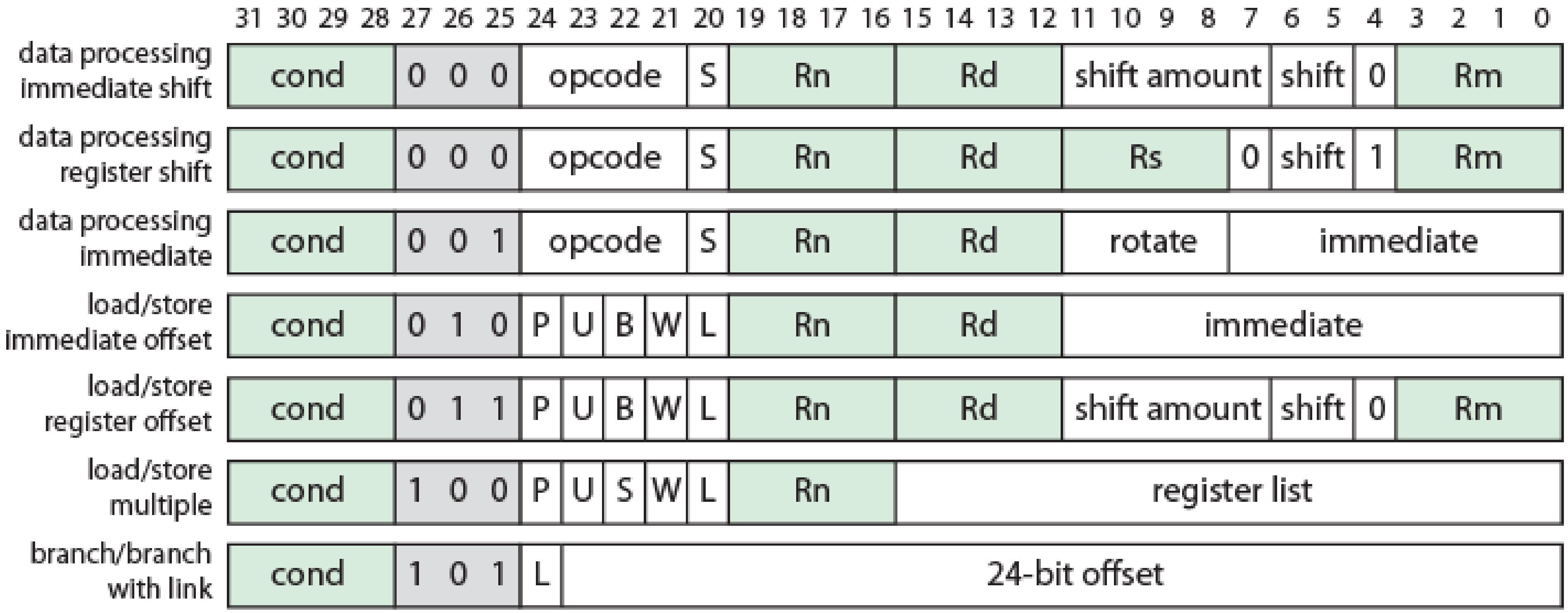
Pentium: Instruction format

(Sta09 Fig 11.9)

(Sta06 Fig 11.8)



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										

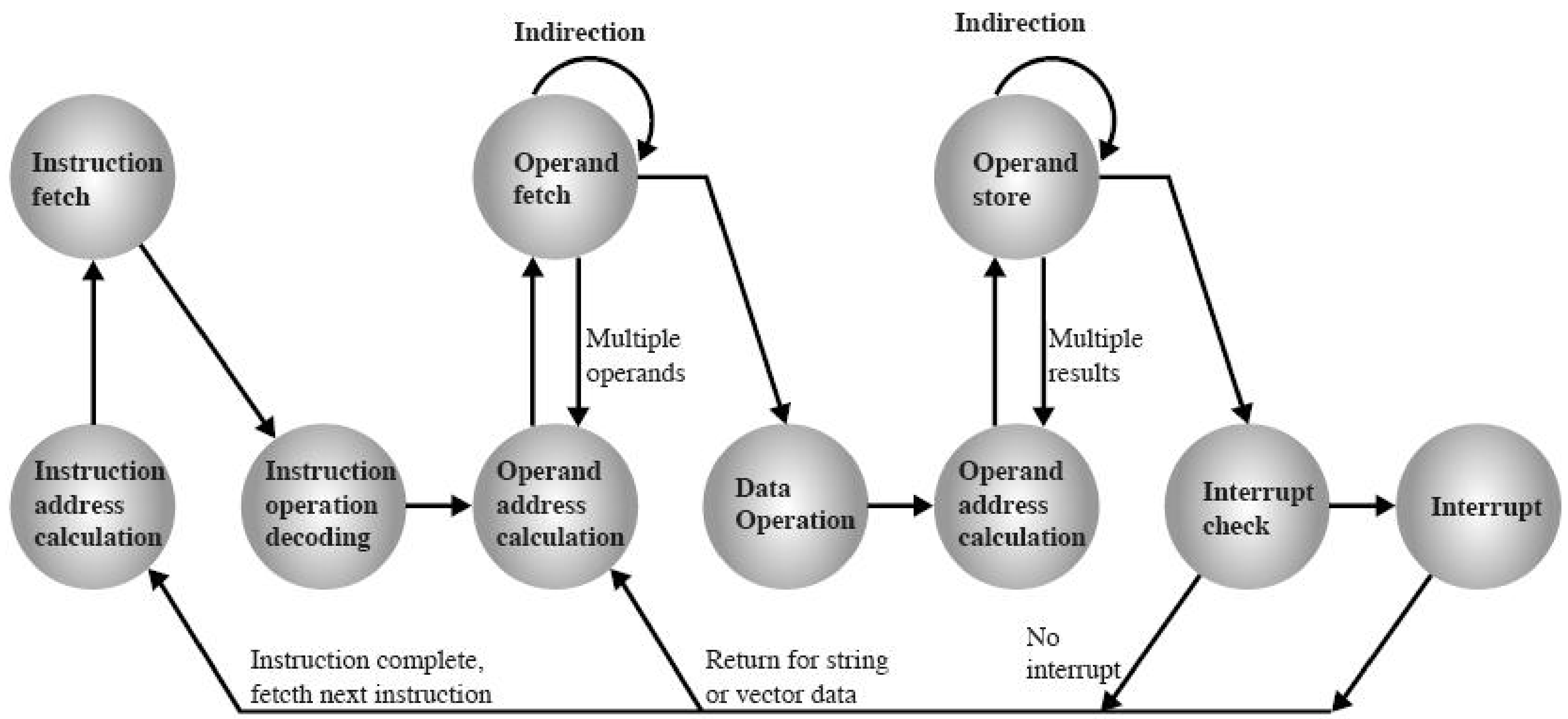
- S = For data processing instructions, updates condition codes
- S = For load/store multiple instructions, execution restricted to supervisor mode
- P, U, W = distinguish between different types of addressing mode
- B = Unsigned byte (B==1) or word (B==0) access
- L = For load/store instructions, Load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

