DNA

May 2008

Rahuvaran Pathmanathan

DNA 2008

Syllabus: Computer Architecture – Spring 2008

[SCO] Kap 1: 1-13
[SCO] Kap 2: 51-82
[SCO] Kap 3: 135-189
[SCO] Kap 4: 231-311
[SCO] Kap 5: 331-339, 348-357, 360-371
[SCO] Appendix A: s. 679-688

Computer Architecture

1M: Digital logic.
Explain how basic digital logic circuits are made and how they relate to Boolean algebra.
Redegør for opbygningen af digitale kredsløb og relationen til Boolsk algebra.

2M: Microarchitecture.
Explain the components that make up a microarchitecture.
Redegør for et eksempel på en mikroarkitekturs bestanddele.

3M: Microarchitecture.
Explain the structure and function of a micro instruction.
Redegør for formatet af en mikroinstruktion.

4M: Microarchitecture.
Explain the concept behind pipelining and how it is implemented.
Redegør for principperne omkring pipelining, samt teknikker til implementation.

5M: ISA layer.
Explain a typical structure of an ISA instruction and how it is executed on a micro computer.
Redegør for en typisk opbygning af en ISA-instruktion og for afvikling af denne på en mikrodatamat.

1C: RISC versus CISC.
Define and explain the difference between RISC and CISC architectures.
Redegør for RISC og CISC-arkitekturer og forklar forskellen.

2C: Micro Architecture.
Explain the structure and function of an Instruction Fetch Unit.
Redegør for opbygningen og anvendelsen af en "Instruction Fetch Unit".

3C: CPU Architecture.
Explain the structure and function of a data path.
Redegør for opbygningen og anvendelsen af en "Data Path".

4C: Cache Memory.
Explain the principle of cache memory.
Redegør for begrebet "Cache Memory" og dets anvendelse.

5C: Bus Architecture.
Explain the use of latch A, B and C in the MIC-3 machine.
Redegør for anvendelsen af A, B og C latch i MIC-3.

6C: Memory.
Explain the concept behind the memory hierarchy.
Redegør for princippet om et hukommelses-hierarki.

7C: Digital logic.
Define and explain the fundamental Boolean operations.
Redegør for de fem grundlæggende Bool'ske funktioner.

8C: The Von Neumann machine.
Explain the structure of a classical Von Neumann machine.
Redegør for opbygningen af en klassisk von Neumann maskine.

9C: Instruction Set
Explain shortly how Java code can be translated into binary code.
Redegør kort for oversættelsen af Java kode til binær kode.

10C: ISA layer.
Explain the concept behind an expanding opcode.
Redegør for princippet bag "expanding opcode".

M: Main Question C: Control Question

Syllabus: Network Architecture – Spring 2008

Chapter 1: Introduction
Overview of the Internet, client/server paradigm, circuit switching, packet switching, physical media, queuing delay and packet loss, TCP/IP and OSI reference models, Internet Protocol Stack
Readings: Chapter 1 (Kurose and Ross) pp 1 - 52

Chapter 2: Application Layer I:
Conceptual implementation of application protocols; Protocols - HTTP, FTP, SMTP/POP3/IMAP.
Readings: Chapter 2 (Kurose and Ross) pp 81 - 129

Application Layer II:
DNS; P2P; Socket Programming with TCP/UDP
Readings: Chapter 2 (Kurose and Ross) pp 130 - 140; 144 - 146; 149 - 177

Chapter 3: Transport Layer I:
Principles behind transport layer - multiplexing, demultiplexing, reliable data transfer
Readings: Chapter 3 (Kurose and Ross) pp 196 - 239

Transport Layer II:
Connection-Oriented TCP; Principles of congestion control.
Readings: Chapter 3 (Kurose and Ross) pp 241 - 248; 252 - 266; 268 - 285; (3.7.1) 287 - 290.

Chapter 4: The Network Layer I:
Forwarding and Routing; Virtual circuits and datagram networks; Router; The Internet Protocol.
Readings: Chapter 4 (Kurose and Ross) pp 310 - 322; 324 - 332; 335 - 365

The Network Layer II:
Routing Algorithms; Routing in the Internet
Readings: Chapter 4 (Kurose and Ross) pp 368 - 400.

Chapter 5: The Link Layer and LANs I:
Link layer Services: Error detection and correction techniques: Multiple Access Protocols; Link Layer Addressing.
Readings: Chapter 5 (Kurose and Ross) pp 437 - 468

The Link Layer and LANs II:
Ethernet; Switches; PPP; Link Virtualization
Readings: Chapter 5 (Kurose and Ross) pp 470 - 499

Chapter 6: Wireless and Mobile Networks
WiFi: 802.11 Wireless LANs
Readings: Chapter 6 (Kurose and Ross) pp 514 - 518, 526 - 547

Network Architecture Exam Questions – Spring 2008

1M. The Internet.
Explain the OSI model, how is it related to the Internet Protocol Stack (5 layers).
Explain Circuit and Packet switching (pros and cons, how do they differ?)

2M. Application Layer.
In the Internet Protocol Stack, What does the Application Layer do?
Explain any two of the following protocols, HTTP, FTP, SMPT/POP3/IMAP, and DNS

3M. Explain socket programming with TCP and UDP.

4M. Transport Layer.
What is the main function of the transport layer protocols? How does TCP implement the reliable transmission in the Internet?

5M. Explain TCP, flow control, and congestion control.

6M. The Network Layer
Explain VC/Datagram Networks, structure of a router (switching techniques) and any three of the following: IP, IP addressing, subnetmask, and NAT.

7M. Routing Algorithms: Explain Link-state (Dijkstra's), ASs, RIP, BGP **or** Distance vector, ASs, RIP, BGP
Demonstrate Dijkstra's (or distance vector).

8M. The Link Layer and LANs
What is ARP protocol? Which information can a ARP table contain possibly? Please describe in detail how the translation is done and how to send a datagram to a host over the Internet.

9M. We can classify multiple access protocol into three categories: channel partitioning protocols, random access protocols and taking turns. Please explain their principle respectively.

10M. Wireless and Mobile Networks
Explain the principle of 802.11 media access protocol - CSMA/CA.
Compare with Ethernet's CSMA/CD.

1C. TCP/UDP: Isn't TCP always preferable to UDP since TCP provides a reliable data transfer service and UDP does not?

2C. Routing: What is the difference between routing and forwarding?

3C. Compare and contrast link-state and distance-vector routing algorithms.

4C. DHCP: What is DHCP? How can it be used?

5C IP: How can one transition from IPv4 to IPv6? What techniques could be used?

Figurhenvisninger

Computer Architecture

1M	fig. 3-10	3-2, 3-22, 3-23, 3-24, 3-25, 3-26
2M	fig. 4-6	4-1
3M	fig. 4-5	4-3
4M	fig. 2-4,	4-34, 2-5, 2-6
5M	fig. 4-11	4-14, 4-15, 4-17, 5-12, 5-13
1C		
2C	fig. 4-27	
3C	fig. 4-1	
4C	fig. 2-16	
5C	fig. 4-31,	4-34 (pipelining)
6C	fig. 2-18	
7C	fig. 3-2	
8C	fig. 1-5	
9C	fig. 4-11,	4-14, 4-15
10C	fig. 5-12,	5-13

Network Architecture

1M	fig. 1-19	1-8, 1-10
2M	fig. 2-7	2-14
3M	fig. 2-30	2-31, 2-32, 2-34
4M	fig. 3-2	3-9, 3-10, 3-11, 3-12, 3-13, 3-14, 3-15, 3-31, 3-36, 3-37
5M	fig. 3-2	3-38, 3-51,
6M	Table 319	4-8, 4-22
7M	fig. 4-27	4-30, 4-36-4,38
8M	fig. 5-18	5-19
9M	fig. 5-11	5-12, 5-14
10M	fig. 6-10	5-14
1C	fig. 2-5	
5C	fig. 4-24	(gamle bog)

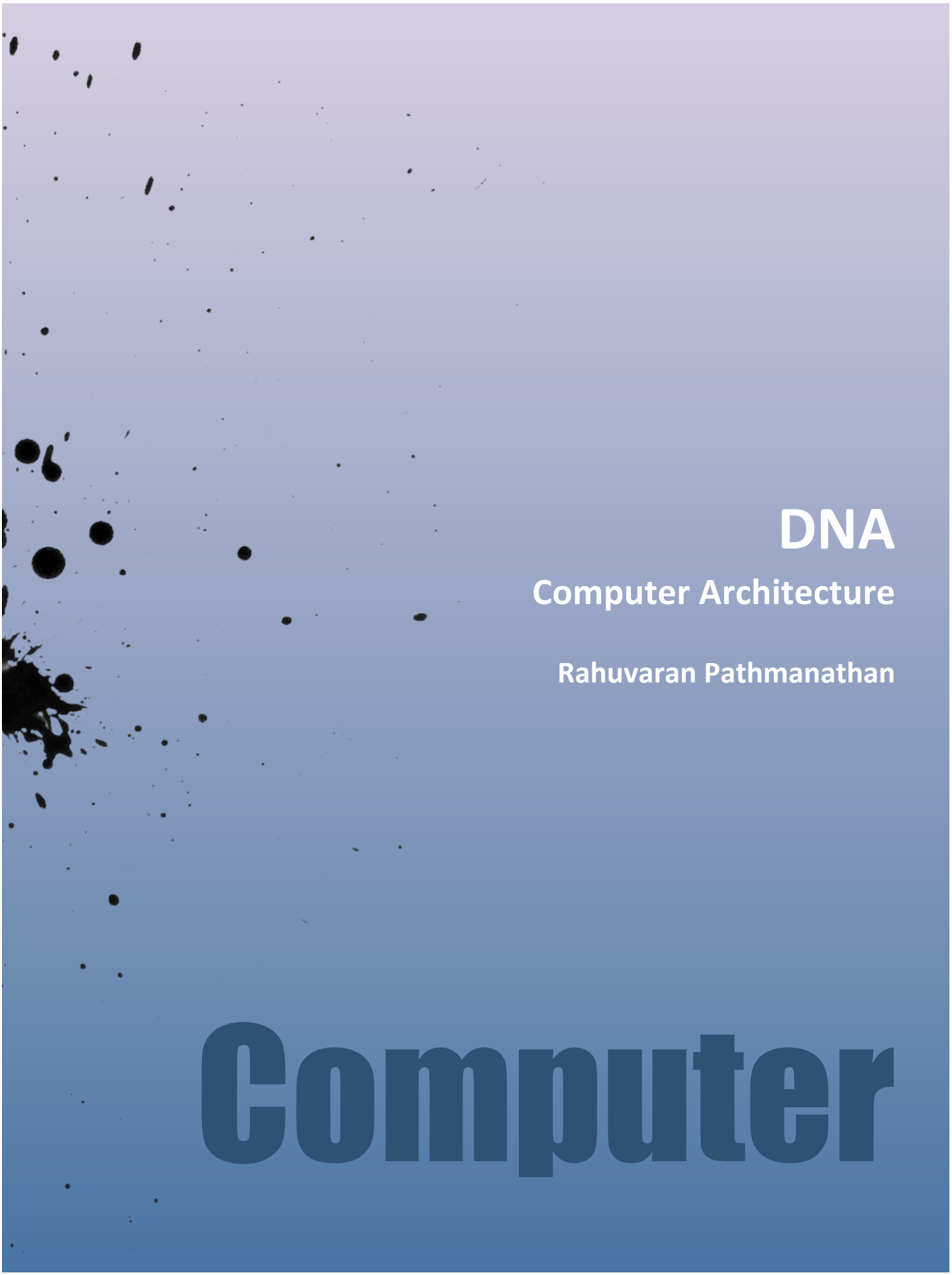
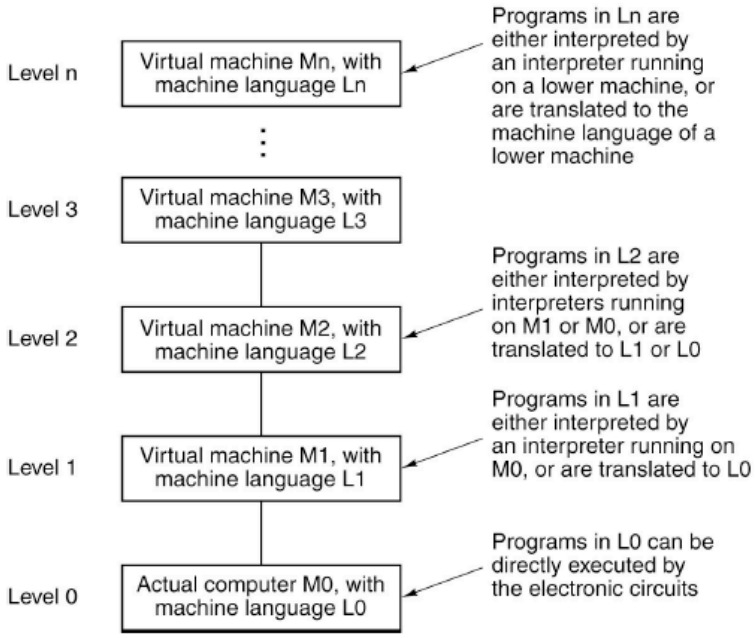

DNA
Computer Architecture
Rahuvaran Pathmanathan
Computer

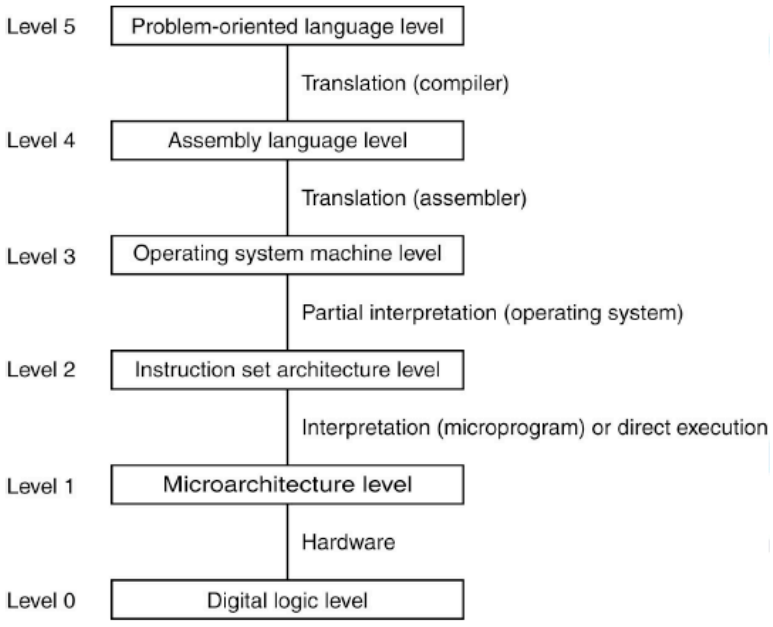
Table of Contents

Introduction



One method of executing a program written in L1 is first to replace each instruction in it by an equivalent sequence if instructions in L0. This technique is called translation.

