
NobleProg

Mikrokontrolery – wprowadzenie
Elementy architektury komputerów

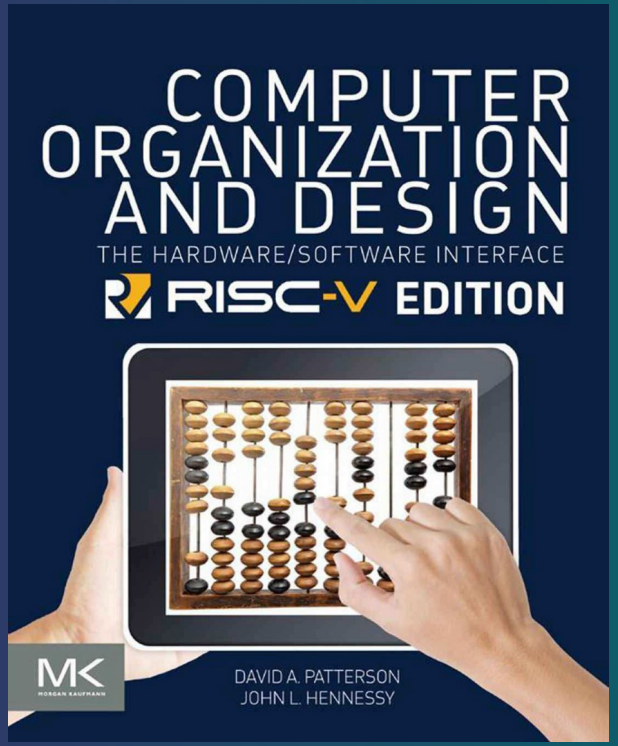
Grzegorz Mazur

NobleProg

Plan zajęć

- Komputer - architektura
- Pamięci komputerów
- Dane
- Języki maszynowe i asemblerowe
- Potrzeby programów w języku wysokiego poziomu
- Wyjątki
- Działanie procesora

Literatura



- Patterson, Hennessy – Computer Organization and Design, the Hardware/Software Interface, RISC-V Edition – Elsevier
- Dostępna w sieci (...)

