Elementy logiczne zawarte w typowym komputerze (cont.)

- Jednostka arytmetyczno-logiczna(ALU) – część CPU
  - Wykonuje obliczenia arytmetyczne (dodawanie, odejmowanie...) i podejmuje decyzje logiczne
- Układ sterowania (CU) - część CPU
  - Nadzoruje oraz koordynuje pracę innych części komputera
- Zapasowe urządzenia do przechowywania danych
  - Tanie, długi okres przechowywania, bardzo duża pojemność ,
  - Zawiera nieaktywne programy

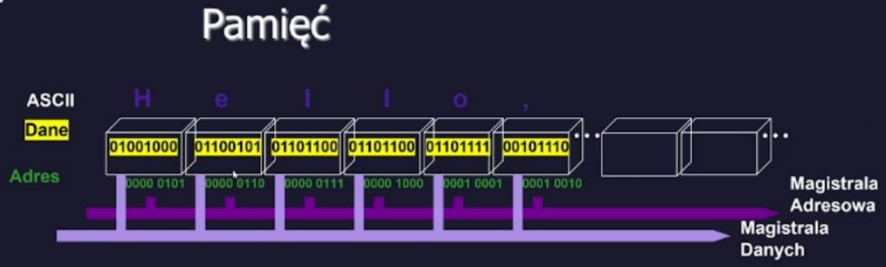
Elementy logiczne zawarte w typowym komputerze (kont.)

- Jednostka centralna - Central Processing Unit (CPU) mózg "komputera, składający się z :
  - jednostki arytmetyczno- logicznej (ALU)
  - jednostki sterującej (CU): dekoduje każdą instrukcję maszyny i wysyła sygnały sterujące do innych komponentów, aby zrealizować , instrukcję.

Elementy logiczne zawarte w typowym komputerze (cont.)

- Pamięć
  - Komórki (słowa): jednostki zarządzalne; typowy rozmiar 8 bitów (1 bajt), inne maszyny są 16 bitowe (2 bajtowe) i jeszcze inne są 32 bitowe lub 64 bitowe
    - Bajt (8 bitów), KB (kilobajt, $10^3 \approx 210$ bajtów), MB (Megabajt, $10^6 \approx 220$ bajtów), GB (Gigabajt, $10^9 \approx 230$ bajtów).
    - Uwaga: k ≠ K ponieważ 1000 ≠ 1024.

Elementy logiczne zawarte w typowym komputerze (cont.)



- Organizacja oparta o bajtowy rozmiar komórki pamięci
- Pamięć komputera jest porównywalna do kolekcji ponumerowanych skrzynek pocztowych. Dla identyfikacji poszczególnych komórek w pamięci maszyny, każda komórka ma przypisane unikalne "nazwisko" - to znaczy jego adres

Magdalena Szymczyk

Architektura Von Neumanna

- Wykorzystywana jest jedna droga do przenoszenia danych i instrukcji pomiędzy pamięcią, I/O i CPU.
- Zaimplementowana jest ona jako magistrala co powoduje powstanie wąskiego gardła znanego jako wąskie gardło architektury von Neumanna.
- Odmianą tej architektury jest architektura Harvardzka, która rozdziela dane i instrukcje na dwie ścieżki (jak na przykład procesory Microchip PIC)
- Inna odmiana, stosowana w większości komputerów, jest wersją magistrali systemowej, w której istnieją różne magistrale między CPU i pamięcią i pamięcią oraz urządzeniami I/O
- Architektura von Neumanna działa w **cyklu pobierz – wykonaj.**
  - **Faza pobrania - instrukcja jest pobierana z pamięci o adresie wskazanym przez licznik programu.**
  - **Faza dekodowania – dekodowanie instrukcji w jednostce sterującej, operandy danych potrzebnych do wykonania instrukcji są pobierane z pamięci**
  - **Faza wykonania – wykonanie instrukcji w ALU, przechowywując wynik w rejestrze. Przeniesienie wyniku z powrotem do pamięci w razie potrzeby.**

Magdalena Szymczyk

Ogólne spojrzenie

Sprzęt i oprogramowanie składa się z warstw ułożonych hierarchicznie, z których każda niższa ukrywa szczegóły warstwy z poziomu wyżej.

Jednym z kluczowych interfejsów pomiędzy poziomami abstrakcji jest architektura zestawu rozkazów: interfejs pomiędzy sprzętem i oprogramowaniem niskiego poziomu.

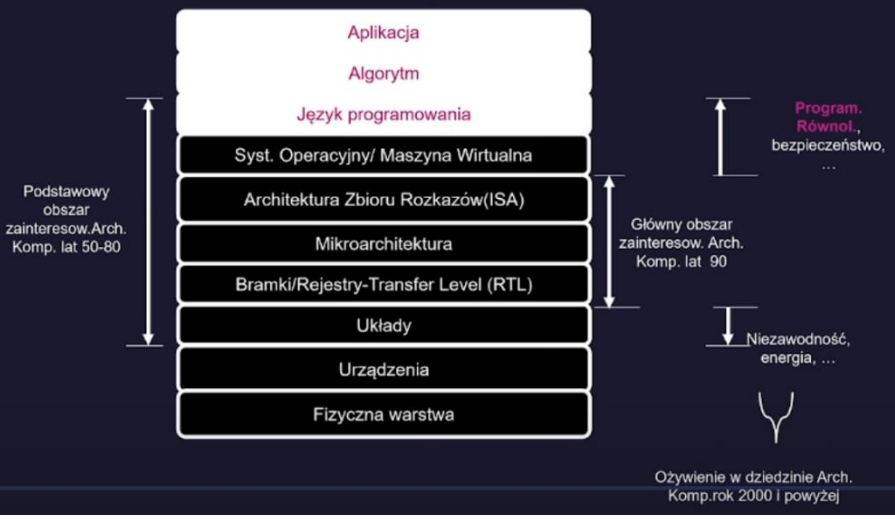
Ta zasada abstrakcji jest sposobem radzenia sobie ze złożonością systemów komputerowych zarówno dla projektantów sprzętu jak i projektantów oprogramowania.

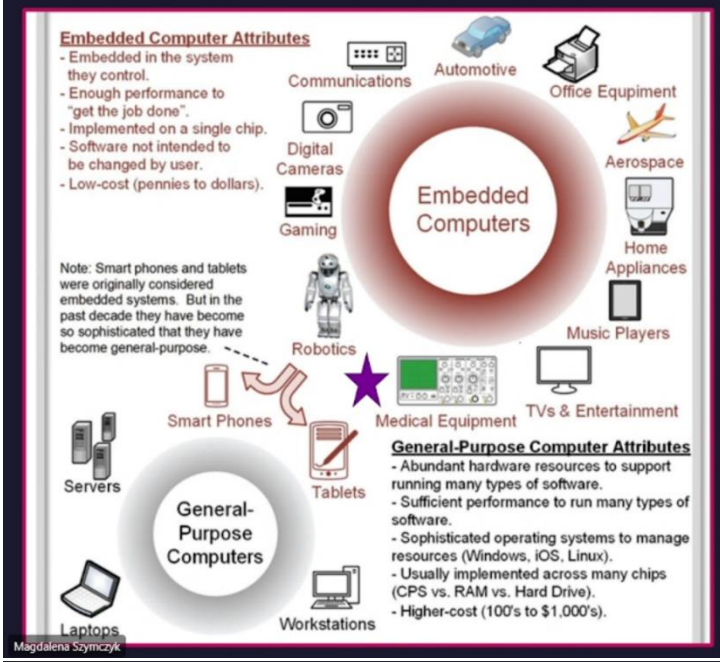
Ten abstrakcyjny interfejs umożliwia realizację wielu implementacji różniących się kosztami i wydajnością, a uruchamiających identyczne oprogramowanie.

John L. Hennessy

David A. Patterson

Poziomy abstrakcji w nowoczesnych systemach komp.



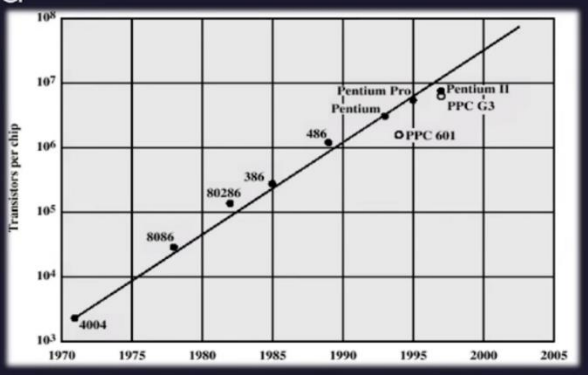


Obszar zastosowań systemów wbudowanych oraz systemów komputerowych ogólnego zastosowania

Trendy – prawo Moore'a

Gordon Moore (założyciel Intela) zauważył, że gęstość upakowania tranzystorów zwiększa się o współczynnik 2 co 2 lata

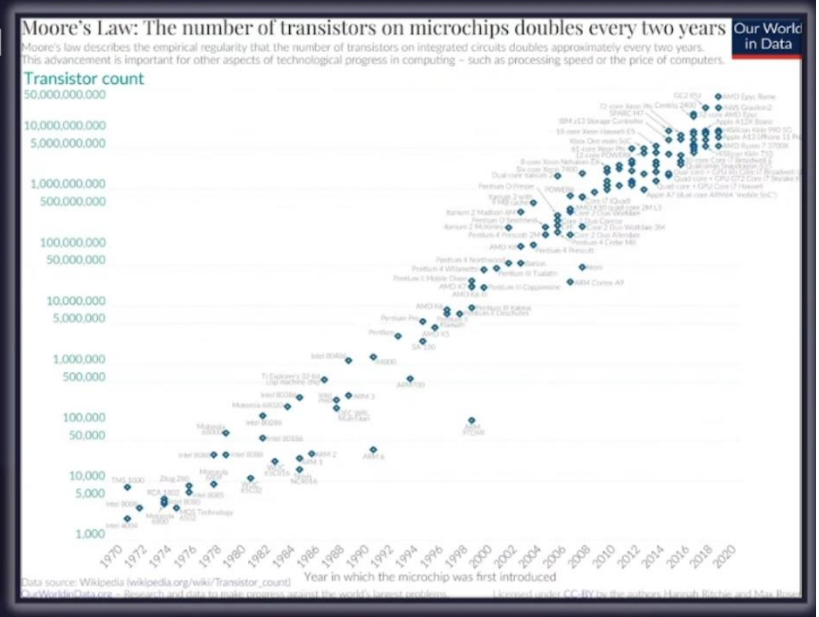
Tej obserwacji dokonał w 1965 roku



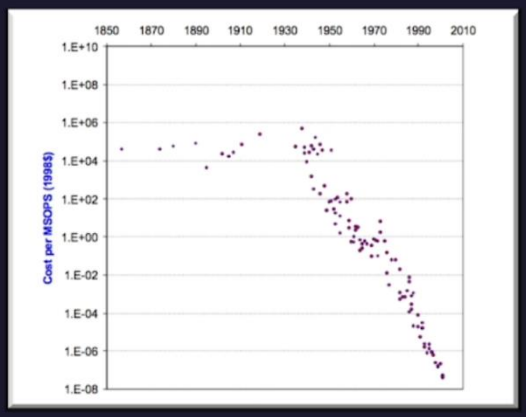
Wzrost upakowania oznacza wzrost liczby tranzystorów (a więc także pojemności pamięci i wydajności CPU).

Czy prawo Moore'a działa nadal?

Prawo Moore'a



Koszt procesorów /MOPS

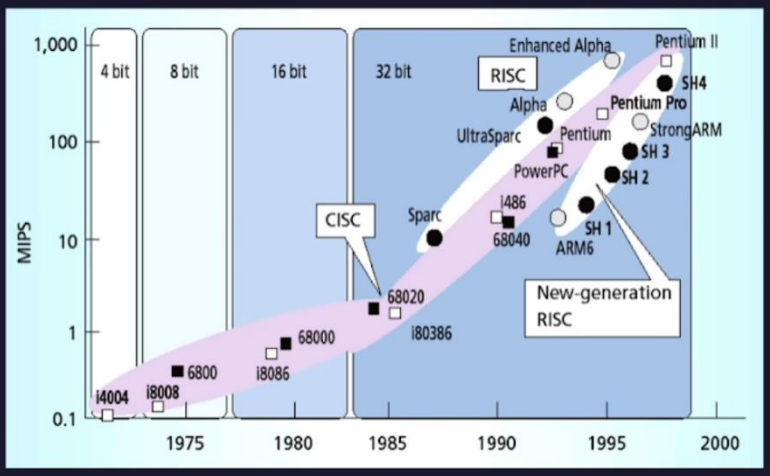


Prawo Moore'a

- Zwiększona gęstość elementów w jednym układzie
- Liczba tranzystorów w układzie scalonym podwaja się co roku
- Ponieważ rozwój od 1970 roku zwolnił trochę, liczba tranzystorów podwaja się co 18 miesięcy
- Koszt chipów pozostał na prawie niezmienionym poziomie
- Wyższa gęstość upakowania oznacza skrócenie połączeń elektrycznych, zapewniając wyższą wydajność
- Mniejszy rozmiar daje większą elastyczność
- Redukcja wydatków na energię i chłodzenie
- Mniejsza liczba krótszych połączeń zwiększa niezawodność

Magdalena Szymczyk

Wydajność Procesorów



("The Cooler the Better: New Directions in the Nomadic Ages," *Computer*, April 2001.)

