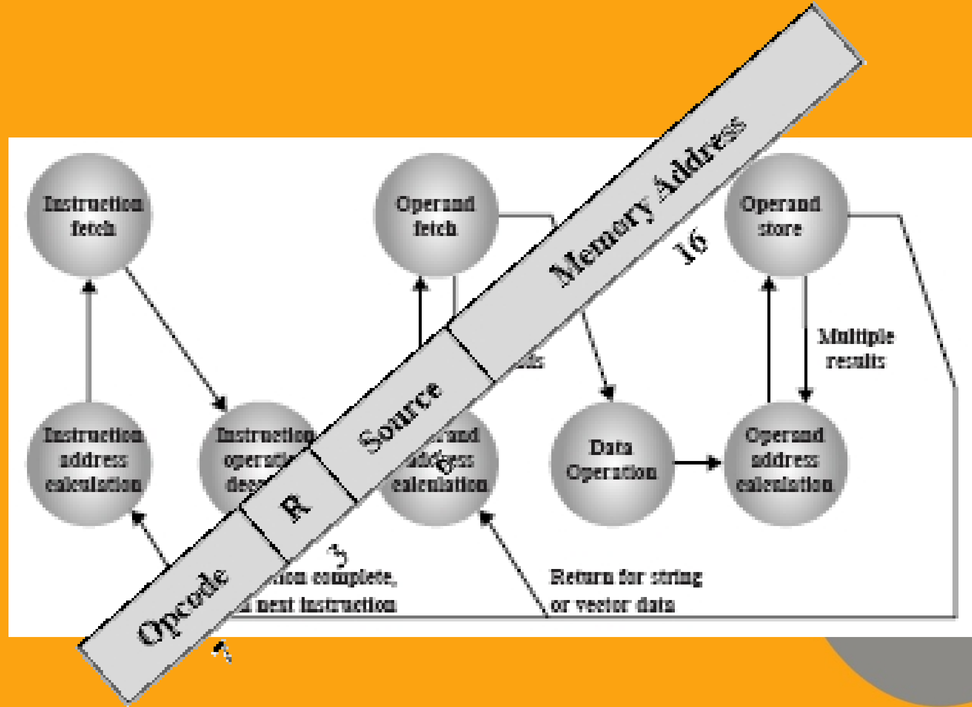
Lecture 6

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Instruction sets (*Käskykannat*)

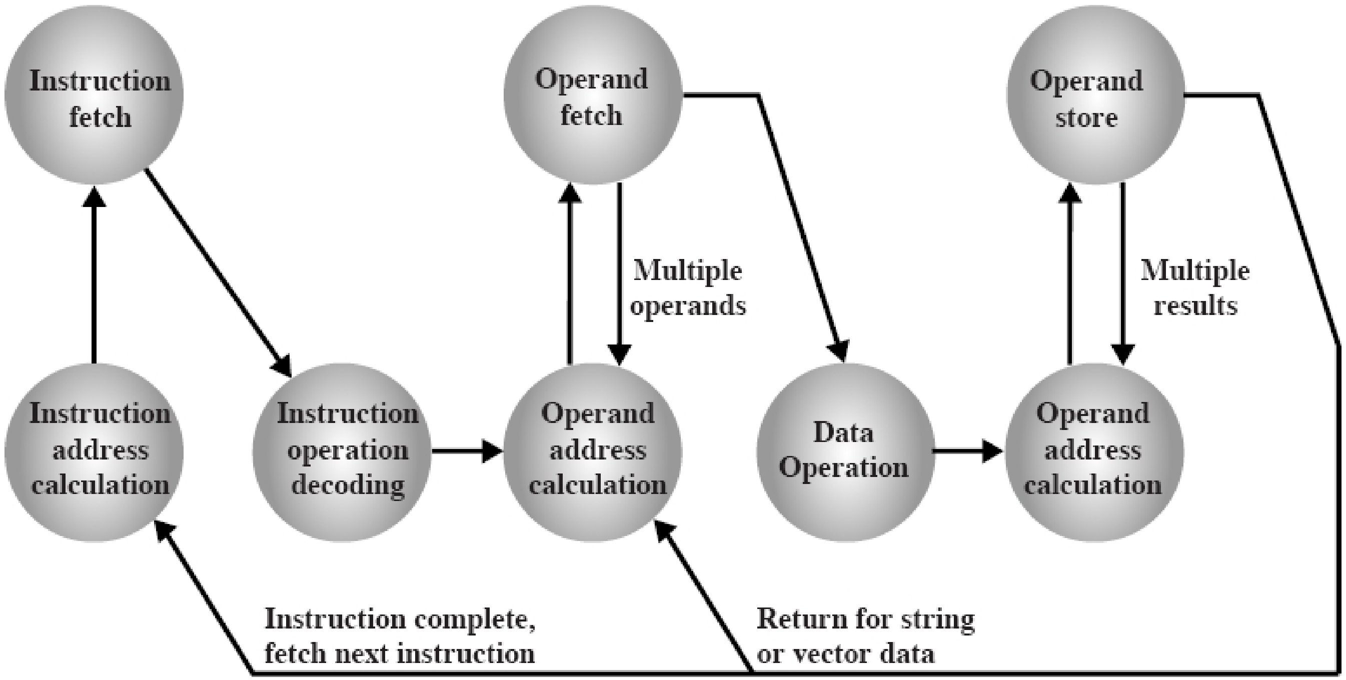


Ch 10-11 [Sta10]

- Operations
- Operands
- Operand references (osoitustavat)
- Pentium / ARM

Instruction cycle

- CPU executes instructions "one after another"
- Execution of one instruction has several phases (see state diagram). The CPU repeats these phases



(Sta10 Fig 10.1)

Computer Instructions (*konekäskyt*)

- Instruction set (*käskykanta*) =
  - Set of instructions CPU 'knows'
- Operation code (*käskykoodi*)
  - What does the instruction do?

Covered on Comp. Org I

- Data references (*viitteet*) – one, two, several?
  - Where does the data come for the instruction?
    - Registers, memory, disk, I/O
  - Where is the result stored?
    - Registers, memory, disk, I/O

Access time?
Access rate?