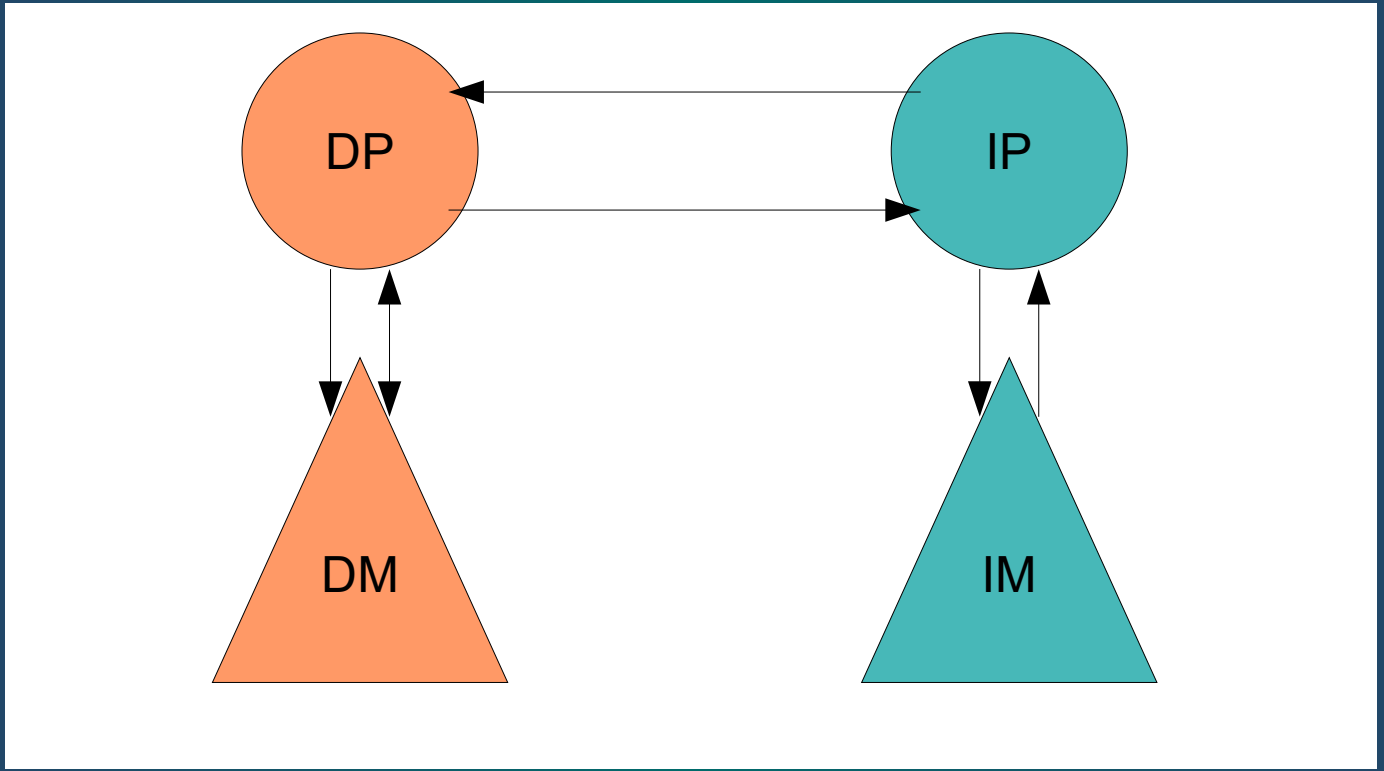
Komputer

- Urządzenie do przetwarzania danych, wyposażone w możliwość wprowadzania, przechowywania i wyprowadzania danych
  - wprowadzanie i wyprowadzanie danych może być realizowane w postaci odpowiedniej dla człowieka (klawiatura, ekran) lub właściwej dla współpracy z jakimś obiektem (np. czujnik temperatury, grzejnik)
- Składniki logiczne komputera:
  - Procesor – przetwarza informację (dane, na podstawie programu)
  - Pamięć – przechowuje dane i programy
  - Wejście-wyjście – zapewnia wymianę informacji z otoczeniem

Taksonomia Skillicorna

- Zaproponowana około 1988 roku przez Davida Skillicorna, opisuje budowę komputera jako strukturę złożoną ze składników
- Abstrakcyjne składniki architektury
  - “abstrakcyjne” - bo nie ma bezpośredniej odpowiedniości pomiędzy składnikami modelu Skillicorna i fizycznymi elementami komputera
  - Procesory instrukcji (IP) – pobierają i analizują instrukcje
  - Procesory danych (DP) - wykonują operacje na danych
  - Hierarchie pamięci instrukcji (IM) – przechowują instrukcje
  - Hierarchie pamięci danych (DM) – przechowują dane – argumenty operacji i wyniki

Taksonomia Skillicorna - składniki



Współpraca składników

- Procesor instrukcji – hierarchia pamięci instrukcji
  - Procesor przesyła do hierarchii pamięci żądania pobrania instrukcji
  - Hierarchia pamięci instrukcji zwraca instrukcje
- Procesor instrukcji – procesor danych
  - procesor instrukcji przesyła instrukcje w postaci zrozumiałej dla procesora danych
  - procesor danych zwraca informacje o stanie przetwarzania oraz informacje dotyczące przebiegu wykonania programu
- Procesor danych – hierarchia pamięci danych
  - procesor przesyła żądania odczytu i zapisu danych
  - hierarchia pamięci przesyła dane – argumenty operacji
- Procesor danych – procesor danych – wymiana danych

Hierarchia pamięci

- Nie jest możliwe zbudowanie pamięci o dowolnie dużej pojemności i dowolnie krótkim czasie dostępu
- Czas dostępu rośnie z pojemnością
  - pojemność wpływa na wymiary pamięci, a długość drogi dostępu do danych określa czas niezbędny do wykonania dostępu
- Struktura hierarchiczna-warstwowa
  - umożliwia zróżnicowanie parametrów pamięci – pojemności i czasów dostępu
  - kolejne warstwy mają coraz większe pojemności i czasy dostępu
- W skład hierarchii pamięci wchodzą wszystkie zasoby dostępne dla komputera, które mogą służyć do przechowywania danych i programów

Warstwy hierarchii pamięci

- rejestry
- kieszenie (1..3 poziomów)
- pamięć operacyjna
- pamięć masowa (rozszerzenie p. operacyjnej w systemie pamięci wirtualnej)
- pamięć masowa- system plików
- nośniki wymienne, zasoby sieciowe

