1. 5 komponentów komputera:
   1. Procesor – control, datapath
   2. Memory (dane i program podczas wykonywania)
   3. Urządzenia zewnętrzne – input (klawiatura, mysz, output (monitor, drukarka)
2. Komputer liczy output na podstawie input

Int1, int2 🡪 CPU 🡪 output

Zmienna 🡪 memory (RAM) 🡪 CPU 🡪 memory (RAM)

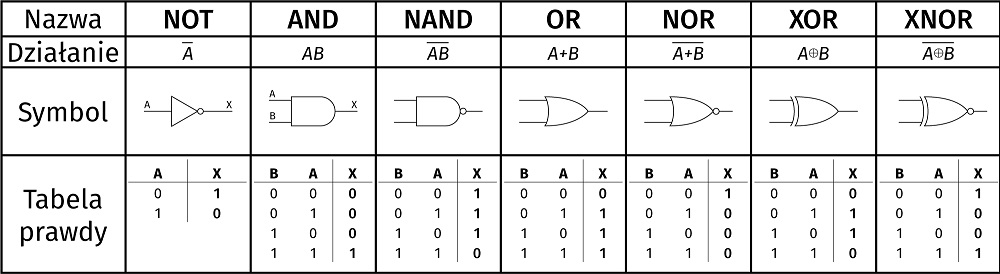
Wszystko dzieje się w systemie 0,1 – ciąg bitów.

1. Programowanie:

High-level language program (in C) 🡪 C compiler 🡪 assembly language program 🡪 assembler 🡪 binary machine language program

1. Bramki logiczne: (każda ma kilka tranzystorów)

Współczesny procesor zawiera >100



1. Taktowanie
   1. 1 takt to jeden wykonany rozkaz przez CPU
   2. Takty wyznaczone są przez zegar
   3. Stan CPU jest zapisywany w specjalnych rejestrach
   4. Procesor CPU ma więc wiele wejść różnych typów:
      * Do pobierania / zapisywania danych (np. 2, 3, etc)
      * Pobierania kolejnych instrukcji (np. read a; c=a+b; etc)
      * Dla zegara do wyznaczania kolejnych taktów

Zegar jest zbudowany na krysztale kwarcu, który pobudzony wysyła impulsy: zera i jedynki na przemian

1. Internet

CPU + pamięć 🡪 karta sieciowa, graficzna, dźwiękowa 🡪 internet 🡪 web server

1. Kompilacja i wykonywanie programów:
   1. Programy są pisane w języku zrozumiałym dla człowieka
   2. Muszą być przetworzone (skompilowane) do języka maszynowego do rozkazów w postaci bajtów
   3. System operacyjny wykonuje binarnych kod zarządzając hardwarem. W tym samym czasie może być wykonywane wiele programów (aplikacji) chociaż CPU wykonuje rozkazy sekwencyjne tj. jeden rozkaz w jednym takcie.
2. Elementy komputer:
   1. Procesor – mózg
   2. Memory – notatki
   3. Disk – trwała pamięć
   4. I/O – komunikacja (poprzez zmysły)
   5. Software – sposoby, metody, algorytmy
3. Pamięć:
   1. Wielkie tablice komórek (słowa kilku bajtowe)
   2. RAM : Random Access Memory
   3. W RAM jest przechowywana informacja (dane) podczas działania komputera:
      1. Dane jako wartości zmiennych
      2. Te dane to także program (sekwencja instrukcji z instrukcjami kontrolnymi tj. skokami), który jest wykonywany
4. Bus (magistrala) – prosty i szybki sposób komunikacji tj. przesyłania bajtów pomiędzy urządzeniami (autostrada, bus pas)

- Na magistrali znajdują się zapytania dla poszczególnych urządzeń, które sprawdzają czy to co się tam znajduje dotyczy ich, jeżeli tak, opowiadają

III 17.10

* + - 1. Control Unit State Machine
         1. Fetch – pobierz instrukcje
         2. Execute – wykonaj instrukcje
         3. Repeat

