Samenvatting

TI1406 Computer Organization

Door: David Alderliesten

Disclaimer

1 – Parallel and Distributed Systems plus Intro

- Organization = All physical aspects of the computer
    - Circuits, memory types, signals, transistors, etc.
- Architecture = All logical aspects
    - Let programmers interact with systems.
    - Input/Output, etc.
- Data deluge
    - Generated by humans and devices (IOT).
        - Interacting
        - Understanding
        - Deciding
        - Creating
    - By 2020, too much data will exist for storage capacity.
- IEEE, ACM, organizations that govern Computer Science and standards curriculum, including educational standards.

A little history of computers

- A history of performance
    - Improving speed of operation by mechanical means.
        - Walking, Biking (=5x walking, 1 order of mag), Flying (=200xwalking, 2OM).
    - Multiplication of two 10 figure numbers (ex: 5,678,912,343)
        - By hand: 10 minutes.
        - By computing: 10 nanoseconds.
- A computer is not a single invention, it has ideas from math, physics, mechanical, and electrical engineering.
- Pre-History (before 1930s)
    - 17^{th} century… development of calculation machines started.
        - Pascal: Two bit operation machine.
    - Binary system… 1705.
    - Mechanical Turk (1770), programmable device.
    - A
    - Programmable devices became widespread, used punching cards.
    - Jaquard Loom
        - Punching card with holes in it.
        - Operator would provide energy by turning a wheel.
        - System would read card and weave based on it.
    - Difference engine
        - Invented by Johann Helfrich von Muller, 1786.
        - Extended by Charles Babbage, 1822.
    - Ada, Countess of Lovelace
        - First computer programmer, 1840s.
        - Knows assembly, as card used that system.

    - Herman Hollerith
        - Punch cards for processing data of the 1890 United States census.
        - Founder of Tabulating Machine Company (1889). Became IBM.
    - George Boole, 1854
        - Showed that logic could be reduced to a simple algebraic system.
    - Formal logic was developed by Alan Turing.
- Analog computers, 1930s
    - Analog computers predated digital computers.
        - Mechanical integrators used in differential analyzers (Vannevar Bush, 1931).
        - Slide rule.
    - First systems that enabled significant reduction of calculation times.
- Spying-Killer app of 1940s, and 1940s.
    - COLOSSUS, UK, 1944, could decrypt secret codes by algorithms.
    - ASCC, First general purpose digital computer.
        - 750,000 components, 5 tons.
        - 3-5 times faster than work/calculations by hand.
    - ENIAC, first all-electronic computer (although not legally).
        - 1,000 bits of memory.
        - On each boot, 10 tubes out of 1000 failed.
        - Not flexible, difficult to program, too small a memory to use.
    - First IP lawsuit: John Mauchly/John Presper Eckert VERSUS John Atansoff.
    - EDVAC, electro-mechanical devices.
        - Von Neumann Architecture. (Memory ←→ CPU ←1 →0.
- Transistors, 1955 – 1975.
- Microprocessors, 1960s - today

2 – History of Computers, Introduction to Algebra/Principles, Laws

- Transistors and Microprocessors (1955-1975 and 1960s-now).
    - Transistors
        - Reliable
        - Less Power
        - Smaller
        - First Supercomputer: Cray's CDC 6600, 10 Mega FLOPS.
    - Integrated Circuits
        - Enabled small, low-cost microprocessors.
        - Apple I/II
        - MOS Tech, ATARI.
        - Current Technology.
- Flop = number of floating point operations per second.
    - Number of arithmetic operations a computer can do per second.
- Law of Computers
    - What is deemed as a supercomputer today, will be in a pocket format within 10 years.
    - Not official law, but more of a parody/reference.
- Devices in the transistor/micro era.
    - First monitor, 1951.
        - US Army's display system.
        - Developed as part of the WHIRLWIND program.
    - First Mouse, 1968.
        - Doug Engelbart's "X-Y Position Indicator for a Display System."
        - "What if any human could have a computer with virtual input, with voice commands, sharing information between these computers."
        - *Mother of all demoes*, developed things not even understood today, developed before PCs even existed.
        - Developed for an in-house project at XEROX.
    - IBM personal computer
        - Release in the 1980s.
        - The blueprint for today's PCs.
        - Was a market influence, and was not protected by law.
            - This meant that open standards and friendliness to third-party hard/software developers.
            - Most platforms were protected by law, preventing development on them.
- Moore's law
    - *Density of a silicon chip doubles every 1.5-2 years.*
    - Moore's law no longer applies today, as we're reaching the (approx..) 18 nanometer density, which is the current limit to them, as they interfere with each other when put closer.
        - IBM Research-Almaden, Jan 2012
            - 96-atom byte.