Another method is to write a program In L0 that takes programs in L1 as input data and carries them out by examining each instruction in turn and executing the equivalent sequence of L0 instructions directly. This technique is called interpretation and the program that carries it out is called an interpreter.



Most modern computers consist of two or more levels. Level 0, at the bottom, is the machine's true hardware. Its circuits carry out the machine-language programs of level 1. At the Digital logic level, the interesting objects are called gates. Although build from analog components, such as transistors gates can be accurately modeled as digital devices.

Each gate has one or more digital inputs (signals representing 0 or 1). A small number of gates can be combined to form a 1-bit memory. Again the 1-bit memory can be combined with other 1-bit memories to form a 16, 32, 64 register.

Micro architecture level (level 1)

At the micro architecture level we see collection of typically 8-32 registers that form a local memory and circuit called an ALU, which is capable of performing simple arithmetic operations. The registers are connected to the ALU to form a data path, over which data can flow. On some machines a program called a micro program controls the operation of the data path. On others the data path is controlled directly by hardware.

On machines with software control of the data path, the micro program is an interpreter for the instructions at level 2. It fetches, examines, and executes instructions one by one, using the data path to do so.

Example, for an ADD instruction, the instruction would be fetched, its operands located and brought into registers, the sum computed by the ALU, and finally the result routed back to the place it belongs.

Instruction Set Architecture (ISA level) (level 2)

Every computer manufacture publishes a manual for each of the computers it sells. These manuals are really about the machine's instruction set, not the underlying levels. When they describe the machine's instruction set, they are in fact describing the instructions are carried out interpretively by the micro program or hardware execution circuits.

Operating system machine level (level 3)

Usually a hybrid level. Here we find both level 2 and level 3 instructions, the level 3 is interpreted by the operating system and the level 2 is interpret by the micro program.

Level 2 and 3 are always interpreted. Level 4, 5 and above are usually supported by translation.

The language of level 1, 2 and 3 are numeric. Starting at level 4 the language contains words and abbreviations meaningful to people. Level 4, the assembly language level, is really a symbolic form for one of the underlying languages. This level provides a method for people to write programs for level 1, 2 and 3 in a form that is not as unpleasant as the virtual machine languages themselves. Programs in assembly language are first translated to level 1, 2 or 3, and the interpreted by the appropriate virtual or actual machine. The program that performs the translation is called an assembler.

Level 5 usually consist of languages designed to be used by applications programmers. Such languages are often called high-level languages (C, C++ and Java). These languages are generally translated to level 3 or 4 by translators called a compiler.

1M Digital logic

Explain how basic digital logic circuits are made and how they relate to Boolean algebra.

An integrated circuit chip contains a number of gates, and is characterised as follows:

- SSI (Small-Scale Integration): 1-10 gates.
- MSI (Medium-Scale Integration): 10-100 gates.
- LSI (Large-Scale Integration): 100-100.000 gates.
- VLSI (Very-Large-Scale Integration): over 100.000 gates.

Gates

Digital logic is build up by assembled gates, where each chip, has some pins. These pins can be input, output, power and ground. The different types of gates can be combined with each other by these pins and to create a Boolean algebra and functions. The same effect can be found, by assembling the different type of gates.

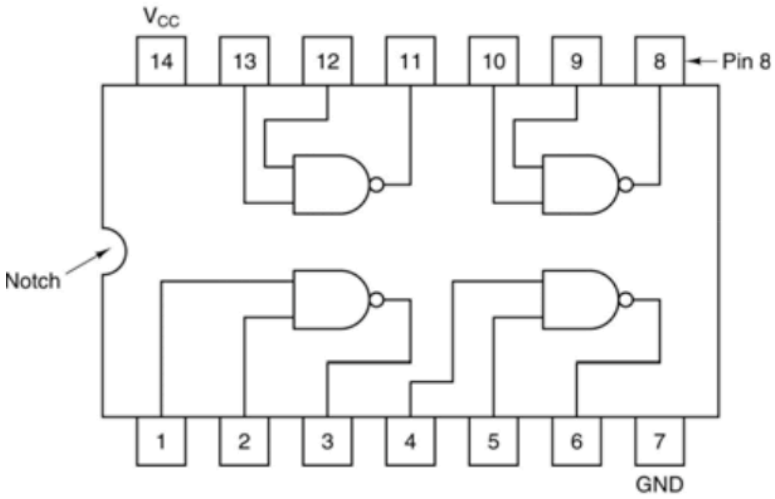


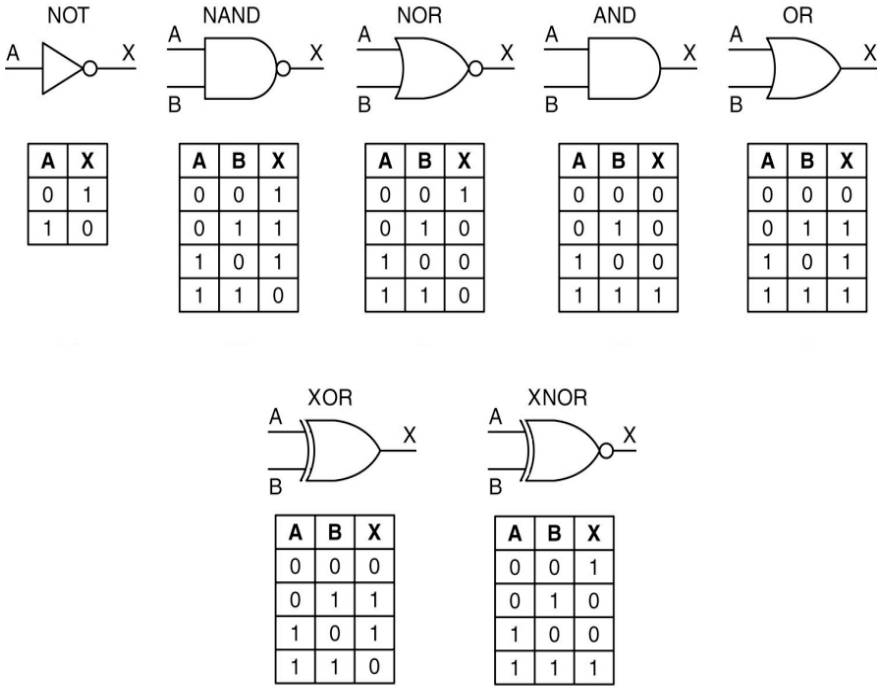
fig 3-10 (page 147 in the book)

It is therefore a question of getting a small amount of gates, or different types of gates as possible in a circuit.

The most used gates today is NAND and NOR, while they only uses 2 transistors, and AND & OR gates uses 3 transistors.

Transistor (fig. 3-1) – Truth table for NAND, NOR and NOT.

The relation to the Boolean algebra lays in the different types of gates, and the work with the values 0 and 1. These gates sent out a voltage of 0-1 for the value of 0, and between 3,3-5v for the value of 1. They can be used for creating bigger digital circuits.



A	X
0	1
1	0

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

fig 3-2 (page 137 in the book)

The most known digital circuits today is listed down under:

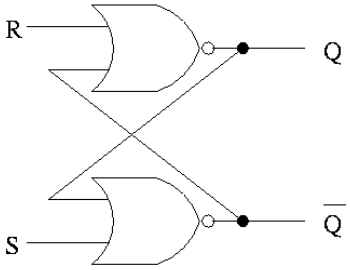
- Multiplexers (more input less output)
- Demultiplexers (less input more output)
- Encoders and decoders
- Comparators (compares to inputs)
- Arithmetic (Shifter and adders)
- Latches (SRLatch)
- Flip-flops (memory)

Latches

Example of gates could be latches. Figures 3-22, 3-23, 3-24 pages 159-161.

SR-latch

A latch is circuit, which can “remember” previous values – it’s memory. There exist three types of latches: SR Latch, which has two inputs, S and R, used for setting and resetting the latch, a Clocked SR latch, which has a clock, and a D latch. SR latches (clocked or not) can be problematic because they have inconsistent states (e.g. when both inputs are 1). The D latch does not have this problem.



SR latch operation		
S	**R**	**Action**
0	0	Keep state
0	1	Q = 0
1	0	Q = 1
1	1	Restricted combination

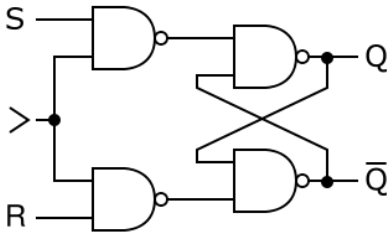
fig. 3-23 (a) +(b) (page 159 in the book)

An SR Latch can be set and reset. When set Q = 1, and when set Q = 0. To set the latch S = 1, and to reset the latch R = 1.

The outputs of a latch are determined by both the inputs and the outputs. If S, R and Q are 0 then $\bar{}Q$ is also 0 (check by going through fig. 3-22). If Q = 1 and R = S = 0, $\bar{}Q = 0$. Setting and resetting can be proven by trying to use S = 1 and R = 1 respectively.

SR Clocked-latch

To be able to control when a latch shall change state a clock (does not to be controlled by an actual clock) can be added – we will get a clocked SR Latch. If the clock is 0 the inputs into the NOR gates will be 0 and the state will not be changes. When the clock is 1, S and R matters and the latch works as before. See figure 3-23.

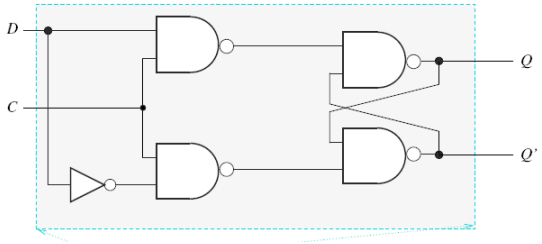


Gated SR latch operation	
E	**Action**
0	No action (keep state)
1	The same as non-clocked SR latch

The problem with an SR Latch is when both inputs are 1. Then it is not possible to determine what happens. If both inputs are 1 both outputs need to be 0. When the inputs return to 0, the latch has to be set or reset. Depending on which input that goes to 0 first the latch is either set or reset. If they both go to 0 simultaneously the state is picked at random – this is very unlikely.

D-latch

To avoid the problem with both inputs being 1 we can use a clocked D Latch. It is basically a clocked SR Latch, where one of the inputs is substituted by a NOT gate – see figure 3-24. In that way the two inputs are never 1 at the same time.



D-latch truth table			
E/C	**D**	**Q_{next}**	**Comment**
0	X	Q_{prev}	No change
1	0	0	Reset
1	1	1	Set

A clocked D Latch is a true 1-bit memory. The value of D is stored and available at Q. the latch is set when D = 1 and reset when D = 0.

Flip-flop

A flip-flop is kind of an expanded D latch. A flip-flop is triggered on an edge rather on a level – a D latch is triggered when the clock is 1, a flip-flop is triggered when the clock is going from 0 to 1 (or 1 to 0).

Figures 3-25 and 3-26 (page 162-163 in the book).

The AND gate should never output 1 since the two inputs are always different. However the NOT gate delays the b signal, and therefore the AND gate can output 1 for a short period of time setting d = 1. That way a signal can be fed into a D latch on the rising edge of the clock (or any other time).

2M Micro-Architecture

Explain the components that make up a micro architecture.

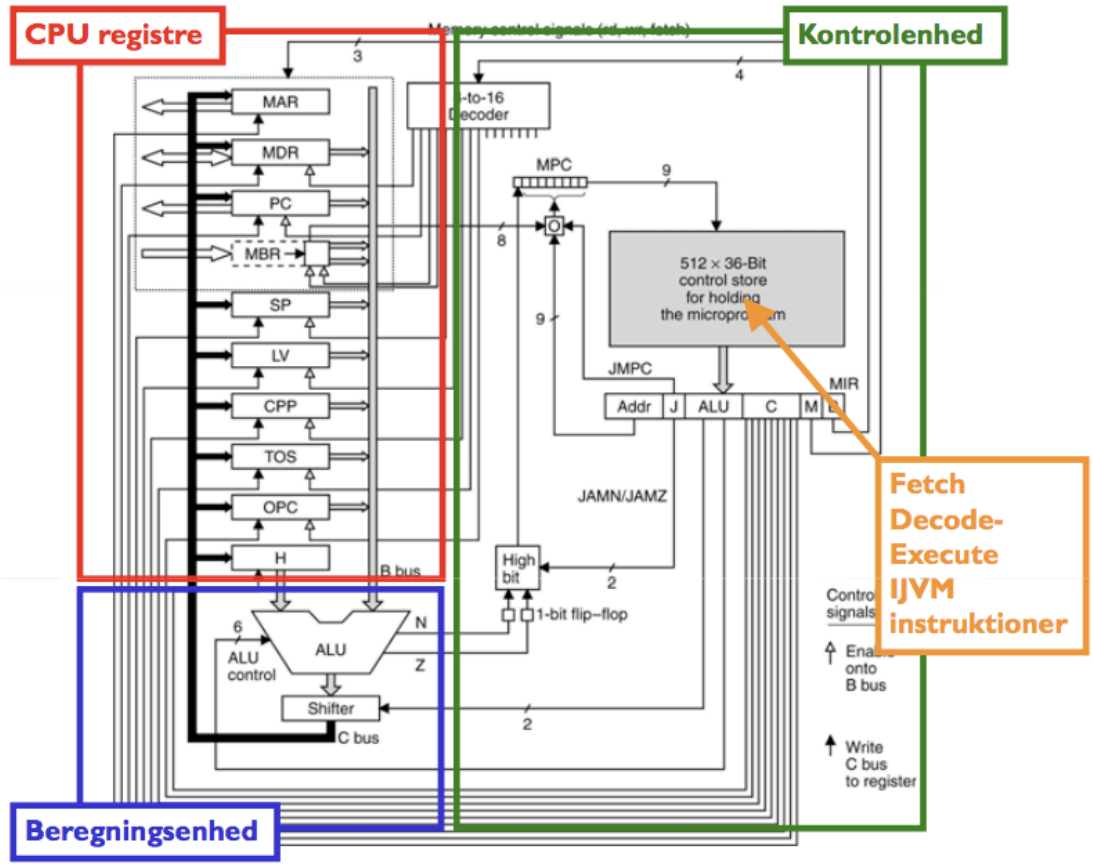


fig. 4-16 (page 242 in the book)

If we have to take a look on a micro architecture, we see on Mic1. It is splitted up in three parts:

- CPU registers (Data path)
- Calculating unit (Data path)
- Control unit

CPU registers

MAR (Memory Address Register)

The register holding the memory address to read or write

MDR (Memory Data Register)

Contains the data to be read or written to memory

PC (Program Counter)

Points to the next instruction to be executed

MBR (Memory Buffer Register)

A buffer that is not as flexible as MDR, memory can't read from it, but only write to it. As seen by the control signal (small arrows under the registers). MBR has no black arrow, which enables it on the C-bus, only white arrow to enable it on the B-bus.