- What instruction is executed next?
  - Implicit? Explicit?
- I/O?
  - Memory-mapped I/O → I/O with memory reference operations

Instructions and data (*käskyt ja data*)

instructions

Address	Contents			
101	0010	0010	0000	0001
102	0001	0010	0000	0010
103	0001	0010	0000	0011
104	0011	0010	0000	0100
201	0000	0000	0000	0010
202	0000	0000	0000	0011
203	0000	0000	0000	0100
204	0000	0000	0000	0000

data

(a) Binary program

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Hexadecimal program

Address	Instruction	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

address

value

(c) Symbolic program

Label	Operation	Operand
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

Symbolic name

(d) Assembly program

(Sta10 Fig 11.13)

Instruction types?

Sta10 Table 10.3

- Transfer between memory and registers
  - LOAD, STORE, MOVE, PUSH, POP, …
- Controlling I/O
  - Memory-mapped I/O - like memory
  - I/O not memory-mapped – own instructions to control
- Arithmetic and logical operations
  - ADD, MUL, CLR, SET, COMP, AND, SHR, NOP, …
- Conversions (*esitystapamuunnokset)*
  - TRANS, CONV, 16bTo32b, IntToFloat, …
- Transfer of control (*käskyjen suoritusjärjestyksen ohjaus),* conditional, unconditional
  - JUMP, BRANCH, JEQU, CALL, EXIT, HALT, …
- Service requests (*palvelupyyntö)*
  - SVC, INT, IRET, SYSENTER, SYSEXIT, …
- Privileged instructions (*etuoikeutetut käskyt*)
  - DIS, IEN, flush cache, invalidate TLB, …

What happens during instruction execution?

Data Transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of Control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

(Sta10 Table 10.4)

What kind of data?

- Integers, floating-points
- Boolean (*totuusarvoja*)
- Characters, strings
  - IRA (aka ASCII), EBCDIC
- Vectors, tables
  - N elements in sequence
- Memory references
- Different sizes
  - 8 /16/32/ 64b, …
  - Each type and size has its own operation code

Operation Mnemonic	Name	Number of Bits Transferred
L	Load	32
LH	Load Halfword	16
LR	Load	32
LER	Load (Short)	32
LE	Load (Short)	32
LDR	Load (Long)	64
LD	Load (Long)	64
ST	Store	32
STH	Store Halfword	16
STC	Store Character	8
STE	Store (Short)	32
STD	Store (Long)	64

IBM EAS/390

(Sta10 Table 10.5)

Instruction representation (*käskyformaatti*)

- How many bits for each field in the instruction?
  - How many different instructions?
  - Maximum number of operands per instruction?
  - Operands in registers or in memory?
  - How many registers?
- Fixed or variable size (*vakio vai vaihteleva koko*)?

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

AC = accumulator
A, B, C = memory or register locations
T = top of stack
(T − 1) = second element of stack

(Sta10 Table 10.1)

How many registers?

- Minimum 16 to 32
  - Work data in registers
- Different register (sets) for different purpose?
  - Integers vs. floating points, indices vs. data, code vs. data
  - All sets can start register numbering from 0
  - Opcode determines which set is used
- More registers than can be referenced?
  - CPU allocates them internally
    - Register window – virtual register names
  - Example: function parameters passed in registers
    - Programmer thinks that registers are always r8-r15,
    - CPU maps r8-r15 somewhere to r8-r132
    - (We'll come back to this later)

Architectures

- Accumulator-based architecture (*akkukone*)
  - Just one register, accumulator, implicit reference to it
- Stack-based (*pinokone)*
  - Operands in stack, implicit reference
  - PUSH, POP
- Register-based (*yleisrekisterikone*)
  - All registers of the same size
  - Instructions have 2 or 3 operands
- Load/Store architecture
  - Only LOAD/STORE have memory refs
  - ALU-operations have 3 regs

See : Appendix 10A
in Ch10 [Sta10]

Example: JVM

```
LOAD   R3, C
LOAD   R2,B
ADD    R1,R2,R3
STORE R1,A
```

Byte ordering (*tavujärjestys*): Big vs. Little Endian

See : Appendix 10B (Sta10)

- How to store a <u>multibyte</u> scalar value?

0x1200:

0x1200	0x1201	0x1202	0x1203

(*sanaosoite*) Word

Byte (*tavuosoitteet*)

"Isoimmassa lopputavu"

STORE 0x11223344 ,0x1200 ???

<u>Big-Endian:</u>
Most significant byte in lowest byte addr →

0x11	0x22	0x33	0x44
0x1200	0x1201	0x1202	0x1203

<u>Little-Endian</u>:
Least signifant byte in lowest byte addr →

0x44	0x33	0x22	0x11
0x1200	0x1201	0x1202	0x1203

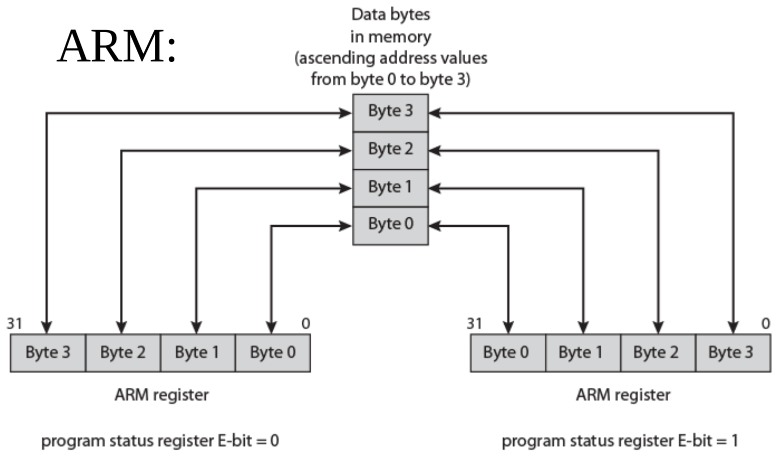
"Pienimmässä lopputavu"

0x00000044 =

0x44	0x00	0x00	0x00
0x1200	0x1201	0x1202	0x1203

Big vs. Little Endian

ARM:



(Sta10 Fig 10.5)