        - U.New South Wales, Feb 2012
            - 1-atom transistor.
    - "The sweet spot" of transistor density was reached in 2006., changed overall development and technology philosophy.
- Rock's Law
    - *The cost of equipment to produce chips with double power every 4 years in the same space, is becoming increasingly more expensive.*
    - With each chip generation you have a doubling of the production cost.
    - Counter-balance to Moore's law.
- The internet: Early History
    - 1965-1969 ARPANET
        - Developed by Leonard Kleinrock, who made the queuing theory.
        - Consisted between four computers at UC Santa Barbara and UC Los Angeles, U of Utah, and Standford Research inst.
    - 1972, ARPANET is public, along with e-mail.
    - 1974, TCIP/IP developed at Stanford.
    - 1982, ARPANET + TCP/IP public, the early internet.
    - The early internet was based on a postal system, with dispatching, queuing, etc.
    - Early theory relied on packet handling, a few million packets per second.
    - System worked well with a few (10) computers, not with today's six billion + nodes.
- Metcalfe's Law
    - The usefulness of a network, n^2.
    - Value of a network increases quadratically with each person on the network.
        - Two people on a network can talk to one other. Add one person, there are two options per user, also known as n^2, etc.
- Peer-to-Peer file dharing
    - Tribler.org, developed at TU Delft.
        - Bittorrent supporting client.
    - Bittorrent has a central component (a sort of master). Failed in Christmas in 2003, exposed fatal flaw in central point of failure theory.
    - "One goes down, the whole entire network fails."
- ABILENE: Backbone research network.
    - Was a test for the land speed record of networks.
    - 7GB in a single TCP steam from Geneva to Caltech.
- Grid Computing
    - Just plug into the computing grid and get results.
    - The Grid…
        - Integrations of computers as day-to-day computing utility, similar to phone, water, and electricity.
        - Economy of scale: Better service at lower cost.
        - Large-Scale-Reality: operational overhead, functionality (robustness + manageable), real heterogeneity.
        - Modern example could be cloud computing.

TAKE-HOME MESSAGE…

-Metcalfe’s law.

-Moore’s and Rock’s law.

-Single and/or distributed computers.

-PC and/or Mobile computing.

-The Von Neumann and/or Harvard architectures.

-Digital Computers are not a single invention.

3 – Digital Logic

What are the mathematical and electrical engineering concepts that enable digital computers?

- Definition of digital computers.
  - Programmable devices that compute arbitrary arithmetic or logical operations, thus being able to perform more than one function.
  - Use digital, rather than analog, components.
- Atanasoff's principles of digital computers.
  - Use binary information bits (radix 2 versus 10).
  - Use electricity and electronics instead of mechanical devices.
  - Memory based on capacitors, with a regenerative process.
  - Computation by Boolean Algebra (derived from1).
- Unit of Information
  - Computer consist of digital circuits.
  - The unit of information is a bit (Binary digit), 0 and 1.
  - There are two interpretations of 0 and 1:
    - As data values.
    - As truth values (such as *true* and *false*).
- Bit Strings
  - By grouping bits together we obtain bit strings.
    - 10001
      - Can mean 17.
      - Can be the letter 'a.'
  - Meaning of a bit string must be assigned to the bit string by programmer.
- Boolean algebra
  - A specific meaning of a bit string.
  - We want a computer that can calculate, i.e, transform strings to other strings.
    - 1 + 2 = 3 ←→ <01> + <10> = <11>
  - To calculate we need an algebra being able to use only two valuables.
  - Goerge Boole (1854), showed that logic (or symbolic reasoning), can be reduced to a simple algebraic system.
- Boolean Algebra: Basics and Boole's definitions
  - X + y = y + x
  - X * y = y * x
  - X(y+z) = x*y + x*z
  - (x + y) + z = x + (y + z)
  - All these rules are default. BUT… one exception…
    - X * x = x.
    - 0 * 0 is still 0.
    - 1 * 1 is still 1.
- Boolean Algebra (equivalence to logic).
  - The operation ".", or x.y = (x ^ y), AND
    - X = TUD, Y = Students, then x.y = TUD students.
  - If x is a class of objects, then 1 – x is the complement of that class.

        - 1-x is also known as xbar.
- Boolean algebra (computing functions)
    - A nice property is that this system can write any function f(x) as:
        - F(x) = a * x + b(1 – x)
    - We can show this by observing that virtually every mathematical function can be written in a polynomial form.
- Boolean algebra, computing any function in previous form.
    - FOR $f(x) = a_0 + a_1x$
    - LET $b = a_0$ and $a = a_0 + a_1$
    - Then we have, f(x) = a * x + b(1-x)
    - FROM THIS, WE SEE..
        - F(1) = a.
        - F(0) = b.
    - SO, f(x) = f(1) * x + f(0) * xbar.
- Minterm = x * y OR x * ybar OR xbar * Y OR xbar * ybar.
- EXAMPLE: Binary Addition
    - We apply this for the modulo-2 addition, exclusive or (XOR).
    - The result is 1 if either one or the other parameter is 1.

X	Y	+/
0	0	0
1	0	1
0	1	1
1	1	0

    - X +/ y = (x * ybar) + (y * xbar)
- Gates
    - Basic transistor based components to represent primary logic operations.
        - X →O—xbar [Invert]

X	NOT / xbar
0	1
1	0

    - Gates are nothing more than a graphical/CS representation of the mathematical equivalency in truth tables.
- NAND and NOR gates
    - NAND and NOR gates are universal.
        - NAND is universal, it can compile any function.
        - See DeMorgan's laws.
    - They are easy to produce/realize.
- Minimization of expressions (partial optimization)
    - Logic expressions can be minimzed, optimized.
        - Saves components, resources.
    - F = xbar * ybar * zbar + xbar * ybar * z + xbar * y * z + x * y *z
    - F = xbar * ybar(zbar * z) + y * z(xbar + x)
    - F = xbar * ybar * 1 + y * z * 1
    - F = xbar * ybar + y * z

- Karnaugh maps optimization, geometrical method.