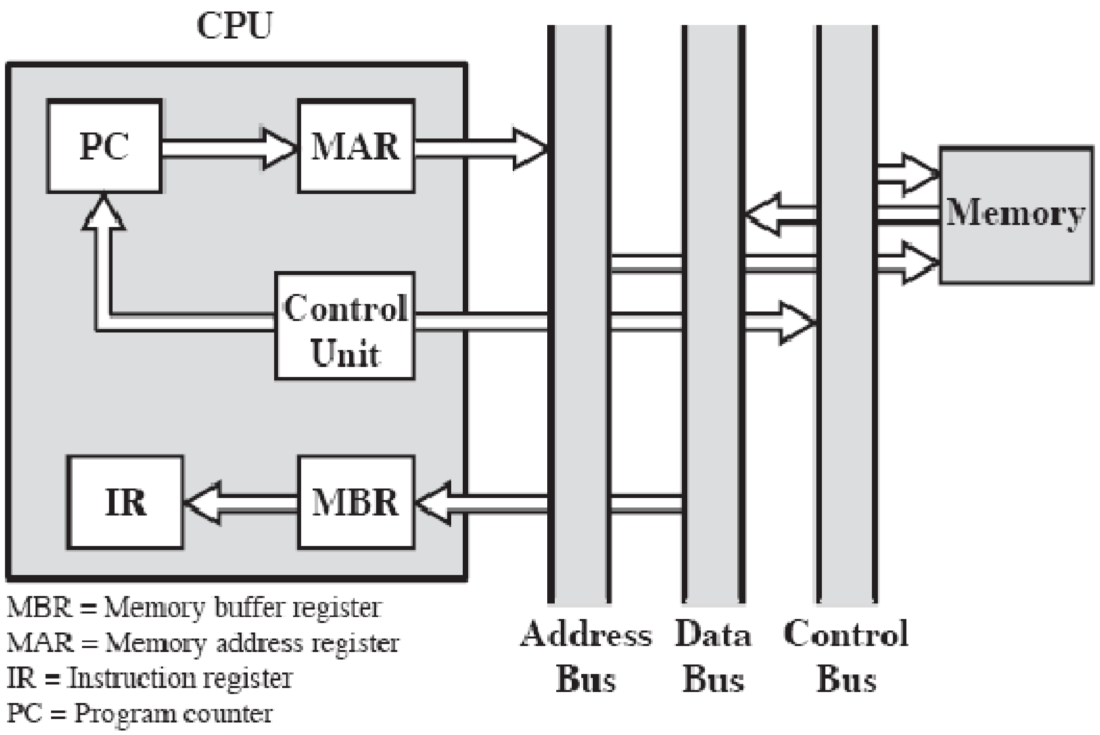
Instruction cycle (*käskysykl*i)


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction
Return for string or vector data
No interrupt

(Sta06 Fig 12.5)

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


CPU
PC
MAR
Control Unit
IR
MBR
Memory
Address Bus
Data Bus
Control Bus
MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Cache (*välimuisti*)!
Prefetch (*ennaltanouto*)!

(Sta06 Fig 12.6)

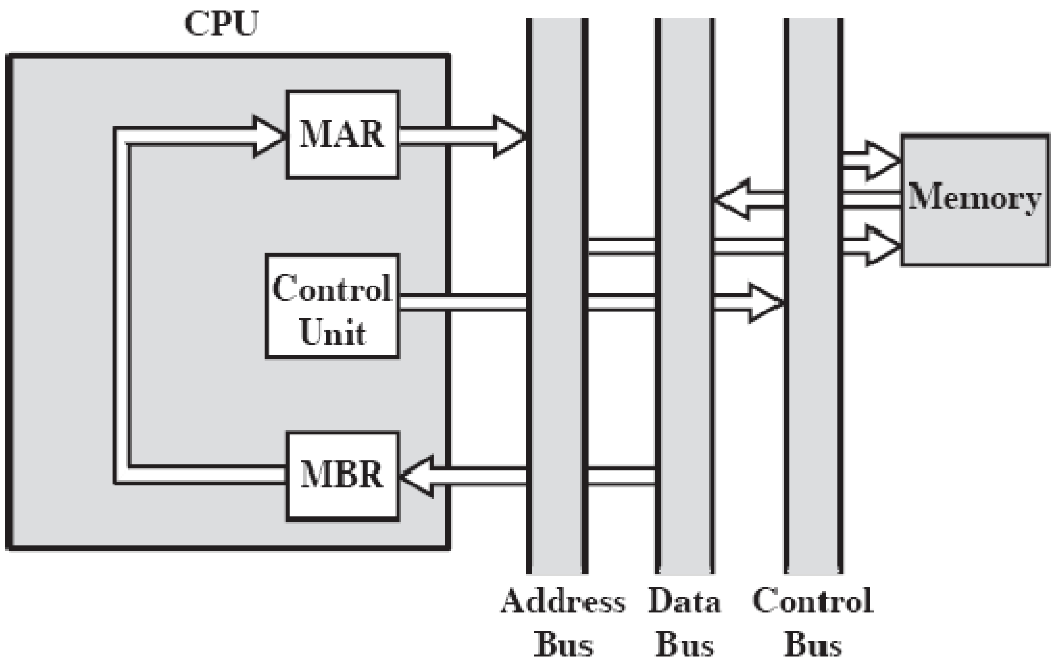
Operand fetch, Indirect addressing (Operandin nouto, epäsuora osoitus)

- 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

Cache!