SP (Stack Pointer)

Indicates where we are in the stack, a part of memory reserved to stack.

LV (Local Variable pointer)

Is pointing on the bottom element in the stack, compared to the procedure.

CPP (Constant Pool Pointer)

Is pointing on the bottom of the constant pool stock.

TOS (Top Of Stack register)

A register over what is in the top of the stack.

OPC (OPCode register)

"Operation Code register". Contains those operations, which is available.

H (Holding register)

Contains the data for the ALU input, the only way to use both inputs on the ALU.

Calculating Unit

Busses

A-bus:

Connection between H register an the ALU, also called H-bus

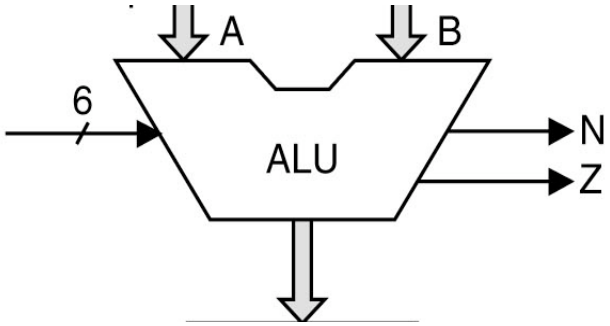
B-bus

Connection between the CPU registers one input on the ALU, only one register ca accesses the bus at a time.

C-bus

Connection between the CPU registers and the output on the ALU through the shifter.

ALU



- F0 and F1 controls how the ALU calculates
- ENA and ENB controls if the A-bus and B-bus should be connected to the ALU
- INC adds 1 to the result
- Function: the Boolean expression for the result

The ALU is controlled by the 6-input line (F0, F1, ENA, ENB, INVA and INC)

When the ALU has done one calculation N and Z will be set. If the result is negative, N will be set to 1, and Z to 0.

Shifter

The calculation unit has a shifter, which has some functions as the ALU has, but it can also replace the result from the ALU back to the registers with help from the C-bus. It has to functions, SSL8 and SRA1.

SSL8 (Shift Left Logical): It shifts all one bit to the left, and replace the 8 most un significant bits out with 0, while it double up the value of the number.

SRA1 (Shift Right Arithmetic): It shifts all one bit to the right, and let the most significant bit stay untouched, and reduce the value of the number by half.

Control Unit

To control the control signals, it has to use a “sequencer” to enable and disable which operations there has to run for en ISA-instruction. In each cycle it has these functions.

- Know what each control signal does.
- The address for the next instruction.

The control unit has a “control store”, which include the micro program. It has to registers **MPC** and **MIR**.

MPC contains information about the address for the next microinstruction and **MIR** contains information about the present microinstruction.

MIR

The MIR contains 6 elements. ADDR, J, ALU, C, M and B.

ADDR and **J** manage the next microinstruction.

ALU contains 8-bit which manage the ALU function and shifter.

C contains the different registers ALU output on the C-bus.

M manages the memory operations.

B manages what the decoder shall put on the B-bus.

MPC

MPC can only have 2

1. The value of the NEXT_ADDRESS
2. The value of the NEXT_ADDRESS with a high-order bit OR’ed with 1.

ROM

The words which comes, get down in the long list in the ROM.

The different control words, defines them self, after which is after which. Sometimes is it though able to manipulate the top bit.
The adress is typical about 8 bit long, it is just able to jump to the bottom part of the control store.

3M Micro-Architecture

Explain the structure and function of a microinstruction.

A microinstruction controls **2 things**:

1. *How all the control signals will be set in the clock cycle.*
2. *The address on the next microinstruction.*

If we look at an example in Mic1, we have **29 signals.**

- 9 signals, which controls which of them, is going to be used on the B-bus, and through the ALU.
- 8 signals, which control the ALU and the shifter, and 9 signals that indicates the registers the output of the ALU will be written on.
- 2 signals that indicates the memory read/write through MAR and MDR (those are just not shown on the example).
- 1 signal that indicates the memory fetch through PC/MBR (not shown on the example).

The B-bus is only able to take 1 input at a time from a register. Therefore we want to reduce the amount of bits, which has to be used to control the B-bus. We do this by encoding the information about which registers there will be used. It is possible to do it by 4 bits instead of the 9 bits that should have been used.

Example

We have to send 8th bit out on the B-bus, can we use the binary (1000) to send it out to the bus, instead of use the 9 bits. A 4-16 decoder is later able to translate the 4 bits to the 16 bit, where only the one out of the 4 bit are set to 1. If we only had 8 registers, would it be easy to use 3 bits, while we can use them to show the numbers from 0-7.

We use a 4-16 decoder, where it is only 9 of the pins that are being used, and all of the registers signalized. Because of the reduce of amount of bits, we now in the Mic1 have 24 bits, but we still don't know where we have to get the next microinstruction from, and therefore we need to have more information about it. We use the JAM, which is used to give information about the NEXT_ADDRESS. In the book on page 240 in fig. 4-5 is shown how the microinstruction is divided up 6 six groups in Mic1.

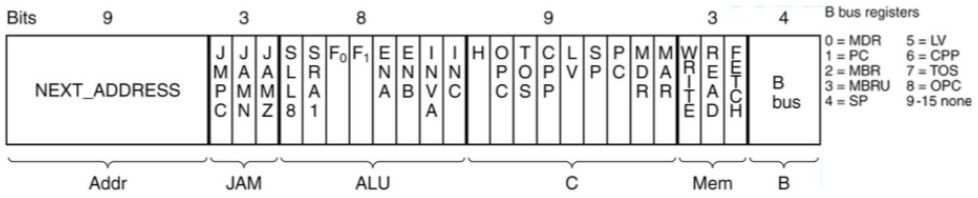


fig. 4-5 (page 240 in the book)

Addr – Contains the address of a potential next microinstruction.

Jam – Determines how the next microinstruction is selected.

ALU – ALU and shifter functions.

C – Selects which registers are written from the C-bus.

Mem – Memory functions.

B – Selects the B bus source; it is encoded as shown.

Addr (NEXT_ADDRESS)

The Addr is 9 bits wide and gives 512 possible instructions. Not all microinstructions contains a NEXT_ADDRESS, if it is needed for the instruction. Some instructions use JAMZ/JAMN/JMPC for determine the address for the next microinstruction.

JAM (JMPC, JAMN and JAMZ)

The JAM bits control what microinstruction is called next. If all bits are 0 the address for the next microinstruction is simply the one in Addr group.

The MPC (microprogram counter) is the address register witch holds the address for the next microinstruction. JAM bits controls the high-bit in that register.

If JAMZ is high the Z flip-flop (high if ALU outputs zero as result) is OR'ed with high-bit of MPC.

If JAMN is high the N flip-flop (high if ALU outputs negative result) is OR'ed with high-bit of MPC.

If JMPC is high

C and B, Addressing for the buses

Group C controls the registers that will read from the C-bus. As there is a need for loading several registers at the same time 9 bits are needed.

Group B controls the B bus. Only 4 bits is needed as the B-bus only used for reading the registers, so there is no need for enabling more registers at a time. We have 8 registers to control, and MBR has 2 control lines, so 9 options are needed therefore 4 bits (16 options) are used. To convert the 4 bits address to the 9 control signals a 4-16 decoder is used.

Data path timing

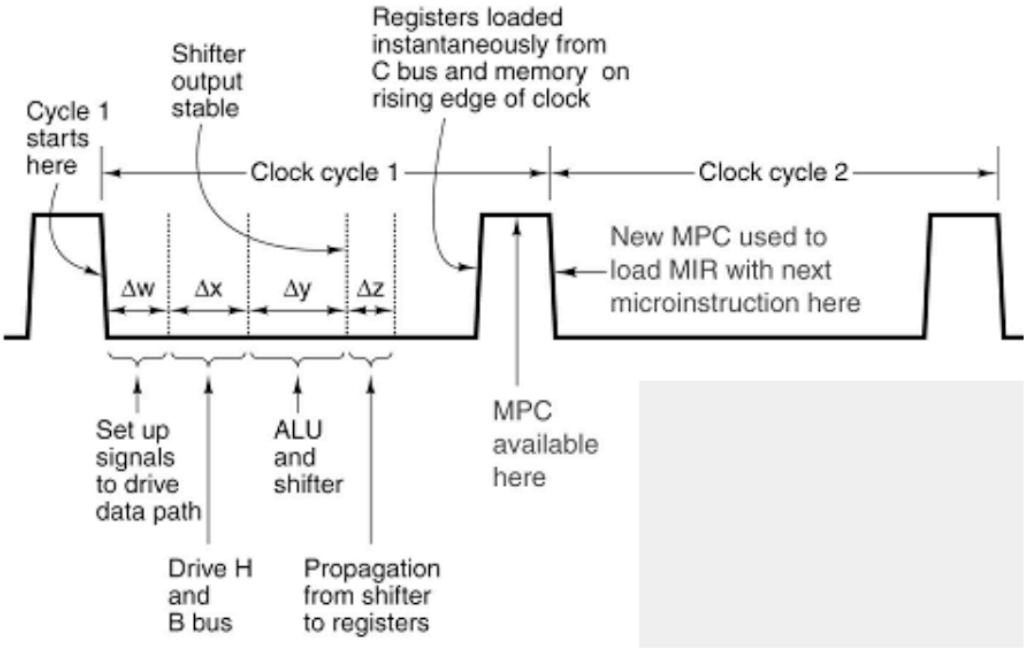


fig. 4-3 (page 236 in the book)

First of all the new instruction of the control store is being loaded, and then the control signals is sent out. The ALU is being loaded and it is calculating and the result is being writing to the registers and the next address of the instruction is being calculated.

T0

To the time of T0, the MPC is pointing on a special address in the 512 lines of code.

Δw

The signals are out from the micro program, which forwards them. It set up the signals, after which ones who has to be calculated, and the control signals to the shifter. The address field on 9 bits does all have information about the next address in the 512 lines of code.

Δx

The registers are now loaded onto the B bus and stay stabile.

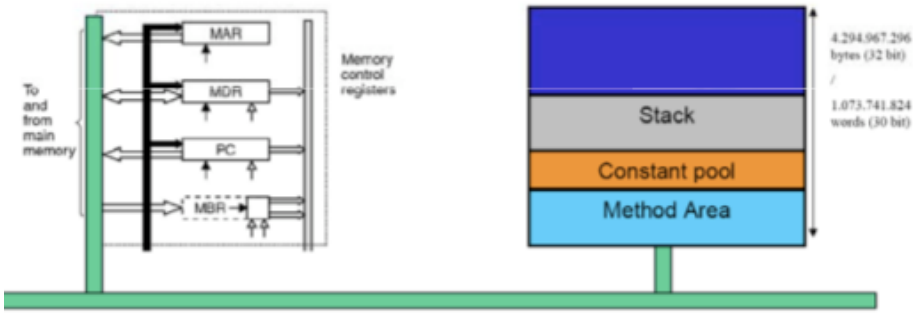
Δy

The ALU and shifter operate and the result is being sent to the C-bus.

Δz

The results propagate along the C bus back to the registers. It all happens when the MPC is high, and are ready to get the result from the C bus. On this time, it is ready to be loaded, and there will be a new value on MPC, and ready for the next calculation.

Stock replenishment



The accession of to the stock, are through the 4 registers, MAR, MDR, PC and MBR.

On the top we have the amount of bytes that are available. The PC and MBR are used to read the method area, by using the byte addresses. MAR and MDR is used to read the stack and constant pool by using word addresses. Timing: If the PC (Mar) is set in the cycle contains MBR (MDR) contains the stock cell cycle k+2

4M Micro-Architecture.

Explain the concept behind pipelining and how it is implemented.

The concept behind pipelining is to run functions sequential and the concept of it, can been seen on figure 4-34 (page 287 in the book). The idea is to have the opportunity to run the functions more efficient, by making the components run most of the time. It is made for not wasting time of waiting, and pipelining is therefore today used for optimizing and "paralyzing", where more instructions run on the same time.

Pros

- Processor cycle time will be reduced.
- Sometimes it means more bandwidth.

Cons

- There can be a waiting time on branching, because it has to wait for fulfilling the needed instructions.
- By use of more flip-flops, it increases the latency in a pipelined processor.
- Because of the waiting time, it can be difficult to know how it performs.

Look at figure 2-4

9 clock cycles are being used.

Superscalar architectures

Dual pipeline, (fig. 2-5) –

To be able to run in parallel the two instructions must not conflict over resource, and neither must depend on the result of the other.

These are most used in RISC machines. Original Pentium had as shown on the figure.

Fig. 2-6 – Superscalar is used to describe processors that issue multiple instructions often 4-6 six in a single clock cycle.

It is possible to have multiple ALU's in stage 4.

Mic3

An example on pipelining is available in the Mic3, where it is the extension of the Mic2, where the IFU was introduced, and the extension of the three latches in the A, B and C busses.