○ Xy→ VW down	○ 00	○ 01	○ 11	○ 10
○ 00	○ 1	1	○ 0	○ 1
○ 01	○ 1	○ 1	○ 0	○ 1
○ 11	○ 0	○ 0	○ 0	○ 0
○ 10	○ 1	○ 1	○ 1	○ 0

  - Construction:
    - Consecutive rows/columns differ in 1 bit.
    - One cell per minterm
    - Cell = 1 term from truth table.
  - Optimization rules:
    - Group all 1s.
    - Adjacent 1s, horizontal/vertical. NOT DIAGONALLY.
    - Group size is power-of-2 (1 is -0^{th} power of 2).
    - Take the largets groups possible.
  - THUS, FOR THAT TABLE:
    - F = vdash * xdash + V * wdash * xdash + v * wdash * y + vdashf * x * ydash
      - ALL NOTATION X/Y.
    - First group comes from 00/00, 01/00, 00/01, 01/01.
    - Third group comes from 01/10 and 11/10.

4 – Mathematical and Electrical engineering concepts that enable digital computers.

- Circuits are not instantaneous.
- Functional (arithmetic and logic) Units.
    - It would be uneconomical to construct separate combinatorial circuits for each function needed.
    - Hence, functional units are **parameterized**.
- Propagation Delay
    - Every network of gates has delays.
    - **Transition time** is the time needed to switch between states 0 and 1.
    - **Propagation delay** is the time between the midpoint of the transition time and the midpoint of the output change.
    - Identical types of gates (NAND/NAND, NOR/NOR), will have identical times for propagation delays.
- Functional Units.
    - Take any function.
    - Find its logic variables (eg. X and y), and express it as a Boolean function.
    - Write its truth table.
    - Optimize
        - Simplify the function as a sum-of-products.
        - Also known as XOR.
    - Implement as digital logic.
    - Package and combine where possible [sell as products].
- Combinational Circuits.
    - Combinational circuits have outputs depending entirely on system inputs.
- Sequential Circuits.
    - Repeated addition/counters requires feedback.
    - This can only be done with intermediary storage.
- Set/Reset Flip-Flop
    - Storage elements are not transient and are able to hold a logic value for a certain period of time.
    - **State** of flip-flop depends on current and previous inputs
        - $Q(t + 1) = f(Q(t), S, R)$.
- Clocks
    - In many circuits it is very convenient to have the state changed only at **regular points** in time.
    - This makes design of systems with memory elements easier.
    - Also, reasoning about the behavior of the system is much easier.
    - This is done via a clock signal, with a fixed period.
    - Has rising edge (rising) and falling edge (falling).
- Triggers
    - Level-triggered latches occur at clock highs and lows.
    - Edge-triggered flip flops occur at clock edges.

- D flip flop
    - D flip flop samples (sets output) when clock is high and stores when the clock is low.
- State diagrams.
    - We call the change from one state to another a **state transition**.
    - These can be represented as a state diagram.
- Procedure FST
    - Make state diagram.
    - Make state table/truth table.
    - Give the states binary code.
    - Put state update functions in Karnaugh map, and optimize.
    - Make combinatorial circuit to realize functions, or implement it.
- Circuit Design
    - Inputs and outputs in general line.
    - Holding states around main blok/area.
- Multiplexer
    - Takes n inputs, and selects only 1 of the inputs to output.
    - Y=A, if m=1.
    - Y=B, if m=0.
- Decoder
    - Takes n, and selects only 1 combination.
    - Output $Y_a = 1$, the rest is 0.

5 – Arithmetic and Data representation

Numerical data

- We want to convert values to bit strings.
- Integer Numbers
  - Integers (Z): Set of natural numbers including 0 and their non-zero negatives.
  - Are represented by:
    - Alfabet
    - A string X of n elements from E: X_n is an element of E^n.
  - Radix system
    - $317 = 3 * 10^2 + 1 * 10^1 + 7 * 10^{0.}$
    - $26_7 = 2 * 7^1 + 6 * 7^0 = 20_{10}$
- If one radix is power of other radix…
  - Binary to octal
    - $10100101001 = 010\ 100\ 101\ 001 = 2451_{oct}$
  - Binary to hexadecimal
    - $10100101001 = 0101\ 1010\ 1001 = 5A9_{hex} = 0x05A9$
  - Binary to decimal
    - $10110101001 = 2^{10} + 2^8 + 2^7 + 2^5 + 2^3 + 2^0 = 1449$
- Hexadecimal
  - Counts from 0, 1,….,9, but then, must keep going, so uses letters.
  - A, B, C, D, E, F
    - F = 15
    - E = 14
    - D = 13
    - C = 12
    - B = 11
    - A = 10
    - 9 = 9
    - Etc.
- Binary Addition
  - Can utilize a **ripple carry adder** (slow).
    - Two inputs
    - One carry over
    - One output
  - Binary coded decimal
    - 0, 0000
    - 1, 0001
    - 2, 0010
    - 3, 0011
    - 4, 0100
    - 5, 0101
    - 6, 0110,
    - 7, 0111

        - 8, 1000
        - 9, 1001
- Criteria for comparing representations
    - *Let* $X_n = < x_{n-1}, x_{n-2}, \dots , x_0>$
    - **Range** of natural numbers of x.
    - **Representation** of 0_{10}.
    - **Efficiency** of implementation
        - Sign inversion: A → -A
        - Addition: A + B
        - Extension: n bit → m bit representation, m > n
- Sign-magnitude / 3-bit example
    - Intuition: Represent sign, then the magnitude/value of the number.
    - F: sign(x_{n-1}) {etc.}
        - Sign(1) = - and sign(0) = +
            - $110 = -2^1 = -2$
            - $010 = 2^1 = 2$
            - First bit represents sign, second-n represents magnitude.
        - Zeroes are 100 and 000.
        - Inversion: Simply flip the sign bit.
- One's complement (1C) / 3-bit example
    - Intuition: diminished radix complement, r^n-1-N
    - F: $-x_{n-1}${etc.} * {et.}
        - $110 = -2^2 + 1 + 2^1 = -1$
        - $010 = 2^1 = 2.$
    - Zero: 111 and 000.
- Two's complement (2C)
    - Intuition: Radix complement, 0 iff. N=0, else $r^n – N$
        - $110 = -2^2 + 2^1 = -2$
        - $010 = 2^1 = 2.$
    - Zero: 000.