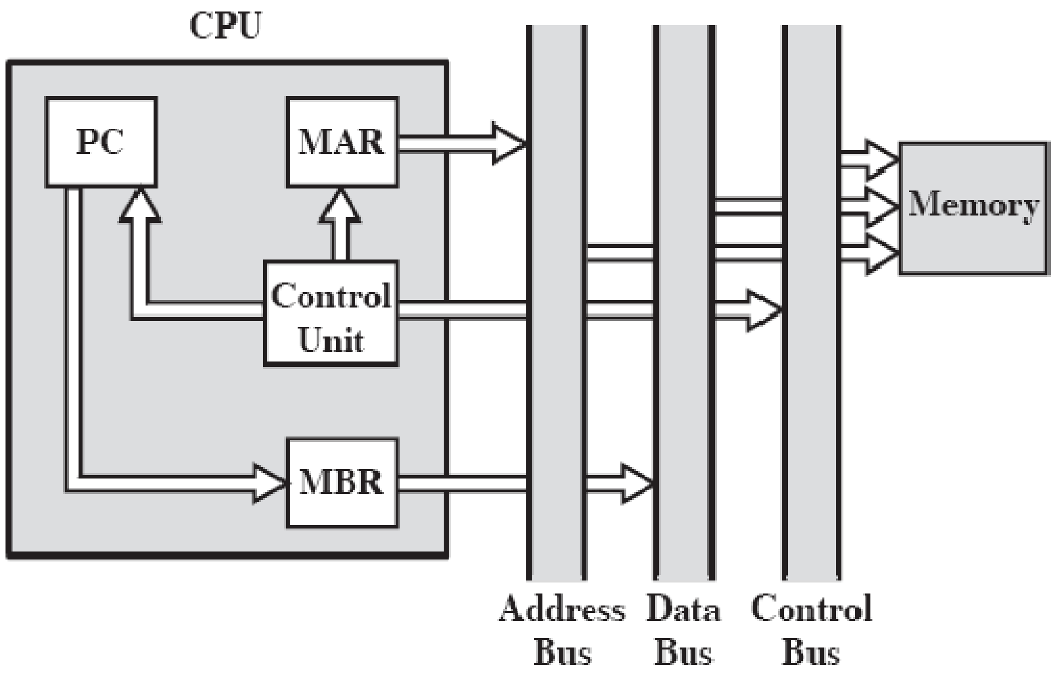


(Sta06 Fig 12.7)

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]
- Control Bus ← Release

No address translation!



SP = Stack Pointer

(Sta06 Fig 12.8)

Problems, design issues

- Structural dependency (*rakenteellinen riippuvuus*)
  - Several stages may need the same HW
  - Memory: FI, FO, WO
  - ALU: CO, EI

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

- Control dependency (*kontrolliriippuvuus*)
  - Jump destination of conditional branch known only after EI-stage
  - → Prefetched wrong instructions

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

- Data dependency (*datariippuvuus*)
  - Instruction needs the result of the previous non-finished instruction

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

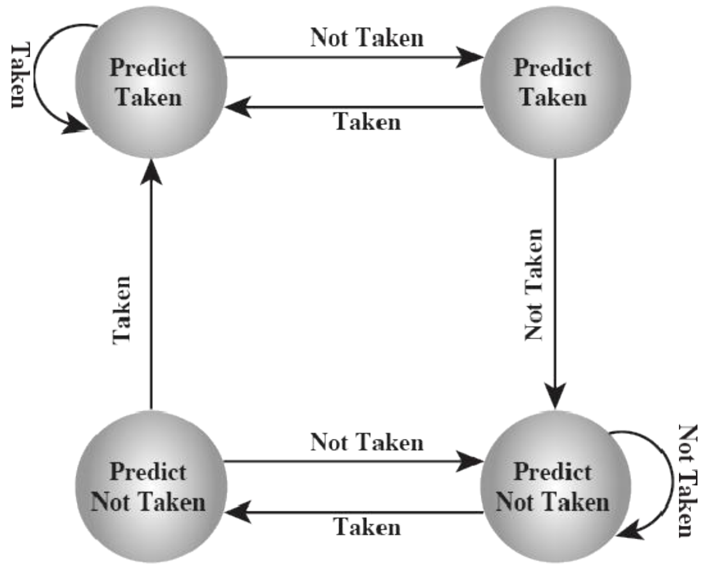
Data dependency

- Read after Write (RAW)   (a.k.a true or flow dependency)
  - Occurs if succeeding read takes place before the preceeding write operation is complete
- Write after Read (WAR)   (a.k.a antidependency)
  - Occurs if the succeeding write operation completes before the preceeding 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

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



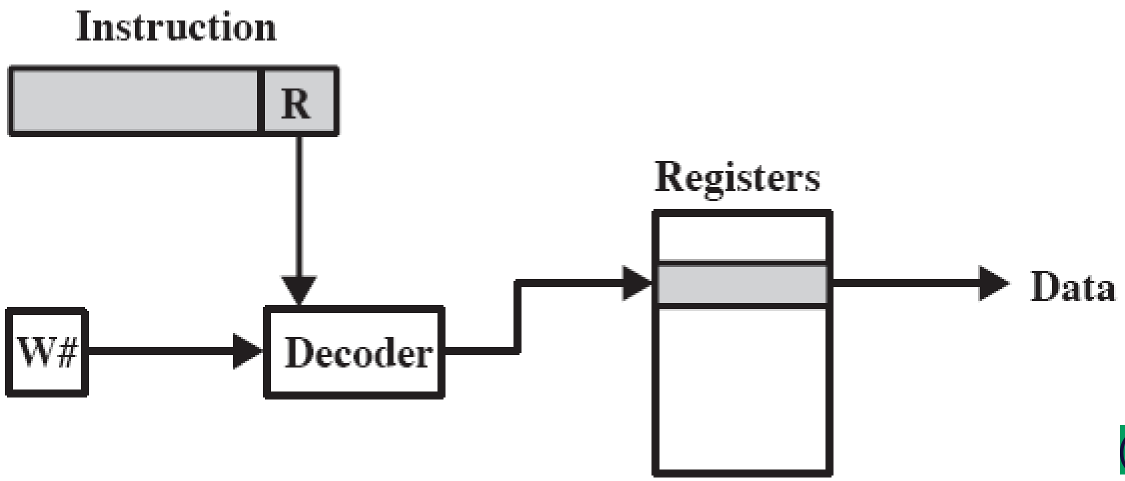
Lecture 8: RISC, Chapter 13

- 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

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



Current Window Pointer

(Sta06 Fig 13.3)

RISC-pipeline, Delayed Branch

	1	2	3	4	5	6	7	8
100 LOAD X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 105				I	E			
103 ADD rA, rB					I			
105 STORE rA, Z						I	E	D

Traditional

100 LOAD X, rA	I	E	D				
101 ADD 1, rA		I	E				
102 JUMP 106			I	E			
103 NOOP				I	E		
106 STORE rA, Z					I	E	D

RISC with inserted NOOP

Two port MEM

100 LOAD X, Ar	I	E	D			
101 JUMP 105		I	E			
102 ADD 1, rA			I	E		
105 STORE rA, Z				I	E	D

Cond. branch (*ehdollinen hyppy*):

JZERO 105, rA ??