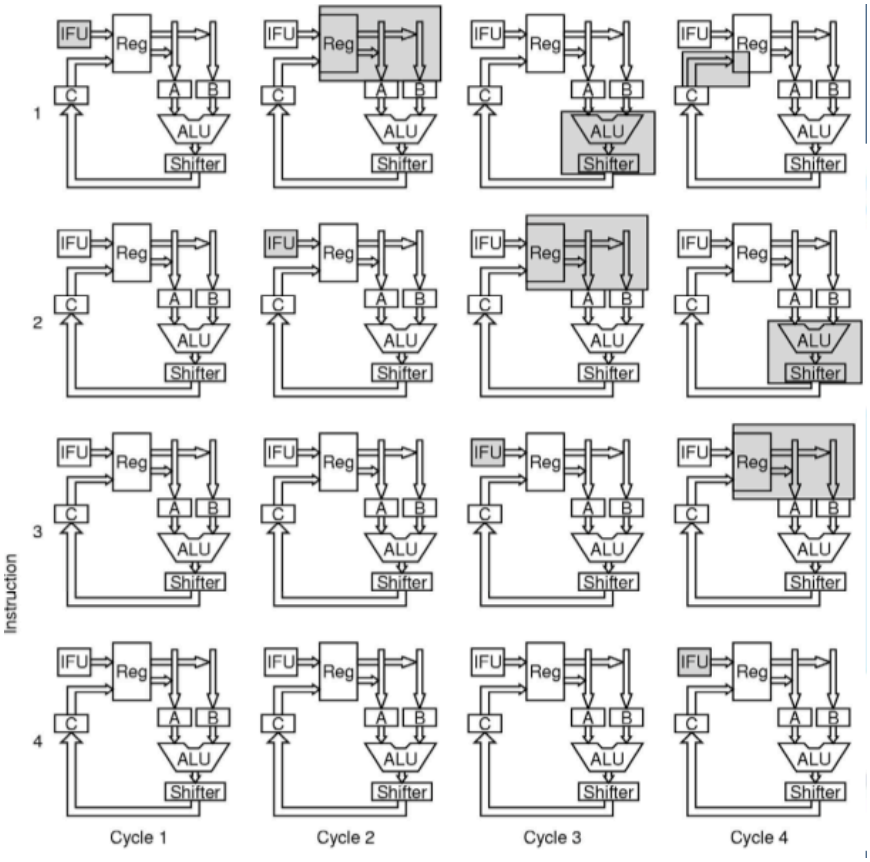


fig. 4-34 (page 287 in the book)

In Mic2, did one instruction take about 1 cycle, and in Mic3, an instruction take up to 3 cycles, because the frequency of the clock is higher while the max delay is shorter, and we are therefore able to use all parts of the data path in each cycle. On the figure there is shown 4 cycles, where the

1st cycle The IFU is working on it.

2nd cycle selects the registers, which has to be on the A- and B-bus (saves in A- and B-latch).

3rd cycle selects the registers and the ALU and shifters I set, and it saves in the C-latch.

4th cycle does send the result back to the registers to be saved.

The more new extensions about the new scalar, where there is being used more ALU'es (use of floating point, a load and a store).

An analogy

The production of a car on a car factory, with the assembly line. After the 4th cycle, there will be produced a car, when every cycle ends.

5M ISA-layer

Explain a typical structure of an ISA instruction and how it is executed on a microcomputer.

An ISA instruction (Instruction Set Architecture) is on level 2, opposite of the Microinstructions which is on level 1, is it now able to access smaller set of registers the machine has. An example can be the PC or SP, which is being used to saving results and addresses in. PSW (Program Status Word), contains information about the ALU:

- N is 1, if the result is negative.
- Z is 1, if the result is 0.
- V is 1, if the result gave an overflow.
- C is 1, if the result gave a carry out on leftmost bit.
- A is 1, if the result gave a carry out on 3rd bit.
- P is 1, if the result is the same.

An ISA instruction consists of an opcode and 0-3 addresses. An opcode is the function, and examples on these functions is ILOAD, ISTORE or IADD, which all are opcodes in IJVM, which is the ISA which is used in the Mic machine.

IJVM instruction set

We look at the IJVM instruction set, where the operands byte, const and varnum are 1 byte each, and the operands disp, index, and offset are 2 bytes.

To go from Java to binary, do we convert many times, where we first start to convert from Java to Java assembly (Mnemonic), then to hexadecimal.

The figure shows the instruction set for the IJVM, where the first column shows the HEX code, and the next column shows the Mnemonic (Java assembly code) of the instruction. The 3rd column shows the meaning of the code of the Hex and Mnemonic code.

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Fig 4-11 (page 250 in the book)

If we have to show how the translation is going on, we can see at the example on 4-14, where we can se the translation from a) a java fragment, to b) a java assembly code to the final c) IJVM in hexadecimal.

```
i = j + k;                1      ILOAD j          // i = j + k      0x15 0x02
if (i == 3)               2      ILOAD k                         0x15 0x03
   k = 0;                 3      IADD                            0x60
else                      4      ISTORE i                        0x36 0x01
   j = j – 1;             5      ILOAD i          // if (i == 3)    0x15 0x01
                          6      BIPUSH 3                        0x10 0x03
                          7      IF_ICMPEQ L1                    0x9F 0x00 0x0D
                          8      ILOAD j          // j = j – 1      0x15 0x02
                          9      BIPUSH 1                        0x10 0x01
                         10      ISUB                            0x64
                         11      ISTORE j                        0x36 0x02
                         12      GOTO L2                         0xA7 0x00 0x07
                         13  L1: BIPUSH 0         // k = 0          0x10 0x00
                         14      ISTORE k                        0x36 0x03
                         15  L2:
          a)                          (b)                              (c)
```

fig. 4-14 (page 254 in the book)

An example of the Java fragmentation can we see on the figure 4-15 from the first line 0, to the 15th line.

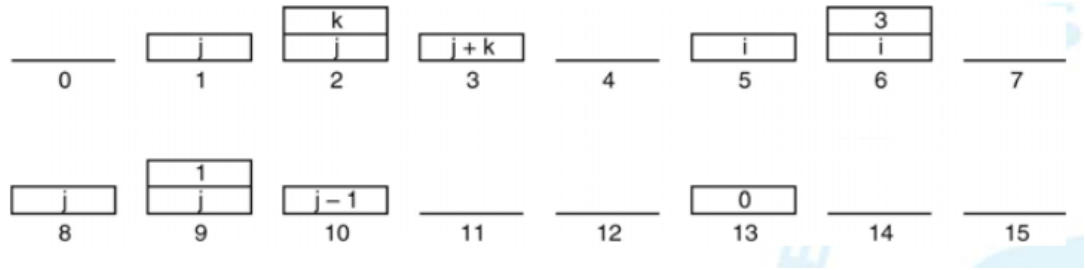


fig. 4-15 (page 255 in the book)

Look at page 262 (fig 4-17), and 282 (fig 4-30) for the instruction set for the Mic1 and Mic2.

ISA instruction to microinstructions

An example is on the ILOAD k, where it is compiled from 1 ISA instruction to more microinstructions. First of all the value of J is loaded from the memory and then pushed on the stack. Look at fig. 4-17 (page 263 in the book).

```
iload1    H = LV                    MBR contains index; copy LV to H
iload2    MAR = MBRU + H; rd        MAR = address of local variable to push
iload3    MAR = SP = SP + 1         SP points to new top of stack; prepare write
iload4    PC = PC + 1; fetch; wr    Inc PC; get next opcode; write top of stack
iload5    TOS = MDR; goto Main1     Update TOS
```

After j and k is pushed on the stack, the IADD is being interpreted opcode of Mic1. Look at fig. 4-17 (page 263 in the book).

```
iadd1     MAR = SP = SP – 1; rd     Read in next-to-top word on stack
iadd2     H = TOS                   H = top of stack
iadd3     MDR = TOS = MDR + H; wr; goto Main1   Add top two words; write to top of stack
```

Expanding opcode

Short instructions are **better** than long ones. A program consisting of *n* 16-bit instructions takes up only half as much memory space as *n* 32-bit instructions.

Furthermore, minimizing the size of the instructions may make them **harder to decode** or **harder to overlap**. Therefore, achieving the minimum instruction size must be weighed against the time required to decode and execute the instructions. Another reason **minimizing the instructions** length is with **today's fast processors** needs **larger memory bandwidth** (number of bit/sec the memory can supply). This is a bottleneck problem.

One level 2 instructions considers of one opcode telling what the instruction does, There can be **zero, one, two** or **three** addresses present. The addressing specifies where the operands came from, and where the result goes.

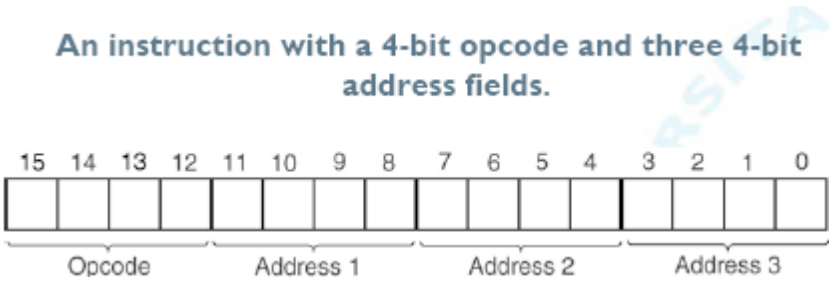


fig. 5-12 (page 355 in the book)

For most machines the instructions have the same length, this makes it simpler and easier to decode them, but often waste space, since all instructions have to be as long as the longest one. Look at figure 5-11 on page 352.

The concept of an expanding opcode can be most clearly seen by this simple example.

Consider a machine in which instructions are 16 bits long and addresses are 4 bits long. This might be okay for a machine that has 16 registers (4-bit register address instruction) on which all arithmetic operations take place. (Figure 5-12) One design would be a 4-bit opcode and tree addresses in each instruction, giving 16 tree-address instructions.

However, if the designer needs (**figure 5-13**):

- 15 three-address instructions
- 14 two-address instructions
- 31 one-address instructions
- 16 instructions with no address at all.

They can use opcode 0-14 as tree-address instructions but interpret opcode 15 differently. On this figure it possible to see the tradeoff between the opcode and the addresses.

1C RISC vs. CISC

Define and explain the difference between RISC and CISC architectures.

CISC

Complex Instruction Set Computer.

Many addressing modes, many operations. Instructions need to be interpretive.

Set of more complex instructions may be broken into separate parts, which can then be executed as a sequence of microinstructions. This extra steo skis down the machine.

RISC

Reduced Instruction Set Computer.

Few instructions. Instructions **don't need to** be interpretive. So shorter executing time per instruction, but more instructions needed, in contrast to CISC.

All instructions can be executed in **one data path cycle**.

Pipelining is easier, as every instruction is executed in one data path cycle.

Hybrid

A lot of today's CPU is hybrid, as the complex instructions are interpretive as in CICS and simple instructions are executed directed.

2C Micro Architecture

Explain the structure and function of an Instruction Fetch Unit.

Each instruction consists of number of steps. Some of these steps do not require the full "power" of the ALU. Therefore it can be useful to add other paths, which do not pass through the ALU. The IFU is used to increment/adjust the PC. The IFU is used because it is a waste to use the ALU to perform such a simple task.

The IFU can interpret the opcode and can figure out how many fields must be fetched. Therefore it can assemble the fields into a register that can be used by the main execution unit.

The structure is simpler than an ALU, therefore it is a better solution than using a second ALU.

3C CPU Architecture

Explain the structure and function of a data path.

The data path is the part of the CPU where data is stored and computed. It consists of CPU registers, ALU, shifter and the buses (A, B and C).

CPU registers

MAR (Memory Address Register)

The register holding the memory address to read or write

MDR (Memory Data Register)

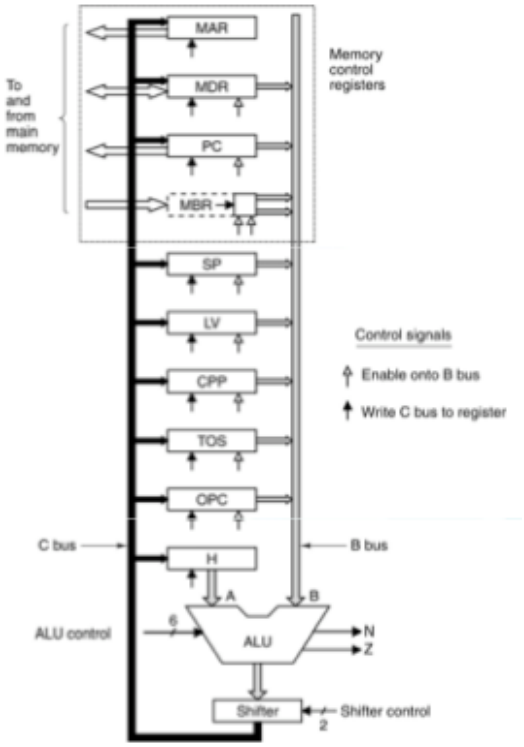
Contains the data to be read or written to memory

PC (Program Counter)

Points to the next instruction to be executed

MBR (Memory Buffer Register)

A buffer that is not as flexible as MDR, memory can't read from it, but only write to it. As seen by the control signal (small arrows under the registers). MBR has no black arrow that enables it on the C-bus, only white arrow to enable it on the B-bus.



SP (Stack Pointer)

Indicates where we are in the stack, a part of memory reserved to stack.

LV (Local Variable pointer)

Peger på det nederst element i stacken i forhold til proceduren.

CPP (Constant Pool Pointer)

(Dodge it)

TOS (Top Of Stack register)

Et register over hvad der er i toppen af stakken.

OPC (OPCode register)

"Operation Code register". Indeholder de operationer der er til rådighed.

H (Holding register)

Contains the data for the ALU input, the only way to use both inputs on the ALU.

Busses

A-bus:

Connection between H register an the ALU, also called H-bus

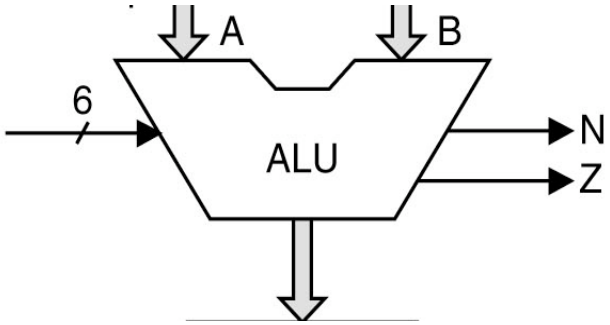
B-bus

Connection between the CPU registers one input on the ALU, only one register ca accesses the bus at a time.