6 – Digital Data, part two

- Coders at Work, *Reflections on the craft of programming*.
- Karnaugh map minimizer, download program.

- Integer
  - Inversion for SM.
    - F: sign(x_{n-1}){x_{n-2}*etc.}
    - In order to inverse, we need to flip the sign.
      - Change the first bit.
  - Inversion for 1's complement
    - -3 = 100
    - -2 = 101
    - -1 = 110
    - 0 = 111
    - 0 = 000
    - 1 = 001
    - 2 = 010
    - 3 = 011
    - $2^n - 1$
    - Invert all bits from SM for value.
  - Inversion for 2's complement
    - -4 = 100
    - -3 = 101
    - -2 = 110
    - -1 = 111
    - 0 = 000
    - 1 = 001
    - 2 = 010
    - 3 = 011
    - 2^n
    - Add one to all bits from 1C.
  - Excess-16
    - Add 16, convert to binary.
- Real Numbers
  - Fixed Point
    - Real Numbers (R) = value in a continuous space.
    - "Convert umber to fixed point with 2 precision digits."
      - Input: 8.75
      - Step 1: Unsigned binary: +1000
      - Step 2: Ratio: .11 (.75 = $1*2^{-1} + 1*2^{-2}$)
      - Result: +1000.11
    - Some fractions cannot be done exactly in binary, for example, .2.

- Floating Point
  - Is a representation for very large numbers and real numbers.
  - Has a limited number of bits.
    Notation:
    - $G = m * r^e$
      - M = mantissa, or fraction, or significant
      - R = radix (base 2)
      - E = exponent (floating part)
    - R is normally 10 (real life), or 2 (digital devices)
    - Mantissa m is normalized: (1/r) <= m < 1
      - Eg: if r = 2, then .5 <= m < 1
      - E.g: if r = 10, then .1234 * 10^{10} is normalized.
      - E.g: 123,400.0000 * 10^6 is denormalized
      - Think of scientific notation.
  - Metrics
    - Range = ratio largest: smallest number
    - Precision = how many significant bits exist for representation.
      - Found in mantissa/magnitude.
    - Fixed point is better for precision, as more bits exist for mantissa, causing a greater precision.
    - Floating point has a wider range through the exponent.
    - Comes back to the ***no free lunch*** principle.
- IEEE 754
  - Birth of a standard
    - NEEDED
      - Many competing hardware makers.
      - Various range and accuracy.
    - Work group, Intel 1976, W. Kahan.
      - Correct floating point.
      - Developers can write portable code and prove correctness.
    - Resulted in IEEE 754
  - 8 bits given to exponent.
  - 23 bits given to mantissa.
  - 1 bit given to sign bit, totaling at 32 bit.
  - Normalized Mantissa
    - Implicit 1 bit just left of the point, so 24 bit total.
    - Thus, including implicit 1., 1 <= m' < 2
  - Biased exponent
    - E: [1,254] in excess-127 representation → [-126, 127]
    - E = 0 and e = 255 are reserved values.
    - Floating point (radix point is floating)
      - E = 35 → m + 1 means $+2^{35}$ (+34359738368)
      - E = 1 → m + 1 means $+2^1$ (2)

- IEEE example
  - Input: 8.75
  - Step 1: binary fixed point: +1000.11
  - Step 2: normalize: $1.0001100 * 2^3$ [GET NUMBER BETWEEN ½ AND 1]
  - Step 3: Bias the exponent: 3 + 127 = 130 (10000010)
- IEEE 754 TO NUMBER
  - In: 110000000 01010000 00000000 000000
  - Step 1: Split into components
    - [Sign][Exponent][Mantissa]
  - Step 2: Sign… 0 is positive, 1 is negative.
    - Sign in this case is negative.
  - Step 3: Exponent… 128 (10000000)-127 (bias) = 1.
  - Step 4: Mantissa… 1.01010000… = 1.3125
  - Step 5: Result … $(-1)^1 * 1.3125 * 2^1 = -2.625$

7 – ISAs

- ISA = instruction set architectures.
- 7-steps: "How to program a digital computer?"
    - 1. Create a model for a programmable digital computer.
    - 2. Specify a programming interface for each computer.
    - 3. Specify a **generic** programming interface.
    - 4. Design digital computer that match the specific/general interface.
    - 5. Design software engineering techniques to simplify and optimize coding.
    - 6. Design computer engineering techniques to simplify and optimize digital computers.
    - 7. Take class TI1406 and learn basics 1-6.
- A programmable device
    - READ(X) = read next input value from input stream and store as variable x.
    - WRITE(X) = put value in variable X on output stream.
    - ADD(X,Y,Z) = assign value of X+Y to Z.
        - The above three items can add operands and execute addition, then output result.
            - READ(X)
            - READ(Y)
            - ADD(X,Y,Z)
            - WRITE(Z)
- Von Neumann Architecture
    - Had memory and the Central Processing Unit.
    - The memory held data and instructions.
        - X: 1
        - Y: 2
        - Z: 3
        - READ(X)
        - READ(Y)
        - ADD(X,Y,Z)
        - WRITE(Z)
    - The CPU was the programmable device, was actually "reading" the memory and executing instructions. Could output to output device, receive inputs.
    - Control mechanism includes the **program counter**, which identifies or points to the next instruction (remembers location in memory where next instruction is).
    - Bit of storage inside CPU called **instruction register**. Once instruction is understood (location in main memory), it must be fetched and placed in the CPU. It will be stored locally and processed.
    - CPU provides units of memory (register), to store operands and result.
    - Other component of the CPU is the ALU, or arithmetic unit.
- Instruction set architecture
    - Instructions a computer can perform (instruction type) and instruction format.
    - Common instruction types
        - Data copy (memory to register)