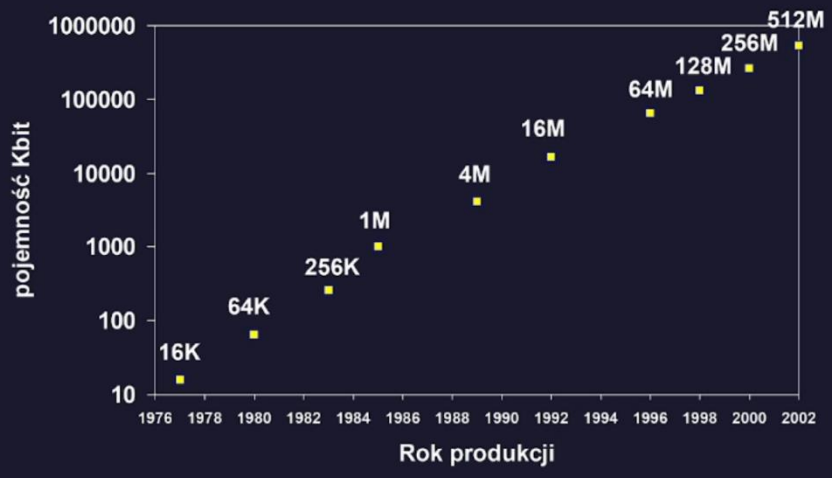
Magdalena Szymczyk

Przyczyny tak znacznego wzrostu wydajności procesorów

- Technologia
  - Więcej tranzystorów na układ
  - szybsze logika cyfrowa
- Organizacja komputerów / Implementacja
  - potokowanie
  - więcej instrukcji wykonywanych równolegle
- Instruction Set Architecture
  - Komputery o zredukowanej liście instrukcji(RISC)
  - Rozszerzenia dla multimediów
  - Jawna równoległość
- Technologia kompilatorów
  - znalezienie więcej równoległości w kodzie
  - więcej poziomów optymalizacji

Magdalena Szymczyk

Wzrost pojemności pamięci DRAM

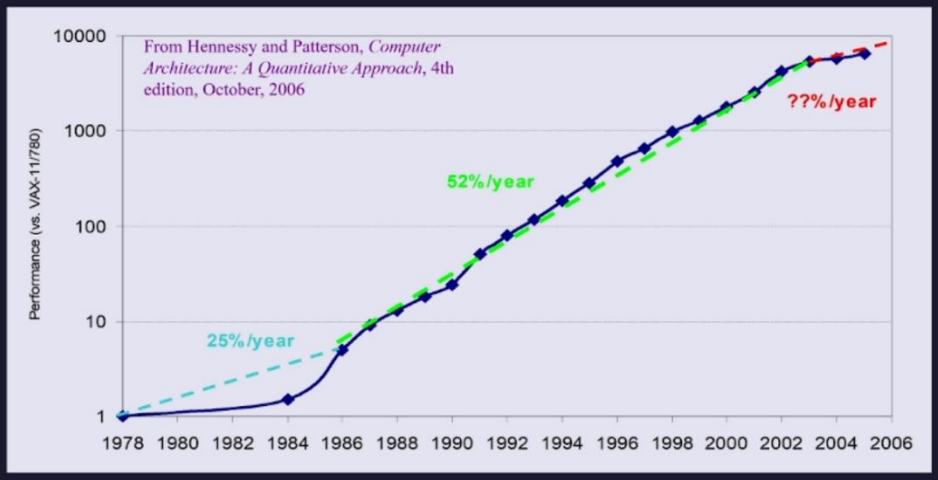


Magdalena Szymczyk

Wpływ coraz to lepszej technologii na elementy systemu komputerowego

- Procesor
  - pojemność ukł.cyfrowych: zwiększa się około 30% na rok
  - wydajność: 2x na każde 1.5 roku
- Pamięć
  - Pamięć DRAM : 4x na każde 3 lata, teraz 2x na każde 2 lata
  - Prędkość pamięci : 1.5x na każde 10 lat
  - Koszt na bit: zmniejsza się około 25% na rok
- Dysk
  - pojemność: zwiększa się około 60% na rok

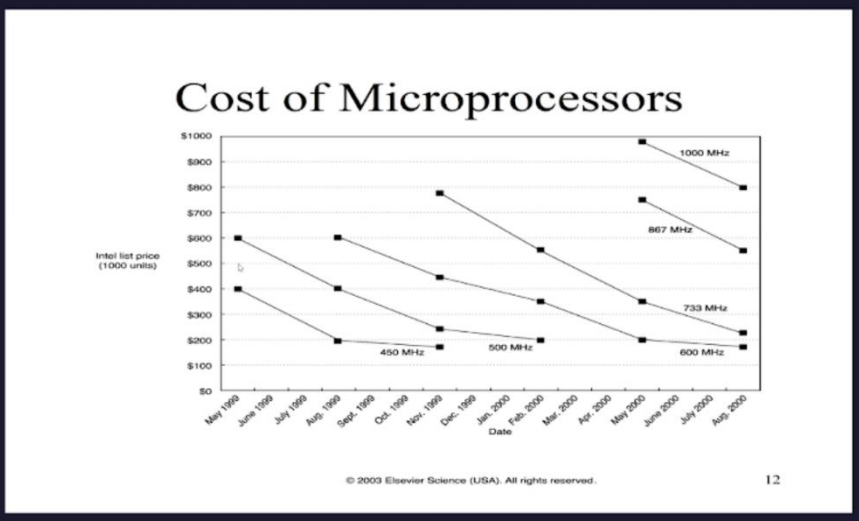
Wydajność procesorów jednordzeniowych



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

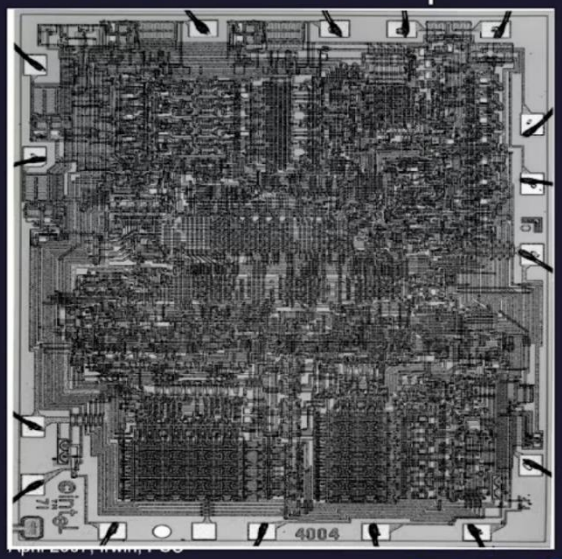
Magdalena Szymczyk

Koszt mikroprocesorów



Magdalena Szymczyk

Intel 4004 Microprocessor



1971

0.2 MHz clock

3 mm^2 matryca

10,000 nm rozmiar elementu

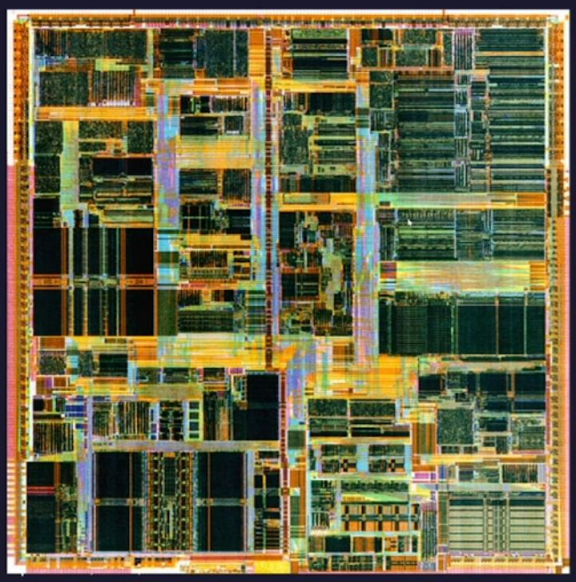
~2,300 tranzystorów

2mW zasilanie

Courtesy, Intel ®

Magdalena Szymczyk

Mikroprocesor Intel Pentium (IV)



2001
30 (15*2) lat

1.7 GHz zegar
8500x szybszy

271 mm^2 matryca
90x większa matryca

180 nm rozmiar elementu
55x mniejsze elementy

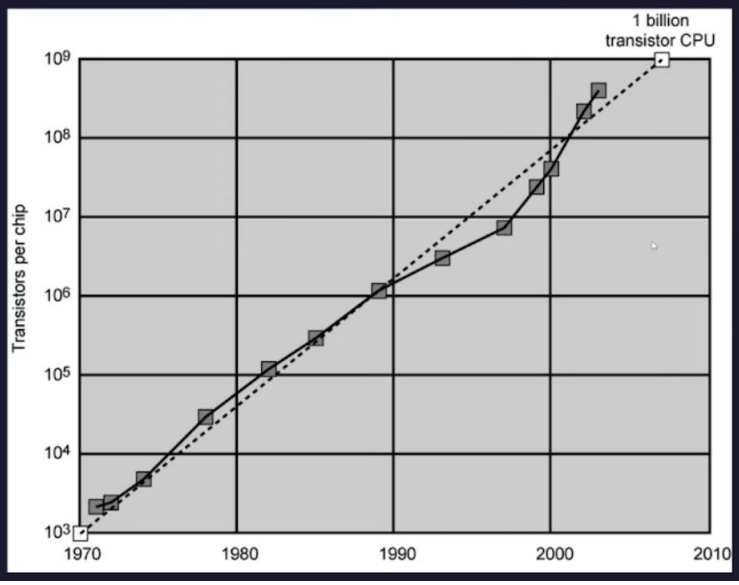
~42M tranzystorów
18,000x więcej tr.

64W zasilanie
32,000x (2^{15}) większe zasilanie

Courtesy, Intel ®

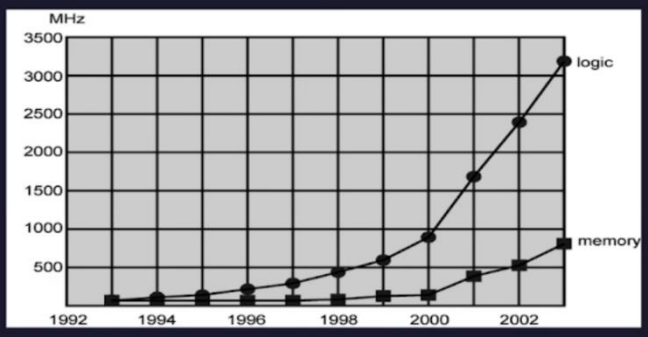
Magdalena Szymczyk

Zwiększenie liczby tranzystorów na chip



Magdalena Szymczyk

Przepaść technologiczna pomiędzy logiką cyfrową a pamięcią

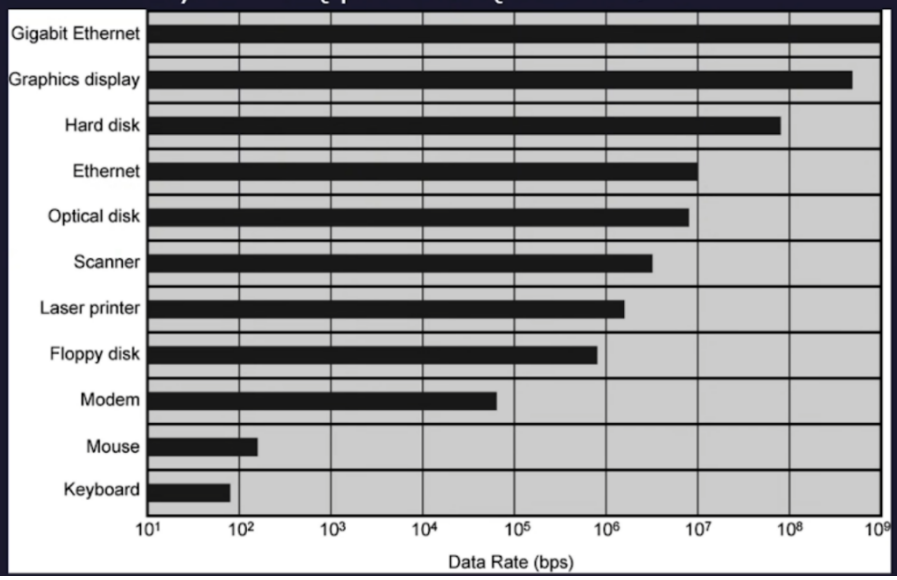


Szybkość procesora wzrosła
Pojemność pamięci wzrosła
Szybkość pamięci pozostaje w tyle za szybkością procesora

Rozwiązanie tego problemu

- Zwiększenie liczby bitów pobieranych naraz
  - Dołożyć DRAM "szerszy" niż "głębszy"
- Zmiana interfejsu pamięci DRAM
  - cache
- Zmniejszenie częstotliwości dostępu do pamięci
  - Bardziej złożony cache
  - pamięć podręczna umieszczona na chipie
- Zwiększenie przepustowości połączeń
  - Wysoka prędkość magistral
  - Hierarchia magistral

Typowe czasy dostępu urządzeń I/O



Magdalena Szymczyk

Peryferia (urządzenia I/O)

- Peryferia o dużych żądaniach I/O
- Wymagana jest duża przepustowość danych
- Procesory mogą sobie z tym poradzić
- Problem jest z przenoszeniem danych
- Rozwiązania:
  - pamięć podręczna
  - buforowanie
  - wyższe prędkości działania magistral
  - bardziej złożone struktury magistral
  - konfiguracje wieloprocesorowe

Klucz to równowaga

- Elementy procesora
- Pamięć główna
- Urządzenia I/O
- Struktura połączeń

Poprawa organizacji i architektury układu scalonego

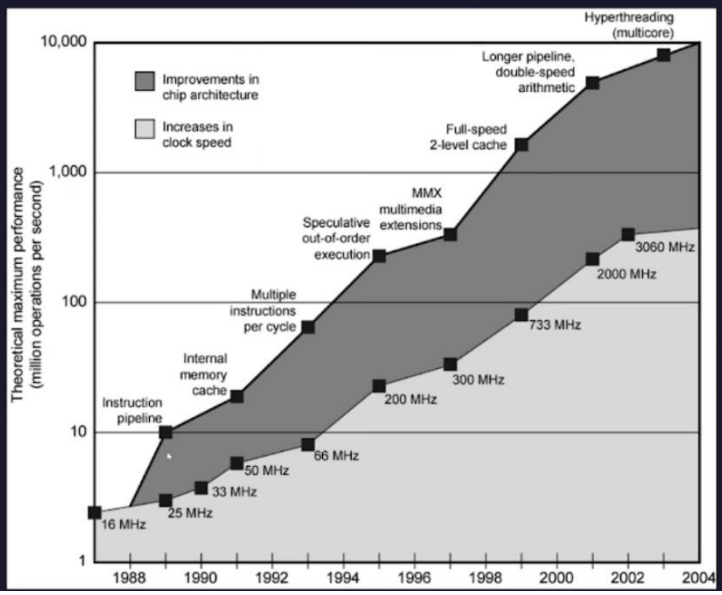
- zwiększenie prędkości działania sprzętu procesora
  - Zasadniczo spowodowane kurczeniem się wielkości bramek
    - Więcej bramek, spakowane mocniej, zwiększa się szybkość zegara
    - Czas propagacji sygnałów zmniejsza się
- Zwiększenie wielkości i szybkości działania pamięci podręcznej
  - część powierzchni procesora zajmuje cache
  - czas dostępu do pamięci podręcznej znacznie zmniejszył się
- Zmiana organizacji procesora i jego architektury
  - Zwiększenie efektywnej prędkości realizacji instrukcji
  - równoległość

Problemy z prędkością taktowania zegara oraz z gęstością upakowania ukł. cyfrowych

- Moc
  - Zapotrzebowanie na moc zwiększa się wraz z gęstością logiki i prędkością zegara
  - rozpraszanie ciepła
- Opóźnienia typu RC
  - Prędkość, z którą przepływają elektrony jest ograniczony rezystancją i pojemnością przewodów metalowych
  - Opóźnienie zwiększa się wraz z zwiększającym się iloczynem RC
  - Przewody łączące cieńsze, zwiększenie oporności
  - Przewody bliżej siebie, zwiększa się pojemność
- Opóźnienia w dostępie do pamięci
  - Prędkości pamięci ma wpływ na szybkość działania procesora
- Rozwiązanie:
  - Większy nacisk na rozwiązania organizacyjne i architektoniczne

Magdalena Szymczyk

Wydajność mikroprocesora Intel



Magdalena Szymczyk

Zwiększenie pojemności pamięci podręcznej

- Zwykle dwa lub trzy poziomy pamięci podręcznej pomiędzy procesorem i pamięcią główną
- Zwiększa się gęstość upakowania na chipie
  - Więcej pamięci cache na procesor
    - Szybszy dostęp do pamięci podręcznej
  - Procesor Pentium przeznacza około 10% obszaru układu dla pamięci podręcznej
  - Pentium 4 poświęca około 50%

Bardziej złożona budowa logiki wykonawczej

- Włączenie równoległego wykonywania instrukcji
- Potokowanie działa podobnie jak jak linia montażowa
  Na różnych etapach realizacji różne instrukcjie w tym samym czasie wzdłuż potoku
- Superskalarność pozwala na wiele potoków w jednym procesorze
- Instrukcje, które nie zależą od siebie mogą być wykonywane równolegle

Prognozy rozwoju procesorów

Organizacja wewnętrzna bardzo złożona współczesnych procesorów
Dalszy znaczący wzrost może być stosunkowo niewielki
Korzyści z pamięci podręcznej są bliskie osiągnięcia limitu
Zwiększanie częstotliwości zegara to problem strat mocy
Niektóre podstawowe fizyczne granice zostały już osiągnięte

Architektura syst. komputerowego

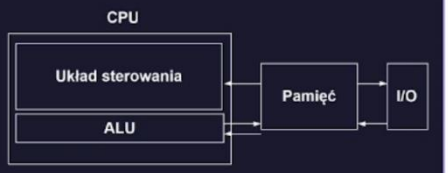
Architektura syst. komputerowego to atrybuty systemu, które mają bezpośredni wpływ na logiczne wykonanie programu.