Sterowanie hierarchią pamięci

- Najczęściej używane obiekty przemieszczane w górę hierarchii
  - każdy potrzebny obiekt powinien znaleźć się jak najbliżej procesora
  - obiekty chwilowo niepotrzebne będą spychane w dół hierarchii pamięci
- Sterowanie przemieszczaniem:
  - rejestry↔reszta hierarchii - programista piszący program/kompilator
  - kieszenie↔pamięć operacyjna - sprzęt
  - pamięć operacyjna↔pamięć wirtualna - system operacyjny
  - pamięć wirtualna↔lokalny system plików - program użytkowy/użytkownik
  - lokalny↔zdalny system plików – użytkownik
- Styki poszczególnych warstw charakteryzują się również różną granularnością przemieszczanych obiektów

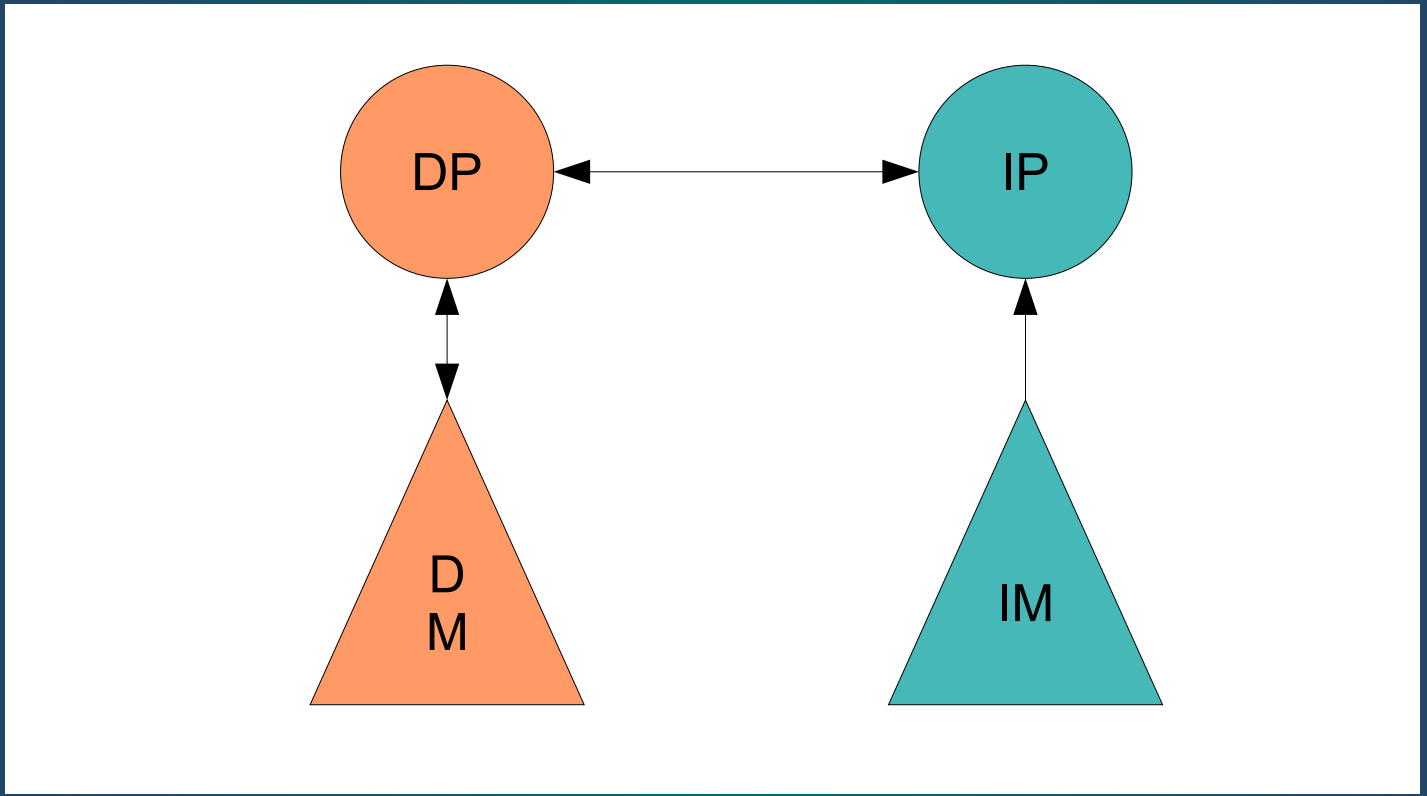
Warianty organizacji hierarchii pamięci w komputerach