        - Data processing (arithmetic, etc.)
        - Program control (IF statements, etc.)
    - Common instruction formats
        - General format (opcode + operands)
        - Other fomrats.
- Types of ISAs
    - 1. How are instructions aligned with memory boundaries?
    - 2. Which types of instructions?
    - 3. How many explicit operands to allow per instruction?
    - 4. Where are the operands stored in the CPU (reg/acc/stk)?
        - Accumulator = Storing 1 unit of information in processor.
        - Register = Multiple registers for data in processor.
        - Stack = Providing restricted access to certain instructions of processor.
    - 5. How to access operands?
    - Other questions, trivia…
        - "Which instructions can access memory?"
- Legacy Problem = Once an ISA becomes standard, it will be the standard for as long as such processors are in circulation. Must also be supported by newer processors.
- Criteria for Comparing ISAs (flaming)
    - 1. Complexity of what the ISA can do (flexibility)
        - Types of operations the ISA provides.
    - 2. Complexity for the programmer (programmability)
        - Length of code, etc.
        - Naïve implementation is the cause of the 10% peak performance.
    - 3. Complexity for the hardware (implementation cost)
        - Length of instructions, etc.
    - 4. Other questions…
        - Wasteful in memory occupancy?
        - Order of bytes in memory?
        - What kind of hardware needs to exist?
        - Etc.
- Memory Layout
    - Main memory layout
        - Bit
            - 00000000 00000000 00000000 00000000

              8 16 24 31
        - Byte
            - 00000000 00000000 00000000 00000000

              1 2 3 4
        - Word, non-standardized [8bit, 16bit, etc.]
            - 00000000 00000000 00000000 00000000

- Notice about kilobytes (KB), rounded from 1024 to 1000, the 2% rule.
- When converting bits to bytes, convert with a power of 2^3.
- For word 32 bits, subtract two from byte count.
- Instruction and Word alignment
    - Equal versus Variable Length, Word-aligned
        - Equal length gives an equal amount of space for all instructions.
        - Variable length provides size depending on the requirements.
        - Word aligned aligns instructions at the beginning of a word.
    - Variable length, not word-aligned
        - Instructions have different lengths.
        - Words start mid-sentence or mid instruction.
- 32-bit word formats
    - An integer number in 2C
        - 31,30, 28, 1, 0
    - 4 ASCII characters
        - Byte, Byte, Byte, Byte
    - A machine instruction
        - Opcode specifier, operand specifiers.
            - Opcode = Description of operation that will come.
            - Operand = Details or specifics for this operation.
- Byte Ordering (endianness)
    - Little Endian
        - Intel
        - Little Endian places most important bit in most right position.
        - MSB to LSB.
    - Big Endian, MSByte in smallest point.
        - IBM, IPV6.
        - Big Endian places most important bit in most left position.
        - MSB to MSB
    - NUXI problem
        - When trying to port data between machines with big-endian and little-endian machines, for example, a network.
        - Due to a lack of correct translation, NUXI can become UNIX in translation.
        - Networking protocols communicate type of representation alongside the actual data.
- Types of instructions
    - Common modern computers have 4 types of instructions.
        - Data copy operations
            - Between memory and registers
            - Between memory locations
            - Between registers
        - Arithmetic and Logic operations

        - Program flow operations
        - I/O operations.
    - Symbolic Notation
        - Copy instructions
            - $[R_1] \leftarrow M(LOC)$
                - Register R_1
                - LOC is a memory address
                - M(address) means contents of memory location at address.
        - Arithmetic operations
            - $M(C) \leftarrow M(A) + M(B)$

8 – Instruction Set Architectures, Part 2

- Symbolic Notation
  - Copy Instruction
    - $[R_1]$ ← M(LOC)
  - Arithmetic Operations
    - M(C) ← M(A) + M(B)
- Operand specification formats
  - Three-address instructions
    - Format: INSTR source#1,source#2,destination
    - Example: Add A, B, C
    - Means: M(C) ← M(B) + M(A)
    - PROBLEM: 3=address instructions means long instruction words.
      - If k bits are needed for memory addressing, then 3k bits are needed for addressing operands.
  - Two-address instructions
    - Format: INSTR source, destination
    - Example: Add A, B
    - Means: M(B) ← M(B) + M(A)
    - PROBLEM: 2-Address instructions mean somewhat long word or multiple. FETCHes per operation.
    - PROBLEM: Operand Override
      - We destroy content at B.
      - Solution is to store B in a new variable C.
  - One-address instructions
    - Have implicit source (called an accumulator)
    - Example:
      - Load A
      - Add B
      - Store C
    - Means:
      - [Accu] ← M(A)
      - [Accu] ← [Accu] + M(B)
      - M(C) ← [Accu]
- Register namen 32/64 bit namen
  - EAX [RAX in 64 BIT]
  - EBX
  - ECX
  - EDX
  - Bij intel heten ze accumulator (A), base op (B), Counter op (C), en Data op (D).
- Geheugenadressen zijn meestal in hexecimal.
- Registers, multiple accumulators
  - Many computers have a number of general-purpose registers inside the CPU.
  - Access to registers is faster than to memory locations.