- ALU uses only one of them
  - Little-endian: x86, Pentium, VAX
  - Big-endian: IBM 370/390, Motorola 680x0 (Mac),
    most RISC-architectures
  - ARM, a bi-endian machine, accepts both
    - System control register has 1 bit (E-bit) to incidate the endian mode
    - Program controls which to use
- Byte order must be known, when transfering data from one machine to another
  - Internet uses big-endian format
  - Socket library (*pistokekirjasto*) has routines htoi() and itoh()
    (Host to Internet & Internet to Host)

Data alignment (*kohdentaminen*)

0010…1001<u>0</u>
0010…101<u>00</u>
0010…11<u>000</u>

- 16b data starts with even (*parillinen*) (byte)address
- 32b data starts with address divisible (*jaollinen*) by 4
- 64b data starts with address divisible by 8
- Aligned data is easier to access
  - 32b data can be loaded by one operation accessing the word address (*sanaosoite*)
- Unaligned data would contain no 'wasted' bytes, but
  - For example, loading 32b unaligned data requires two loads from memory (word address) and combining it

```
load r1, 0(r4)
```

r4 →

11	22	33	44

r1:

11	22	33	44

or

```
load r1, 2(r4)
shl  r1, =16
load r2, 4(r4)
shr  r2, =16
or   r1, r2
```

r4 →

		11	22
33	44		

r1:

11	22	33	44

Memory references

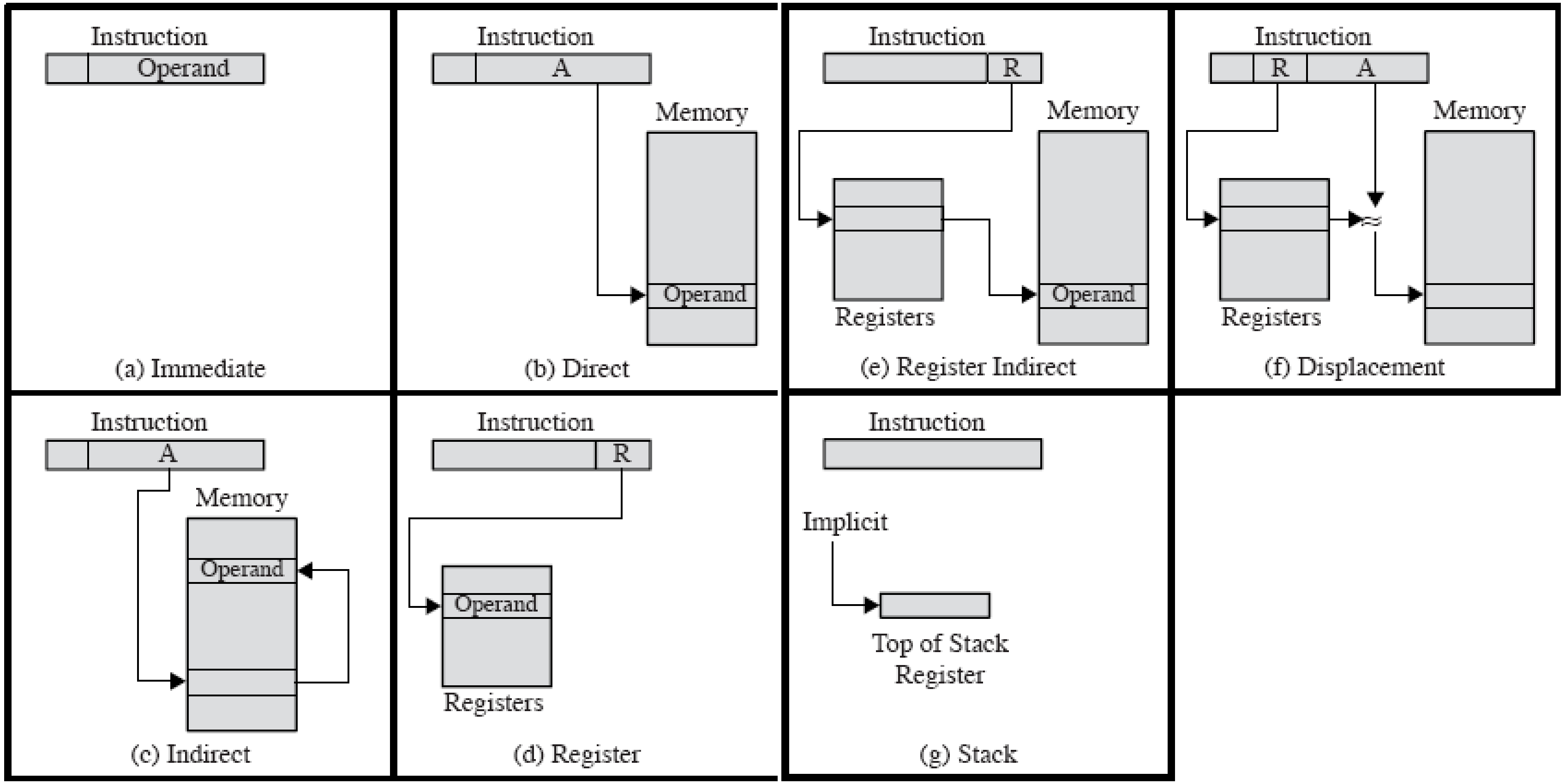
(*Muistin osoitustavat*)

Ch 11 [Sta10]

Where are the operands?