C-bus

Connection between the CPU registers and the output on the ALU through the shifter.

ALU



- F0 and F1 controls how the ALU calculates
- ENA and ENB controls if the A-bus and B-bus should be connected to the ALU
- INC adds 1 to the result
- Function: the Boolean expression for the result

The ALU is controlled by the 6-input line (F0, F1, ENA, ENB, INVA and INC)

4C Cache-Memory

Explain the principle of Cache memory

Is a **small amount** of **very fast memory** combined with ordinary memory, the **words most heavily** used are stored in the cache, when the CPU needs the word it first look in the cache. If a substantial fraction of the words are in the cache, the average access time can be greatly reduced.

Fig 2-16. -

Main memory and cache are divided up into fixed-size blocks. Commonly referred to as cache lines. When a cache miss occurs, the entire cache line is loaded from main memory into the cache, not just the word needed. Example, whit a 64-byte line size, a reference to memory address 260 will pull the line consisting of bytes 256 to 319 into cache line. With a little luck, some of the words will be needed shortly. This operating is faster because it's faster to fetch k words all at once, than one-word k times.

Issue:

- The bigger cache, the better it performs, but also the more it cost.
- Size of the cache line. A 16 KB cache can be divided up into 1024 lines of 16 bytes, or 2048 lines of 8 bytes, and other combinations.
- Whether instructions and data are kept in the same cache or in different ones. Having a unified cache is a simpler design and automatically balances instruction fetches and data fetches.

Unified cache: Data and instructions in one cache.

Splitted cache: Data and instructions en each cache. (Harvard architecture)

5C Bus-Architecture

Explain the use of latch A, B and C in the MIC-3 machine.

Figure 4-31, page 284

The latches in the MIC-3 machine are used to speed up the machine and making it more efficient. The data path is split into three parts. That means that every part of the data path can be used in every cycle. The data path requires three cycles to complete, but because the path is split up the max delay is smaller and the clock speed higher.

Each piece of the data path is called a micro step. If a micro step has to wait for a previous step to read a register (the previous step has to write the register) it is called stalling. There is RAW (Read After Write) Dependence between the micro steps. See figure 4-33 on page 286

The use of latches is called pipelining and is used in all modern CPUs. Pipelining is illustrated on figure 4-34.

6C Memory hierarchy

Explain the concept behind the memory hierarchy:

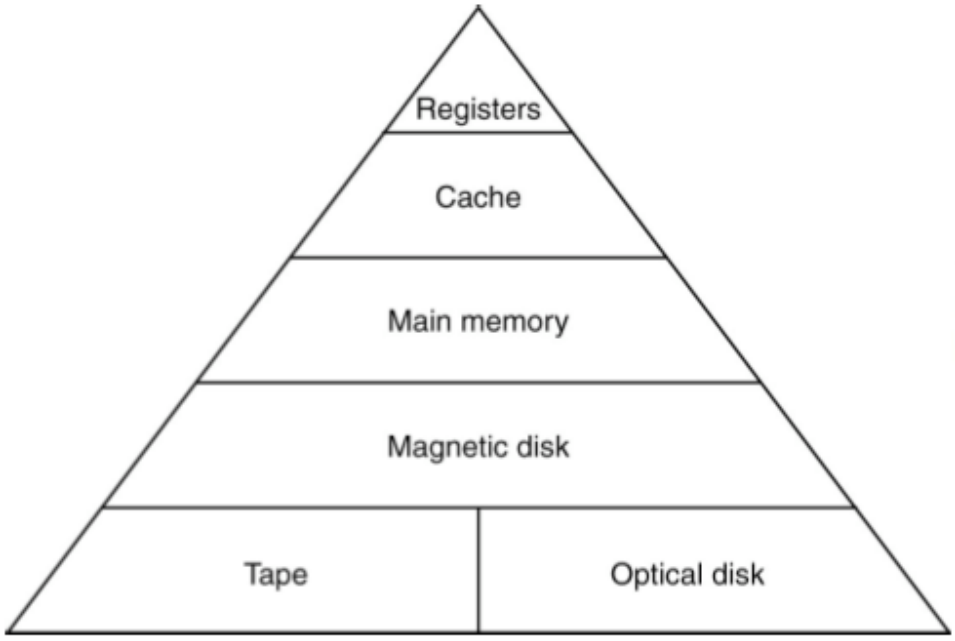


fig. 2-18 (page 81-82 in the book)

The memory has a special hierarchy, which is illustrated in fig. 2-18.
It contains the following 5 levels:

- CPU-registers (cpu full speed)
- Cache, (32kb to few megabytes)
- Main memory (from 16mb to tens of gigabytes)
- Magnetic disk
- Tapes and optical disks.

The memory hierarchy is the traditional solution to store a great deal of data, and is based on three important key parameters:

1) The access **time gets bigger** as we move down the hierarchy

2) The storage **capacity increases** as we go downward.

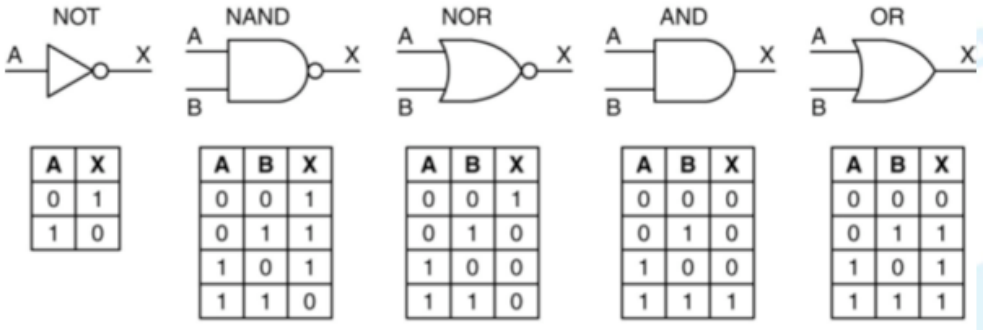
   (**1 + 2** = the memory **becomes larger and slower**)

3) The number of bits you get per dollar kroner spent increases down the hierarchy. Ex. CPU registers are good for perhaps 128 bytes and are more expensive than for example the main memory, with sizes ranging from 16 MB to gigabytes.

Because the smaller memories, as registers and cache, are more expensive and faster, we want to use smaller memories to try to hold the most recently accessed items close to the registers, and larger (and slower and less expensive) memories as we move away from the CPU.

7C Digital logic

Define and explain the fundamental Boolean operations.



A	X
0	1
1	0

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

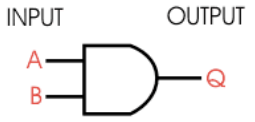
fig. 3-2 (page 137 in the book)

A range of logic gates exists and they are represented as symbols, each with its own truth table (sometimes called a logic table). Gates have inputs and produce outputs and these are in the form of **1s** and **0s**. Remember, a **1** represents an **input or output** of electrical current. Each truth table clearly shows the 'state' of inputs and outputs at any one time.

The **AND** gate will only output current (produce a 1 at Q) if both logic states at inputs A and B change to 1.

AND gate

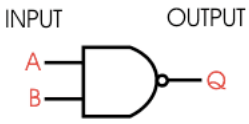
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



The **NAND** gate has the opposite outputs to the AND gate.

NAND gate

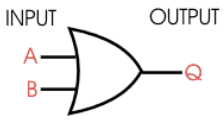
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



The **OR** gate will output current at Q if either of the logic states of inputs A and B change to1.

OR gate

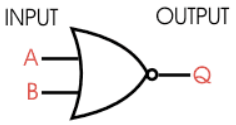
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1



The **NOR** gate has the opposite outputs to the OR gate.

NOR gate

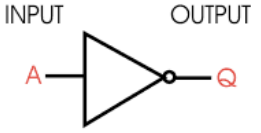
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



The **INVERTER** gate reverses input. For example, if the input is 1 then the output is 0. This is a very useful gate especially when designing logic circuits.

INVERTER gate

A	Q
0	1
1	0



8C Von Neumann vs. Harvard

Explain the structure of a classical Von Neumann machine.

(A short paper about the Von Neumann machine: http://www.pcmech.com/article/von-neumann-architecture/)

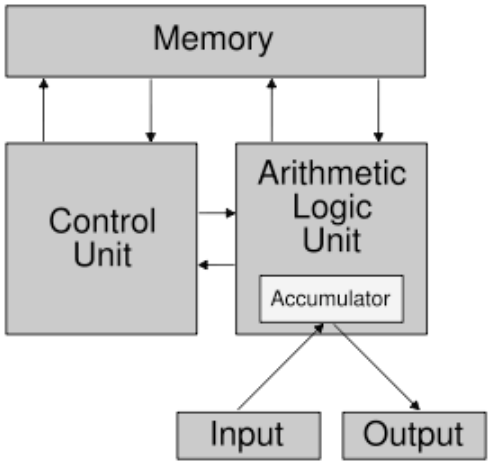


Fig. 1-5 (page 19 in the book)

The classic design is based on 4 parts:

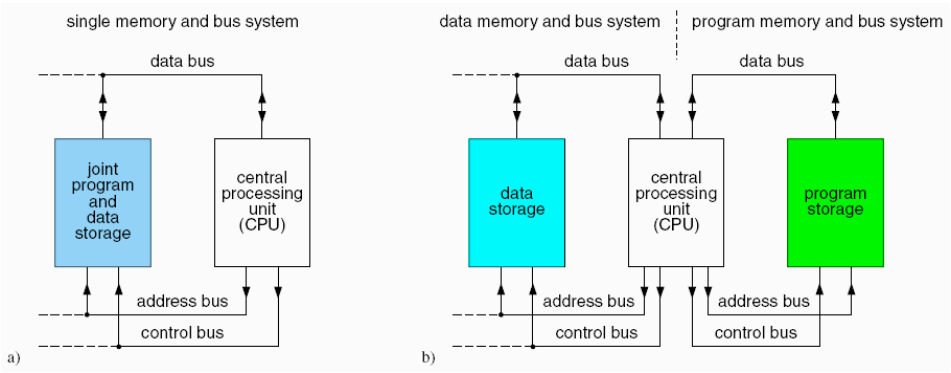
- **ALU**
  Calculation unit and internal register (Accumulator) (today its part of the CPU)

- **Control Unit**
  Responsible for fetching instructions from memory, decode the instructions, control the ALU for execution. (Today its part of the CPU)

- **Memory**
  Well exactly ;-)

- **Input/output**
  Communication to external devices, like hard disks etc.

Von Neumann has only **one memory (and bus)** for both data storage and program storage, as opposite **of Harvard** which **split the memory.**



Von Neumann design has a **bottleneck problem** because of the single memory design. It **can't** both **fetch instructions** and **data** at the **same time.** To **reduce that problem** a solution is to add **fast cache memory**. A lot of today's computers is using Von Neumann architecture for the external memory, and is using Harvard architecture for its internal cache memory.

9C Instruction set for the IJVM

Explain the translation of Java to binary code.

I will start with looking at the example seen on 4.11, where it's shown how the translating of Java to binary is.

We look at the IJVM instruction set, where the operands byte, const and varnum are 1 byte each, and the operands disp, index, and offset are 2 bytes.

To go from Java to binary, do we convert many times, where we first start to convert from Java to Java assembly (Mnemonic), then to hexadecimal.

The figure shows the instruction set for the IJVM, where the first column shows the HEX code, and the next column shows the Mnemonic (Java assembly code) of the instruction. The 3rd column shows the meaning of the code of the Hex and Mnemonic code.

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Fig 4-11 (page 250 in the book)

If we have to show how the translation is going on, we can see at the example on 4-14, where we can se the translation from a) a java fragment, to b) a java assembly code to the final c) IJVM in hexadecimal.

```
i = j + k;                1     ILOAD j          // i = j + k      0x15 0x02
if (i == 3)               2     ILOAD k                         0x15 0x03
    k = 0;                3     IADD                            0x60
else                      4     ISTORE i                        0x36 0x01
    j = j – 1;            5     ILOAD i          // if (i == 3)  0x15 0x01
                          6     BIPUSH 3                        0x10 0x03
                          7     IF_ICMPEQ L1                    0x9F 0x00 0x0D
                          8     ILOAD j          // j = j – 1    0x15 0x02
                          9     BIPUSH 1                        0x10 0x01
                         10     ISUB                            0x64
                         11     ISTORE j                        0x36 0x02
                         12     GOTO L2                         0xA7 0x00 0x07
                         13 L1: BIPUSH 0         // k = 0        0x10 0x00
                         14     ISTORE k                        0x36 0x03
                         15 L2:
        a)                          (b)                              (c)
```

fig. 4-14 (page 254 in the book)

An example of the Java fragmentation can we see on the figure 4-15 from the first line 0, to the 15th line.

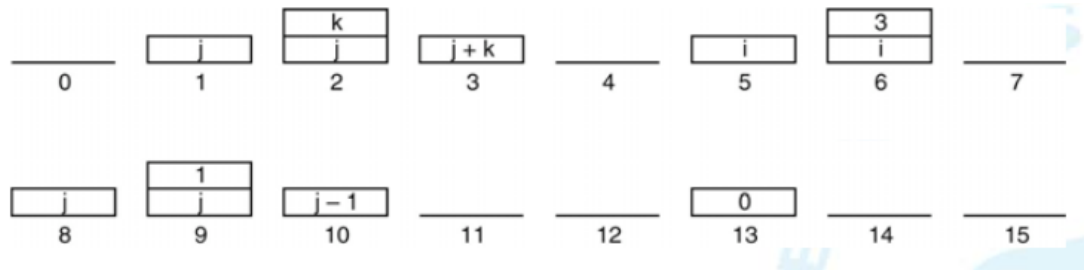


fig. 4-15 (page 255 in the book)

Look at fig. 4-17 (page 262 in the book) and fig. 4-30 (page 282 in the book) for the instruction set for both Mic1 and Mic2.

10C ISA-Layer

Explain the concept behind an Expanding opcode

Short instructions are **better** than long ones. A program consisting of *n* 16-bit instructions takes up only half as much memory space as *n* 32-bit instructions.

Furthermore, minimizing the size of the instructions may make them **harder to decode** or **harder to overlap**. Therefore, achieving the minimum instruction size must be weighed against the time required to decode and execute the instructions. Another reason **minimizing the instructions** length is with **today's fast processors** needs **larger memory bandwidth** (number of bit/sec the memory can supply). This is a bottleneck problem.