1. Programy i instrukcje:
   1. Program składa się z sekwencji instrukcji
   2. CPU wykonuje jedną instrukcje w każdym cyklu zegarowym – współczesne procesory wielordzeniowe wykonują więcej
   3. Poziomy programowania
      * Najniższy: Machine Language – ciągi bitów
      * Pośredni: Assembly Language – specyfikuje instrukcje niskiego poziomu do postaci mnemonicznej
      * Do tworzenia aplikacji: High-level Language – jest dekodowany na wiele instrukcji niskiego poziomu
   4. Złożony zestaw instrukcji:
      * CISC – Complex Instruction Set Computer
      * RISC - Reduced Instruction Set Computer
2. Machine Language:
   1. Różne rodzaje procesorów posiadają różne zestawy instrukcji:  
      instrukcje są różnie kodowane w machine/assembly
      * Pentium family / celeron / xeon / AMD K6 / Cyrix / (Intel x86 family)
      * Power PC (mac)
      * DragonBall (Palm Pilot)
      * StorngArm/MIPS (WinCE)
      * Wiele innych (specjalizowanych i uniwersalnych)
3. Kodowanie instrukcji
   1. Maszynowe instrukcje to również liczby 16-bitowe
   2. Dzielimy te bity na grupy:
      * Pierwsze 4 bity są nazywane OP-Code (Kody operacji):

Wskazują na typ instrukcji

?

* 1. Instrukcje:
     + 0: halt
     + 1: add
     + 2: substract
     + 3: multiply
     + 4: bus output
     + 5: jump
     + 6: jump if positive
     + 7: jump & count
     + 8: bus input
     + 9: load
     + A: store
     + B: load direct/addr
     + C: NAND
     + D: AND
     + E: Shift Right
     + F: Shift Left

1. Stos
2. Rejestr IP i rejestr FLAGOWY
   1. IP (wskaźnik rozkazów)- rejestr ten zawiera zawsze offset pamięci, w którym zawarty jest następny rozkaz do wykonania. Bazowy adres segmentu kodu zawarty jest w rejestrze CS. Stąd pełny adres logiczny wykonywanego rozkazu wskazywany jest parą rejestrów CS:IP.
   2. Oznacza to, że jeśli wykonywany jest rozkaz to wskaźnik rozkazów ustawiany jest do następnego adresu pamięci, pod którym znajduje się rozkaz do wykonania. Wyjątek stanowią rozkazy wywołania i skoku.
   3. FLAGS (rejestr znaczników, rejestr flagowy)- rejestr ten jest zbiorem poszczególnych bitów kontrolnych (znaczników), które wskazują wystąpienie określonego stanu procesora
3. Historia języków programowania
   1. Fortran (1954) John Backus IBM, do obliczeń naukowych (głównie w Fizyce) ale też atomowych i termojądrowych
   2. Cobol (1959) (akronim od ang. common business-oriented language) - wysokopoziomowy język programowania stworzony i używany do tworzenia aplikacji biznesowych. COBOL jest językiem imperatywnym, proceduralnym, oraz od 2002 roku, obiektowym.
   3. Algol (1958) bardziej uniwersalny niż Fortran - nadal są skoki w formie „go to"
   4. Lisp (1958) programowanie na listach
   5. A Programming Language (APL) - język programowania wysokiego poziomu, znany ze swojej zwięzłości i możliwości generowania macierzy. Opracował go w połowie lat 60. Kenneth E. Iverson
   6. Algol + Fortran 🡪 PL/1 (1964)
   7. Basic (1964) dla każdego
   8. Simula (1967) + Algol 🡪 Smalltalk (1969) obiektowy
   9. BCPL 🡪 B 🡪 C (1971)
   10. Algol 🡪 Pascal (1971)🡪 Modula 1,2,3,
   11. C++ (1983) = C z obiektami - C jest nadal użwany
   12. Awk (1978) 🡪 Perl (1987) - Web programming language
   13. Java (1991) - Web applets
   14. Visual Basic(1991) macros and programs - Core of Microsoft systems
4. Jaki powinien być dobry język programowania?
   1. Łatwy do kodowania
   2. Chroni przed popełnianiem błędów
   3. Obsługuje debugowanie, gdy jest to potrzebne
   4. Ma wszechstronny zestaw narzędzi
5. Podsumowanie
   1. Programowanie jest trudne
   2. Programiści muszą:
      * dokładnie zrozumieć zadanie
      * przewidzieć wszystkie możliwości
      * pisać dobry (?) kod
      * przewidzieć, co mogą zrobić użytkownicy (trudne)
   3. Języki (i frameworki) programowania pozwalają używać narzędzi do budowania kodu. Tam też mogą być błędy
   4. Koszt błędu może być bardzo duży
   5. Nie istnieje prawo Moore'a dotyczące oprogramowania.

IV 24.10