- In the memory
  - Variable of the program, stack (*pino*), heap (*keko*)
- In the registers
  - During the instruction execution, for speed
- Directly in the instruction
  - Small constant values
- How does CPU know the specific location?
  - Bits in the operation code
  - Several alternative addressing modes allowed

Addressing modes (*osoitusmuodot*)



(Sta10 Fig 11.1)

Addressing modes

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	EA = 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

(Sta10 Table 11.1)

- 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)

Displacement Address (*siirtymä*)

- Effective address = (R1) + A

(*tehollinen muistiosoite*)

register content + constant in the instruction

- Constant relatively small (8 b, 16 b?)
- Usage
  - Relational to PC
  - Relational to Base
  - Indexing a table
  - Ref to record field
  - Stack content

    (*e.g., in activation record*)

JUMP *+5

CALL SP, Summation(BX)

ADDF F2,F2, Table(R5)

MUL F4,F6, Salary(R8)

STORE F2, -4(FP)

More addressing modes

operand size

- Autoincrement (before/after)
  - Example CurrIndex=i++;

$$EA = (R),\ R \leftarrow (R) + S$$

- Autodecrement (before/after)
  - Example CurrIndex=--i;

$$R \leftarrow (R) - S,\ EA = (R)$$

- Autoincrement deferred
  - Example Sum = Sum + (*ptrX++);

$$EA = Mem(R),\ R \leftarrow (R) + S$$

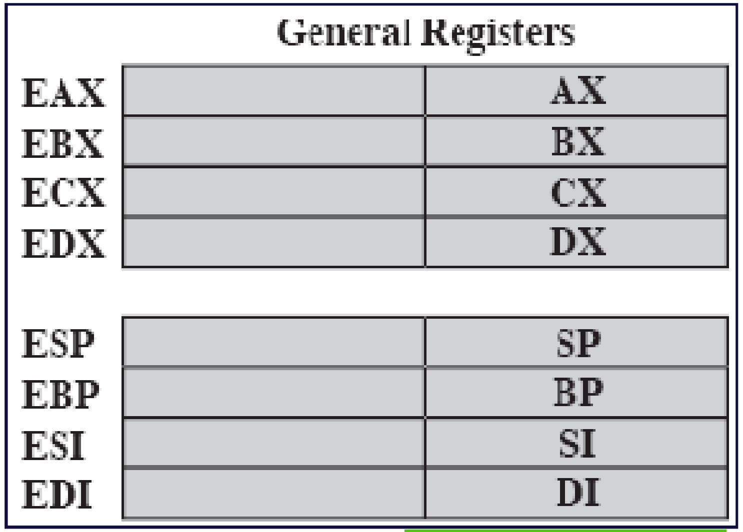
- Autoscale
  - Example Double X;
    ….
    X=Tbl[i];

$$EA = A + (R) * S$$

Pentium

Pentium: Registers

- General registers (*yleisrekisterit*), 32-b
  - EAX, EBX, ECX, EDX accu, base, count, data
  - ESI, EDI source & destination index
  - ESP, EBP stack pointer, base pointer
- Part of them can be used as16-bit registers
  - AX, BX, CX, DX, SI, DI, SP, BP
- Or even as 8-bit registers
  - AH, AL, BH, BL, CH, CL, DH, DL
- Segment registers 16b
  - CS, SS, DS, ES, FS, GS
    - code, stack, data, data, ...
- Program counter (*käskynosoitin)*
  - EIP Extended Instruction Pointer
- Status register
  - EFLAGS
    - overflow, sign, zero, parity, carry,…



(Sta10 Fig 12.3c)

x86: Data types

- Not aligned
- Little Endian

Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value co4ntained in a byte, word, or doubleword, using twos complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2^{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2^{32} - 1$ bytes.
Floating point	Single / Double / Extended precision
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types

IEEE 754 standard

(Sta10Table 10.2)

Pentium: Operations

(just part of)

Data transfers, arithmetics, moves, jumps, stricts, etc

High-Level Language Support	
ENTER	Creates a stack frame that can be used to implement the rules of a block-structured high-level language.
LEAVE	Reverses the action of the previous ENTER.
BOUND	Check array bounds. Verifies that the value in operand 1 is within lower and upper

Segment Register	
LDS	Load pointer into D segment register.
	System Control
HLT	Halt.
LOCK	Asserts a hold on shared memory so that the Pentium has exclusive use of it during the instruction that immediately follows the LOCK.
ESC	Processor extension escape. An escape code that indicates the succeeding instructions are to be executed by a numeric coprocessor that supports high-precision integer and floating-point calculations.
WAIT	Wait until BUSY# negated. Suspends Pentium program execution until the processor detects that the BUSY pin is inactive, indicating that the numeric coprocessor has finished execution.
Protection	
SGDT	Store global descriptor table.
LSL	Load segment limit. Loads a user-specified register with a segment limit.
VERR/VERW	Verify segment for reading/writing.
Cache Management	
INVD	Flushes the internal cache memory.
WBINVD	Flushes the internal cache memory after writing dirty lines to memory.
INVLPG	Invalidates a translation lookaside buffer (TLB) entry.

(Sta10 Table 10.8)

Pentium: MMX Operations

(just part of)

SIMD

Category	Instruction	Description
Arithmetic	PADD [B, W, D]	Parallel add of packed eight bytes, four 16-bit words, or two 32-bit doublewords, with wraparound.
	PADDS [B, W]	Add with saturation.
	PADDUS [B, W]	Add unsigned with saturation
	PSUB [B, W, D]	Subtract with wraparound.
	PSUBS [B, W]	Subtract with saturation.
	PSUBUS [B, W]	Subtract unsigned with saturation
	PMULHW	Parallel multiply of four signed 16-bit words, with high-order 16 bits of 32-bit result chosen.
	PMULLW	Parallel multiply of four signed 16-bit words, with low-order 16 bits of 32-bit result chosen.
	PMADDWD	Parallel multiply of four signed 16-bit words; add together adjacent pairs of 32-bit results.

No under/overflow.
Use closest representation

Conversion	PACKUSWB	Pack words into bytes with unsigned saturation.
	PACKSS [WB, DW]	Pack words into bytes, or doublewords into words, with signed saturation.
	PUNPCKH [BW, WD, DQ]	Parallel unpack (interleaved merge) high-order bytes, words, or doublewords from MMX register.
	PUNPCKL [BW, WD, DQ]	Parallel unpack (interleaved merge) low-order bytes, words, or doublewords from MMX register.