RISC with reversed instructions

(Sta06 Fig 13.7)

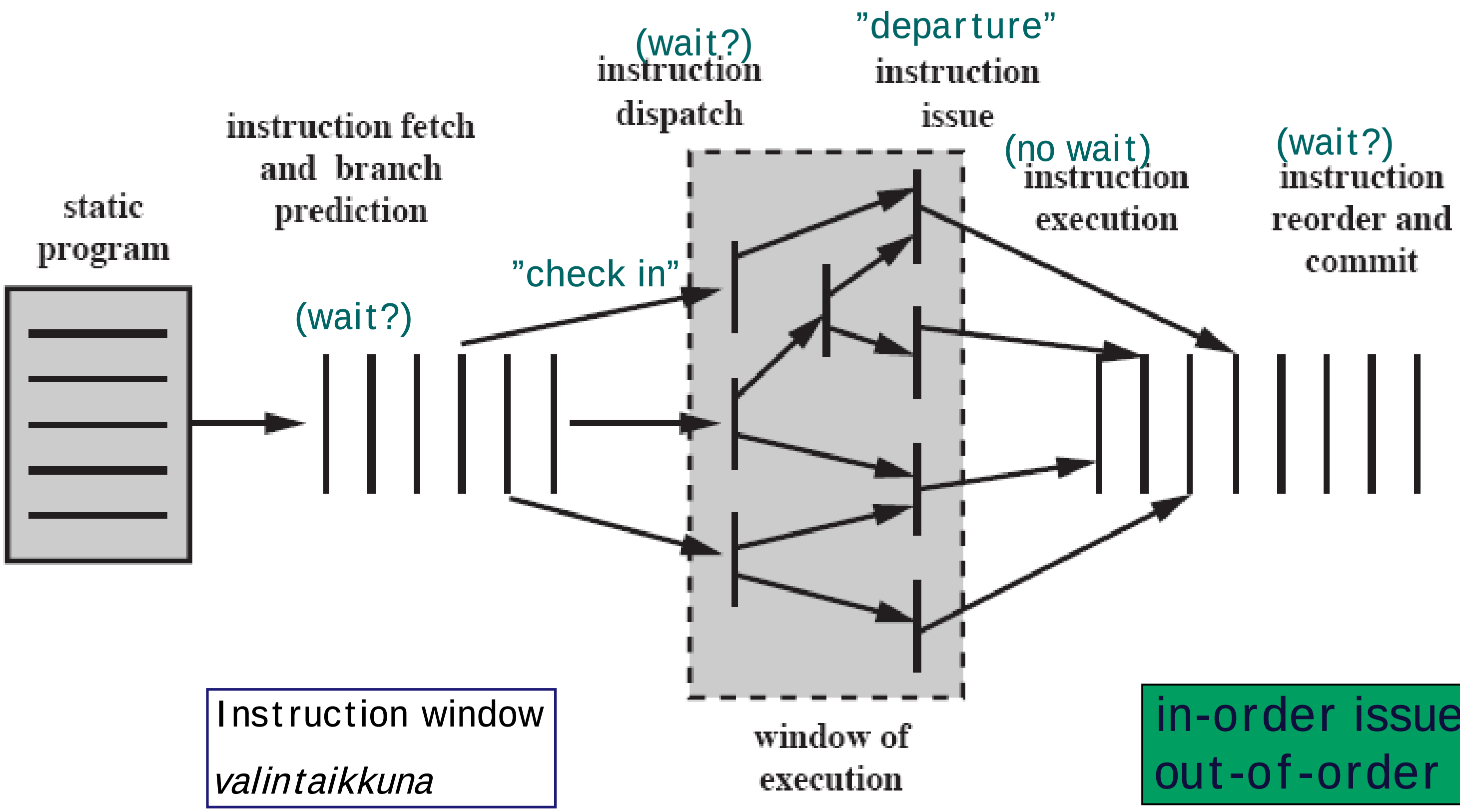
Lecture 9: Superscalar, Chapter 14

- 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)

Superscalar execution



Instruction window
valintaikkuna

in-order issue vs. out-of-order issue

in-order complete vs. out-of-order complete

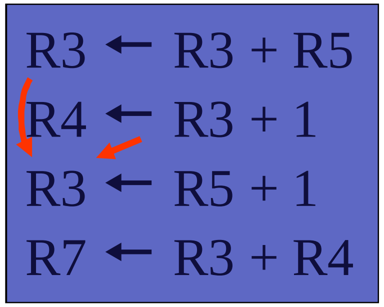
issue ~ *laukaisu, liikkeellelaskeminen*

dispatch ~ *vuorottaminen, lähettää suorittamaan*

(Sta09 Fig 14.6)

Register renaming (*rekistereiden uudelleennimeäminen*)

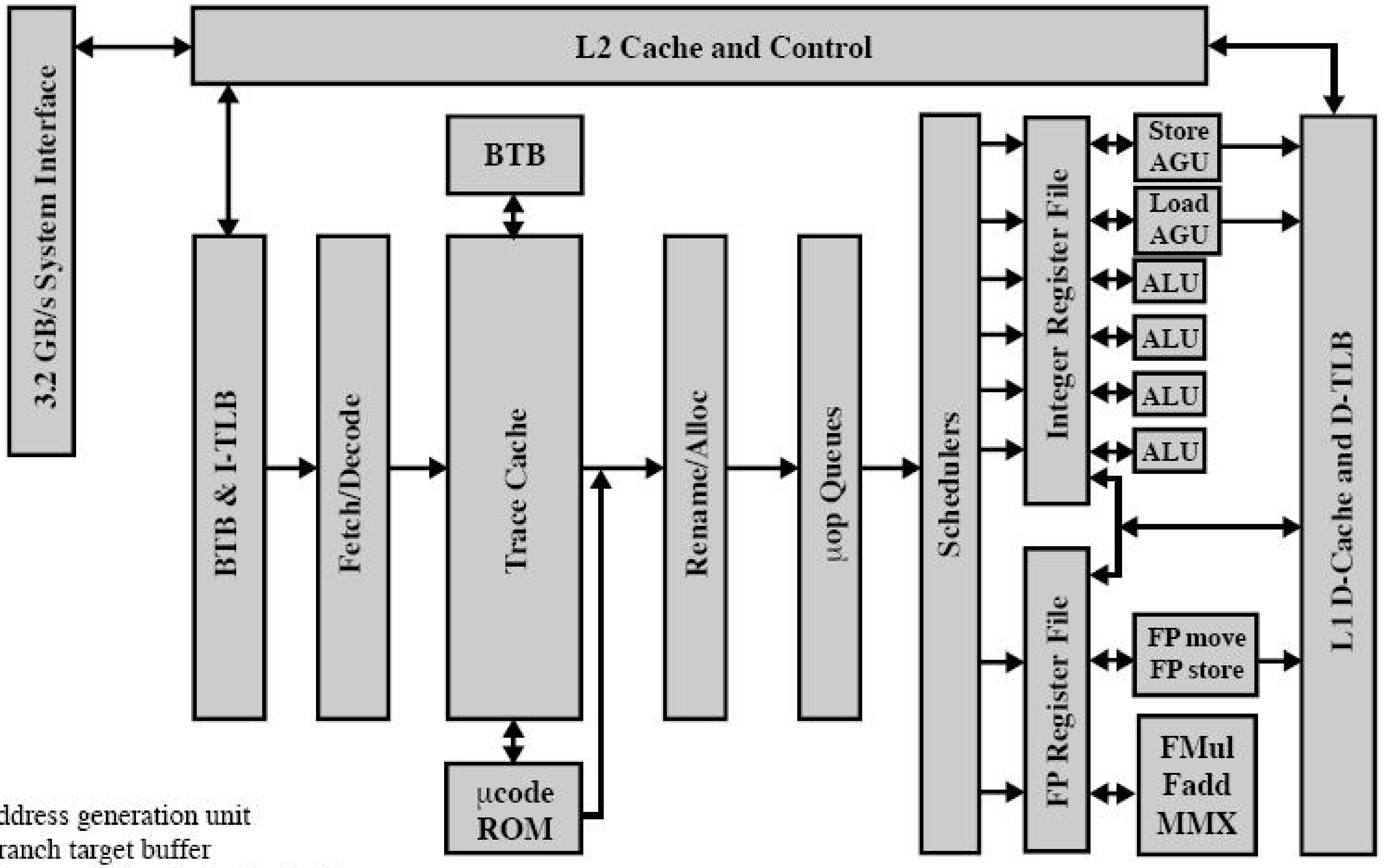
- One cause for some of the dependencies is the usage of names
  - The same name could be used for several independent elements
  - Thus, instructions have unneeded write and antidependencies
  - Causing unnecessary waits