- „Harvard” - oddzielne hierarchie pamięci programu i danych
  - często nie uznawana za maszynę von Neumanna
- „Princeton” - wspólna hierarchia pamięci programu i danych
- Na następnych ekranach, przedstawiających schematy blokowe komputerów będziemy posługiwali się symbolami zapożyczonymi z taksonomii Skillicorna w sposób niezgodny z zasadami tworzenia modeli obowiązującymi w tej taksonomii
  - nie będą to modele architektur wg. Skillicorna!!!

Maszyna von Neumanna

- Nazwa odnosi się do zasady działania komputera zbudowanego na uniwersytecie Princeton w latach 1945..49
- Instrukcje tworzące program są przechowywane w pamięci w taki sam sposób, jak dane
- Pamięć składa się z pewnej liczby ponumerowanych komórek
  - dostęp do pamięci następuje poprzez podanie przez procesor numeru komórki
  - numer komórki nazywamy ADRESEM
- Z powyższych postulatów wynika w praktyce, że
  - zazwyczaj komputer pobiera kolejne instrukcje programu z kolejnych komórek pamięci
  - komórki te są wybierane przez zwiększający się adres, który powinien być przechowywany i inkrementowany w procesorze
  - adres ten jest przechowywany w specjalnym rejestrze – tzw. liczniku instrukcji (Program Counter – PC)