One level 2 instructions considers of one opcode telling what the instruction does, There can be **zero, one, two** or **three** addresses present. The addressing specifies where the operands came from, and where the result goes.

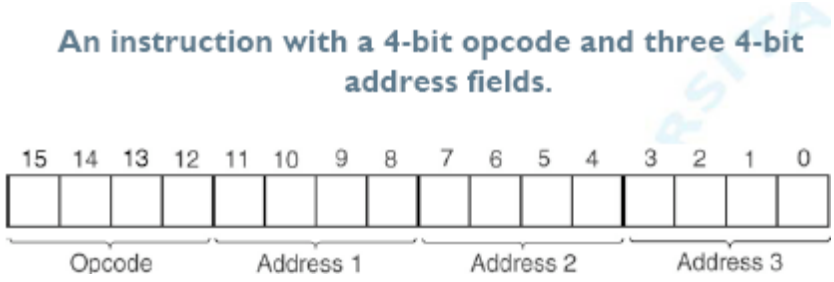


fig. 5-12 (page 355 in the book)

For most machines the instructions have the same length, this makes it simpler and easier to decode them, but often waste space, since all instructions have to be as long as the longest one. Look at figure 5-11 on page 352.

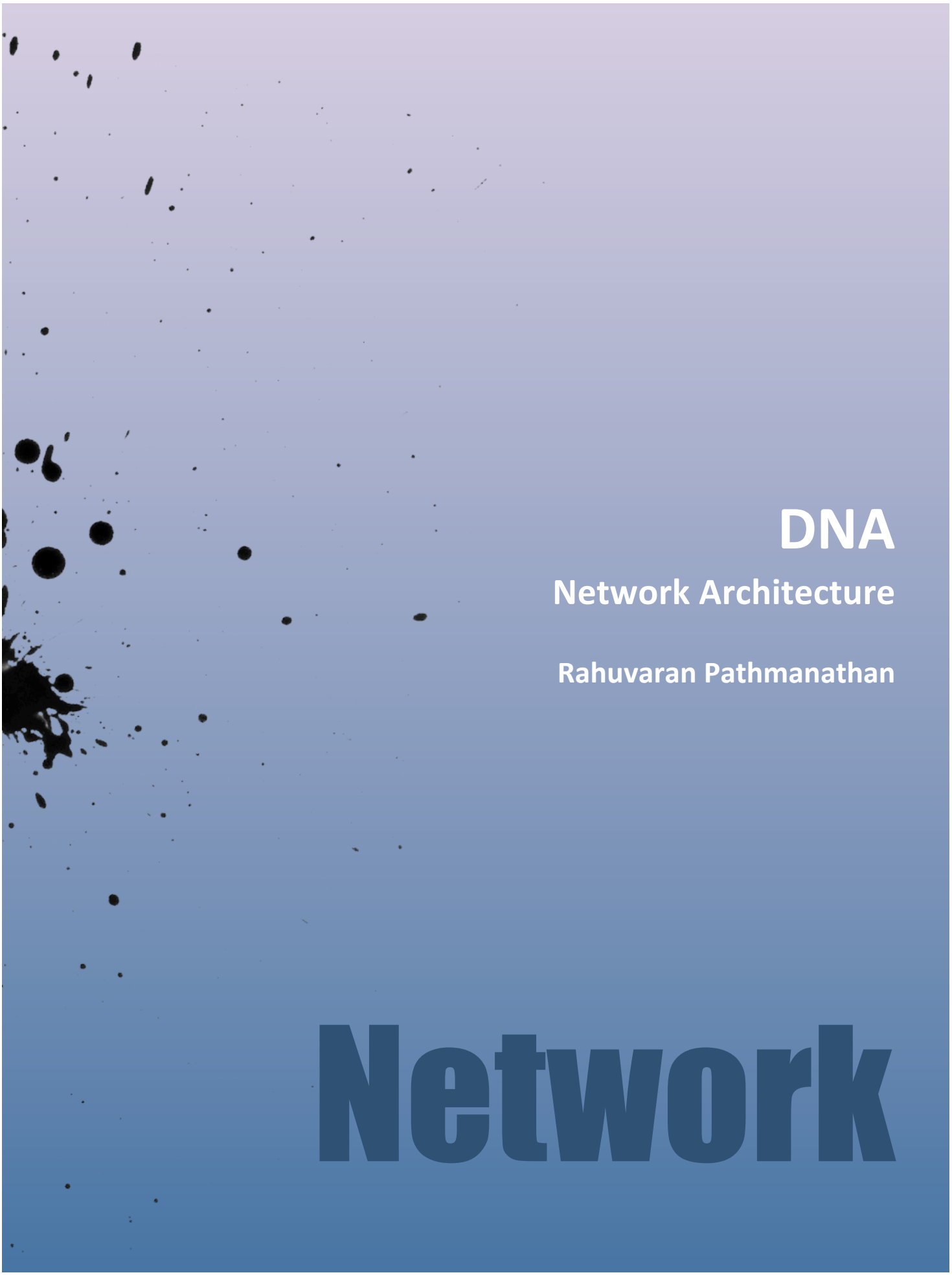
The concept of an expanding opcode can be most clearly seen by this simple example.

Consider a machine in which instructions are 16 bits long and addresses are 4 bits long. This might be okay for a machine that has 16 registers (4-bit register address instruction) on which all arithmetic operations take place. (Figure 5-12) One design would be a 4-bit opcode and tree addresses in each instruction, giving 16 tree-address instructions.

However, if the designer needs (**figure 5-13**):

- 15 three-address instructions
- 14 two-address instructions
- 31 one-address instructions
- 16 instructions with no address at all.

They can use opcode 0-14 as tree-address instructions but interpret opcode 15 differently. On this figure it possible to see the tradeoff between the opcode and the addresses.



DNA

Network Architecture

Rahuvaran Pathmanathan

Network

Table of Contents

1M. The Internet.

Explain the OSI model, how is it related to the Internet Protocol Stack (5 layers).

The Internet protocol stack

Figure 1.19 the Internet protocol stack and OSI reference model.

Each protocol belongs to a layer. Here we are interested in the service that a layer offers to the layer above (**service model**).

Application Layer

- HTTP (Web documentation request and transfer)
- SMTP (transfer of e-mail messages)
- FTP (Transfer of files between two end system)
- DNS (domain name service)

Application-layer protocol is distributed over multiple end systems, with the application in one end system using the protocol to exchange packets of information with the application in another end system. We refer to this packet of information at the application layer as a message.

Transport Layer

Transport application-layer messages between application-layer messages between application endpoints.

- TCP (Provides a connection-oriented service to its applications with flow control and congestion control)
- UDP (Provides a connectionless service to its applications)

Transport-layer packets are referred to as segments.

Network Layer

Responsible for moving network-layer packets known as **datagrams** from one host to another. The Internet transport-layer protocol (TCP or UDP) in a source host passed a transport-layer segment and a destination address to the network layer. The network layer then provides the service of delivering the segment to the transport layer in the destination host.

- IP (defines the field in the datagram as well as how the end system and routers act on these fields. All Internet components that have a network layer must run IP protocol)
- Routing protocols (determine the router that datagram take between source and destinations)

The Internet has many routing protocols. Internet is a network of networks.

Link Layer

The Internet network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node to the next node in the route, the network layer relies on the services of the link layer. The link layer delivers the datagram to the next node along the route. At this next node.

- Link-layer protocol provides reliable delivery, from transmitting node, over one link, to receiving node. (This service is different from the one that TCP provides, which is reliable from one end system to another)

The Link-layer packets are called frames.

Physical layer
While the job of the link layer is to move entire frames from one network element to an adjacent network element, the job of the physical layer is to move the individual bits within the frame from one node to the next.

- The protocols in this layer are again link dependent and further depend on the actual transmission medium of the link.

The OSI model
In the late 1970s, the ISO proposed that computer network be organized around seven layers, called the

Open Systems Interconnection
- **The presentation layer**

  The role of the presentation layer is to provide services that allow communication applications to interpret the meaning of data exchanged. This service includes data compression and data encryption and data description. This frees the application from having to worry about the internal format in which data are stored.
- **The session layer**

  The session layer provides for delimiting and synchronization of data exchange, including the means to build a check pointing and recovery scheme.

If these layers are need in the Internet is all up to the application developer, is it important is up to the developer to build the functionality into the application.

Explain Circuit and Packet switching (pros and cons, how do they differ?)

Fig 1.8 page 25 & fig. 1.10 page 28 in CN v.4

There are two different technologies for making connections and get data sent, when you need to transfer data between 2 computers over a large Internet.

Circuit switching
Circuit switching can be described as a set up between 2 devices. The circuit may either be fixe done that is always present, or it may be a circuit that is created on an as-needed basis. Even if many potential paths through intermediate devices may exist, only one will be used for any given dialog.

In circuit switching you sue either frequency-division multiplexing (FDM) or time-division multiplexing (TDM) – multiplexing is used to enable multiple users on one network. In **FDM** the bandwidth is divided into portions (typically 4 kHz), which are reserved, to each individual – this is also uses by radio stations to share the FM band. In **TDM** time is split into frames of fixed duration. When a connection is made, the network dedicates one slot in every frame to the connection.

An example on the Circuit switching is the phonesystem; where the sender tells the central who he wants to speak to, by pressing the unique number, and the central find its way direct to the receiver.

Pros

- Dedicated connection, where there is no disturbing.
- Almost no delay (might be the cable-transmission delay)

Cons

- It takes some time to establish connection
- When there is someone one the line, it closes for others.
- Unless you have exactly the same type of connection at both ends, so it will wasting time in the big end.

Packet switching

In Packet switching, there isn't any specific path it used for data transfer. Here is the data chopped into small pieces called packets, and sent over the network. Each time a package comes to a router, the router look at it and sends the package forward in the right direction.

An example on a Packet switching, is the postal system, where the packets is received on the post central, which sent the packets out on different routes to the source destination.

Pros

- Optimum use of the bandwidth
- Relative small delays in routers (1 bit delay in theory)
- On faulty equipment, can packets be sent in another way.

Cons

- Complex protocol with high requirements for the equipment.
- Sometimes is retransmission necessary, while the packets disappear in congested unit.

2M. In the Internet Protocol Stack, what does the Application Layer do?

Explain any 2 of the following protocols, HTTP, FTP, SMTP/POP3/IMAP, and DNS

Fig 2.7 page 100 fig. 2.14 page 114 in CN v.4
The application is the topmost layer. The application layer is where network applications and their application-layer protocols are. Examples of Internet application-layer protocols are HTTP, FTP and DNS. Application-layer protocols are often implemented in software (protocols can be implemented in both hardware and software).

HTTP

Explanation of **HTTP**: The Hyper Text Transfer Protocol is the heart of the WWW. It's an application-layer protocol. HTTP is used to serve websites from web servers to clients/browsers.

In general: e.g. when a user clicks on a hyperlink, the client (browser) sends an HTTP request message to the server. The server responds with an HTTP respond message that contains the requested object.

HTTP uses TCP. TCP provides reliable data transfer to HTTP. This ensures that all requests sent by the client to the host are received intact and vice versa. Shows the advantage of a layered architecture – HTTP doesn't worry about reliable data transfer.

HTTP is a stateless protocol because the servers don't remember any info about the clients. If a client requests the same file twice in a row, the server serves the file twice in a row.

HTTP can use either Non-persistent or Persistent connections. When using a persistent connection, the same TCP connection is used for every request/response pair – this is the default. When using Non-persistent connections a new TCP connection is used for each request/response pair – web servers can be configured to do this.

A webpage consists of objects – e.g. the base HTML-file, images, videos etc.

Non-persistent connections can be slower than persistent ones. Example on figure 2.7 on page 100. The RTT (round-trip time) is the time is takes for a packet to travel from the client to the host and back to the client. When using non-persistent connections each object on a webpage will take two RTTs. If using Persistent the initial RTT will only be needed once.

FTP

Explanation of **FTP**: the File Transfer Protocol is used to transfer files between a user (at local host) and a remote host. The user must be identifies by the server using username and password (FTP can be public).

FTP uses TCP, like HTTP. FTP uses two TCP connections, a control connection and a data connection. Control is used for identifying the user, and for sending other control commands, like changing directory, and put/get. The data connection is used for moving the actual files – it is non-persistent, a new TCP connection is opened for every file.

Since FTP uses dual connections it is said that it sends control information out-of-band. Hence HTTP sends control information in-band.

3M. Socket programming

Explain socket programming with TCP and UDP.

Socket

A socket is a sort of door between the process (the program) and transport layer.

Therefore socket programming to TCP and UDP are not the same.

Before programming, the developer must decide which port to use, if the developer programs eg. a web-browser, the standard port for HTTP is 80. If it is a non-standard protocol like Skype, the Port chosen must be one outside of the standard scope.

Socket programming TCP

For socket programming, we have to program 2 programs – a client program and a server program, which create a client and server process, when both programs are executed.

Both programs have to follow a protocol as RFC, or else there will not be a connection between client and server, and use of the ports, which is defined in RFC.

Fig. 2.30 Processes communicating through TCP sockets.

Connection between client & socket

- Client must contact server (server process must first be running and then create a socket)
- Client contacts server (Client Socket, specifying IP & port, and establish connection when socket is created)
- When server has been contacted by client, it creates socket, for the communication process (the three way handshaking) *look at fig. 2.31.*

An example on socket programming with TCP is seen on fig. 2.32. Step by step.

- Send request using *clientSocket* (getInputStream (keyboard))
- Read request fron *connectionSocket* and Write reply to *connectionSocket* (converts input to UPPER case)
- Read reply form *clientSocket* (System.out.println (monitor))

Socket programming UDP

- UDP connectionless – sends independent packets of data, without any guarantees about delivery (NO handshaking).
- On each datagram, it has to be batched by an IP address and a port number.
- UDP uses DatagramSocket instead of TCP which uses Socket.

(Fig. 2.34 p. 172)

- Socket is created on the server of type: DatagramSocket, port number is supplied.
- Socket is created on the client of type DatagramSocket.
- Client transfer data to server with method sendPacket() which require IP number and port.
- Server receives the data from client with receive().
- Server transfer data to client with method sendPacket() which require IP number and port.
- Client receives the data from server with receive().
- Client closes the connection with. close(), as it is UDP NO message i sent to server.