R3 $\leftarrow$ R3 + R5
R4 $\leftarrow$ R3 + 1
R3 $\leftarrow$ R5 + 1
R7 $\leftarrow$ R3 + R4

- 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

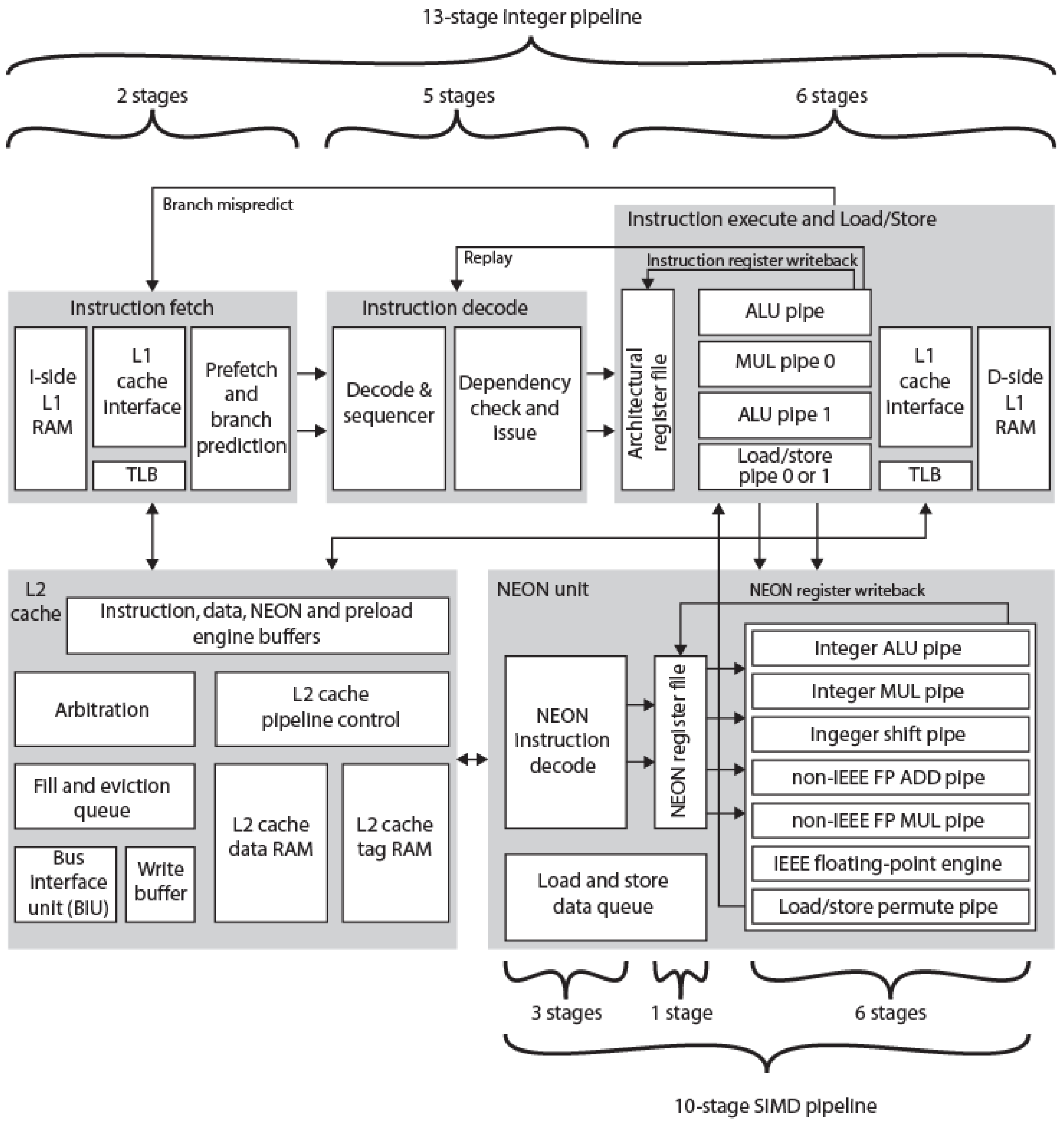
Pentium 4



AGU	= address generation unit
BTB	= branch target buffer
D-TLB	= data translation lookaside buffer
I-TLB	= instruction translation lookaside buffer

(Sta09 Fig 14.7)