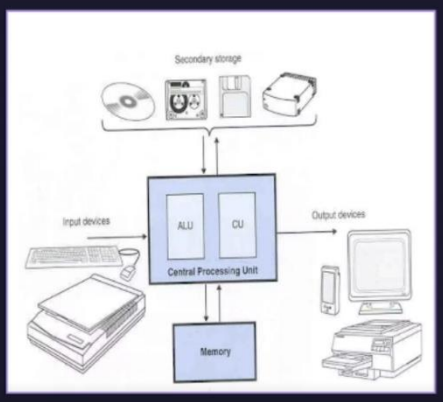
- Architektura komputera jest widoczna dla programisty poprzez:
  - zestaw instrukcji
  - reprezentację danych
  - mechanizmy I/O
  - Sposób adresowania pamięci

Organizacja syst. komputerowego

- Są to szczegóły budowy fizycznej, które są przezroczyste dla programisty, takie jak:
  - realizacja sprzętowa instrukcji
  - sygnały sterujące
  - technologia wykorzystywanych pamięci

Typowa architektura Von-Neumanna





1. Urządzenia wejściowe
2. Urządzenia wyjściowe
3. Pamięć
4. Jedn. arytm-logiczna(ALU)
5. Jednostka sterująca(CPU)
6. Zapasowe urządzenia do przechowywania danych

Elementy logiczne zawarte w typowym komputerze

- Urządzenie wejściowe
  - Pobiera informacje (klawiatura, myszka)
- Urządzenie wyjściowe
  - Wyprowadza informacje (na ekran, na drukarkę, jako sterowanie do innych urządzeń)
- Pamięć
  - Szybki dostęp, mała pojemność, składuje informacje wejściowe
  - ROM (Read Only Memory): CMOS, EPROM …
  - RAM (Random Access Memory): SRAM, DRAM, SIMM, DIMM …

Hierarchiczna budowa komputera

Każda maszyna wirtualna jest warstwą abstrakcji dla poziomu poniżej.

Maszyna na każdym poziomie realizuje własne szczególne instrukcje, używając maszyn na niższych poziomach do wykonywania zadań zgodnie z wymaganiami.

Układy elektroniczne komputera ostatecznie wykonują pracę.

Magdalena Szymczyk

Hierarchiczna budowa komputera

Poziom 6: Poziom użytkownika

- Wykonywanie programów i poziom interfejsu dla użytkownika.
- Poziom, z którymi jesteśmy najbardziej zaznajomieni.

Poziom 5: Poziom języka wysokiego poziomu

- Poziom z którym współpracujemy podczas pisania programów w językach takich jak C, Pascal, Lisp i Java.

Poziom 4: Poziom asemblera

- Pracuje na instrukcjach asemblera wyprodukowanych na bazie Poziomu 5, jak również instrukcji zaprogramowanych bezpośrednio na tym poziomie.

Poziom 3: Poziom systemu operacyjnego

- Kontrola realizacji procesów w systemie.
- Ochrona zasobów systemowych.
- Instrukcje asemblera często przechodzą poziom 3 bez zmian.

Poziom 2: Poziom maszynowy

- Znany także jako poziom architektury zestawu rozkazów (ISA) .
- Składa się z instrukcji, które są szczególne dla architektury maszyny.
- Programy napisane w języku maszynowym, nie potrzebują kompilatorów, interpreterów, czy też asemblerów

Magdalena Szymczyk

Hierarchiczna budowa komputera

Poziom 1: Poziom sterowania

- Układ sterowania dekoduje i wykonuje instrukcje oraz przemieszcza dane porzez system.
- Jednostka sterująca może być zrealizowana programowo lub sprzętowo.
- Mikroprogram to program napisany w języku niskiego poziomu, który jest realizowany przez sprzęt.
- Jednostki sprzętowe układu sterującego zbudowane są z elektroniki, która bezpośrednio wykonuje instrukcje maszyny.

Poziom 0: Poziom logiki cyfrowej

- Jest to poziom układów cyfrowych (chipów).
- Układy cyfrowe składają się z bramek oraz ścieżek.
- Te komponenty realizują logikę matematyczną dla wszystkich innych poziomów.

Magdalena Szymczyk

Hierarchiczna struktura oprogramowania komputera

- Komputery składają się z wielu dodatkowych elementów oprócz układów scalonych.
- Komputer aby wykonać polecenie, musi także korzystać z oprogramowania.
- Pisanie skomplikowanych programów wymaga strategii "dziel i rządź", w której każdy moduł programu rozwiązuje mniejszy problem.
- Złożone systemy informatyczne wykorzystują podobną technikę poprzez szereg wirtualnych warstw maszyny.

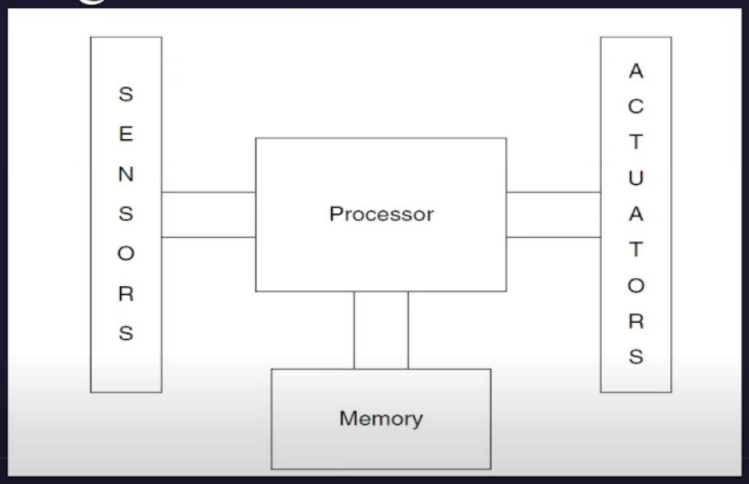
Magdalena Szymczyk

Motto

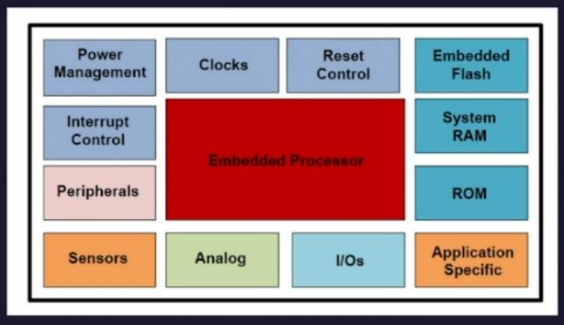
Abstrakcja jest naszym jedynym psychologicznym narzędziem w celu opanowania złożoności.

Edsger Dijkstra

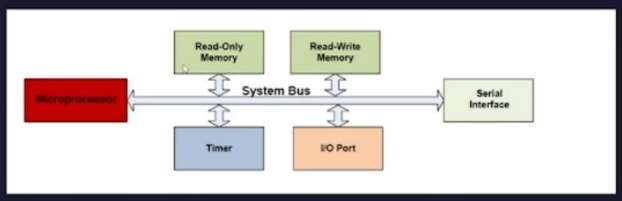
Ogólny schemat blokowy systemu wbudowanego



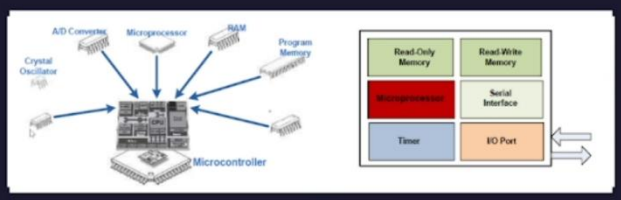
Elementy systemu wbudowanego



Mikroprocesor kontra mikrokontroler
System bazujący na mikroprocesorze



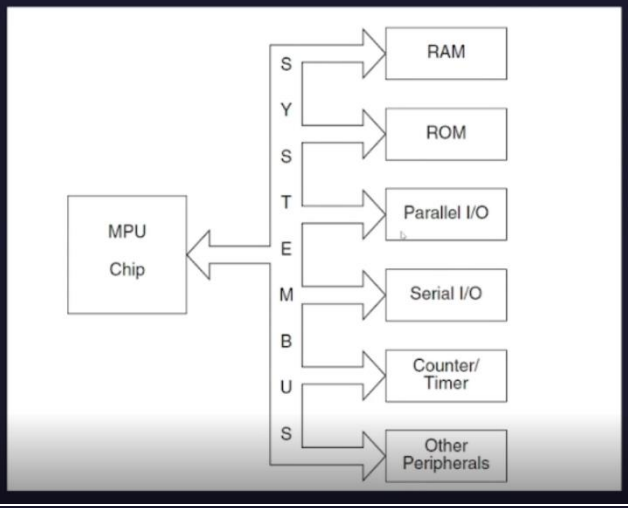
Mikroprocesor kontra mikrokontroler
System bazujący na mikrokontrolerze



Co to jest mikroprocesor?

- Układ krzemowy reprezentujący Jednostkę Centralną (CPU), która jest zdolna do wykonywania operacji arytmetycznych, jak również logicznych zgodnie z wcześniej zdefiniowanym zestawem instrukcji, który jest specyficzny dla danego producenta.
- Ogólnie rzecz biorąc, CPU zawiera jednostkę arytmetyczno-logiczną (ALU), jednostkę sterującą i rejestry robocze
- Mikroprocesor jest jednostką zależną i do prawidłowego funkcjonowania wymaga połączenia z innym sprzętem, takim jak pamięć, timery, kontroler przerwań itp.

Co to jest mikroprocesor?Cd.



Co to jest mikrokontroler?

- Mikrokontroler to komputer na chipie lub, jeśli wolisz, komputer jednoukładowy. Mikro sugeruje, że urządzenie jest małe, a kontroler mówi, że urządzenię może być używane do kontrolowania obiektów, procesów lub zdarzeń. Innym terminem opisującym mikrokontroler jest kontroler wbudowany, ponieważ mikrokontroler i jego obwody pomocnicze są często wbudowane w urządzenia, którymi sterują."

The Microcontroller Idea Book, Jan Axelson

Co to jest mikrokontroler? Cd.

- Wysoce zintegrowany układ krzemowy zawierający procesor, pamięć RAM, tablice rejestrów specjalnego i ogólnego przeznaczenia, pamięć ROM/FLASH do przechowywania programu, timer i jednostki sterujące przerwaniami oraz dedykowane porty I/O.
- Mikrokontrolery mogą być uważane za nadzbiór mikroprocesorów.
- Mikrokontroler może być ogólnego przeznaczenia (jak Intel 8051, zaprojektowany dla ogólnych aplikacji i domen) lub specyficzny dla aplikacji (jak Automotive AVR z Atmel Corporation. Zaprojektowany specjalnie dla aplikacji samochodowych)
- Mikrokontrolery są tanie, efektywne kosztowo i są łatwo dostępne na rynku

Kryteria wyboru mikrokontrolera

- Wybór mikrokontrolera (MCU) jest kluczowym elementem projektowania systemu wbudowanego.
- Niewłaściwy wybór może prowadzić do opóźnień, wzrostu kosztów i awarii systemu.

Cele procesu wyboru

- Celem jest wybór najtańszego MCU, który spełnia wymagania systemu i zapewnia niezawodność.
- Projektant powinien dążyć do równowagi między kosztami, wydajnością i dostępnością.

Etapy selekcji mikrokontrolera

- 1. Określenie wymagań funkcjonalnych.
- 2. Wybór architektury.
- 3. Analiza dokumentacji.
- 4. Ocena kosztów i narzędzi.
- 5. Testy prototypowe.

Pytanie kluczowe

- „Co mikrokontroler ma robić w moim systemie?" – to pytanie definiuje wymagane cechy i funkcje MCU.

Klasyfikacja mikrokontrolerów

- Mikrokontrolery dzielą się na 8-, 16- i 32-bitowe.
- Rozmiar magistrali wpływa na wydajność, zużycie energii i koszt.

Wybór 8-, 16-, czy 32-bitowego MCU

- 8-bit: niski koszt, mała moc, proste aplikacje.
- 16-bit: średnia wydajność, dobra równowaga.
- 32-bit: duża moc obliczeniowa, aplikacje zaawansowane (np. medyczne).

Częstotliwość zegara

- Wyższa częstotliwość zwiększa szybkość przetwarzania, ale rośnie pobór mocy.
- W systemach niskomocowych stosuje się zegary o zmiennej częstotliwości.

Pamięć mikrokontrolera

- Wielkość pamięci programu (Flash) i danych (RAM) musi odpowiadać złożoności aplikacji.
- Niektóre MCU oferują możliwość rozszerzenia pamięci zewnętrznej.

Zasoby wbudowane (peryferia)

- MCU zawiera wbudowane bloki funkcjonalne, np. timery, A/D, D/A, UART, SPI, I2C.
- Zwiększają one integrację i zmniejszają liczbę układów zewnętrznych.

Znaczenie peryferiów

- Dobór peryferiów wpływa na możliwości systemu.
- Należy analizować rozdzielczość timerów, dokładność przetworników i liczbę portów I/O.

Architektura mikrokontrolera

- Harvard: oddzielna pamięć programu i danych – szybsze działanie.
- Von Neumann: wspólna pamięć – większa elastyczność.
- Często stosowana modyfikowana architektura Harvard.

Przykłady rodzin mikrokontrolerów

- AVR – Atmel (Microchip)
- PIC – Microchip
- ARM Cortex-M – różni producenci
- MSP430 – Texas Instruments
- STM32 – STMicroelectronics

Zestaw instrukcji i rejestry

- Zestaw instrukcji decyduje o łatwości programowania i wydajności.
- Warto zwrócić uwagę na dostępność instrukcji arytmetycznych, bitowych i logicznych.

System przerwań

- Przerwania umożliwiają reakcję w czasie rzeczywistym.
- Kluczowe parametry: liczba źródeł przerwań, poziomy priorytetów, czas reakcji.