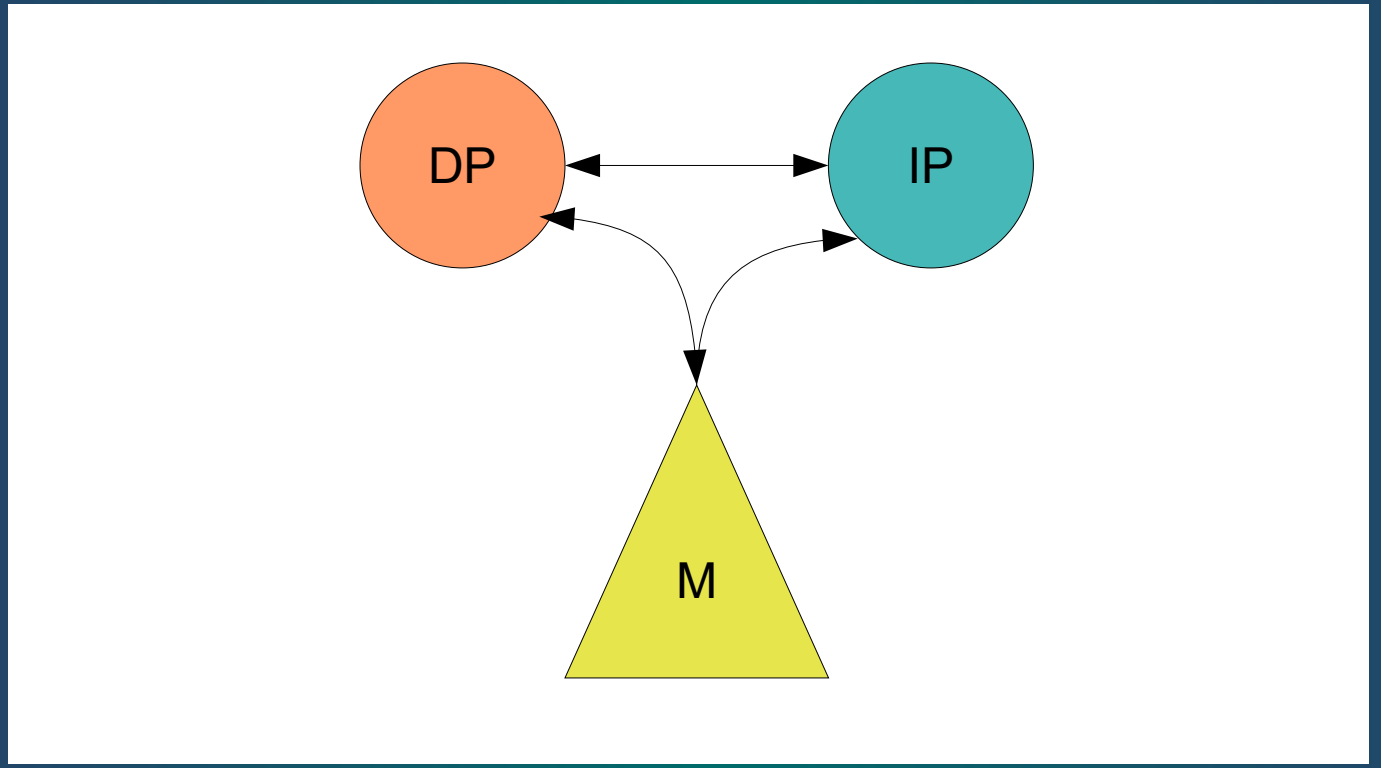
Architektura Harvard



Architektura Harvard

- Realizacja maszyny von Neumanna z oddzielnymi hierarchiami pamięci programu i danych
  - często uznawana za architekturę nie-vonneumannowską ze względu na dyskusyjność zachowania postulatu o jednakowym składowaniu instrukcji i danych
- Wysoka wydajność dzięki możliwości równoczesnego pobierania instrukcji i operacji na hierarchii pamięci danych
- Brak możliwości zapisu instrukcji do hierarchii pamięci instrukcji
  - czyli brak możliwości programowania
  - komputer dostarczany ze stałym programem
  - dopuszczalne tylko w zastosowaniach wbudowanych

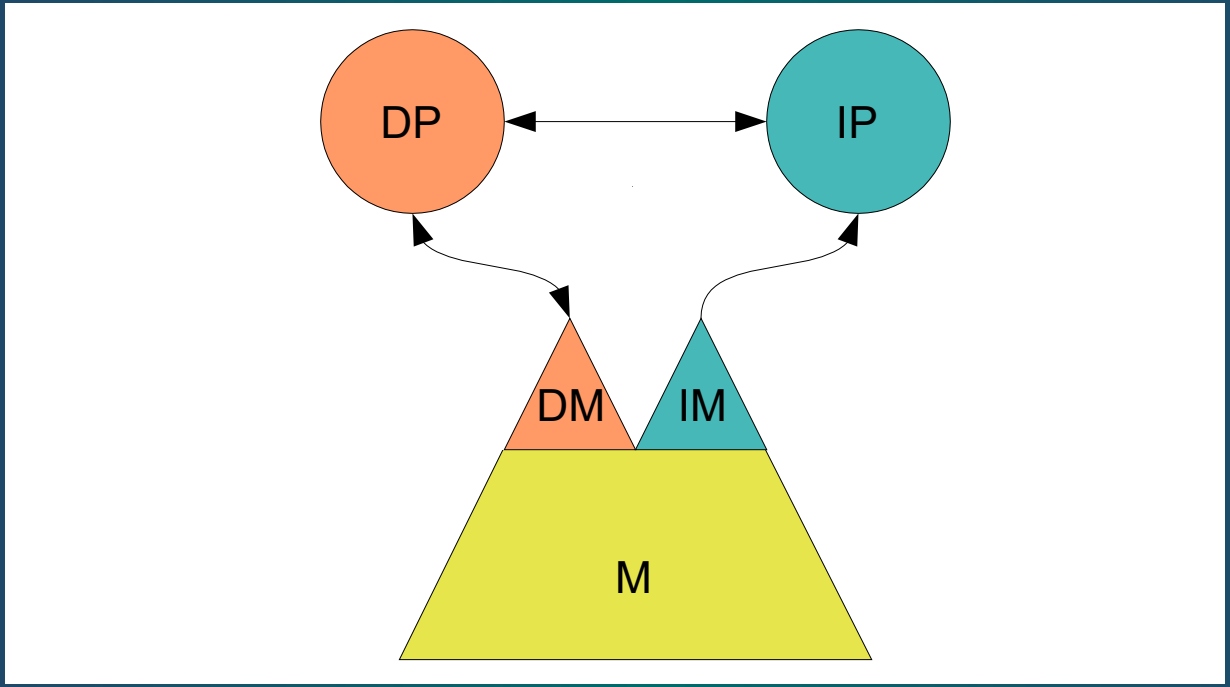
Architektura Princeton



Architektura Princeton

- “Wzorcowa” realizacja maszyny von Neumanna ze wspólną hierarchią pamięci instrukcji i danych
- Wspólna hierarchia wyklucza równoczesne pobieranie instrukcji i operacje na danych
  - tzw. *von Neumann bottleneck*
- Nieograniczone możliwości modyfikacji programu
  - obiekt zapisany przez procesor danych do hierarchii pamięci jako dany może być następnie pobrany przez procesor instrukcji jako instrukcja
  - możliwość programowania – potrzebna w komputerach uniwersalnych
  - program może sam siebie modyfikować (automodyfikacja)
    - nie zawsze jest to pożądana cecha