ARM Cortex-A8 Block Diagram



Lecture 10: Control-Unit, Chapters 15 & 16

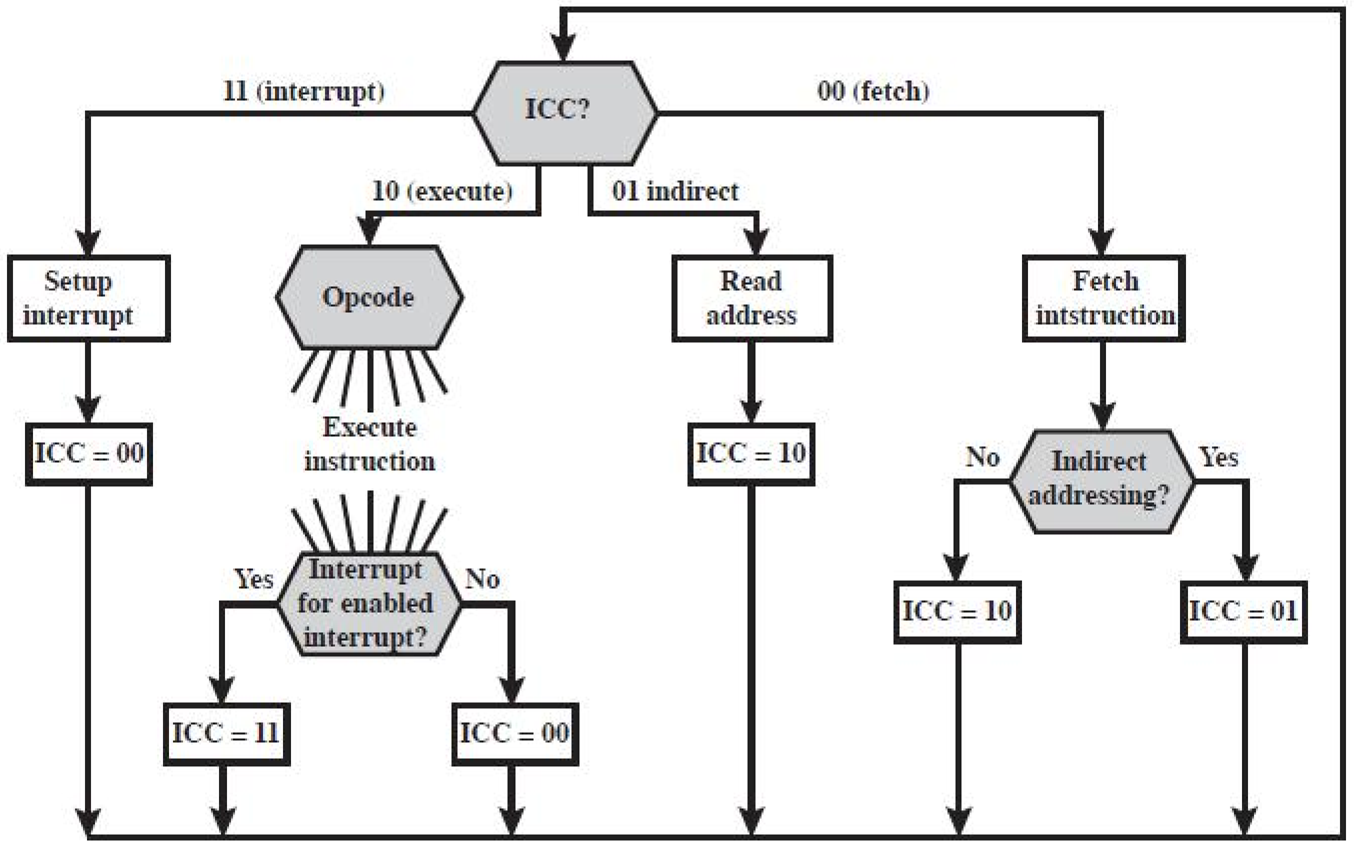
- Chapter 15 Everything but the tiny details of processors
- Chapter 16: 16.1 – 16.3

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

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

Instruction cycle flow chart (as state-machine?)

- ICC: Instruction Cycle Code register 's state



(Sta09 Fig 15.3)

(Sta06 Fig 16.3)