Narzędzia deweloperskie

- • Programatory i debugery
- • IDE (np. MPLAB, STM32CubeIDE, Atmel Studio)
- • Kompilatory i symulatory
- • Wsparcie dla systemów RTOS

Aspekty fizyczne i obudowa

- Obudowy: DIP, QFP, BGA.
- QFP – łatwiejszy montaż.
- BGA – mniejszy rozmiar i lepsze chłodzenie.
- Wybór zależy od przestrzeni, kosztu i bezpieczeństwa projektu.

Czynniki ekonomiczne i dostępność

- Należy uwzględnić koszt jednostkowy, dostępność w długim czasie i wsparcie producenta.
- Analiza TCO (Total Cost of Ownership) pomaga w wyborze optymalnym ekonomicznie.

Niezawodność i bezpieczeństwo

- MCU z watchdogiem, wykrywaniem błędów i zabezpieczeniami pamięci zwiększają niezawodność.
- W systemach krytycznych liczy się certyfikacja i odporność EMC/EMI.

Podsumowanie

- Wybór MCU to proces analityczny i iteracyjny.
- Decyzja wpływa na całość systemu wbudowanego.
- Świadomy wybór gwarantuje sukces projektu, niezawodność i efektywność kosztową.

System komputerowy ogólnego przeznaczenia vs System wbudowany

System będący połączeniem sprzętu ogólnego przeznaczenia i systemu operacyjnego ogólnego przeznaczenia do wykonywania różnych aplikacji.

- Zawiera system operacyjny ogólnego przeznaczenia (GPOS).
- Aplikacje są modyfikowalne (programowalne) przez użytkownika (możliwe jest, aby użytkownik końcowy przeinstalował system operacyjny i dodał lub usunął użytkownika).
- Wydajność jest kluczowym czynnikiem decydującym o wyborze systemu. Zawsze "szybciej znaczy lepiej".
- Mniej/niezupełnie dostosowane do zmniejszonego zapotrzebowania na energię operacyjną, opcje różnych poziomów zarządzania energią.
- Wymagania dotyczące czasu reakcji nie są krytyczne
- Nie musi być deterministyczny w wykonaniu zachowania

System będący połączeniem sprzętu specjalnego przeznaczenia i wbudowanego systemu operacyjnego do wykonywania określonego zestawu aplikacji.

- Może zawierać lub nie system operacyjny.
- Firmware systemu wbudowanego jest zaprogramowany i nie jest zmieniany przez użytkownika końcowego.
- Wymagania dotyczące aplikacji (takie jak wydajność, zapotrzebowanie na energię, wykorzystanie pamięci itp.) są głównymi czynnikami decydującymi
- W dużym stopniu dostosowane do wykorzystania trybów oszczędzania energii obsługiwanych przez sprzęt i system operacyjny
- Dla niektórych kategorii systemów wbudowanych, takich jak systemy krytyczne dla misji kosmicznych, wymagania dotyczące czasu reakcji są bardzo krytyczne
- Wykonanie jest deterministyczne dla niektórych typów systemów wbudowanych, takich jak systemy typu "Hard Real Time".

Sposoby reprezentacji liczb

- Liczby całkowite bez znaku
- Liczby całkowite ze znakiem
- BCD (Binary Coded Decimal)
- Liczby ze stałym przecinkiem
- Liczby zmiennoprzecinkowe

- Inne rodzaje danych:
  - Znaki (kody ASCII, Unicode)
  - Piksele (grafika)
  - Grupa bitów

Liczby całkowite bez znaku

- Dlaczego liczby całkowite?

  Dostęp do pamięci: rejestry PC, SP, RA

  W języku C - unsigned int

Jak je reprezentować?

Pozycyjny system liczbowy

Przykład system dziesiętny: 10 różnych symboli: 0 1 2 3 4 5 6 7 8 9

liczba= $$\sum_{i=0}^{n-1}(d_i * 10^i)$$

Liczby całkowite bez znaku

Ogólnie system liczbowy o k-wadze

liczba = $$\sum_{i=0}^{n-1}(d_i * k^i)$$

- Jak wiele różnych d_i ?
  Jaka jest największa liczba?
  Jaka jest najmniejsza?
- Szczególne przypadki:
  - → o podstawie 2 – binarny
  - → o podstawie 8 (ósemkowy)
  - → o podstawie 16 (heksadecymalny)

System pozycyjny – dziesiętny

Przykład:

$541.25_{10} = 5 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$

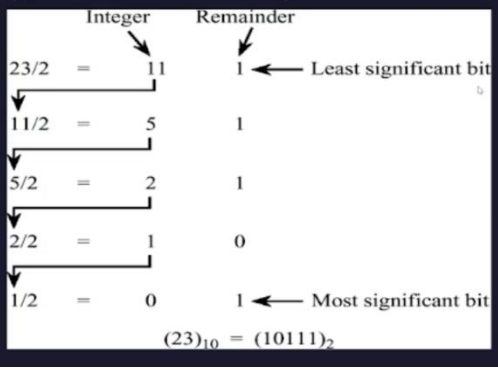
$= (500)_{10} + (40)_{10} + (1)_{10} + (2/10)_{10} + (5/100)_{10}$

$= (541.25)_{10}$

Sposób konwersji liczby dziesiętnej na binarną

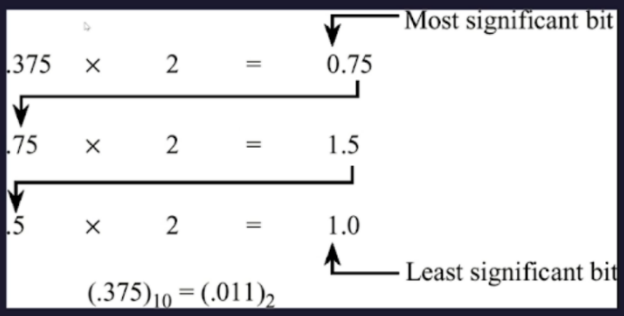
Konwersja 23.37510 do zapisu binarnego

Rozpoczynamy konwersję od części całkowitej:



Sposób konwersji liczby dziesiętnej na binarną dla części ułamkowej

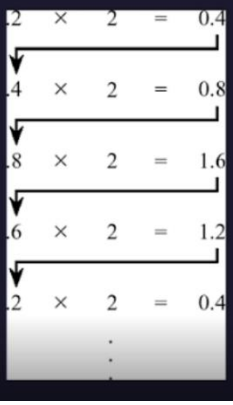
Dokonujemy konwersji części ułamkowej:



Łącząc wyniki razem dostajemy **$23.375_{10} = 10111.011_2$**

Ułamki okresowe w notacji binarnej

Nie zawsze możemy przekonwertować ułamek dziesiętny na nieokresowy ułamek binarny:



Liczby całkowite bez znaku

Konwertowanie liczb w zapisie heksadecymalnym na dwójkowy oraz ósemkowy:

$010100=(0*2^2+1*2^1+0*2^0)*2^3+(1*2^2+0*2^1+0*2^0)$

$=2*8^1+4*8^0=24_8$

Binarnie: 0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0

Magdalena Szymczyk

Więcej informacji na temat konwersji

- Ile bitów pobiera się dla każdej podstawy np. 4, 8, itd., ?

  Dla podstawy 2, w którym $2 = 2^1$, wykładnik jest 1 i jeden bit jest używany dla tej podstawy.

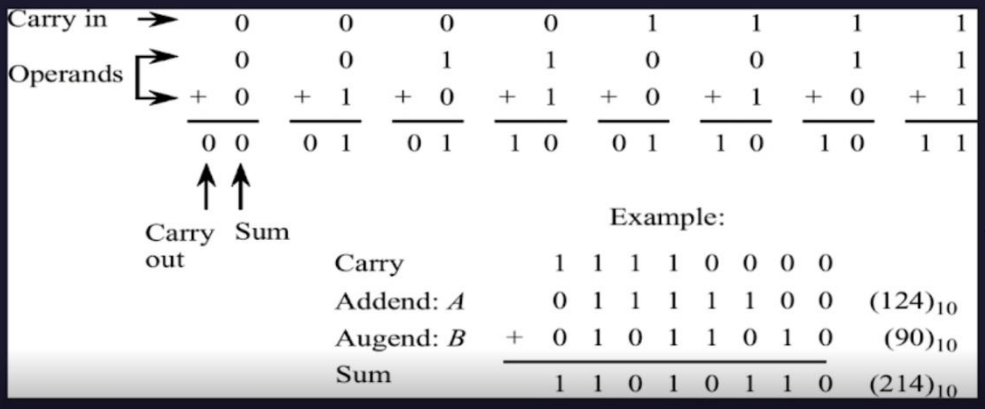
- Dla podstawy 4, w którym $4 = 2^2$, wykładnik jest 2 i dwa bity są używane dla tej podstawy.

- Podobnie, dla podstawy 8 i podstawy 16, $8 = 2^3$ i $16 = 2^4$, a więc 3 bity i 4 bity są używane dla bazy (inaczej podstawy) 8 i bazy 16 odpowiednio.

Której podstawy użyć?

- Liczby dziesiętne: szczególnie dla ludzi, zwłaszcza do liczenia
- Hex: jeśli człowiek patrzy na długie ciągi liczb binarnych i jest dla niego łatwiejszy zapis 4 bity to pewien symbol
- Binarnie: używana przez komputery, dowiesz się, jak komputery do +, -, *, /
  Komputer (procesor) zawsze korzysta z liczb binarnych.

Dodawanie binarne



Stałoprzecinkowe liczby ze znakiem

- Dla 8-bitowych liczb, jest $2^8 = 256$ możliwych wzorców bitowych.
- Te wzorce bitowe mogą reprezentować też liczby ujemne.
- Możemy przypisać połowę wzorców bitowych do liczb ujemnych i pół wzorców bitowych do liczb dodatnich.
  Cztery typy reprezentacji liczb ze znakiem które omówimy to:

Kod prosty - ang. Signed Magnitude

Kod uzupełnień do 1 – ang. One's Complement

Kod uzupełnień do 2 - Two's Complement

Z przesunięciem – ang. Excess (Biased)

Kod prosty

Znany również jako "znak i wartość bezwzględna,"

skrajny lewy bit jest znakiem (0 = dodatnia, 1= ujemny), a pozostałe bity są wartościami .

Przykład:

+25$_{10}$ = 00011001$_2$

-25$_{10}$ = 10011001$_2$

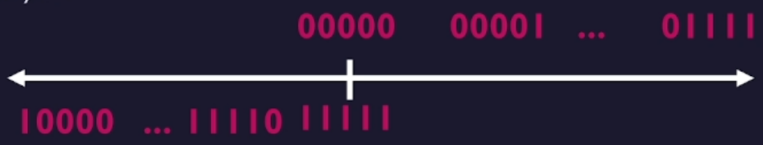
- Dwie reprezentacje wartości 0: +0 = 00000000$_2$, -0 = 10000000$_2$.
- Największa liczba to +127, najmniejsza liczba to -127$_{10}$, przy użyciu 8-bitowej reprezentacji danych

Braki tej notacji (kod prosty)

- Obwody realizujące operacje arytmetyczne dość skomplikowane
- Specjalne działania w zależności od tego, czy znaki są takie same czy też nie
- Dodatkowo, dwa zera
  - 0x00000000 = +0$_{ten}$
  - 0x80000000 = –0$_{ten}$
  - Problemy w programowaniu
- Dlatego też ten sposób zarzucono

Inna próba: kod uzupełnień do jeden

- Przykład: 7_{10} = 00111$_2$ -7_{10} = 11000$_2$
- Dopełnienie bitów
- Uwaga: liczby całkowite posiadają wiodące zera, liczby ujemne mają wiodące jedynki.



- Co oznacza -00000 ? Odp: 11111

Standardowa reprezentacja liczb całkowitych ze znakiem

- Reprezentacja **kod uzupełnień do dwóch**

Jaki będzie wynik dla liczb bez znaku, gdy chcemy odjąć

Wiekszą liczbę od mniejszej?

- $3 - 4 \Rightarrow 00\ldots0011 - 00\ldots0100 = 11\ldots1111$

- **Z dostępnych alternatyw lepiej wybrać reprezentację realizowaną na prostym sprzęcie**
- Podobnie jak w kodzie prostym, wiodące 0a ⇒ dodatnia liczba, wiodące 1a ⇒ ujemna
  - 000000...xxx to ≥ 0, 111111...xxx to < 0
  - Z wyłączeniem 1…1111 to -1, nie -0 (jak w kodzie prostym)

Magdalena Szymczyk

Kod uzupełnień do dwóch

Skrajny lewy bit jest znakiem (0 = dodatnia, 1 = ujemna).

Liczbę ujemną uzyskuje się przez dodanie 1 **do jej uzupełnienia do jeden.**

Zachodzi to w obie strony, w celu konwersji pomiędzy liczbami dodatnimi i ujemnymi.

Przykład (pamiętamy że -25_{10} w kodzie uzupełnień do jeden to 11100110_2):

$+25_{10} = 00011001_2$

$-25_{10} = 11100111_2$

- Jedna reprezentacja zera: $+0 = 00000000_2$, $-0 = 00000000_2$.
- Największa liczba $+127_{10}$, najmniejsza liczba -128_{10}, przy użyciu reprezentacji 8-bitowej.

Koncepcja programu

- Systemy programowane sprzętowo są nieelastyczne
- Komputer ogólnego przeznaczenia może wykonywać różne zadania, na postawie prawidłowych sygnałów sterujących
- Zamiast ponownego (innego) okablowania, dostarcza się nowy zestaw sygnałów sterujących

Co to jest program?

- Sekwencja kroków
- W każdym kroku, są wykonywane pewne operacje arytmetyczne lub logiczne
- Dla każdego działania - instrukcji, potrzebny jest inny zestaw sygnałów sterujących

Procesor komputera

(Computer Processing Unit inaczej CPU) jest to automat stanów skończonych - taki że wykonuje instrukcje przechowywane w pamięci.

Stan komputera jest określony przez wartości **w pamięci i rejestrach.**

Instrukcje

- Każda instrukcja określa szczególny sposób zmiany stanu systemu.
- W ramach nowego stanu określa się, która instrukcja powinna być wykonana jako następna.

Komputery z przechowywanym programem

- Instrukcje i dane są przechowywane w tej samej pamięci.
- Instrukcje mogą być traktowane jako dane w razie potrzeby (np. kompilator produkuje instrukcje jako dane wyjściowe. Są one przechowywane jako dane w pamięci i później wykonywane jako instrukcje).