Architektura Harvard-Princeton



Klasyfikacja pamięci

- Ze względu na trwałość informacji:
  - Ulotne – utrzymanie zawartości wymaga dostarczania energii
    - Dynamiczne – utrzymują zawartość do kilkudziesięciu ms, wymagają odświeżania zawartości
    - Statyczne
  - Nieulotne – trwała zawartość, energia potrzebna tylko do zmiany zawartości pamięci
- Ze względu na mechanizm dostępu:
  - bezpośredni (przez adres)
  - sekwencyjny (wg kolejności)
  - Asocjacyjny – przez porównanie z wzorcem
  - Hybrydowy – np. bezpośrednio-sekwencyjny

Jednostki miary i krotności

2^{10}	Ki	kibi
2^{20}	Mi	mebi
2^{30}	Gi	gibi
2^{40}	Ti	tebi
2^{50}	Pi	pebi
2^{60}	Ei	exbi

- Zdefiniowane przez standardy IEEE 1541 i IEC 80000-13
- Jednostki informacji:
  - bit (b)
  - oktet (o) – 8 bitów
  - bajt (B) – najmniejsza jednostka adresowalna, zwykle 8 bitów
- Krotności binarne – używane do wyrażania parametrów o wartościach wynikających z użycia w komputerze systemu binarnego, np. pojemności pamięci

Półprzewodnikowe pamięci nieulotne

- Technologia Flash EEPROM
  - Duża pojemność, niski koszt jednostkowy
  - Zapis wolniejszy od odczytu
  - Przed zapisem wymagane skasowanie poprzedniej zawartości
    - Jest to najbardziej czasochłonna i energochłonna operacja
    - Układ pamięci ma ograniczoną żywotność - liczbę operacji kasowania
      - NOR Flash – 10 000..1 000 000 cykli kasowania, pojemności MiB..GiB
      - NAND Flash – 1 000..10 000 cykli, pojemności do setek GiB
- Technologie magnetyczne – MRAM, FRAM
  - Mała pojemność, wysoki koszt jednostkowy
  - Szybki zapis (mała energia zapisu), trwałość ok. 10^{14} cykli

Charakterystyka technologii pamięci

Typ	Nieulotna	odczyt/zapis	kasowanie	Pojemność układu (max. AD2020)
DRAM (DDR4)	nie	10 ns		8 Gib
SRAM	nie	50 ns		64 Mib
MRAM	tak	35 ns		32 Mib
NOR Flash	tak	50 ns	20 ms, 1000000×	1 Gib
NAND Flash	tak	50 ns	10 ms, 10000×	512 Gib

Dane, na których operuje komputer

- Wartości logiczne (prawda/fałsz)
- Znaki pisarskie
- Liczby
  - całkowite
    - nieujemne
    - ze znakiem
  - niecałkowite
    - stałopozycyjne
    - zmiennopozycyjne
- Dźwięki i inne sygnały jednowymiarowe
- Obrazy rastrowe

Binarna reprezentacja danych

- Komputer rozpoznaje tylko dwa symbole – cyfry binarne – 0 i 1
  - BInary digiT → bit
- Komputer operuje wyłącznie na grupach bitów (cyfr binarnych), które mogą być interpretowany jako liczby binarne
  - współczesne komputery operują najczęściej na słowach binarnych, których długość wynosi 8×2^n bitów (np. 8, 16, 32, 64)
- Wszelkie inne symbole (np. znak liczby) mogą być zapisane wyłącznie przy użyciu bitów – wartości 0 lub 1
- Dane nieliczbowe (znaki, sygnały) muszą być najpierw zapisane przy użyciu liczb