(Sta10 Table 10.11)

Pentium: Addressing modes (muistin 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

1, 2, 4, 8B

Registers: 1, 2, 4, 8B

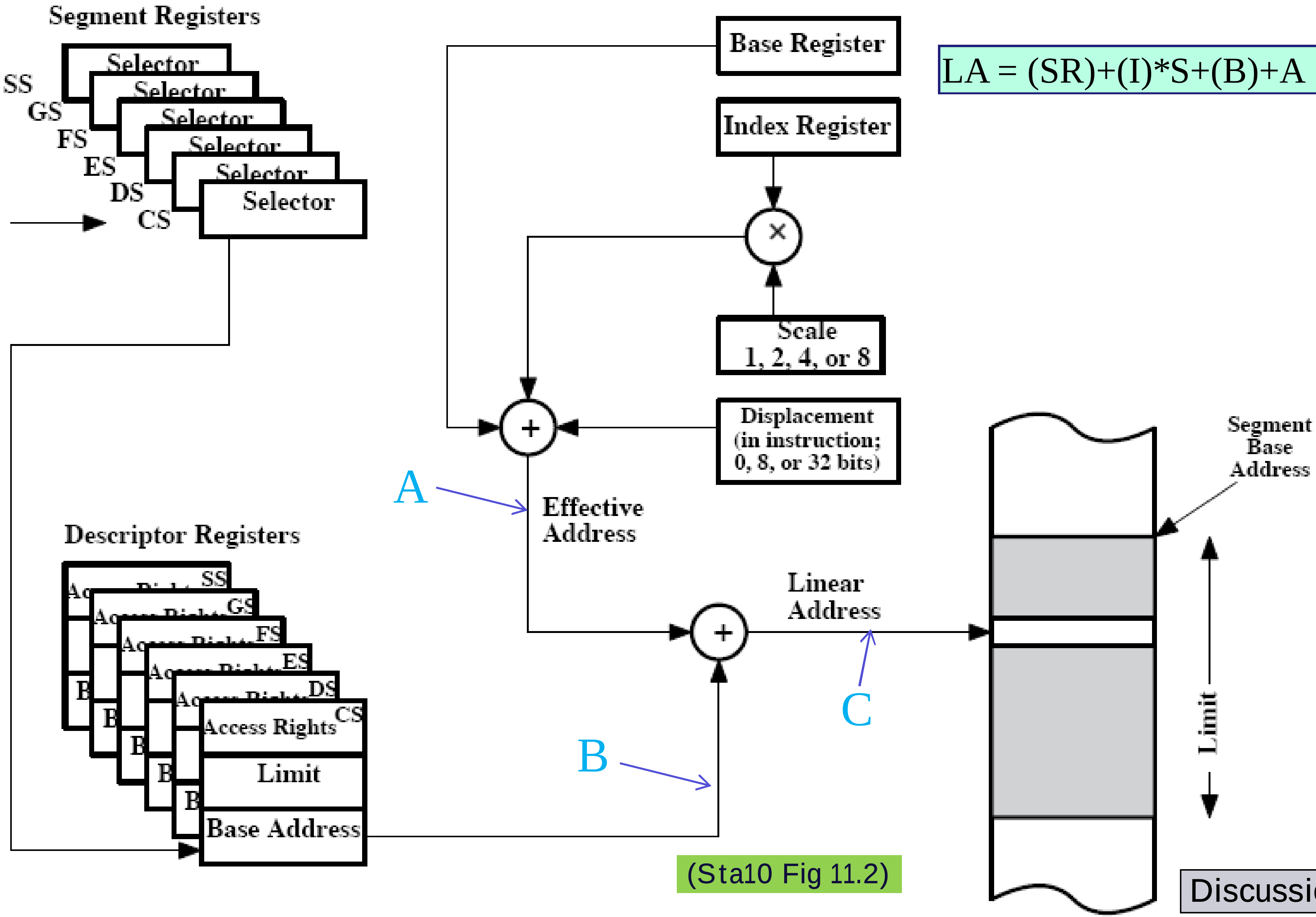
LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction

R = register
B = base register
I = index register
S = scaling factor

(Sta10 Table 11.2)

indexing arrays?
arrays in stack?
two dimensional arrays?

Pentium: Addressing Mode Calculation



LA = (SR)+(I)*S+(B)+A

(Sta10 Fig 11.2)

Discussion?