Architektura zbioru rozkazów - Instruction Set Architectures (ISA)

- CPU ma stałą liczbę instrukcji, które może wykonywać.
- Każda instrukcja jest realizowana za pomocą obwodów elektronicznych zawartych w CPU
- Różne procesory mają własne (indywidualne) zestawy instrukcji.
- Zaskakująco mało instrukcji potrzebnych jest do zbudowania uniwersalnej maszyny Turinga

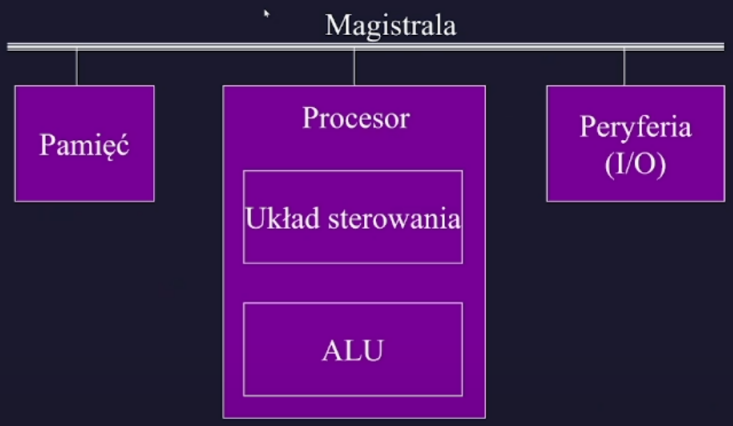
Model architektury von Neumanna

- Architektura oparta o przechowywanie programu spełnia co najmniej następujące warunki:
  - Składa się z trzech części :
    - jednostki centralnej (ang. CPU) w tym : jednostki sterującej, jednostki arytmetyczno-logicznej (ang. ALU), rejestrów (małe pamięci magazynowe) i licznika programu;
    - głównej pamięci systemu, która zawiera programy które kontrolują działanie komputera,
    - systemu I/O.

Posiada pamięć niezbędną do wykonywania kolejnych instrukcji

Zawiera jedną ścieżkę, fizyczną lub logiczną, pomiędzy głównym systemem a układem sterowania procesora, wymuszając na przemian cykl modyfikacji instrukcji i cykl jej wykonania. Ta jedna ścieżka jest często określana jako wąskie gardło von Neumanna.

Architektura von Neumanna



Architektura von Neumanna

- Zarówno dane jak i instrukcje programu reprezentowane są za pomocą bitów, a więc mogą być przechowywane we wspólnej pamięci.
- Rozróżnienie danych i instrukcji poprzez mechanizm dostępu i kontekst:
  - licznik programu daje dostęp do instrukcji,
  - adres rejestru daje dostęp do danych

Architektura von Neumanna

Architektura ta uruchamia programy w tzw. cyklu von Neumanna (zwanym także cyklem pobierz-dekoduj-wykonaj).

Jedna iteracja cyklu jest następująca:

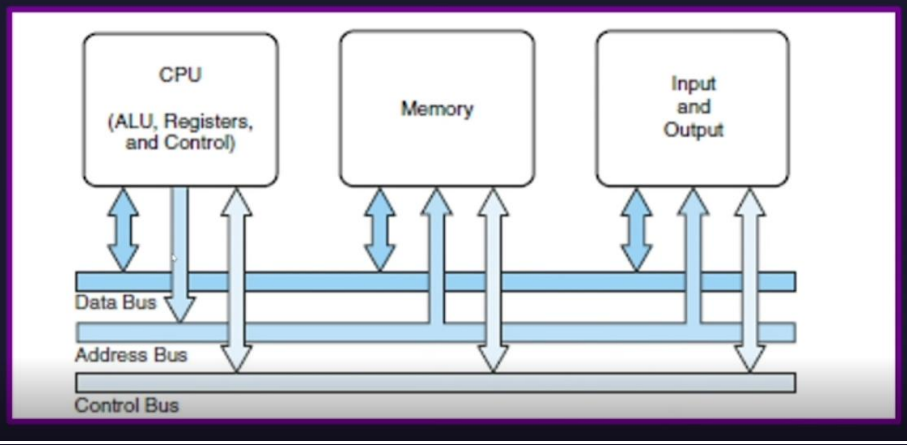
- Jednostka centralna pobiera następną instrukcję programu z pamięci, przy wykorzystaniu licznika programu (PC) w celu ustalenia, gdzie znajduje się ta instrukcja.
- Instrukcja jest dekodowana na język zrozumiały przez ALU.
- Wszelkie argumenty czyli dane potrzebne do wykonania instrukcji są pobierane z pamięci i umieszczone w rejestrach CPU.
- ALU wykonuje instrukcje i umieszcza wyniki w rejestrach lub pamięci.

Architektura von Neumanna

Rozszerzenia architektury von Neumanna

- Idea architektury von Neumanna została zmodyfikowana. Obecnie programy i dane przechowywane na nośniku o wolnym dostępie jak np. Dysk twardy, można przekopiować na ulotne szybkie pamięci takie jak np. RAM tuż przed realizacją polecenia.
- Architektura ta została również usprawniona magistralą systemową:
  - Magistrala danych przenosi dane z pamięci do rejestrów procesora (i odwrotnie).
  - Magistrala adresowa przechowuje adres danych, do których szyna danych ma aktualnie dostęp.
  - Magistrala sygnałów sterujących przesyła niezbędne sygnały sterujące, które określają, w jaki sposób przekazywanie informacji ma się odbywać.

Zmodyfikowana architektura von Neumanna wzbogacona o magistralę systemową



Inne modyfikacje architektury von Neumanna

Inne ulepszenia architektury von Neumanna to:

- rejestry indeksowe dla adresowania,
- dodanie danych zmiennoprzecinkowych,
- przerwania i asynchroniczne operacje I/O,
- pamięć wirtualna,
- dodanie rejestrów ogólnego przeznaczenia,
- równoległe operacje I/O,
- wieloprocesorowość.

Uwagi krytyczne do architektury von Neumanna

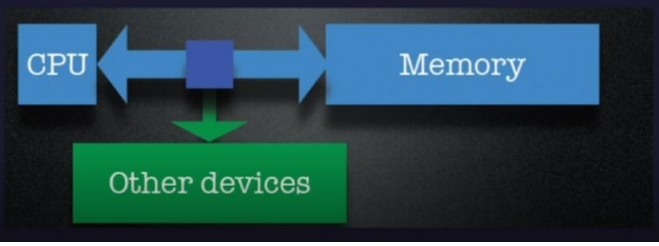
- Dane i program są nierozróżnialne z punktu widzenia zawartości pamięci.
- Kłopot z dostępem do tablic wielowymiarowych
- Znaczenie danych nie jest określone co prowadzi do bardziej złożonych programów

Architektura Harvardzka

- Podobna do architektury von Neumanna, z wyjątkiem tego, że program i dane umieszczone są w pamięci oddzielne wraz z osobnymi połączeniami do CPU.
- Potencjalnie szybsza niż architektura von Neumanna.

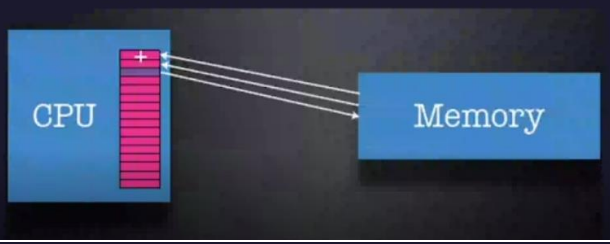
Pamięć komputera

- Programy są przechowywane w pamięci
- Programy zazwyczaj działają na danych przechowywanych w tej samej pamięci
- Istnieją też inne urządzenia (np. dyski, karty sieciowe) połączone ze sobą przez magistralę i pewnymi układami logiki

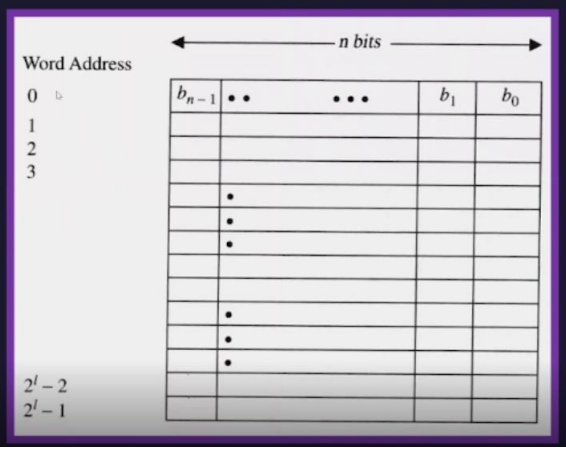


Hierarchia pamięci

- CPU nie działają bezpośrednio na danych w pamięci (zazwyczaj)
  - Używają rejestrów
- Wykorzystywane są instrukcje load, store do operacji czytania i zapisu do pamięci



Ilustracja graficzna pamięci komputera

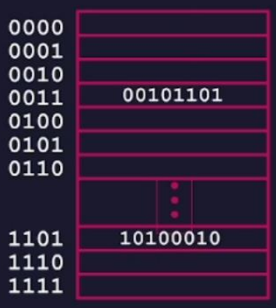


Pamięć

- *Tablica k* x *n* składowanych bitów

(*k* jest zwykle równe 2^l)

- Adres
  - unikalny (*l*-bitowy) identyfikator lokalizacji
- Zawartość
  - *n*-bitowa wartość składowana w lokalizacji
- Podstawowe operacje:
  - **LOAD**
    - odczytuje wartość z lokalizacji w pamięci
  - **STORE**
    - zapisuje wartość do miejsca w pamięci



Wykonanie instrukcji

- Ogólnie ALU wykonuje instrukcje, które przekształcają dane wejściowe na nowe dane wyjściowe
- Układ sterowania obsługuje kopiowanie i przenoszenia danych
  - na przykład jednostka sterująca wykonuje instrukcję MOVE, tak że pobiera adres pamięci i rejestr jako argumenty i kopiuje dane z jednej lokalizacji do drugiej:
  - Load, jeżeli z pamięci do rejestru
  - Store, jeśli z rejestru do pamięci
- ALU przetwarza dane przechowywane w rejestrach. Rejestry umożliwiają dostęp do danych znacznie szybciej niż poprzez dostęp do pamięci

Operacje na pamięci

- Fetch (adres) lub Load (adres)
  - Uzyskaj kopię tego, co jest zapisane w pamięci pod podanym adresem
  - Zwraca wartość do dalszego przetwarzania
  - Operacja niemodyfikująca wartości w pamięci
- Store (Address, value)
  - Składowanie szczególnej wartości w komórce pod adresem
  - Nowa wartość zastępuje starą
  - Operacja modyfikująca zawartość
- Czas dostępu do pamięci
  - Typowo to 10-75 nanosekund (ns) ???

Współpraca rejestrów z pamięciami

- Dwa rejestry procesora są wykorzystywane do połączenia procesora z pamięcią główną:
- rejestr pamięci adresu - MAR i rejestr pamięci danych - MDR.
  - MDR jest używany do przechowywania danych składowanych i/lub pobieranych do/z miejsca pamięci, a adres tej pamięci jest przechowywany w MAR

Interfejs z pamięcią (jak działa)

- Jak procesor pobiera dane do/z pamięci?
  MAR: Rejestr adresu pamięci
  MDR: Rejestr pamięci danych
- Aby odczytać położenie (A) w pamięci:
  1. Wpisz adres (A) w MAR
  2. Wyślij sygnał „read" do pamięci.
  3. Odczytaj dane z MDR.
- Aby zapisać wartość (X) do lokalizacji (A):
  1. Wpisz dane (X) do MDR.
  2. Wpisz adres (A) w MAR
  3. Wyślij sygnał „write" do pamięci.

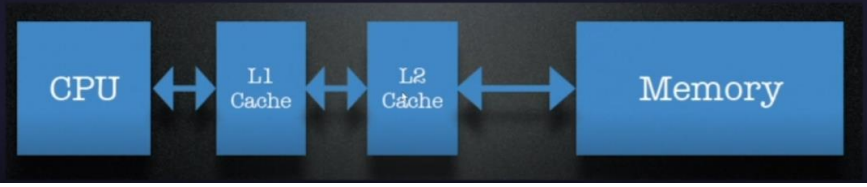
Hierarchia pamięci

• Aktualnie dostęp do pamięci znacznie wolniejszy niż szybkość działania procesora

• W przypadku nowoczesnych procesorów, pobieranie danych z pamięci, powodowałoby że procesor byłby przez 90% czasu bezczynny

Hierarchia pamięci - cache

- Rozwiązaniem jest pamięć podręczna
- Mała ilość szybkiej pamięci umieszczona między CPU i pamięcią
- Zawiera często używane instrukcje i dane



Hierarchia pamięci

Dlaczego tak jest to skomplikowane?

- Szybkie pamięci są drogie
- Nie można wyposażyć komputera w dużą ilość najszybszych pamięci

	Size	Bandwidth [GB/s]	Latency [ns]	Cost [$/MB]
Registers	512 bytes	-	<1	-
L1	32 kB	20	2-3	-
L2	4MB	20	6-10	10
Memory	1GB	2	50	0.10

Strategia działania pamięci podręcznej

- Pamięć podręczna jest niewidocznie obsługiwana sprzętowo
- Pamięć podręczna powiela dane zawarte w pamięci
  - Dane przekazywane w górę hierarchii (w kierunku CPU) na żądanie
  - Pozostają aż do momentu wyparcia przez bardziej pilne żądanie danych
- Jeśli żądanie może być spełnione przez cache, mamy tzw. trafienie
  - CPU wznawia przetwarzanie
- W przeciwnym razie żądanie jest przekazywane w dół hierarchii (brak danych)
  - CPU musi czekać

Typy pamięci

- Pamięć jest kluczowym elementem w architekturze przechowywanego programu komputerowego.
- Pamięć główna zapewnia miejsce do przechowywania informacji dla komputera.
- Istnieją dwa rodzaje pamięci: pamięć o dostępie swobodnym (RAM) i tylko do odczytu - pamięć (ROM).
  - Istnieją dwa rodzaje pamięci RAM, dynamiczne RAM (DRAM) i statyczne RAM (SRAM).
- Pamięć można podzielić na wiele sposobów:
  - Mając na uwadze technikę przechowywania (nowoczesne pamięci są półprzewodnikowe, stare używały rdzeni magnetycznych, linii opóźniających, rejestrów przesuwających i innych technik).
  - Czy dostęp jest swobodny czy sekwencyjny. W pamięci o dostępie swobodnym (RAM), dostęp do dowolnego miejsca w pamięci, niezależnie od jego adresu, zajmuje tyle samo czasu (choć może przybierać różne wartości dla czytania i zapisu).
  - Do odczytu/zapisu (RWM) lub tylko do odczytu (ROM).
  - Czy jest ulotna (tzn. traci zawartość po wyłączeniu zasilania) czy też nie
  - Czy jest destrukcyjna czy też nie (w destrukcyjnych pamięciach, czytanie słowa niszczy jego zawartość).

DRAM (Dynamic RAM)

- Najczęstszy typ pamięci komputera - dynamiczna , ponieważ musi być odświeżana lub ponownie doładowywana setki razy na sekundę, aby zachować dane w słowach pamięci.
- Każdy bit w słowie jest zbudowany w oparciu o mały kondensator, który może przechowywać ładunek elektryczny (naładowany kondensator oznacza bit równy 1).

SRAM (Static RAM)

- Ta pamięć jest pięć razy szybsza, dwa razy droższa i dwa razy większa niż DRAM.
- SRAM nie wymagają odświeżenia jak DRAM (każdy bit SRAM jest przerzutnikiem, układ który ma dwa stany stabilne, po umieszczeniu go w określonym stanie, pozostaje w nim). Pamięci te daje się szybciej czytać i zapisywać niż kondensatory pamięci DRAM, ale zużywają one więcej energii.

DRAM kontra SRAM

Ze względu na niższe koszty i mniejszy rozmiar, DRAM jest używana głównie do budowy pamięci głównej, a szybsze i droższe SRAM służą przede wszystkim do budowy pamięci podręcznej.