Dane alfanumeryczne – tekstowe

- Każdy znak pisarski jest reprezentowany przez liczbę, stanowiącą jego numer w tablicy kodowej
- Najczęściej używane kody:
  - ASCII – 128 pozycji, w tym małe i wielkie litery alfabetu łacińskiego
  - rozszerzone kody 256-pozycyjne na bazie ASCII
    - pierwsze 128 pozycji jak w ASCII, następne 128 pozycji zawiera znaki narodowe lub inne symbole
    - problem: różne kody dla różnych części świata
  - kody rodziny EBCDIC (używane głównie przez IBM)
  - UNICODE
    - pierwotnie 2^{16}, obecnie do 2^{21} możliwych pozycji
    - reprezentacja wszystkich znaków używanych na świecie

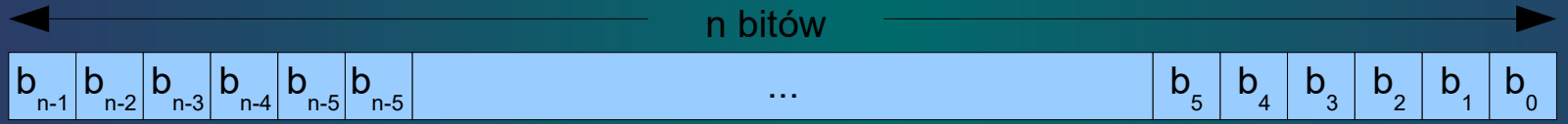
Reprezentacja dźwięków i obrazów

- Dźwięk:
  - chwilowa wartość napięcia reprezentującego ciśnienie akustyczne próbkowana z częstotliwością zależną od potrzeb (zwykle od 8 do 48 kHz)
  - wartości próbek zapisywane jako liczby całkowite
- Obraz rastrowy
  - zapisany w postaci prostokątnej macierzy punktów – elementów obrazu (pikseli – picture element)
  - każdemu pikselowi odpowiada jeden kolor
  - kolor reprezentowany w postaci trzech składowych – dla celów wyświetlania są to jasności świateł podstawowych: czerwonego, zielonego i niebieskiego
  - wartości jasności zapisane w postaci liczb całkowitych bez znaku

Formaty danych

- Komputery operują na słowach – ciągach bitów o długościach wyrażonych najczęściej potęgami liczby 2
  - typowe długości słów dla komputerów uniwersalnych – 8, 16, 32, 64, 128 bitów
  - komputery specjalizowane mogą używać innych formatów danych – np. 24 bity
- Niektóre komputery mogą ponadto operować na pojedynczych bitach i ciągach bitów o dowolnych, niewielkich długościach – tzw. polach bitowych
- Dane są najczęściej zapisywane w postaci słów o długości odpowiedniej dla danego typu komputera

Zapis liczb całkowitych nieujemnych



- Naturalny kod binarny – NKB
- Kod BCD
  - Używany dla liczb dziesiętnych stałopozycyjnych
  - Każda cyfra dziesiętna jest zapisywana w postaci liczby binarnej
  - 4 bity (tetrada) na cyfrę
  - Dozwolone wartości tetrady: 0..9, pozostałe nieważne
  - Postaci:
    - Spakowana – 2 cyfry w bajcie
    - Niespakowana (“ASCII”) - jedna cyfra w bajcie

$$v_{NKB} = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

Zapis ósemkowy i szesnastkowy

- W celu ułatwienia zapisu danych binarnych przez programistę (zmniejszenia liczby zapisywanych cyfr) używa się zapisów: ósemkowego i szesnastkowego.
  - 8 i 16 są potęgami liczby 2; pojedyncza cyfra reprezentuje odpowiednio 3 lub 4 bity
- Zapis ósemkowy
  - Cyfry '0'..'7'
  - W języku C każda stała całkowita rozpoczynająca się od cyfry 0 jest traktowana jako ósemkowa
- Zapis szesnastkowy
  - Cyfry '0'..'9', 'A'..'F'
  - W języku C stałe szesnastkowe poprzedza się sekwencją 0x, np.. 0xA1B, 0xc2d
- Nowy standard języka C (C2x) obejmuje zapis binarny stałych z prefiksem 0b (np. 0b101)