    - Registers require less bits of address than main memory.
    - Used to store temporary data during processing.
- Branching (name for loop)
    - Common conditions
        - They refer to the last operation
            - Decr R1
            - Branch>0L
        - N (negative) set to 1 if result is negative.
        - Z (zero) set to 1 if zero is given.
        - V (overflow) set to 1 if overflow.
        - C (carry) set to 1 if result led to carry-out.
- Immediate Addressing
    - Opcode specifier | operand
        - Add | # - 1
        - JNZ 10 (simple counting loop)
    - ADVANTAGES
        - Fast
        - No additional calculations needed to obtain operand.
    - DISADVANTAGES
        - Operand value must be known.
        - Operand value cannot be changed.
        - Limited number of bits available.
    - Notation: MOVE #200, R0
    - Meaning: $[R_0] \leftarrow 200$
- Direct addressing / Absolute Addressing
    - Opcode specifier | memory or register address.
        - Add | 13
        - JNZ | 10
        - ///
        - # - 1
    - ADVANTAGES
        - Operand separate from instruction
        - Can be changed
        - Full word length available.
    - DISADVANTAGES
        - More memory accesses
        - More store occupation.
    - Notation: ADD R1,R2
    - Meaning: [R2] ← [R2] + [R1]
- Indirect addressing
    - Opcode specifier | memory or register address
        - Operand found within memory or registry address.
            - ADD | (12)
            - JNZ | 10

        - 13
        - # - 1
    - ADVANTAGES
        - Actual address of operand is not in instruction, and can be changed.
    - DISADVANTAGES
        - Even more memory or register references.
        - More memory occupation.
    - Notation: ADD (R1), R2
    - Meaning: [R2] ← [R1] + M([R1]_
- Index addressing
    - Opcode | Register | index
    - ADVANTAGES
        - Allows specification of fixed offset to operand address.
    - DISADVANTAGES
        - Extra addition to operand address.
    - Notation: ADD X(R1), R3 [X = Offset]
    - Meaning: [R3] ← [R3] + M(R1] + X
- Addiitonal modes
    - Some computers have auto-increment loops.
        - Example: R(O) +
        - Meaning: …M(R0) ; [R0] ← R[0]+1
        - Example: - (R0)
        - Meaning: [R0 ← [R0] – 1; … M(R0);
    - Logic Instructions
        - Not R0 (invert all bits in R0)
        - And #$FF000000, R0; AND with bit string.
    - Shift and rotate instructions
        - Many variants for different purposes.
- CISC vs RISC
    - CISC
        - Complex instruction set
        - Traditional architecture
        - Powerful instructions
            - Complex operations
            - Many instructions
        - Memory to Memory operations
        - Programs often use stacks.
        - Example are 68xxx and 80xxx architectures.
        - The Pentium architecture.
    - RISC
        - Reduced instruction set
        - Small number of instructions
        - Load/Store from memory
        - Operations between registers

        - Large register sets.
        - Example is the PowerPC architecture.
    - Advantages of CISC
        - Easier to program.
        - Reduced code size.
        - Complexity in hardware, not in software.
            - HLL support in hardware.
        - (Politics) Legacy
            - CISCs are in all our PCs and servers.
    - Disadvantages of CISC
        - Instruction encoding is complex.
        - Variable number of cycles to load instruction.
            - IA-32 instruction can be 1-17 bytes long.
        - Many instructions are too specific, thus not used.
        - May be slow
            - Stacks are main memory, registers are near the processor
        - May consume more energy
            - Thus, are not found in embedded systems and portable devices.

9 IA-64 focus, assembly and real-world instructions.

- What and why assembly?
  - What?
    - Assembly Is a symbolic notation for machine language.
  - Why?
    - Improves readability.
  - Access to all hardware resources of the machine.
  - Targets for compilers.
  - Speed of program in critical situations improves.
- Assembler statements
  - Declarations
    - No code generation
    - Memory reservation
    - Symbolic data declaration
    - Where to start the code execution
  - Executable statements
    - Are translated to real machine instructions (usually, one-to-one).
- Data declarations
  - S EQU 200
  - Origin 201
    - S is a variable with value 200.
  - N Data 300
  - N RESERVE 300
    - Array N with 300 bits.
- Number notation
  - Numbers can be represented in many formats:
    - ADD $93, R1
    - ADD $%01011101, R1
    - ADD #$5D, R1
- The Stack
  - Large memory space used to store program data.
  - Items are added to the stack through a PUSH operation.
  - Items are removed from the stack through a POP operations.
  - Often, a stack is a contiguous array of memory locations.
  - Often, any number of stacks can be set up by a program.
  - Often, only one stack can be used at a time.
    - Changing the active stack is possible.
  - Een stack gaat naar de nul toe, dus altijd 4-bit/8-bit eraf halen.