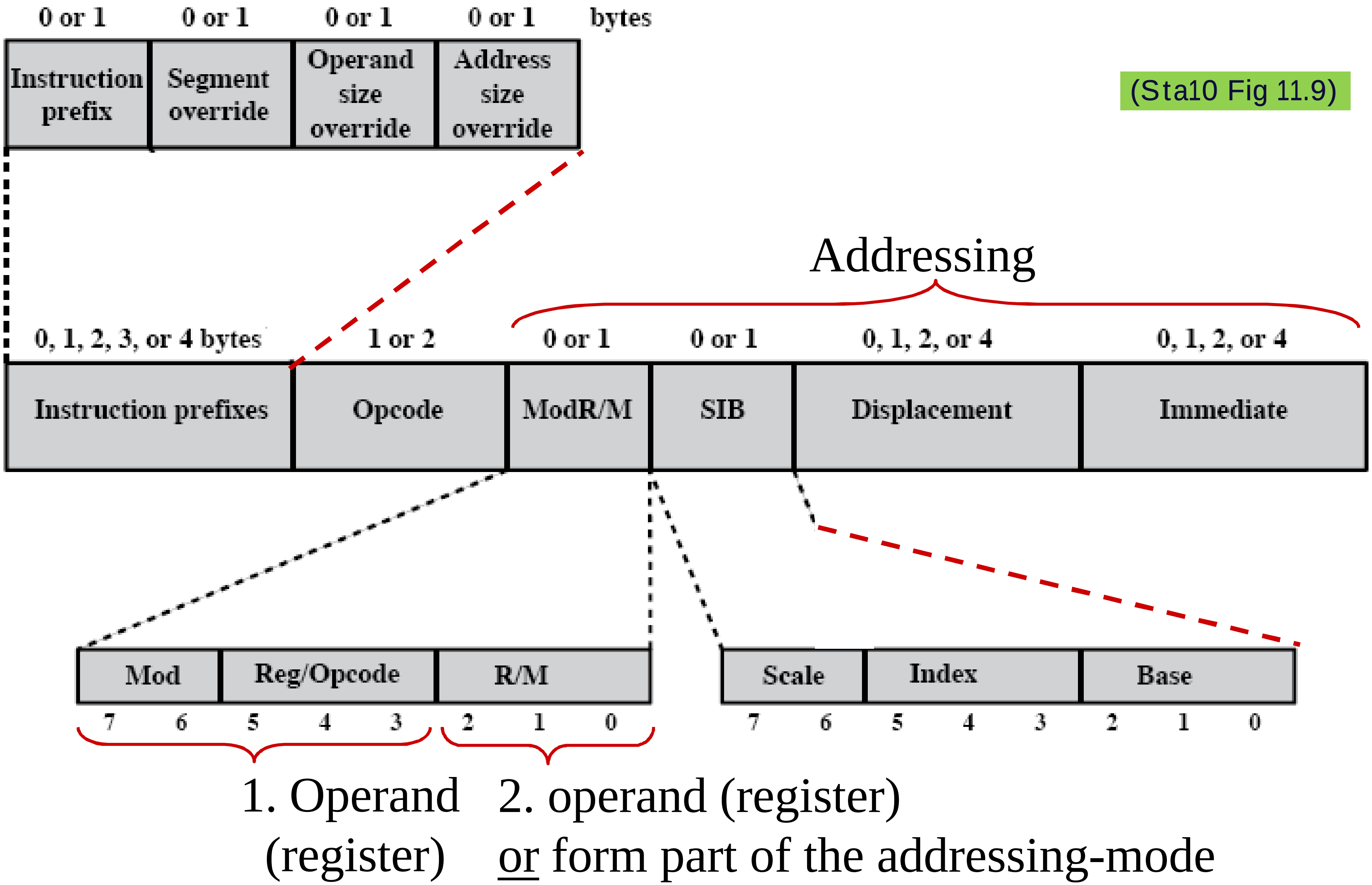
Pentium: Instruction format

- CISC
  - Complex Instruction Set Computer
- Lots of alternative fields

  only op-code always!
  - Part may be present or absent in the bit sequence
  - Prefix 0-4 bytes
  - Interpretation of the rest of the bit sequence depends on the content of the preceding fields
- Plenty of alternative addressing modes (*osoitustapa)*
  - At most one operand can be in the memory
  - 24 different
- Backward compatibility
  - OLD 16-bit 8086-programs must still work
    - How to handle old instructions: emulate, simulate?

Pentium: Instruction format

(Sta10 Fig 11.9)



Pentium: Instruction format

- Instruction prefix (optional)
  - LOCK –exclusive use of shared memory in multiprocessor env.
  - REP – repeat operation to all characters of a string
- Segment override (optional)
  - Use the segment register explicitly specified in the instruction
  - Else use the default segment register (implicit assumption)
- Operand size override (optional)
  - Switch between 16 or 32 bit operand, override default size
- Address size override (optional)
  - Switch between 16 or 32 bit addressing. Override the default, which could be either

Pentium: Instruction format

- Opcode
  - Each instruction has its own bit sequence (incl. opcode)
  - Bits specify the size of the operand (8/16/32b)
- ModR/m(optional)
  - Indicate, whether operand is in a register or in memory
  - What addressing mode (*osoitusmuoto*) to be used
  - Sometimes enhance the opcode information (with 3 bits)
- SIB = Scale/Index/Base (optional)
  - Some addressing modes need extra information
  - Scale: scale factor for indexing (element size)
  - Index: index register (number)
  - Base: base register (number)

Pentium: Instruction format

- Displacement (optional)
  - Certain addressing modes need this
  - 0, 1, 2 or 4 bytes (0, 8, 16 or 32 bits)
- Immediate (optional)
  - Certain addressing modes need this, value for operand
  - 0, 1, 2 or 4 bytes

ARM Instructions

ARM: Instruction set (käskykanta)

- RISC
  - Reduced Instruction Set Computer
- Fixed instruction length (32b), regular format
  - All instructions have the condition code (4 bits)
- Small number of different instructions
  - Instruction type (3 bit) and additional opcode /modifier (5 bit)
  - Easier hardware implementation, faster execution
  - Longer programs?
- Load/Store-architecture
- 16 visible general registers   (4 bits in the instruction)
- Fixed data size

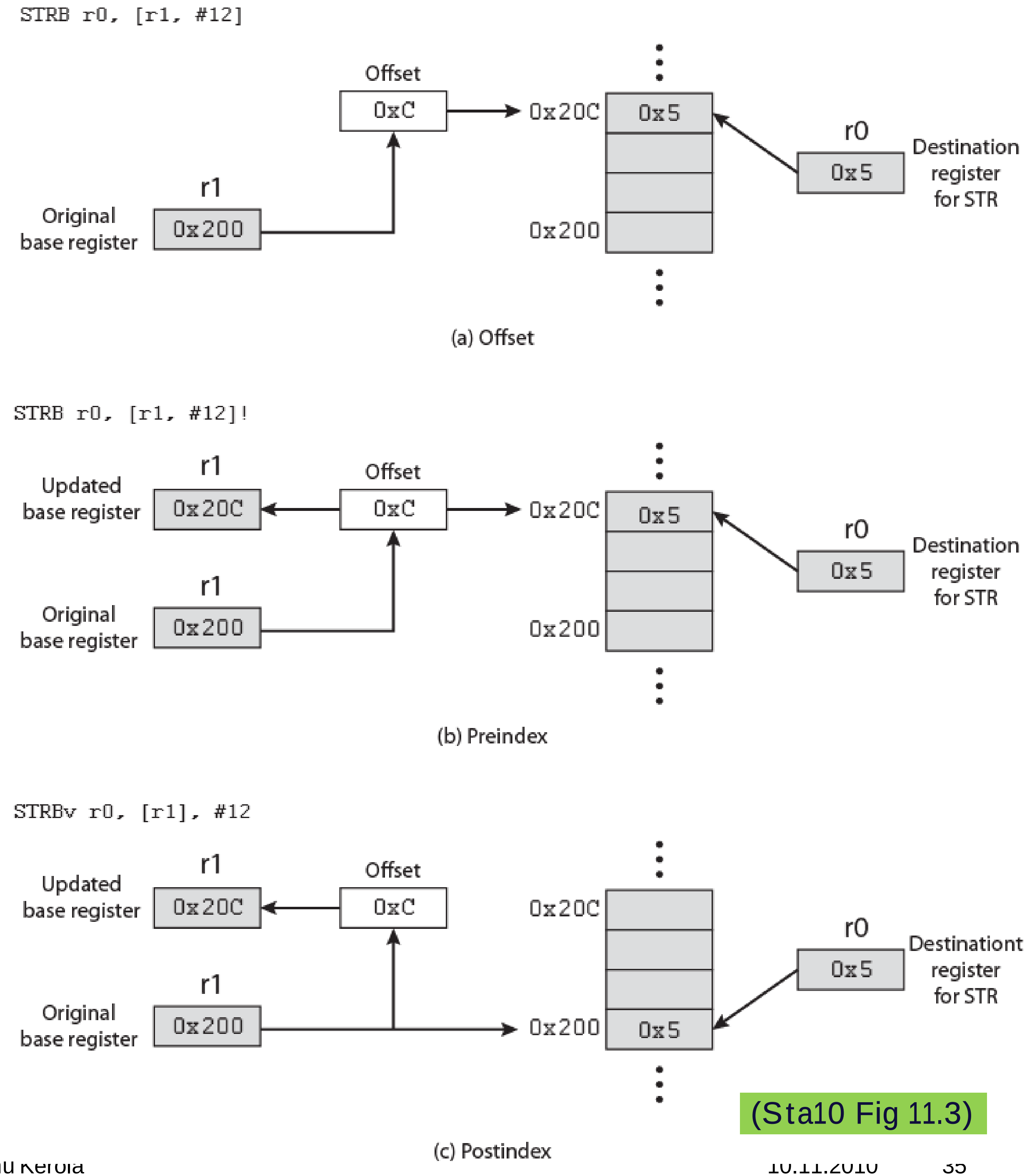
- **Thump** instruction set uses 16 bit instructions