SRAM

- Statyczna pamięć RAM przechowuje każdy bit informacji w zatrzasku W jednym układzie scalonym (chipie) pamięci SRAM mogą się znajdować miliony zatrzasków
- Pojedynczy zatrzask i bramki wokół niego nazywane są komórką bitową (ang. Bit Cell)
- Poszczególne BC są zorganizowane w słowa, linie adresowe są dekodowane, aby stać się liniami selekcyjnymi, które wybierają BC poszczególnego słowa

ROM

- Najprostszy rodzaj pamięci ROM
- Zaprogramowana w momencie produkcji
- Jej zawartość nie może zostać później zmieniona
- ROM składa się z siatki M linii słów i N linii bitów w słowie, bez połączeń między liniami słów i liniami bitów w punktach sieci.
- N wyjść takiej siatki to same zera, dopiero w ostatnim etapie produkcji, dane są zapisywane w pamięci ROM przez wytwarzanie diody w każdym przecięciu siatki, gdzie jest wymagana 1 (tzw. proces maskowania)

PROM

- Kiedy tylko niewielka liczba jednostek ROM jest potrzebna, programowalny ROM (PROM) może być rozwiązaniem.
- Jest wykonany jako jednolita siatka słów i bitów, gdzie na każdym przecięciu siatki jest mała dioda, każde słowo w PROM składa się z samych cyfr równych 1.
- PROM jest zaprogramowany poprzez umieszczenie go w specjalnej maszynie, która łączy diody na tych przecięciach siatki, gdzie są wymagane zera.
- Zawartości PROM nie można zmienić

EPROM - kasowalny PROM

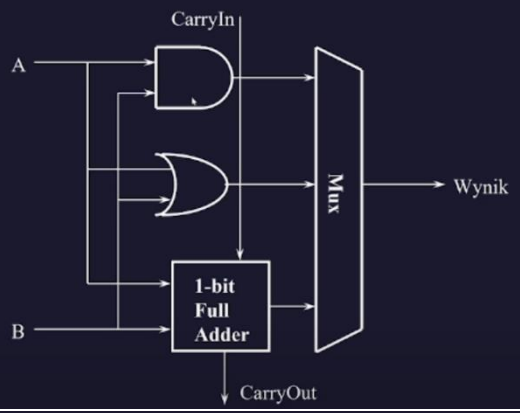
- Zawartość ROM można modyfikować kilka razy, programowanie odbywa się poprzez zatrzymywanie ładunków elektrycznych w tych punktach sieci, gdzie są potrzebne jedynki.
- Kasowanie odbywa się poprzez uwolnienie ładunków albo poprzez długotrwałe wystawianie urządzenia na działanie promieni ultrafioletowych lub przez podłączenie wysokiego napięcia do niego w kierunku przeciwnym, niż ładunki.

Pamięć typu flash

- Pamięć flash wyewoluowała z wymazywalnej programowalnej pamięci do odczytu (EPROM) i elektrycznie wymazywalnej programowalnej pamięci do odczytu (EEPROM).
- Flash jest technicznie odmianą EEPROM, ale branża zastrzega termin EEPROM dla pamięci wymazywalnej na poziomie bajtów i stosuje termin pamięć flash do większych pamięci wymazywalnych na poziomie bloków.
- Pamięć flash w telefonie komórkowym, karcie pamięci cyfrowej, tablecie lub dysku półprzewodnikowym jest formą EEPROM i technicznie jest uważana za ROM, mimo że nie jest nośnikiem tylko do odczytu. W przeciwieństwie do EEPROM, można zapisywać lub wymazywać w bloku, a nie po jednym bicie na raz.

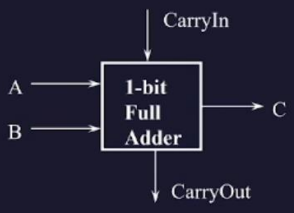
Jedno-bitowe ALU

- To 1-bitowe ALU wykonuje operacje AND, OR i ADD



Jedno-bitowy Full Adder

- Half Adder: brak sygnału CarryIn jak i CarryOut
- Tablica prawdy:



Wejścia			Wyjścia		
A	B	CarryIn	CarryOut	Suma	Komentarz
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

Wyrażenia logiczne dla CarryOut

Wejścia			Wyjścia		
A	B	CarryIn	CarryOut	Suma	Komentarz
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

- CarryOut = (!A & B & CarryIn) | (A & !B & CarryIn) |

(A & B & !CarryIn | (A & B & CarryIn)

- CarryOut = B & CarryIn | A & CarryIn | A & B

Magdalena Szymczyk

Wyrażenia logiczne dla Sumy

Wejścia			Wyjścia		
A	B	CarryIn	CarryOut	Suma	Komentarz
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

- Suma = (!A & !B & CarryIn) | (!A & B & !CarryIn) | (A & !B & !CarryIn) | (A & B & CarryIn)

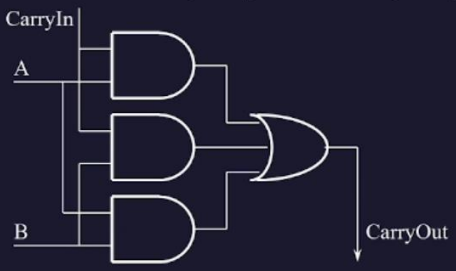
Wyrażenia logiczne dla Sumy (kont.)

- Suma = (!A & !B & CarryIn) | (!A & B & !CarryIn) | (A & !B & !CarryIn)| (A & B & CarryIn)
- Suma = A XOR B XOR CarryIn
- Tablica prawdy dla XOR:

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Diagramy logiczne dla CarryOut i Sumy

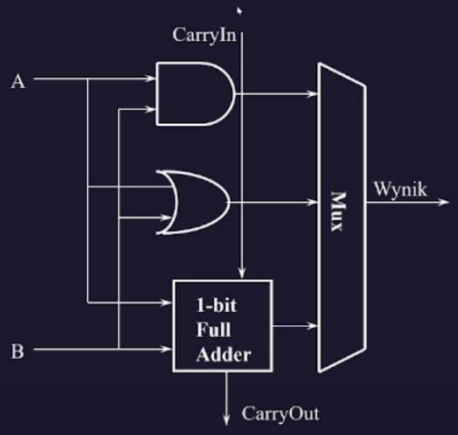
- CarryOut = B & CarryIn ¦ A & CarryIn | A & B



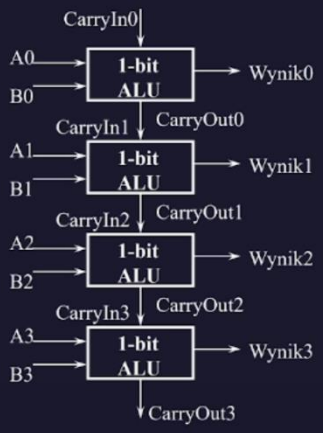
- Suma = A XOR B XOR CarryIn

4-bitowe ALU

- 1-bitowe ALU

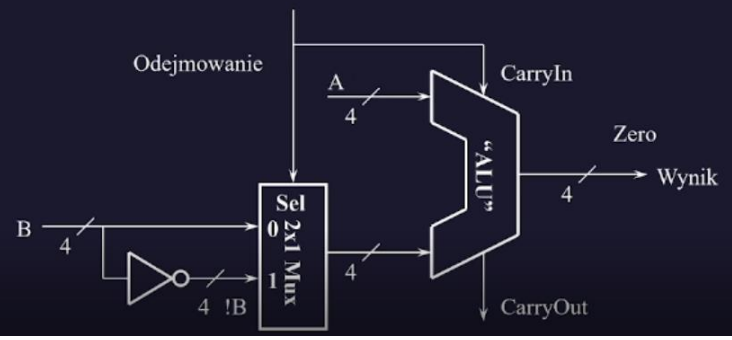


4-bitowe ALU



Odejmowanie

- Pamiętamy, że:
  - (A - B) to to samo co: A + (-B)
    - Kod uzupełnień do dwóch : zaneguj każdy bit i dodaj 1
  - A + !B + 1 = A + (!B + 1) = A + (-B) = A - B

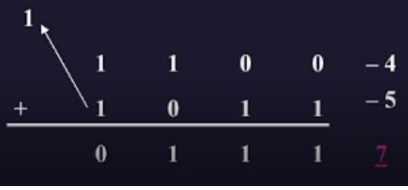


Przepełnienie - Overflow

Dziesiętnie	Binarnie
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Dziesiętnie	Kod uzupełnień do 2
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

- **Przykłady: 7 + 3 = 10 ale ...**
- **- 4 - 5 = - 9 ale ...**

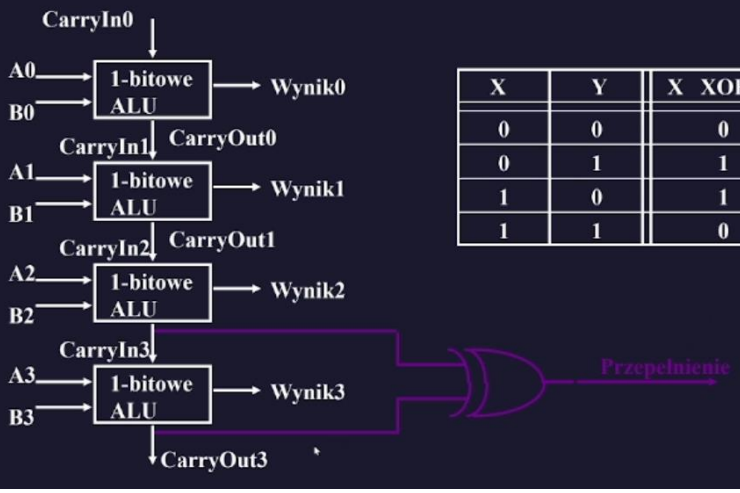


Detekcja przepełnienia (ang. overflow)

- Przepełnienie: wynik jest zbyt duży (lub zbyt mały) aby być reprezentowanym poprawnie
  - Przykłady: - 8 < = 4-bitowa liczba binarna <= 7
- Podczas dodawania argumentów o różnych znakach, przepełnienie nie może wystąpić!
- Przepełnienie występuje gdy dodajemy:
  - 2 dodatnie liczby i suma jest ujemna
  - 2 ujemne liczby i suma jest dodatnia

Układ logiczny do detekcji przepełnienia

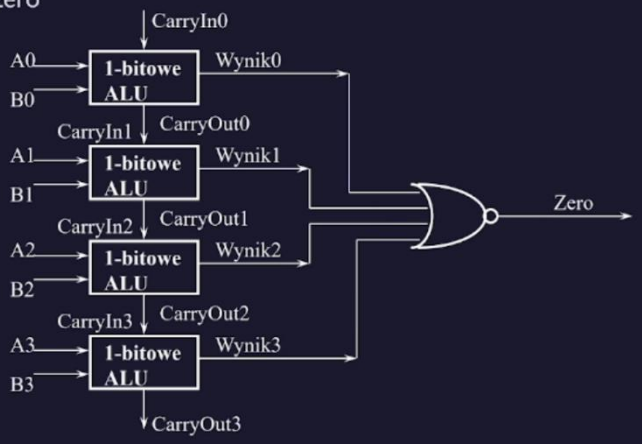
Dla N-bitowego ALU: Overflow = CarryIn[N - 1] XOR CarryOut[N - 1]



X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

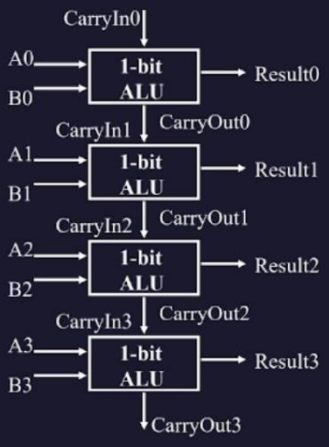
Układ logiczny do detekcji wartości zero

- Układ logiczny do detekcji ZERA to jedna wielka bramka NOR
- Każde niezerowe wejście na bramkę NOR powoduje że wyjście będzie równe zero

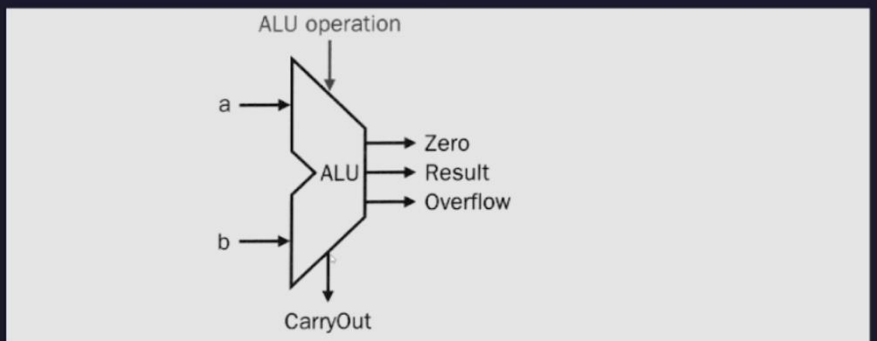


Wydajność

- Ścieżka krytyczna w n-bitowym Rippled-carry adder wynosi n*CP



ALU symbol



The symbol commonly used to represent an ALU, as shown in Figure [redacted]

Mnożenie

- Długotrwały proces mnożenia

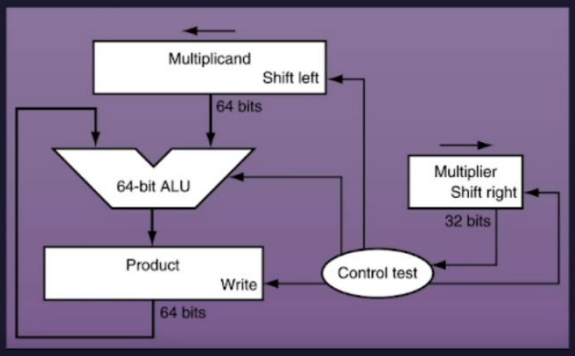
Mnożna multiplicand

Mnożnik multiplier

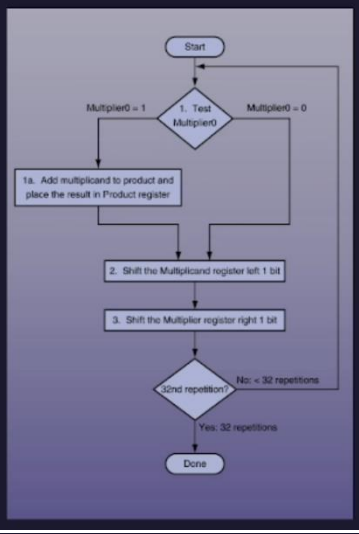
Wynik product

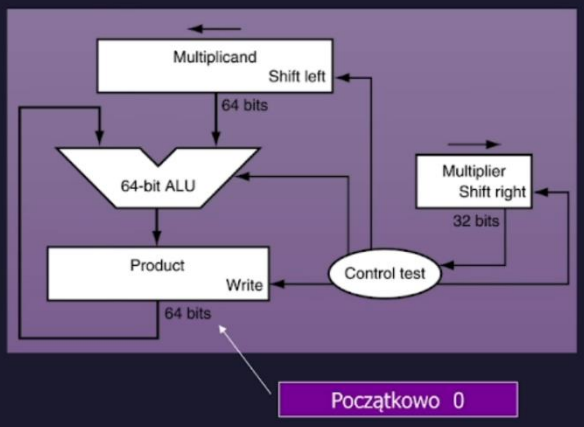
```
  1000
× 1001
------
  1000
 0000
0000
1000
-------
1001000
```