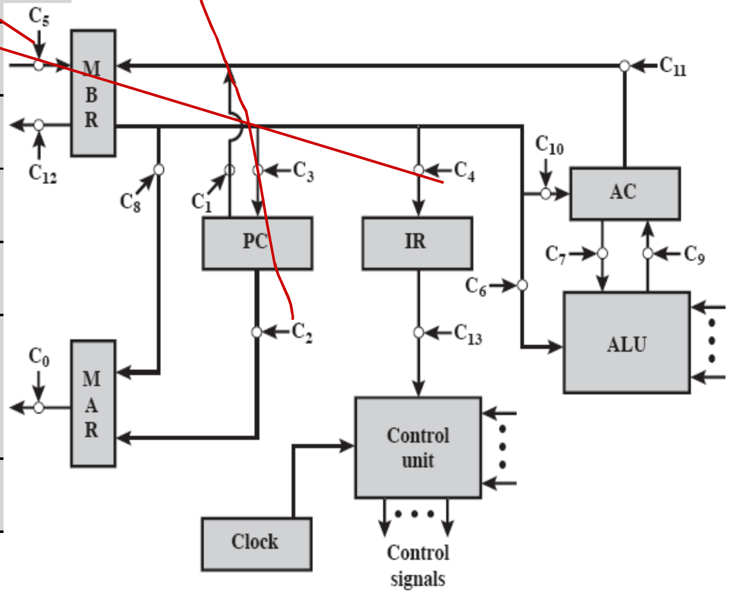
Control signals and micro-operations

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

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

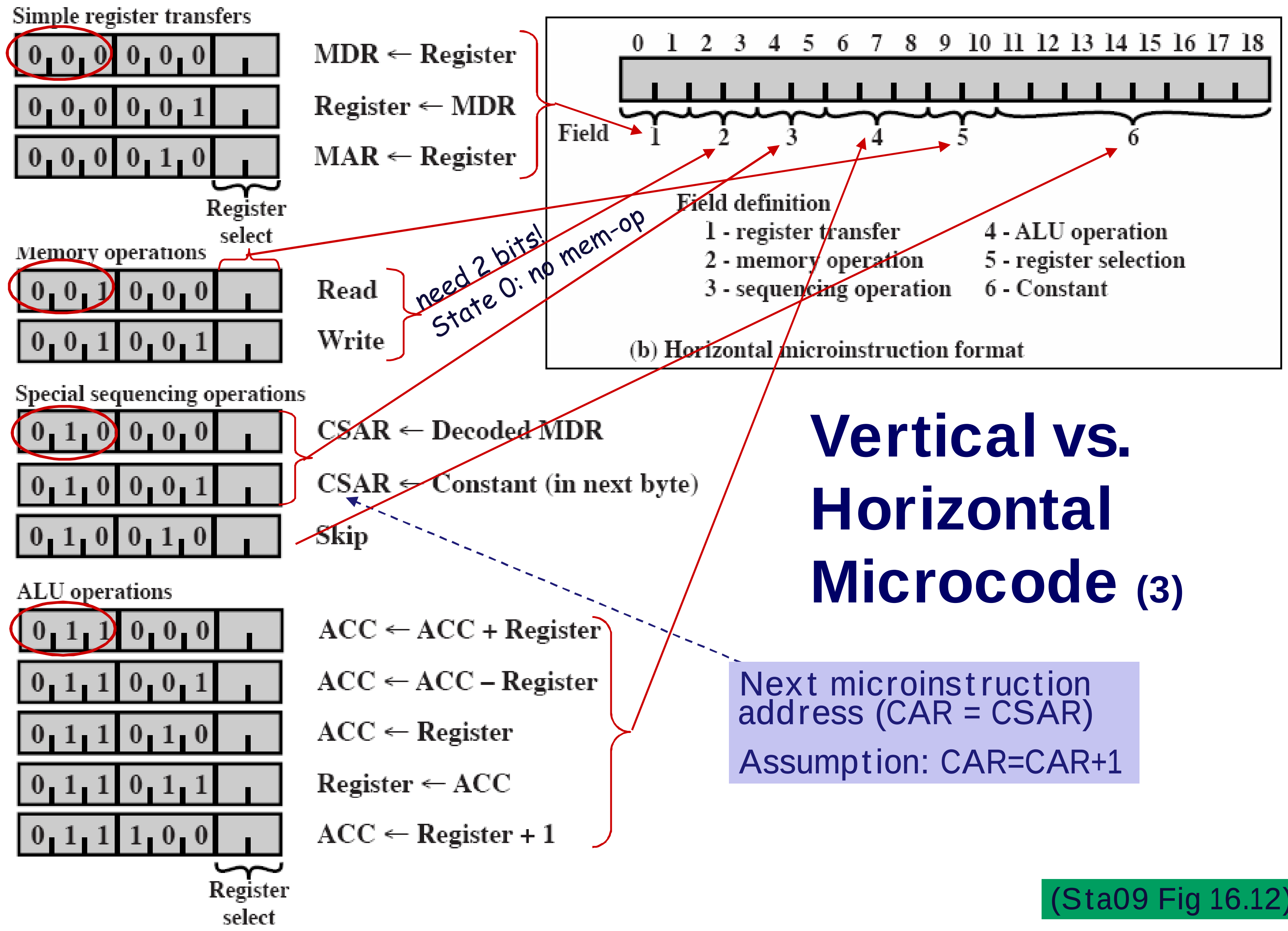
Sta09 Fig 15.5

Sta06 Fig 16.5



(Sta09 Table 15.1) (Sta06 Table 16.1)

Vertical vs. Horizontal Microcode (3)


Simple register transfers
0 0 0 0 0 0 — MDR ← Register
0 0 0 0 0 1 — Register ← MDR
0 0 0 0 1 0 — MAR ← Register
Register select
Memory operations
0 0 1 0 0 0 — Read
0 0 1 0 0 1 — Write
Special sequencing operations
0 1 0 0 0 0 — CSAR ← Decoded MDR
0 1 0 0 0 1 — CSAR ← Constant (in next byte)
0 1 0 0 1 0 — Skip
ALU operations
0 1 1 0 0 0 — ACC ← ACC + Register
0 1 1 0 0 1 — ACC ← ACC – Register
0 1 1 0 1 0 — ACC ← Register
0 1 1 0 1 1 — Register ← ACC
0 1 1 1 0 0 — ACC ← Register + 1
Register select
need 2 bits!
State 0: no mem-op
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Field
1 2 3 4 5 6
Field definition
1 - register transfer
2 - memory operation
3 - sequencing operation
4 - ALU operation
5 - register selection
6 - Constant

(b) Horizontal microinstruction format

(a) Vertical microinstruction format (by resource)

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: sequencially to next, bu ’jump target’ in instruction
- Variable format; sepate 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

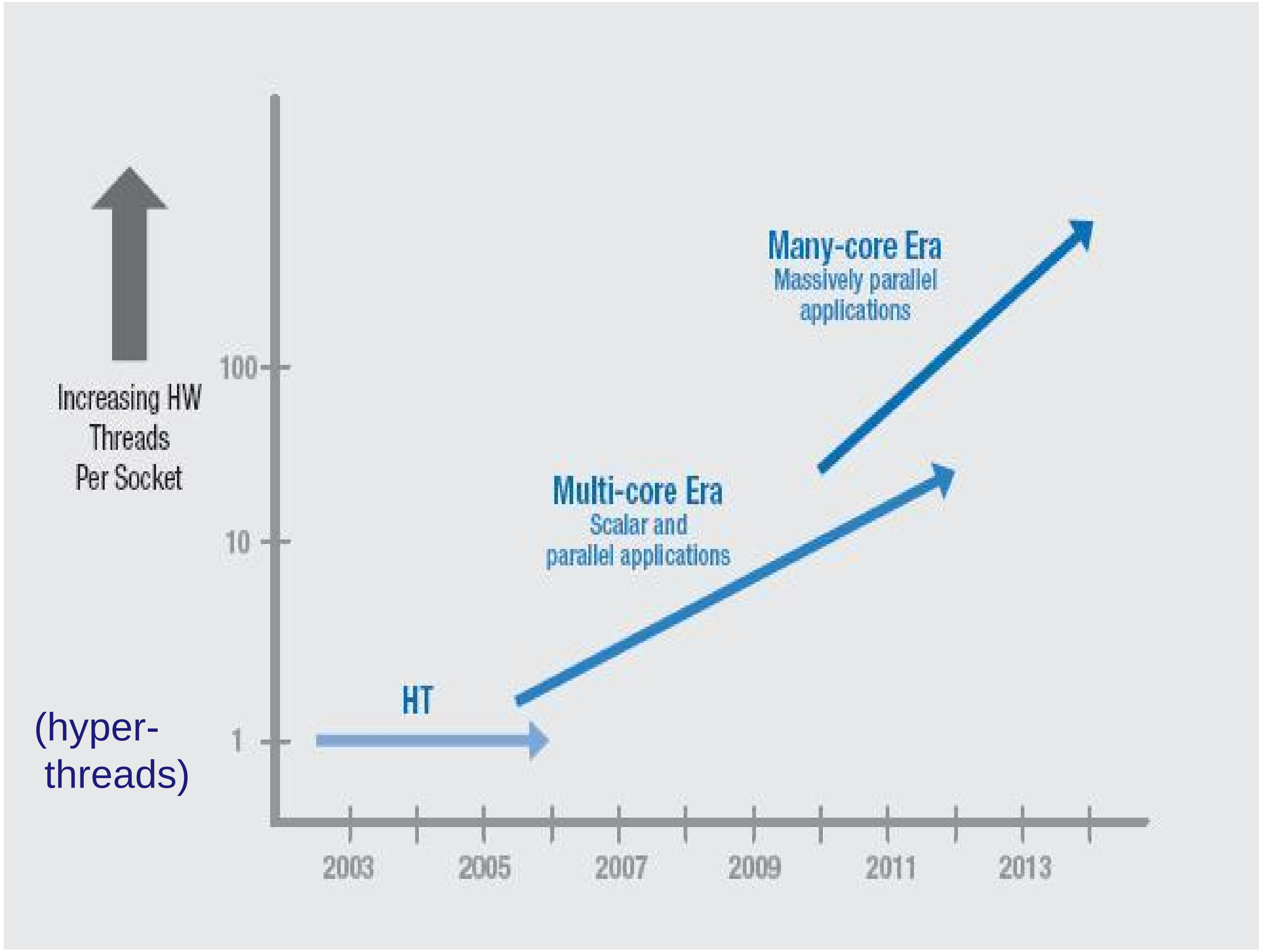
- Most important: cache coherence and MESI