ARM Data Types

- 8 (byte), 16 (halfword), 32 (word) bits - word aligned
- Unsigned integer and twos-complement signed integer
- Majority of implementations do not provide floating-point hardware
- Little and Big Endian supported
  - Bit E in status register defines which is used

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



(Sta10 Fig 11.3)

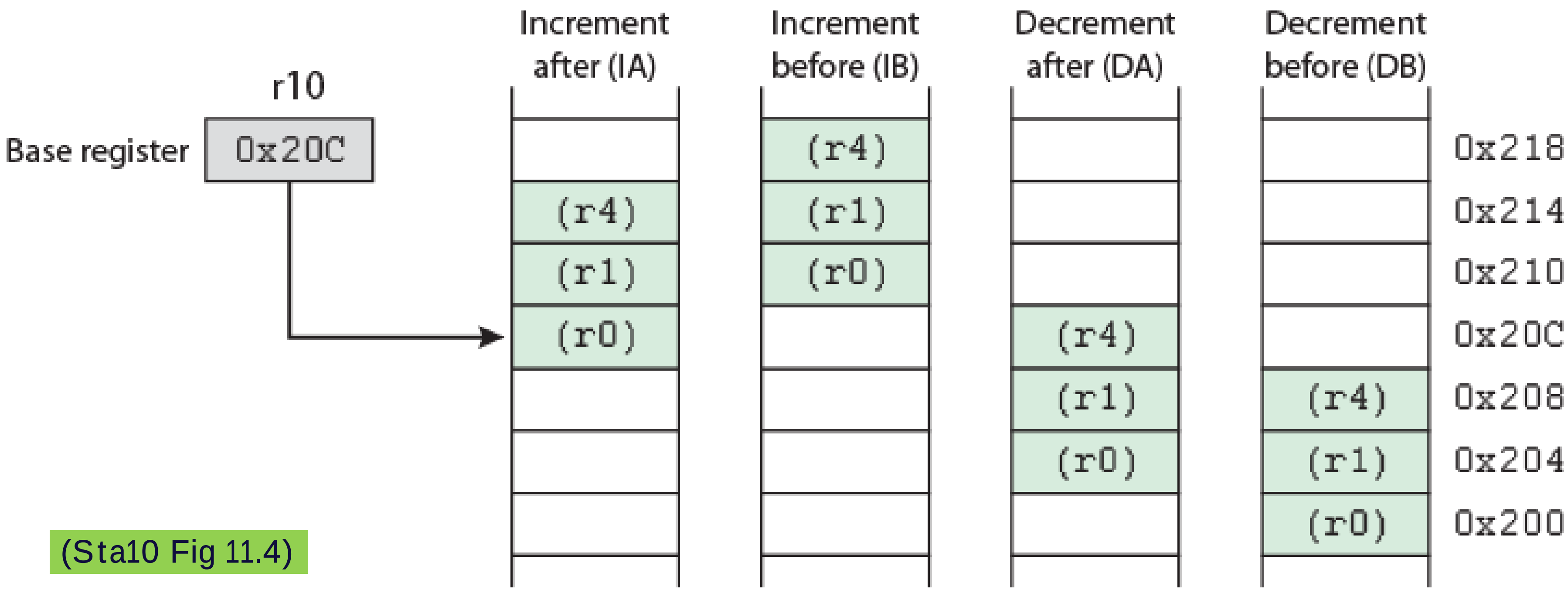
ARM Addressing mode

- Data Processing instructions
  - Register addressing
    - Value in register operands may be scaled using a shift operator
  - Or mixture of register and immediate addressing
- Branch instructions
  - Immediate
  - Instruction contains 24 bit value
  - Shifted 2 bits left
    - On word boundary
    - Effective range +/-32MB from PC.

ARM Load/Store Multiple Addressing

- Load/store subset of general-purpose registers
  - 16-bit instruction field specifies list of registers
  - Sequential range of memory addresses
  - Base register specifies main memory address

```
LDMxx r10, {r0, r1, r4}
STMxx r10, {r0, r1, r4}
```



(Sta10 Fig 11.4)

ARM Instruction Formats

(Sta10 Fig 11.10)

	31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–0
data processing immediate shift	cond	0	0	0	opcode				S	Rn	Rd	shift amount (11–7) | shift (6–5) | 0 (4) | Rm (3–0)
data processing register shift	cond	0	0	0	opcode				S	Rn	Rd	Rs (11–8) | 0 (7) | shift (6–5) | 1 (4) | Rm (3–0)
data processing immediate	cond	0	0	1	opcode				S	Rn	Rd	rotate (11–8) | immediate (7–0)
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 (11–7) | shift (6–5) | 0 (4) | Rm (3–0)
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

Discussion?