Długość wyniku jest sumą długości operandów



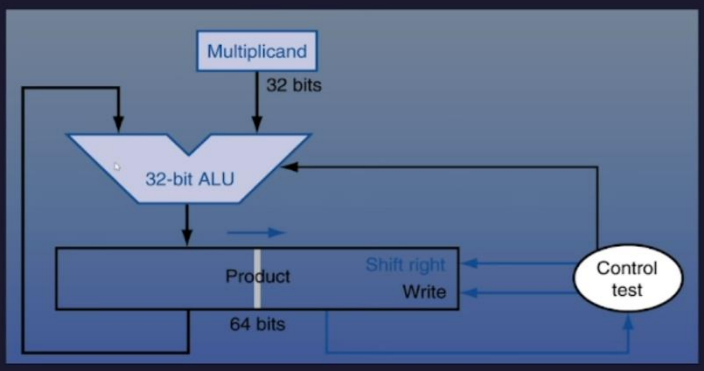
Mnożenie - sprzętowo





Zoptymalizowany ukł. mnożący

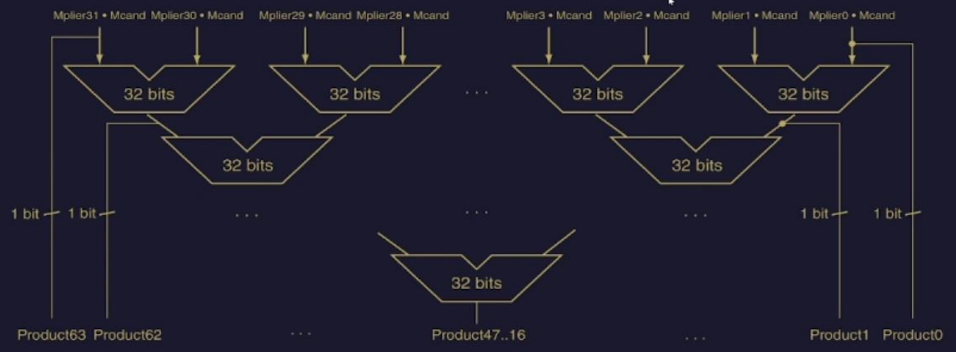
- Wykonuje kroki równolegle: add/shift



- W jednym cyklu jest wykonywane dodawanie częściowego wyniku
  - Takie rozwiązanie dobre, gdy częstotliwość mnożenia jest niska

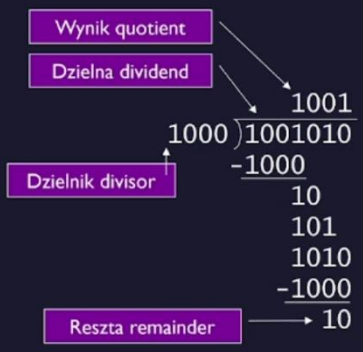
Szybszy układ mnożący

- Używa wielu układów mnożących



- Może być potokowany
  - Wiele mnożeń wykonywanych równolegle

Dzielenie



n-bit operands yield *n*-bit quotient and remainder

- Sprawdzenie czy nie dzielimy przez 0
- Dzielenie pisemne
  - Jeżeli liczba bitów dzielnika ≤ dzielnej
    - Bit równy 1 w wyniku, odejmujemy
  - W p.p.
    - bit równy 0 w wyniku, ściągamy następny bit dzielnej
- Restoring dzielenie
  - Wykonaj odejmowanie i jeżeli reszta < 0, dodaj dzielnik z powrotem
- Dzielenie ze znakiem
  - Dziel wartości bezwzględne
  - Dostosuj znaki wyniku i reszty odpowiednio

Szybsze dzielenie

- Nie można użyć równoległego rozwiązania
  - Odejmowanie jest uzależnione od znaku reszty
  - Szybsze układy dzielące (np. ukł. dzielący SRT) generują wiele bitów wyniku na krok
  - Dalej koniecznym jest wykonanie wielu kroków

Dodawanie zmienno-przecinkowe

Rozważmy 4-cyfrową liczbę dziesiętną

$9.999 \times 10^{1} + 1.610 \times 10^{-1}$

1. Wyrównanie kropek dziesiętnych

   Przesunięcie liczby z mniejszym wykładnikiem

   $9.999 \times 10^{1} + 0.016 \times 10^{1}$

2. Dodanie wartości całkowitych

   $9.999 \times 10^{1} + 0.016 \times 10^{1} = 10.015 \times 10^{1}$

3. Normalizacja wyniku & sprawdzenie przepełnienia/niedomiaru

   1.0015×10^{2}

4. Zaokrąglenie i normalizacja jeżeli potrzebna

   1.002×10^{2}

Dodawanie zmienno-przecinkowe

Rozważmy 4-cyfrową liczbę binarną

$1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)

1. Wyrównanie kropek binarnych

   Przesunięcie liczby z mniejszym wykładnikiem

   $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$

2. Dodanie wartości całkowitych

   $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$

3. Normalizacja wyniku & sprawdzenie przepełnienia/niedomiaru

   $1.000_2 \times 2^{-4}$, nie ma przepełnienia/niedomiaru

4. Zaokrąglenie i normalizacja jeżeli potrzebna

   $1.000_2 \times 2^{-4}$ (bez zmian) => 0.0625

11
Magdalena Szymczyk

Ukł. dodający liczby zmienno-przecinkowe

- Znacznie bardziej złożony niż ukł. dodający liczby całkowite
- Wykonywanie tego w jednym cyklu zegara może potrwać zbyt długo
  - Znacznie dłużej niż dla wersji całkowitej
  - Wolniejszy zegar byłby niekorzystny dla pozostałych instrukcji
- Dodawanie zmienno-przecinkowe trwa zwykle kilka cykli
- Może być potokowane

Mnożenie zmienno-przecinkowe

Rozważmy 4-cyfrową liczbę binarną

$1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5 × –0.4375)

1. Dodanie wykładników

   Bez przesunięcia: –1 + –2 = –3

   Z przesunięciem: (–1 + 127) + (–2 + 127) = –3 + 254 – 127 = –3 + 127

2. Pomnożyć części całkowite

   $1.000_2 \times 1.110_2 = 1.1102 \Rightarrow 1.110_2 \times 2^{-3}$

3. Normalizacja wyniku & sprawdzenie przepełnienia/niedomiaru

   $1.110_2 \times 2^{-3}$ (bez zmian) bez przepełnienia/niedomiaru

4. Zaokrąglenie i renormalizacja jeżeli potrzebna

   $1.110_2 \times 2^{-3}$ (bez zmian)

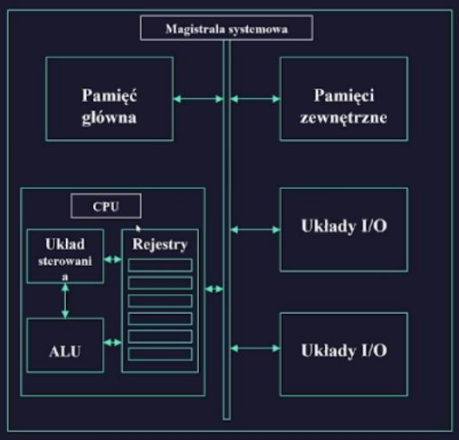
5. Określenie znaku: +ve × –ve ⇒ –ve

   $-1.110_2 \times 2^{-3}$ => –0.21875

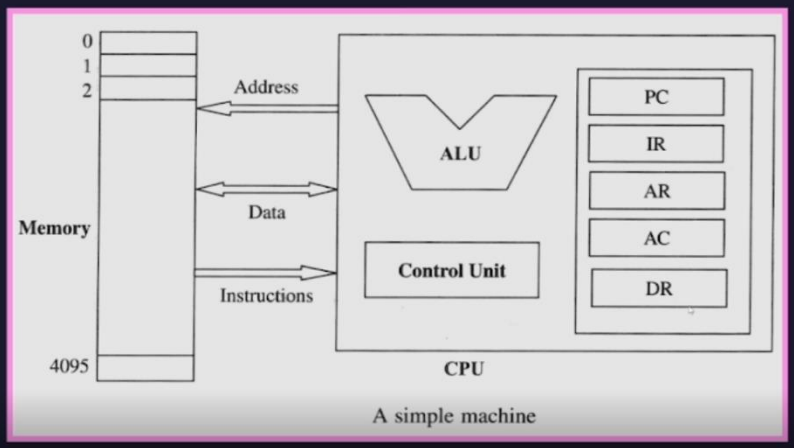
Układ arytmetyczny dla liczb zmienno-przecinkowych

- Ukł. mnożący dla liczb zmienno-przecinkowych jest o podobnej złożoności jak sumator dla tych liczb
  - Ale używa ukł. mnożącego dla części całkowitych zamiast sumatora
- Sprzęt dla arytmetyki na liczbach zmienno- przecinkowych wykonuje zwykle:
  - dodawanie, odejmowanie, mnożenie, dzielenie, odwrotności, pierwiastki kwadratowe
  - Konwersja liczb zmienno-przecinkowych do liczb całkowitych
- Operacja trwa zazwyczaj kilka cykli
  - Może być potokowana

Elementy systemu komputerowego



Prosty procesor



Jak to wszystko działa?

Powtarzaj

A. Pobierz instrukcję z komórki pamięci, której adres znajduje się w PC. Umieść w IR.

B. Zwiększ PC o jeden (następna instrukcja)

C. Zdekoduj kod operacji kodu w IR

D. To co zostało zdekodowane określa co należy zrobić.

Dopóki Halt (stop)

Zbiór instrukcji dla prostego procesora

Kod operacji	Operand	Znaczenie instrukcji
0000		Zatrzymanie wykonania -> STOP
0001	adr	Załaduj operand z pamięci (lokalizacja adr) do AC
0010	adr	Zapisz zawartość AC do pamięci (lokalizacja adr)
0011		Skopiuj zawartość AC do DR
0100		Skopiuj zawartość DR do AC
0101		Dodaj DR do AC
0110		Odejmij DR od AC
0111		AND bitowy DR z AC
1000		Dopełnienie zawartości rejestru AC
1001	adr	Skok do instrukcji pod adresem adr
1010	adr	Skok do instrukcji pod adresem adr gdy AC = 0

Program który dodaje do siebie dwie zawartości w pamięci i wynik umieszcza w następnej

Lokalizacja pamięci (bajty)	Instrukcja binarnie	Opis
0000 0000 0000	0001 0000 0000 1100	Załaduj zawartość lokalizacji 12 w AC
0000 0000 0010	0011 0000 0000 0000	Przenieś zawartość AC do DR
0000 0000 0100	0001 0000 0000 1110	Załaduj zawartość lokalizacji 14 do AC
0000 0000 0110	0101 0000 0000 0000	Dodaj DR do AC
0000 0000 1000	0010 0000 0001 0000	Zapisz zawartość AC w lokalizacji 16
0000 0000 1010	0000 0000 0000 0000	Stop
0000 0000 1100	0000 0001 0101 1110	Dana - wartość 350
0000 0000 1110	0000 0000 0110 0000	Dana - wartość 96
0000 0001 0000	0000 0000 0000 0000	Dana - wartość 0

Program w zapisie heksadecymalnym

Lokalizacja w pamięci Adres (bajty)	Instrukcja heksadecymalnie
000	100C
002	3000
004	100E
006	5000
008	2010
00A	0000
00C	015E
00E	0060
010	0000

Format instrukcji w asemblerze

Etykieta opcjonalnie	Kod operacji wymagany	Operand wymagany w pewnych instrukcjach	Komentarz opcjonalnie

Język asemblera dla prostego procesora

Mnemonik	Operand	Znaczenie instrukcji
STOP		Zakończenie wykonywania STOP
LD	x	Załaduj operand z pamięci (lokalizacja adr) do AC
ST	x	Zapisz zawartość AC do pamięci (lokalizacja adr)
MOVAC		Skopiuj zawartość AC do DR
MOV		Skopiuj zawartość DR do AC
ADD		Dodaj DR do AC
SUB		Odejmij DR od AC
AND		AND bitowy DR i AC
NOT		Dopełnienie zawartości AC
BRA	adr	Skok do instrukcji o adresie adr
BZ	adr	Skok do instrukcji adr if AC =0

Przykład 1 (dodawanie)

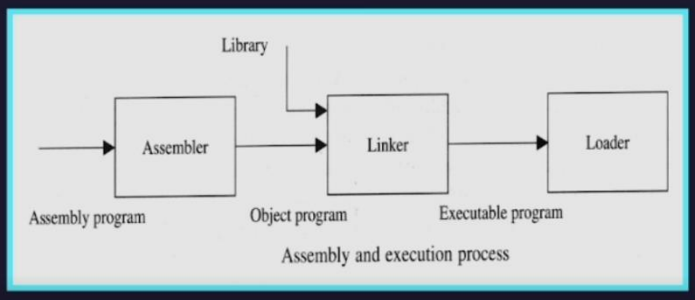
```
    LD X          \ AC ← X
    MOVAC         \ DR ← AC
    LD Y          \ AC ← Y
    ADD           \ AC ← AC + DR
    ST Z          \ Z ← AC
    STOP
X   W 350     \ rezerwacja słowa zainicjalizowanego na 350
Y   W 96      \ rezerwacja słowa zainicjalizowanego na 96
Z   W 0       \ wynik dodawania przechowywany jest tutaj, wyzerowanie tej wart na poczatku
```

Przykład 2 (mnożenie)

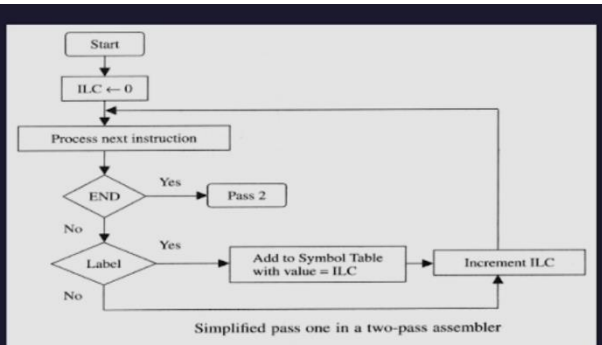
```
        LD  X          \ Load X in AC
        ST  N          \ Store AC (X original value) in N
LOOP    LD N           \ AC N
        BZ EXIT        \ Go to EXIT if AC  == 0 (N reached 0)
        LD ONE         \ AC 1
        MOVAC          \ DR AC
        LD N           \ AC N
        SUB            \ subtract 1 from N
        ST N           \ store decrements N
        LD Y           \ AC Y
        MOVAC          \ DR AC
        LD Z           \ AC Z (partial product)
        ADD            \ Add Y to Z
        ST Z           \ store the new value of Z
        BRA LOOP
        EXIT STOP
X       W   5          \ reserve a word initialized to 5
Y       W 15           \ reserve a word initialized to 15
Z       W 0            \ reserve a word initialized to 0
ONE     W 1            \ reserve a word initialized to 1
N       W 0            \ reserve a word initialized to 0
```