10 – Assembler part 2

- Subroutine Calling mechanism
    - After completion of a subroutine, the program counter calls on the link register to return to its original position.
    - One link register is not enough, as multiple ones may be needed for functions in functions.
- Nesting
    - Calling a function within a function is called **nesting**.
    - Program counter positions can be stored in the stack.
        - MOVE LINK, -(SP)
        - …
        - MOVE (SP+, Link
        - RTS
- Parameter passing
    - Through registers…
        - Fast
        - Limited number of parameters
        - Caller and Callee must both know where parameters are passed.
            - MOVE A, R0
            - Call Sub
            - Sub: Move R0, **C**
            - ….
            - RTS
    - Through memory
        - Very flexible, 20,000 arguments or more.
        - Slower than through registers.
        - Programmer's View
            - Parameters are pushed on stack before calling subroutine.
            - Results are popped from stack after return.
        - System designer's view
            - Extra stack pointers needed.
- Frame Pointer
    - Access of local variables of the subroutine through Index Addressing on the Frame Pointer instead of on the Stack Pointer.
    - Also called the base pointer.
    - Points towards a different point on the stack.
- Re-entrancy
    - Subroutines can be called more than once.
        - Called recursion: subroutine calls itself.
        - Sub A calls Sub B, which in turn calls Sub A.
        - Multiple callers are the same time.
    - Special measures for re-entrancy
        - No change of instructions

        - Each caller must have its own copy of data
        - Use stack(s).
- IA-32. Part II (not 64-bit, yet)
    - IA Family history
        - Intel Architecture is a family of processors.
        - Each processor has the same architecture, but a different organization.
            - Same instruction set
            - Different Performance
        - 32-bit memory addresses and variable length instructions.
        - Very large instruction set (not RISC).
    - Memory = Linear Address Space
        - Memory is byte addressable.
            - Memory is just from 0 to the $2^{32} - 1$.
        - Doublewords can start at any byte location.
        - Data Operands are 8 or 3 bit wide.
        - Mode is little-endian scheme.
    - Numeric Data Types
        - 2's complement numbers are signed integers.
        - IEEE-754 are used for floating points.
    - Registers
        - Floating point register, 80-bits.
            - FP0 – FP7
        - General Purpose registers, 32 bit
            - R0 – R7
            - Can still be used like this ,but called…
                - RAX
                - RBX
                - RCX
                - Etc.
            - EAX refers to a 32-bit register.
            - AX refers to the first/lowest half.
            - ESP and EBP are point registers.
            - ESI AND EDI are index registers.
            - EIP is the instruction pointer (now called RIP, all Es are Is)
            - EFLAGS status register.
    - Status Register
        - CF = Carry
        - ZF = Zero
        - SF = Sign
        - IOPL = I/O privilege level
        - OF = overflow
        - IF = interrupt enable
        - TF = Trap Flag
    - Instructions

        - Variable length and the shortest is 1.
        - Five Types of instructions
            - Data transfer
            - Arithmetic
            - Flow Control
            - Processor Control
            - I/O
- Immediate versus Direct
    - Immediate
        - [EAX] ← #loc
    - Direct
        - [EAX] ← m(loc)
- Compare
    - Used to compare values and leave registers values/contents unchanged.
        - CMP dst, src
        - [dst] – [src]
- Flow Control
    - Two basic branch instructions:
        - Jmp
        - Ret

11 – Basic processing unit, part 1

- MIDTERM WARNING
  - Chapter 1, COMPLETE
  - Chapter 2, COMPLETE
  - Chapter 9, ALL BUT 9.2, 9.5, 9.6, *.*.*
  - Appendix A, all but A.5, A.11-2
- Problem
  - How do we get from the notion of instructions, to a notion of packaged networks of gates.
- Von Neumann architecture
  - Unit of memory stores data instructions.
  - Center of device is processing unit, which fetches operations from memory and performs them one by one.
- Organization and Basic processing cycle
  - Execution of a complete instruction 4-stage execution
    - Fetch Instruction
    - Fetch the Operand
    - Perform Operation
    - [USE MEMORY OPERAND / NOP]
    - Store result
  - Components of a basic Processing unit
    - Instruction address generator [PC]
    - Register File [IR]
    - Processor memory interface
    - ALU
    - CPU Bus
      - System that moves information between CPU components.
    - Control circuitry
      - Control all CPU components
  - The control circuitry must issue signals, but these are directed by a central clock.
  - Components may have internal clocks, but all still follow central clock.
- Organization in a practical processor

PC		Decoder [Control circuit]
Mar [Memory Bus]		IR [Control circuit/IR]
MDR [Memory Bus]		Register files…
Y		R0
ALU		R1… Rn-1
Z		Bus

- Register gating – Intuition
    - Bus (as a concept), is a vehicle that can be driven by one driver at a time.
    - May change driver.
    - Mechanism needed that ensures mutually exclusive use.
    - Through gates/semaphores, it can be ensured through 3 states that only one item is controlling the bus.
    - In order to store information through register gating, both the switch of the register and the switch of the desired logic unit must opened at the same time.
- Simplified design through equal number of stages/instructions.
    - Stages are small sections that perform sub-parts of an operation.
- A collection of wires is called a “port.”
- Types of Operations
    - Transferring information between registers
        - Copy contents of R1 to R3.
        -