4M. Transport layer

What is the main function of the transport layer protocols? How does TCP implement the reliable transmission in the Internet?

Figure 3.2 Transport-layer multiplexing and de multiplexing

The Internet and the TCP/IP network make available two distinct transport-layer protocols.

- TCP (**Transmission Control Protocol**)
    - Provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination
    - Flow control (sender/receiver speed matching)
    - TCP also break long messages into shorter segments and provide a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested.
- UDP (**User Datagram Protocol**) provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control.

*The transport-layer converts the messages from the application-layer into smaller packets called **segments.***

The most fundamental responsibility of UDP and TCP is to extend IP's delivery service between two end systems to a delivery service between two processes running on the end system. Extending host-to-host delivery to process-to-process delivery is called transport-layer **multiplexing** and **demultiplexing.**

Multiplexing requires
- Socket have unique identifiers
- Each segment has fields that indicate the socket to which the segment is to be delivered.
    - Source port number field. (0-16 bit)
    - Destination port number field. (0-16 bit) (**number from 0-1023 are called well-known port numbers,** and are restricted, which means they are reserved for well-known application protocols such as HTTP (80) and FTP (21))

Other header fields depending on which protocol is used **TCP 20 bytes header, UDP 8 bytes header.**

Both UDP and TCP provide integrity checking by including error-detection fields in their segments' header. These two minimal transport-layer services process-to-process and data delivery and error checking are the only two services that UDP provides.

TCP ensures that data is delivered from sending process to receiving process, correctly and in order. TCP converts IP's unreliable service between end systems into a reliable data transport service between processes.

TCP provides **reliable data transfer**. Using **flow control**, **sequence number**, **acknowledgment**, and **timers**.

RDT (Reliable Data Transfer)

TCP is a reliable data transfer protocol that is implemented on top of a unreliable (IP) end-to-end network layer.

- Characteristics of unreliable channel will determine complexity of RDT.
    - RDT 1.0 Underlying channel perfectly reliable (figure 3.9)
    - RDT 2.0 Underlying channel with bit errors (figure 3.10) (Stop and wait protocol)
        - Flaw: What if the ACK or the NAK packet is corrupted?
    - RDT 2.1 Same as 2.0 but with sequence number (figure 3.11 & 3.12)
    - RDT 2.2 Same as 2.1 but NAK free (figure 3.13 & 3.14)
    - RDT 3.0 Channels with errors and loss. Same as 2.2 but with a timer (figure 3.15)
        - How long must the sender wait to be certain that something has been lost? The sender must clearly wait at least as long as a round-trip delay between the sender and receiver.
        - Still a stop and wait protocol, therefore to slow.

Figure 3.31 Sequence and acknowledgement numbers for a simple Telnet application over TCP

First sequence number is random.
Acknowledgement number is the next byte expected from other side.

The Timeout Interval is calculated from EstimatedRTT and DevRTT.

Figure 3.36 A cumulative acknowledgment avoids retransmission of the first segment.

Cumulative acknowledgment

Doubling the Timeout Interval

Each times TCP retransmits; it sets the next timeout interval to twice the previous value. Thus the interval grows exponentially after each retransmission. However, whenever the timer is started after eight of two other events(data received from above or ACK received), the Timeout Interval is set from the most recent calculated values of RTT.

Figure 3.37 Fast retransmit: retransmitting the missing segments before the segment's timer expires

Fast retransmit

When a segment is lost, this long timeout period forces the sender to delay resending the lost packet, thereby increasing the end-to-end delay.

Detect lost segments via duplicate ACKs.

- Sender often sends many segments back-to-back
- If segment is lost, there will likely be many duplicate ACKs.

5M. TCP

Explain TCP, flow control, and congestion control.

Figure 3.2 Transport-layer multiplexing and de multiplexing

The Internet and the TCP/IP network make available two distinct transport-layer protocols.

TCP is one of two transport-layer protocol provides for logical communication between application processes running on different hosts. (For the application's perspective there are no worries about the space between the hosts)

The most fundamental responsibility of UDP and TCP is to extend IP's delivery service between two end systems to a delivery service between two processes running on the end system. Extending host-to-host delivery to process-to-process delivery is called transport-layer **multiplexing** and **demultiplexing.**

Multiplexing requires

- Socket have unique identifiers
- Each segment has fields that indicate the socket to which the segment is to be delivered.
    - Source port number field. (0-16 bit)
    - Destination port number field. (0-16 bit) (**number from 0-1023 are called well-known port numbers,** and are restricted, which means they are reserved for well-known application protocols such as HTTP (80) and FTP (21))
    - Other header fields depending on which protocol is used **TCP 20 bytes header, UDP 8 bytes header.**

TCP (**Transmission Control Protocol**)

- Provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination.
- Flow control (sender/receiver speed matching).
- TCP also break long messages into shorter segments and provide a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested.

The transport-layer converts the messages from the application-layer into smaller packets called **segments.**

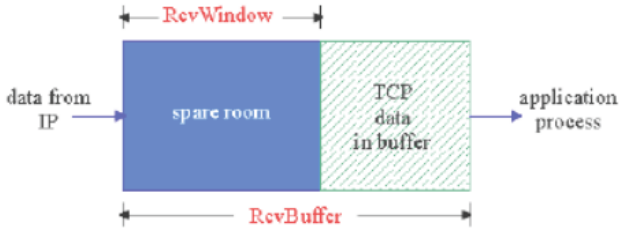
TCP provides **reliable data transfer**. Using **flow control**, **sequence number**, **acknowledgment**, and **timers**. TCP ensures that data is delivered from sending process to receiving process, correctly and in order. TCP converts IP's unreliable service between end systems into a reliable data transport service between processes.

TCP provides **flow control** that eliminate the possibility of the sender overflowing the receiver's buffer . Flow control is a speed-matching service, matching the rate at which the sender is sending against the rate at which the receiver application is reading.

Figure 3.38 the receiver window

TCP provides flow control by having the sender maintain a variable called the receiver window. Informally this window is used to give the sender an idea of how much free buffer space is available at the receiver. Because TCP is full duplex, the sender at each side of the connection maintains a distinct receiver window.

Suppose that Host A is sending a large file to Host B over a TCP connection. Host B allocate a receiver buffer to this connection.



- **LastByteRead**: the number of the last byte in the data stream read from the buffer by the application process in B
- **LastByteRcvd**: the number of the last byte in the data stream that has arrived from the network and been placed in the receive buffer at B

Because TCP is not permitted to overflow the allocated buffer, we must have

LastByteRcvd – LastByteRead <= RcvBuffer

The receiver window, denoted RcvWindow, is set to the amount of spare room in the buffer:

RcvWindow = RcvBuffer – [LastByteRcvd – LastByteRead]

Because the spare room changes over time, RcvWindow is dynamic.
Host B tells Host A how much spare room it has in the connection buffer by placing its current value of RcvWindow in the receiving window field of every segment it sends to A. Initially, Host B sets RcvWindow = RcvBuffer. Note that to pull this off, Host B must keep track of several connection-specific variables.

Host A in turn keeps track of two variables, LastByteSent and LastByteAcked, which have obvious meanings. Note that the difference between these two variables, **LastByteSent – LastByteAcked**, is the amount of unacknowledged data that A has sent into the connection. By keeping the amount of unacknowledged data less than the value of RcvWindow, Host A is assured that it is not overflowing the receiver buffer to Host B.

LastByteSent – LastByteAcked <= RcvWindow

There is one minor technical problem with this scheme. Suppose Host B's buffer become full so that RcvWindow = 0. After advertising RcvWindow = 0 to Host A, also suppose that Host B has nothing to send to A. Now consider what will happen. Host B's buffer will be emptied, but Host A will never know this. To solve this problem, the TCP specification requires Host A to continue to send segments with one data byte when Host B's receiver window is zero. These segments will be acknowledged by the receiver.

TCP congestion control

TCP also provides **congestion control**. Congestion control is not so much a service provided to the invoking application, as it is a service for the Internet as a whole, a service for the general good. Loosely speaking TCP congestion control prevents any one TCP connection from swamping the link and routers between communicating hosts with an excessive amount of traffic. The aim is to give each connection an equal share of the link bandwidth, this is done by regulating the rate at which the sending side of the TCP connection can send traffic into the network.

The congestion- control mechanism has each side of a connection keep track of an additional variable, the congestion window (CongWin), imposes a constraint on the rate at which a TCP sender can send traffic into the network. Specifically, the amount of anacknowledged data at a sender may not exceed the minimum of CongWin and RcvWindow, that is:

LastByteSent – LastByteAcked <= min{CongWin, RcvWindow}

TCP congestion-control algorithm. (TCP is said to self-clocking) have three major components:

- Additive-increase, multiplicative-decrease
- Slow start
- Reaction to timeout events

Figure 3.51 Additive-increase, multiplicative-decrease

Basic idea is to reduce sending rate for the sender (decreasing its congestion window size, CongWin) when a loss event occurs. This is called “multiplicative-decrease” and is approach halving the current value of the CongWin to a minimum of 1 MSS.

When no congestion is detected TCP will increase its sending rate (CongWin) that is , when ACKs arrives for previously yet-to-be-acknowledged data (there is likely available bandwidth) but at a very slow rate, a step for each ACK with the goal of increasing CongWin by 1 MSS every RTT.

Figure 3.51 TCP slow start

When a TCP connection begins, the value of CongWin is typically initialized to 1 MSS. resulting in an initial sending rate of roughly MSS/RTT. Example, if MSS is 500 bytes and RTT is 200msecs the result will be 20kbps. In this initial face the sending rate will be increased exponentially by doubling its value of CongWin every RTT. TCP will keep doubling the rate on CongWin until a loss event occurs, at which time CongWin will be cut in half and then grows linearly.

Reaction to Timeout Events

- If the loss event is detected via receipt of triple duplicate ACKs, TCP will behaves as described- the CongWin is cut in half and then increases linearly.
- If it’s a timeout event, a TCP sender enters a slowstart phase, CongWin is set to 1 MSS an then grows exponentially until reaching one half of its previous size (before slow start). At this point CongWin will grow linearly.

TCP manages these more complex dynamic by maintaining a variable called Threshold, which determines the window size at which slow start will end and congestion avoidance will begins. Whenever a loss event occurs, the value of Threshold is set to one half of the current value of CongWin. As indicated above, a TCP sender will enter a slow start phase after a timeout event. While in slow start, it increases the value of CongWin exponentially fast until CongWin reaches Threshold.

6M. The Network Layer

Explain VC/Datagram Networks, structure of a router (switching techniques) and any three of the following: IP, IP addressing, subnet mask, and NAT

Virtual circuit networks

Table at p. 319

Is a concept from traditional phone networks?

It provides a end-to-end connection between hosts during the transfer.

An permanent connection is provided from host – router – router – host.

When packages have to be sending over the network, a VC is established, and the packet get an VC number. Each router has an internal forward table containing: Incomming interface, Incomming VC#, Outgoing Interface and Outgoing VC#. The packet is routed over the network and forwarded in the routers, which also rewrites the VC number.

(why different VC numbers: then the routers don't have to worry about choosing an unique number).

Then the packet is arrived the VC is closed.

Datagram Networks

A datagram network can be seen as an postal service. There is no dedicated connection setup before it transmits. Each packet is sent with a destination address and a source address. The host transmit the packet to the gateway router. The router looks at the destination address. Internally it has a forward table, which decides on which port the packet will be forwarded to. There is no initial connection setup like in VC, and therefore no closing of the VC. When the packet is forwarded, the router doesn't worrying about it any more.

Router (switching techniques)

There are several different switching techniques for a router.

Fig. 4.8 p. 329

Memory

The packet is copied from the input to internal memory and afterwards copied to output. One memory pool is used for all the packets. Will give a delay, and is bandwidth limited by how fast the memory can read at write data.

Bus

One single bus is used to connect all the ports. When at input port need to connect to an output port, the bus connects the two ports directly. The other port in the router can't be used during that time. Packets will be dropt or stored in input buffer on the ports. Almost no delay

Crossbar

Works a lot like the bus, but make it possible to have more connections between ports at the same time. Almost no delay

IP addressing and subnet mask

Every host on the Internet has to have an unique IP address. The IP address i an 32-bit binary (4-bytes), about 4 billions possible IP addresses. It i written with dotted-decimal notation e.g. 64.233.187.99 .

IP addresses are parted into subnets, to connect two subnets a router is needed. The subnet is determined by the subnet mask (notation /24) e.g. 223.1.1.0/24. Which means the left most

24 bits defines the subnet, so thereby the rightmost 6 bits defines the host. Therefore 233.1.1.0 to 233.1.1.255 can be used for hosts, 256 possible hosts (254 in reality, as .0 is the reserved and .255 is broadcast). So the 254 hosts can be connected to only one router.

Networks like 200.23.0.0/16 allow 65536 host to be on the same subnet. This i normally subnets assigned to big ISP's etc.

NAT

Fig. 4.22 p. 354.

Network address translation, is a concept to make it possible for nodes on a private network to acces the Internet, as if they had external IP addresses.

NAT is done by the router. It has an interface connected to internet e.g. 138.79.29.7, and interface connected till the private network e.g. 10.0.0.0/24 (The router IP in the example is 10.0.0.1).

Internally the NAT router has an NAT translation table.

Example of how it works

- 10.0.0.1:3335 connects to 128.118.48.189:80
- The router rewrites the source of the package to 138.79.29.7:5001 (makes sure that port number isn't already used), and writes that in the NAT translation table. And routes the package on to Internet.
- A package is return from the receiver: 128.118.48.189:80 connects to 138.79.29.7:5001.
- The router rewrites the destination of the package to 10.0.0.1:3345, according to it's internal NAT translation table. And forwards the packages to the private network interface.

A router with NAT for a home network normally also have an DHCP server, as the nodes on the private network cant get IP numbers from the ISP's DHCP server.

Pros:

- No more running out of Ipv4 addresses
- No need to get more them one IP from your ISP.

Cons:

- Some programs have to be alternative to work through NAT.
- First time connection from outside and in (P2P and internal server), require some workaround, as an entry in the NAT translation table, doesn't exists.
  Solutions: Permanent entry in the NAT translation table, setup by the user (used for internal server).

  An outside master node, which the hosts connects (used by IM and P2P).

7M. Routing Algorithms

Explain ASs, RIP, BGP, Distance vector and demonstrate it.