Magdalena Szymczyk

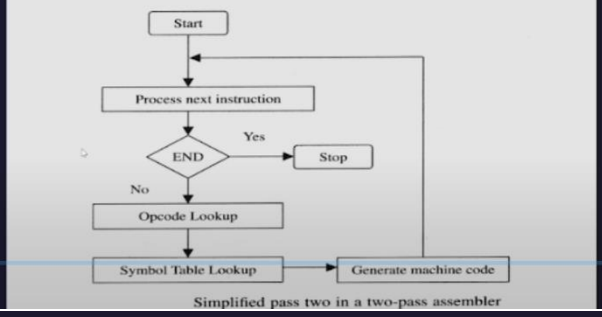
Proces asemblacji, linkowania i wykonania



Assembly and execution process



Simplified pass one in a two-pass assembler



Simplified pass two in a two-pass assembler

Magdalena Szymczyk

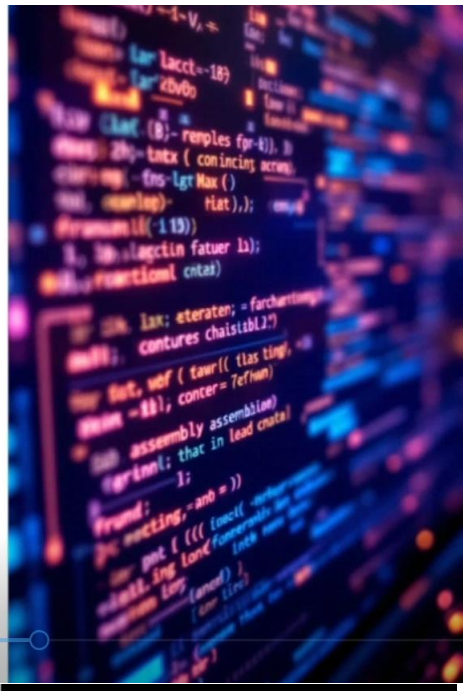
Struktury danych

Tablica Symboli dla mnożenia
(Przykład 2)

Symbol	Wartość (heksadecymalnie)	Inne
Loop	004	
EXIT	01E	
X	020	
Y	022	
Z	024	
ONE	026	
N	028	

Analiza Programu Mnożenia w Asemblerze MARIE.js

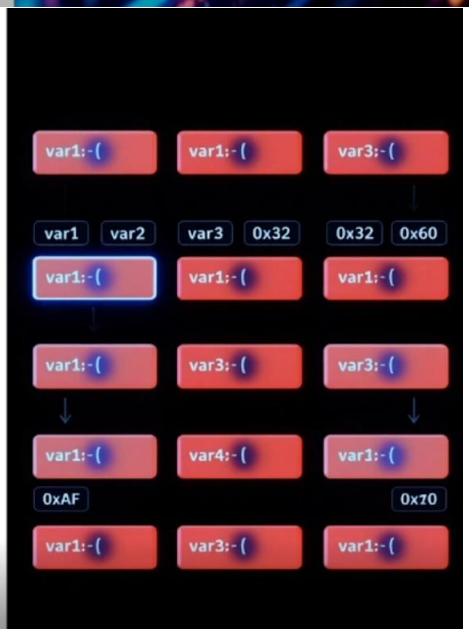
Program w asemblerze dla symulatora MARIE.js służy jako kalkulator mnożenia dwóch liczb całkowitych, również ujemnych, przy użyciu iteracyjnego dodawania. Pobiera dwie liczby całkowite X i Y, mnoży je poprzez iteracyjne dodawanie X, Y razy, obsługuje przypadki, gdy Y jest ujemne lub zerowe, a następnie wypisuje wynik.



Magdalena Szymczyk

Zmienne Używane w Programie

Nazwa	Znaczenie
X	pierwszy czynnik
Y	drugi czynnik (kontroluje ilość dodawań)
one	wartość 1 (do dekrementacji)
negflag	flaga znaku (0: +, 1: -)
result	wynik końcowy



Przykład Działania Programu

Wprowadzenie danych

Użytkownik podaje X = 4 i Y = -3

Przetwarzanie

Y zmieniane na 3 (wartość dodatnia)

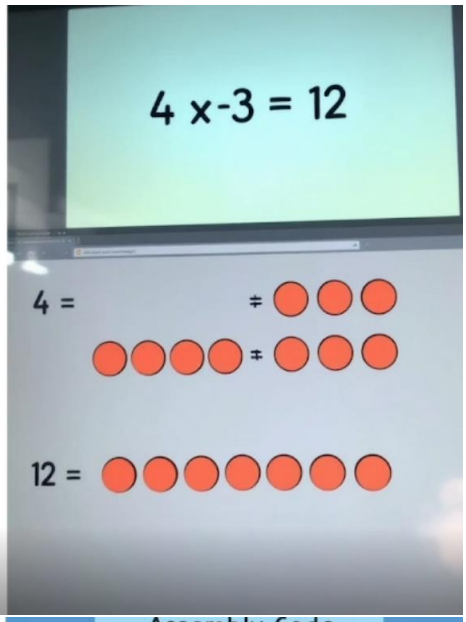
Ustawiana jest flaga negflag = 1

Wykonywane jest dodawanie: 4 + 4 + 4 = 12

Wynik końcowy

Ze względu na flagę negatywną, wynik jest negowany

Ostateczny wynik: -12



Struktura Programu

Wczytanie danych

Resetuje zmienną result i pobiera od użytkownika X i Y

Obsługa liczby ujemnej

Sprawdza czy Y < 0, jeśli tak, zmienia znak Y i ustawia flagę negflag = 1

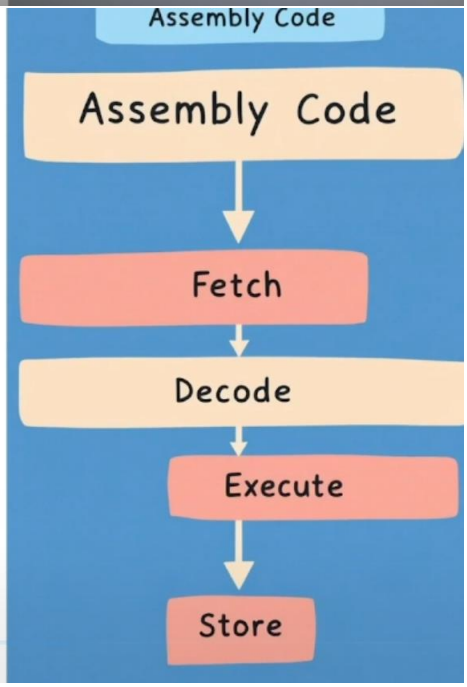
Pętla mnożenia

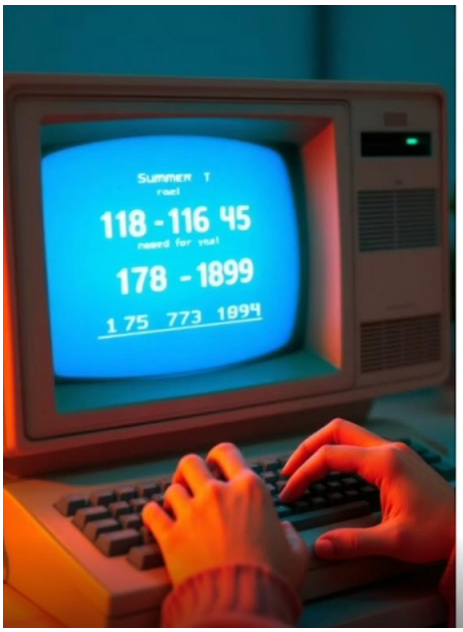
Dodaje X do result, Y razy, zmniejszając Y o 1 po każdej iteracji

Wyświetlenie wyniku

Jeśli negflag = 1, neguje wynik, a następnie wypisuje result i kończy program

Magdalena Szymczyk





Czyszczenie Wyniku i Wczytanie Danych

Resetowanie wyniku

Clear - zeruje akumulator

Store result - zapisuje 0 do zmiennej result

Pobieranie pierwszej liczby

Input - pobiera liczbę od użytkownika

Store X - zapisuje liczbę do zmiennej X

Pobieranie drugiej liczby

Input - pobiera liczbę od użytkownika

Store Y - zapisuje liczbę do zmiennej Y

Obsługa Liczby Ujemnej

1. **Sprawdzenie znaku Y**
   - Load Y - wczytuje Y do akumulatora
   - Skipcond 000 - sprawdza czy Y < 0
   - Jump nonneg - jeśli nie, przechodzi do etykiety nonneg
2. **Zmiana znaku Y**
   - Subt Y - od Y odejmuje Y
   - Subt Y - od 0 odejmuje Y ponownie, co daje -Y
   - Store Y - zapisuje -Y do zmiennej Y
3. **Ustawienie flagi**
   - Clear - zeruje akumulator
   - Add one - dodaje 1 do akumulatora
   - Store negflag - zapisuje 1 do flagi negflag

Magdalena Szymczyk

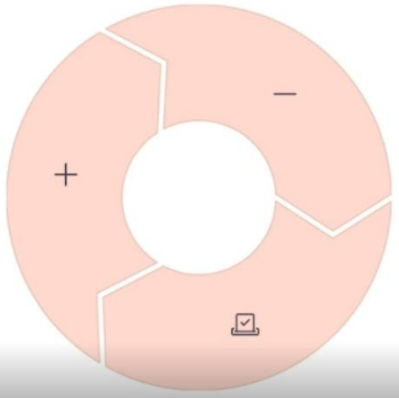
Pętla Mnożenia przez Dodawanie

Dodawanie X do wyniku

Load result - wczytuje aktualny wynik

Add X - dodaje X do akumulatora

Store result - zapisuje nowy wynik



Zmniejszenie licznika

Load Y - wczytuje Y do akumulatora

Subt one - odejmuje 1 od Y

Store Y - zapisuje nową wartość Y

Sprawdzenie warunku

Skipcond 400 - sprawdza czy Y == 0

Jump loop - jeśli nie, wraca do początku pętli

Sprawdzenie Flagi Znaku

Sprawdzenie flagi

Load negflag - wczytuje flagę znaku

Skipcond 800 - sprawdza czy negflag == 0

Negacja wyniku

Load result - wczytuje wynik

Subt result - odejmuje result od result

Subt result - odejmuje result ponownie

Zapisanie wyniku

Store result - zapisuje -result do zmiennej result

Segment programu wykorzystujący procedurę

Podstawowe typy architektur przy uwzględnieniu sposobu adresacji operandów

Podział architektur systemów wbudowanych ze względu na współpracę z rejestrami lub pamięcią obejmuje:

1. Architektury typu akumulatorowego: W tych architekturach obliczenia są wykonywane na zawartości akumulatora, który jest głównym rejestrem. Dane są zapisywane w pamięci, a następnie przesyłane do akumulatora w celu ich przetworzenia.

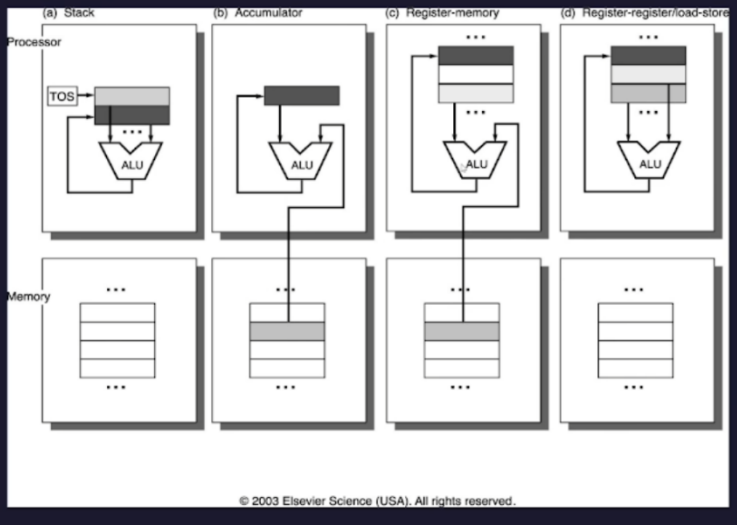
2. Architektury typu rejestrów ogólnego przeznaczenia: W tych architekturach istnieje wiele rejestrów, z których każdy może być używany jako akumulator lub przechowywać dane tymczasowe.

3. Architektury typu stosowego: W tych architekturach dane są przechowywane na stosie, a operacje wykonywane są na jego wierzchołku. Wiele operacji związanych z obsługą stosu jest wbudowanych w procesor.

4. Architektury typu pamięciowego: W tych architekturach istnieje niewielka liczba rejestrów, a większość danych przechowywana jest w pamięci. Operacje są wykonywane bezpośrednio na pamięci, co może prowadzić do wolniejszych obliczeń.

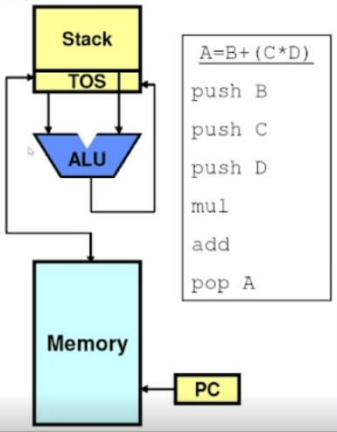
Wybór architektury zależy od wymagań projektowych i ograniczeń, takich jak koszt, czas odpowiedzi, moc obliczeniowa, zużycie energii czy ilość dostępnej pamięci. Architektury typu rejestrów ogólnego przeznaczenia i stosowego są najczęściej stosowane w systemach wbudowanych.

Podstawowe typy architektur przy uwzględnieniu sposobu adresacji operandów



Architektura stosowa

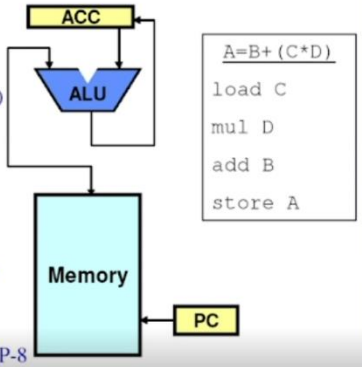
- Stack: First-In Last-Out data structure (FILO)
- Instruction operands
  - None for ALU operations
  - One for push/pop
- Advantages:
  - Short instructions
  - Compiler is easy to write
- Disadvantages
  - Code is inefficient
    - Fix: random access to stacked values
  - Stack size & access latency
    - Fix : register file or cache for top entries
- Examples
  - 60s: Burroughs B5500/6500, HP 3000/70
  - Today: Java VM




```
A=B+(C*D)
push B
push C
push D
mul
add
pop A
```

Architektura akumulatorowa

- Single register (accumulator)
- Instructions
  - ALU (Acc ← Acc + *M)
  - Load to accumulator (Acc ← *M)
  - Store from accumulator (*M ← Acc)
- Instruction operands
  - One explicit (memory address)
  - One implicit (accumulator)
- Attributes:
  - Short instructions
  - Minimal internal state; simple design
  - Many loads and stores
- Examples:
  - Early machines: IBM 7090, DEC PDP-8
  - Today: DSP architectures




```
A=B+(C*D)
load C
mul D
add B
store A
```