- 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



(hyper-threads)

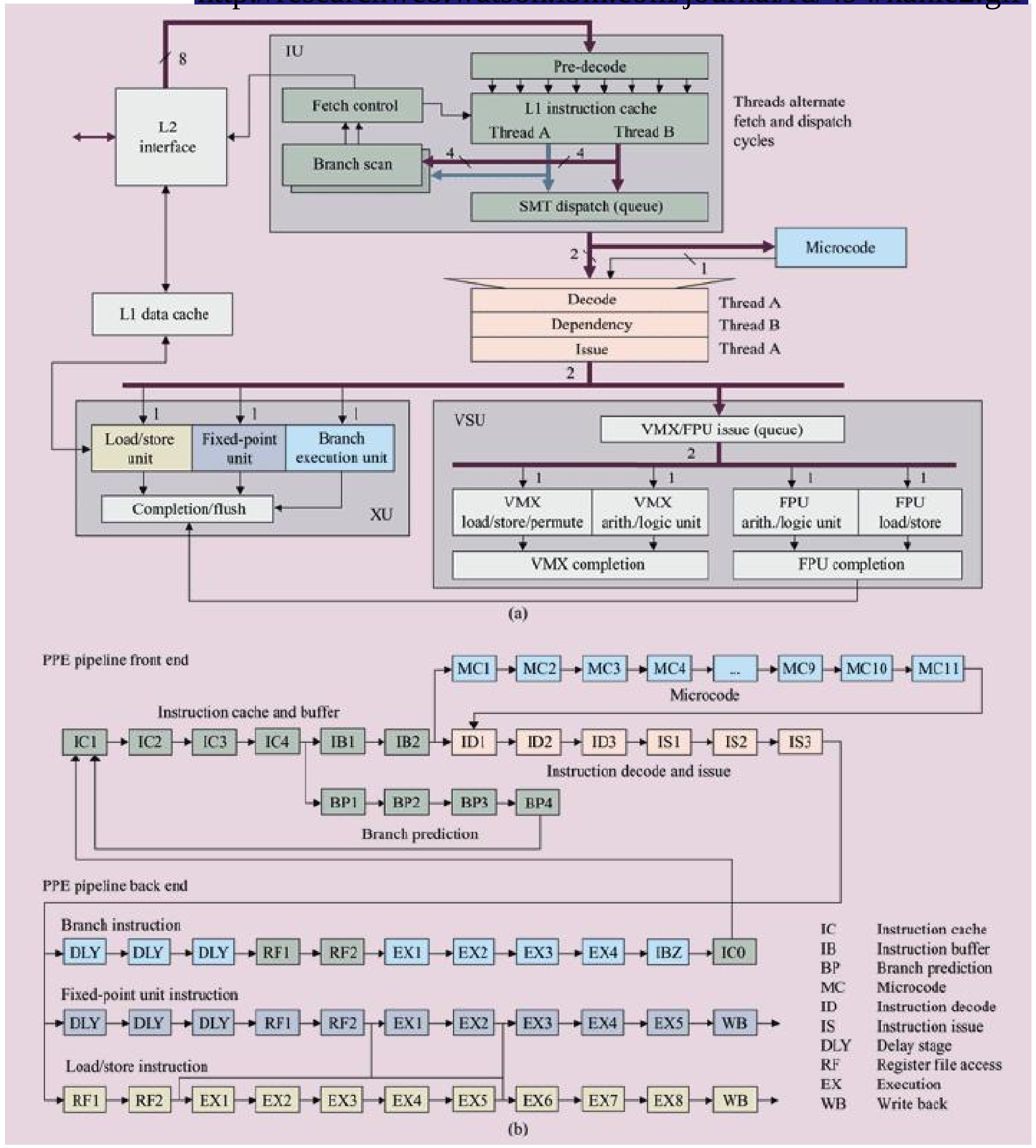
Borkar, Dubey, Kahn, et al. “Platform 2015.” Intel White Paper, 2005. (click)

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

-- The End --

http://researchweb.watson.ibm.com/journal/rd/494/kahle2.gif

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



IU
Pre-decode
Fetch control
L1 instruction cache
Thread A
Thread B
Branch scan
SMT dispatch (queue)
Threads alternate fetch and dispatch cycles
L2 interface
8
4
4
Microcode
2
1
L1 data cache
Decode
Dependency
Issue
Thread A
Thread B
Thread A
2
1
1
1
Load/store unit
Fixed-point unit
Branch execution unit
Completion/flush
XU
VSU
VMX/FPU issue (queue)
2
1
1
1
1
VMX load/store/permute
VMX arith./logic unit
FPU arith./logic unit
FPU load/store
VMX completion
FPU completion
(a)
PPE pipeline front end
MC1
MC2
MC3
MC4
...
MC9
MC10
MC11
Microcode
Instruction cache and buffer
IC1
IC2
IC3
IC4
IB1
IB2
ID1
ID2
ID3
IS1
IS2
IS3
Instruction decode and issue
BP1
BP2
BP3
BP4
Branch prediction
PPE pipeline back end
Branch instruction
DLY
DLY
DLY
RF1
RF2
EX1
EX2
EX3
EX4
IBZ
IC0
Fixed-point unit instruction
DLY
DLY
DLY
RF1
RF2
EX1
EX2
EX3
EX4
EX5
WB
Load/store instruction
RF1
RF2
EX1
EX2
EX3
EX4
EX5
EX6
EX7
EX8
WB
IC Instruction cache
IB Instruction buffer
BP Branch prediction
MC Microcode
ID Instruction decode
IS Instruction issue
DLY Delay stage
RF Register file access
EX Execution
WB Write back
(b)