The job of routing is to determine good paths, from senders to receivers, through the network of routers.
One of the two major classes of routing protocol used in packet-switched networks is **Distance Vector algorithm**. DV is a decentralized routing alg., which means that the calculation of the least-cost paths is carried out in an *iterative, distributed manner*. It does not use global information, so no node has complete information about the costs of all network links. Indeed, the only information a node will have is the costs of the links to its directly attached neighbors, and the information it receives from these neighbors. DV uses the Bellman-Ford equation:

Define
$d_x(y)$:= cost of least-cost path from node x to node y

Then $d_x(y) = \min_v\{c(x,v) + d_v(y)\}$

Where $\min_v$ is taken over all neighbors v of x.

Bellman-Ford example – if we take figure 4.27 p.367:

$d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z)\}$
$= \min \{ 2 + 5, 1 + 3, 5 + 3 \}$
$= 4$
because it is the path with the least-cost. The other paths have a higher cost (7 and 8).

The node that achieves minimum is the next hop in shortest path and the solution to the Bellman-Ford equation **provides the entries in node x's forwarding table**. When a node x receives a **new distance** vector from any of its neighbors v, it saves v's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

$$D_x(y) = \min_v\{ c(x,v) + D^*v(y)\} \quad \text{for each node y in N.}$$

If x's distance vector has changed, x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors. Go through this by figure 4.30.

The Bellman-Ford algorithm does not prevent routing loops from happening and suffers from the count-to-infinity problem. The core of the count-to-infinity problem is that if x tells y that it has a path somewhere, there is no way for y to know if it is on the path. To see the problem clearly, imagine a subnet connected like X-Y-Z-L-M-N. Now suppose that X goes down. In the vector-update-process Y notices that its once very short route of 1 to X is down - Y does not receive the vector update from X.

The problem is, Y also gets an update from Z, and Z is still not aware of the fact that X is down - so it tells Y that X is only two jumps from it, which is false. This slowly propagates through the network until it reaches infinity. This scenario can be avoided by using a technique known as **poisoned reverse**. Y tells Z its (Y's) distance to X is infinite, so Z won't route to Y via X.

ASs (Autonomous Systems)
Routers are aggregated into autonomous Systems (ASs) to scale to millions of users (Hierarchical organization of the Internet). An AS is a collection of routers under the same administrative and technical central. Routers within the same AS all run the same routing algorithm, for example DV, but routers in different AS can run different intra-AS routing protocol. A direct link to a router in another AS is called gateway router.

The routing algorithm within an autonomous system is called an **intra-autonomous system routing protocol**. One of the most used Intra-AS routing protocols is **RIP**, which operates in a manner very close to DV.

RIP (Routing Information Protocol)
RIP uses hop count as a cost metric, so each link has a cost of 1. RIP uses the term Hop, which is the number of subnets traversed along the shortest path from source router to destination router (Max = 15 hop). In RIP, routing updates are exchanged between neighbors approximately every 30 seconds using a **RIP response message**, also known as RIP advertisements. Each advertisement can list of up to 25 destination nets within AS. If no advertisement heard after 180 sec → neighbor/link declared dead. Simply example of how it works, look at figures 4.36 – 4.38 on page390-391 (Husk DV og læg mærke til hvad der sker med destination subnet z). Each router maintains a routing table, which includes both the router's distance vector and router's forwarding table.

BGP (Border Gateway Protocol)
Since the Inter-AS routing protocol involves communication between two ASs, the two communication ASs must run the same inter-AS routing protocol. So, in Internet all ASs run the same inter-AS routing protocol, called **BGP**.
BGP provides each AS a means to:

1. Obtain subnet reach ability information from neighboring ASs.
2. Propagate reach ability information to all AS-internal routers.
3. Determine "good" routes to subnets based on reach ability information and policy.

BGF allows each subnet to advertise its existence to the rest of the Internet and makes sure that all the ASs in the Internet know about the subnet and how to get there. If it wasn't for BFP, each subnet would be alone and unknown by the rest of the Internet.

Se lecture 12 slide 48.

8M. The Link Layer and LANs

What is ARP protocol? Which information can an ARP table contain possibly? Please describe in detail how the translation is done and how to send a datagram to a host over the Internet.

What is ARP?

ARP (Address Resolution Protocol) is a protocol, which is used to translate IP-addresses (layer 3 in OSI) to the MAC-addresses (layer 2 in OSI), and it is uses when 2 hosts on same network communicate with each other.
First time, host A contact host B, does host A send a ARP request in a broadcasting frame on the network, asking which MAC address is the owner to the IP address of host B.
Host B, will send the message back in a standard frame.
The host A will save this address in its ARP-table.

ARP table contains

An ARP table contains information as seen on figure 5.18 p. 467, where its shown the IP-address, MAC-address and the TTL (time-to live) value, which indicates when each mapping will be deleted from the table.

The translation

First it looks at its own ARP table after the given IP-address, and if it isn't in the table over the connected IP address, it sent a query ARP message within a broadcast frame, whereas the response ARP message from the founded host within a standard frame. It then saves the IP address and its MAC address in the table. If it cant find the IP address of the host, the ARP return with an error.

Send a datagram over the Internet

If you want to send a package over the Internet from a Host A, to a host B, you first of all will get the IP address on the Host B. Then you use DNS for finding the router where the Host B IP address is on. You then send the package to the router, and when the router has received it, it then creates its ARP table, and finds the Host B, IP address.

The figure 5.19 p. 468 look like the same, just in subnets and not by the Internet.

9M Protocols

Channel partitioning protocols, random access protocols and taking turns. Please explain their principle respectively.

The concrete problem is that there can arise collisions on the network line, which destroys the packets, which has been collided. It happens that bandwidth will be spoiled and the data shall be sent again and has a delay. There are 3 different protocols which can eliminate this problem.

The purpose with a protocol is to
- Send a package while, using the full bandwidth, which is available.
- When more senders send packages on the same time, shall it again use the full bandwidth.
- No central point, which manage the sending packages.
- Simple method, which is easy to implement.

Channel Partitioning Protocols

Contains 3 different types of sending methods.

- **TDM** (time-division multiplexing)
  - It divides the time into frames and further into timeslots. Each node is assigned to each timeslot.
  - Pros is that is divided on a fair way, and the cons is that the whole bandwidth isn't used, if all timeslots don't sent packages.

- **FDM** (frequency-division multiplexing)
  - It divides the data into different frequencies. Each node is assigned to each frequency.
  - Pros and cons is the same as TDM.

- **CDMA** (Code-division multiple access)
  - Assign different codes to the each node. Every node encodes each package with their own code.
  - More nodes can be sent on the same time over the same channel, if the receiver knows the senders code on the packages it receives.

Random Access Protocols

- **Slotted ALOHA** *fig. 5.11 p.455*
  - All nodes send packages in the start of the time interval.
  - If there is a collision, it tries to send the package in the next time interval
  - Pros are that a node can send with full bandwidth, decentralized, but the cons are the synchronizing of the time interval.
- **ALOHA** *fig. 5.12 p. 456*
  - Same as Slotted ALOHA, just the difference, that on a collision, it doesn't wait for the next time interval, but on a specific time span.
  - Pros are that it doesn't need synchronization but the cons are that it is only half as fast as Slotted ALOHA.
- **CSMA (Carrier sense multiple access)** *fig. 5.14 p.460*
  - If there is any other node, which sends packages, will it wait a specific time for trying? If any nodes begin sends packages while you are doing it, then it stop sending. If you not can see there are others, which sent packages and you try to send there will be a collision.

Taking-turns protocols

- **Polling protocols**
  - Need a master node, which put the other nodes in a circle.
  - A node, which is being the “pollet” may send packages.
  - The master node can see when a node is finished sending, and the pros are that there is no collision, and the bandwidth is used. Cons, can be the polling delay, if the master node dies.

- **Token passing protocols**
  - A token is wandering around the different nodes, and the one, which has the token, is able to send packages. Pros are that it is decentralized and effective. The cons are that one node can break it all by not sending the token further to the other nodes.

10M. Wireless and Mobile Networks

Explain the principle of 802.11 media access protocol - CSMA/CA. Compare with Ethernet's CSMA/CD.

802.11 CSMA/CA

- Using collision avoidance techniques.
- To costly to build hardware that can detect a collision.
- Using a link-layer acknowledgement/retransmission
    - When a station sends a frame, the frame may not reach the destination station intact for a variety of reasons. To deal with this non-negligble chance of failure, it uses link-layer acknowledgement.
    - When the destination receives a frame, it waits a short period of time known as SIFS, and sends back an acknowledgement frame. If it not has been received in an amount of time, it sends an error and retransmits the frame, using the CSMA/CA protocol. If it still doesn't work, it gets a time out.

CSMA/CA

Fig, 6.10 p.533 describes CSMA/Ca

1. It transmits it frame after a shot period of time (DIFS)
2. If the channel is sensed busy, the counter value remains frozen.
3. When the counter reaches 0, the station transmits the entire frame and then waits for ack.
4. If the frame is received, the destination sends an ACK back to the source. If it isn't received, it retransmits from step 2, with a random value chosen from a larger interval.

Ethernet CSMA/CD

- Using collision detection
    - If there is any other node, which sends packages, will it wait a specific time for trying? If any nodes begin sends packages while you are doing it, then it stops sending. If you not can see there are others, which sent packages and you try to send there will be a collision.
    - *(Fig. 5.14 p. 458)*

Difference between Ethernet and 802.11

To stations want to sent a data frame to transmit, but neither station transmits immediately beach each senses that a third station is already transmitting.

Ethernet: They would wait, and send on same time, and then make a collision.

802.11: One of the two stations will transmit, and the other one after, and detect the collision.

1C. TCP/UDP

Isn't TCP always preferable to UDP since TCP provides a reliable data transfer service and UDP does not?

TCP

This type of protocol creates direct connection to the receiver (the handshaking way), and keep the connection open to it isn't useful no more. This means TCP is reliable and send the data in a correct order. It has specifications as:

- Congestion control (It detects errors by calculate the available rate pr. Sec.)
- Flow control (it controls how much data which is being sent, and detect overflow)
- Options for pipelining.
- Connection setup (Establish a stable connection)

UDP

UDP sends packets on the network, with a destination address on them. The package is on a random route, and there isn't any stable connection to the next packet. It means that there is NO handshaking. It doesn't have any Congestion control, which means that it just sends segments on the network.

- It doesn't block the network with an open connection.

A table, which shows the best ways to use TCP or UDP in different applications:

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

Fig. 2.5 p. 92

A developers meaning of using TCP

An application developer may not want its application to use TCP's congestion control, which can throttle the application's sending rate at times of congestion. Often, designers of IP telephony and IP videoconference applications choose to run their applications over UDP because they want to avoid TCP's congestion control. Also, some applications do not need the reliable data transfer provided by TCP.

2C. Routing

What is the difference between routing and forwarding?

Forwarding

Forwarding refers to the router-local action of transferring a packet from an input link interface to the appropriate output link interface.
Example: A packet arriving from Host H1 to Router R1 must be forwarded to the next router on a path to H2.

Routing

Routing refers to the network-wide process that determines the end-to-end paths that packets take from source to destination.
Example: A routing algorithm would determine the path along which packets flow from H1 to H2.

An analogy

Consider a trip from Aalborg to Copenhagen. During the trip the driver passes through many interchanges route to Copenhagen. Forwarding is the process of getting through a single interchange, and the routing as the process of planning the whole trip from Aalborg to Copenhagen.

3C. Routing algorithms

Compare and contrast link-state and distance-vector routing algorithms.

Link-state (uses Dijkstra's algorithm)

Fig. 4.25 and table 4.3

LS is a global routing algorithm, that computes the least-cost path between a source and destination using complete, **global knowledge** about the network. LS is known as an algorithm that must be **aware of the cost of each link in the network.**

Message complexity:	With n nodes, E links, O(nE) messages sent.
Speed of convergence:	$O(n^2)$ algorithm requires O(nE) messages (may have Oscillations)
Robustness:	Node can advertise incorrect link cost, and each node computes only its own table.

Distance-vector (Bellman-ford algorithm)

Fig. 4.25

DV is a decentralized routing algorithm, where the calculation of the least-cost path is carried out in an iterative, distributed manner. DV is known as an algorithm that maintains a vector of estimates of the costs (distances) to all other nodes in the network.

Message complexity:	Exchange between neighbours only (convergence time varies)
Speed of convergence:	Convergence time varies and may be routing loops or count-to-infinity problem.
Robustness:	DV node can advertise path cost, and each node's table used by others can give error propagate through network.

4C. DHCP

What is DHCP? How can it be used?

WHAT is DHCP

DHCP stands for Dynamic Host Configuration Protocol, allows a host to obtain an UP address automatically, as well as to learn additional information, such as its subnet mask, the address of its first-hop router, and the address of its local DNS server.

Use if DHCP

It is used as a service on a server that configured the client TCP/IP options automatically when the machine starts. The server can send an IP-address and net mask to the client. The DHCP server is able to delegate a static IP-address, so the client always gets the same IP address.

5C IP

How is one transition from IPv4 to IPv6? What techniques could be used?

IPv4

IPv4 is the fourth version of the Internet protocol, and it was the first one, which was used so much, and did make a basic foundation for the Internet today. IPv4 uses 32-bit addressing, which has a limit on 4.294.967.296 unique address, where many of them is used for local network (LAN) or multicast addresses. It reduces the amount of addresses, which can be official Internet addresses.

When it one day, will be need to get more IP addresses, it will be solution to use the new version IPv6, and therefore many big companies is starting to convert it to IPv6, which still is backward compatible.

IPv6

IPv4 uses 128-bit addressing. IPv6 then can give much more IP addresses ($3{,}4*10^{38}$ addresses). The IP header has been smaller in the new version, and many field which exist in IPv4, is removed because they didn't been used in the old version. New fields are replaced. Many of the big countries today use the IPv6 and more and more countries will update their IP versions soon.

Techniques for transition from IPv4 to IPv6

- Dual-stack approach (fig. 4.23 in the old book)
  - IPv6 has a IPv4 indside it.
  - When converting from IPv6 to IPv4, it will miss some data fields from IPv6 (flow identifier field).
- Tunnelling (fig. 4.24 the old book)
  - When you want to send between to IPv6, you will be able to pack the datagram into the datafield of an IPv4, then "tunnelling" it through to another IPv4, which extract it, onto the IPv6.