12 – Input / Output

- Data Deluge
    - There is more data in the world (4.4 Zettabytes) than sand in the world.
    - By 2020 it will be in the magnitude of 44 Zettabytes. This will be impossible.
    - **Data generated by humans devices (Internet of Things)…**
        - Interacting
        - Understanding
        - Deciding
        - Creating
- The simplest problem with I/O and CPUs.
    - Computers must be able to communicate with the outside.
    - Large varieties of devices.
        - Size
        - Speed
        - Distance
- Every IO device is just some memory and registers mixed in with other capabilities to perform other operations.
    - The main focus of I/O devices is to handle and find data.
    - Keyboards have small memory, but modern gaming keyboards have macros which store a lot of instructions.
    - Scanners and the Hadron collider (…) need a lot of memory for storing petabytes of information.
- Single bus architecture
    - Processor
        - COMMUNICATE VIA BUS
    - Memory
    - I/O device 1
    - I/O device #n
        - Advantage is there is one bus in the system.
            - Great simplicity, easy to implement.
        - Disadvantage is there is only one bus in the system.
            - One device can write at any moment.
- Multi-bus architecture
    - Memory
        - COMMUNICATE VIA MEMORY BUS
    - Processor
        - COMMUNCIATE WITH I/O BUS
    - I/O Device #1
    - I/O device #n
        - Advantage is that busses can now operate differently.
        - Disadvantage is additional power and more complicated design.
- Bus-Interface organization
    - Consists of address lines, data lines, and control lines.

    - Use I/O busses to read and write.
    - Handled by address decoders, control circuits, data and status registers, etc.
    - These decodes are in I/O interface, which communicates with the I/O device.
- The model and the practice
    - PC motherboards include North and South-bridges.
    - Northbridge is high performance.
        - Memory controller hub
        - Graphics Card
        - RAM memory slots
    - Southbridge is flexible.
        - I/O controller hub
        - Ethernet
    - All components are connected by buses. The North and South bridges are these bus representations.
    - Combining components allows for greater speeds and efficiency, but decimates flexibility and possibilities.
- The video terminal, basic I/O program
    - I/O instructions
        - Move DATAIN, R1
        - Move R1, DATAOUT
    - Busy Waiting (mechanism for polling):
        - READWAIT Branch to READWAIT if SIN = 0
            - Input from DATAIN to R
        - WRITEWAIT Branch to WRITEWAIT if SOUT = 0
            - Output from R1 to DATAOUT
- Main memory mapping
    - Memory mapped
        - CPU can access main memory of all the applications, IOPROC1, and IOPROC2.
        - Registers of the devices are mapped to main memory locations.
    - Separate address spaces
        - Each device and application has its own memory addresses and spaces.
        - CPU must go through each separate bus.
- Programming the I/O
    - Programmed I/O
        - Processor involved in transfer of each data unit.
        - CPU executes the I/O program
    - Non-programmed I/O
        - Processor not involved in transfer of each data unit.
        - Separate entity takes care of data transport.
- Programmed I/O in detail
    - Unconditional I/O
        - **No synchronization** with I/O device.
    - Passive signaling (polling)
        - Synchronization between CPU and Device by programmed **interrogation** by CPU

        - "Are we there?" Not yet. "Now?" No.
    - Active signaling (interrupting)
        - Synchronization between CPU and device by active **interrupt** of device.
        - Keeps going until a parameter is met.
- Non-Programmed I/O in more detail
    - Direct Memory Access (DMA)
    - Special I/O processors
- Handling interrupts
    - Device **raises interrupt** request.
    - Processor interrupts program in execution.
    - Interrupts are **disabled**.
        - The interrupt cannot interrupt the interrupt.
    - Device is informed of **acceptance**, and, as a consequence, lowers interrupt.
    - Interrupt is handled by service routine.
    - Interrupt are enabled again.
    - Execution of interrupt program is resumed.

13 – Memory and memory devices

- Connection between Memory-CPU
  - **Memory Access Time** = response time for a request.
  - **Memory cycle time** = Time between two operations, 10 nanoseconds.
  - **Random-access Memory (RAM)** = memory access time independent of accessed address.
- Organization features
  - Addressable number of bits/bytes/words (capacity)
  - Ordering of bytes/words in memory (storing and access)
  - Access time speed-up techniques
    - Memory interleaving
    - Cache memories
  - Increase capacity (virtual memory)
- Pinning
  - Any circuit needs two extra pins, **ground** and **supply**.
  - The smaller the addressable unit, the fewer pins needed.
- Interleaving
  - Consecutive addresses need to reside in different memory modules
  - This is done by placing identifiers as LSB to the right of address.
- The performance gap between processor and memory
  - Moore's law for processors, until 2006ish.
  - Nothing equivalent for memory.
- Memory hierarchy
  - CPU
    - Really fast memory on core of chip, primary cache, Level 1 cache
    - Memory closer to CPU increases speed.
    - Memory further from CPU increases size.
    - Memory closer to CPU is increased in cost per bit.
  - Primary cache: level 2
    - Outside main part of CPU but still on CPU circuits/core.
  - Secondary cache: Level 3
    - In the order of megabytes
  - Main memory: Level 4
    - Slowest in comparison to upper levels.
  - Secondary memory: Disks, I/O cache
    - Part of the storage hierarchy.
- Caching Basic Principles
  - CPU ←→ Cache ←→ Main memory
  - A cache has a performance closer to that of the CPU.
  - Acts as a mediator between memory.
- How do caches work?
  - READ operations
    - If not in cache (MISS), copy block into cache and read out of cache.

        - If in cache (HIT_, read out of cache.
    - WRITE operation
        - If not in cache (MISS), write in main memory.
        - If in cache (HIT), write in cache, and either:
            - Write in main memory (store through), or…
            - Set modified (dirty) bit, and write later.
- Caching, why does it work?
    - Faster, but smaller (what is stored is faster to access).
    - Store most recent data (Transparent for the user).
    - Works because of the locality principle:
        - Temporal locality:
            - If I access X[i] now, I will access X[i] in the future.
        - Spatial locality:
            - If I access X[i], I will soon access X[i+1].