ARM Condition codes

Condition flags:
N, Z, C and V

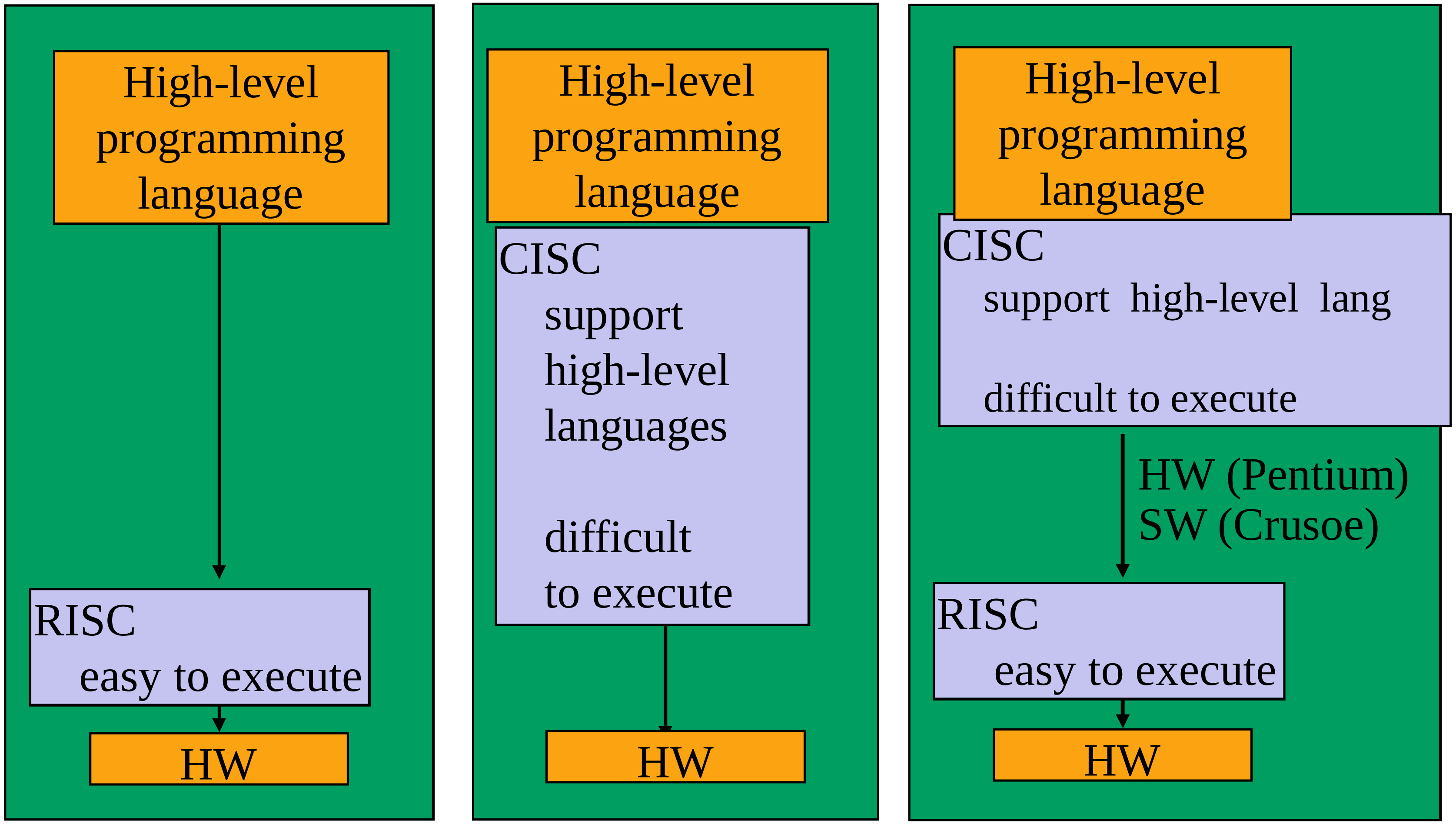
N – Negative
Z – Zero
C – Carry
V – oVerflow

(Sta10 Tbl 10.12)

Code	Symbol	Condition Tested	Comment
0000	EQ	Z = 1	Equal
0001	NE	Z = 0	Not equal
0010	CS/HS	C = 1	Carry set/unsigned higher or same
0011	CC/LO	C = 0	Carry clear/unsigned lower
0100	MI	N = 1	Minus/negative
0101	PL	N = 0	Plus/positive or zero
0110	VS	V = 1	Overflow
0111	VC	V = 0	No overflow
1000	HI	C = 1 AND Z = 0	Unsigned higher
1001	LS	C = 0 OR Z = 1	Unsigned lower or same
1010	GE	N = V [(N = 1 AND V = 1) OR (N = 0 AND V = 0)	Signed greater than or equal
1011	LT	N ≠ V [(N = 1 AND V = 0) OR (N = 0 AND V = 1)]	Signed less than
1100	GT	(Z = 0) AND (N = V)	Signed greater than
1101	LE	(Z = 1) OR (N ≠ V)	Signed less than or equal
1110	AL	—	Always (unconditional)
1111	—	—	This instruction can only be executed unconditionally

RISC vs. CISC

We'll return to this later (lecture 8)



Summary

- Instruction set types: Stack, register, load-store
- Data types: Int, float, char
- Addressing modes: indexed, others?
- Operation types?
  - Arithmetic & logical, shifts, conversions, vector
  - Comparisons
  - Control
    - If-then-else, loops, function calls/returns
    - Conditional instructions
  - Loads/stores, stack ops, vector ops
  - Privileged, os instructions
- Instruction formats
- Intel and Arm case studies

Review Questions / Kertauskysymyksiä

- Fields of the instruction?
- How does CPU know if the integer is 16 b or 32 b?
- Meaning of Big-Endian?
- Benefits of fixed instruction size vs. variable size instruction format?