Reprezentacje liczb całkowitych ze znakiem

- U2 – kod uzupełnieniowy do dwóch

$$v_{U2} = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- U1 – kod uzupełnieniowy do jedynek

$$v_{U1} = -b_{n-1} \cdot \left(2^{n-1} - 1\right) + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- Znak-moduł

$$v_{Z-M} = (-1)^{b_{n-1}} \cdot \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- zapis spolaryzowany (*biased*)
  - zwykle przyjmuje się BIAS = $2^{n-1}-1$

$$v_B = -BIAS + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

Własności kodu U2

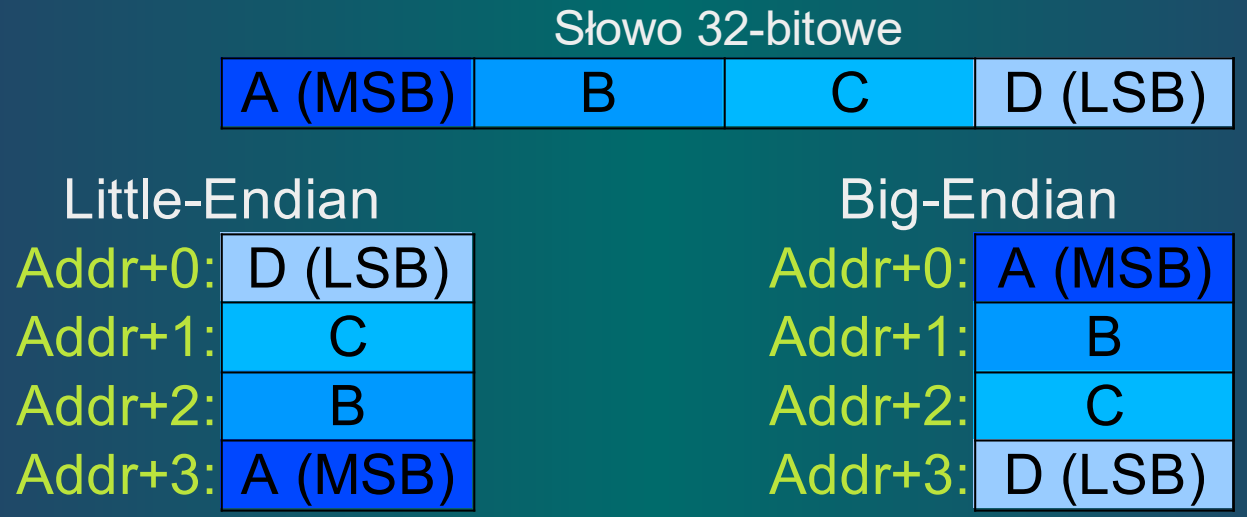
- Dodawanie i odejmowanie wykonywane analogicznie jak w NKB
  - Różne wykrywanie nadmiaru operacji
- N-bitowy wynik mnożenia liczb n-bitowych taki sam, jak w NKB
- Zmiana znaku liczby może zostać zrealizowana poprzez zanegowanie wzorca bitowego i dodanie 1
- Współcześnie liczby stałopozycyjne ze znakiem są zawsze reprezentowane w kodzie U2

Organizacja pamięci

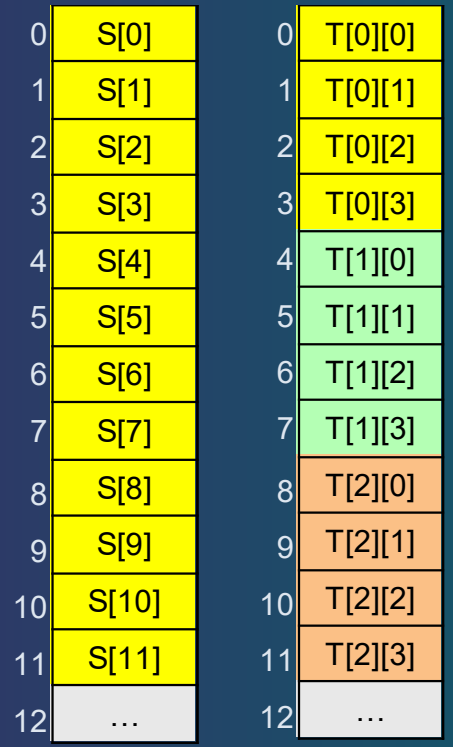
- W komputerach uniwersalnych jednostką adresowaną jest bajt 8-bitowy
- Dane wielobajtowe zajmują odpowiednią liczbę kolejnych komórek pamięci (i adresów)
- Adres danej jest adresem pierwszego bajtu pamięci zajmowanego przez tę daną

Konwencje adresowania danych wielobajtowych

- Little Endian – najmniej znaczący bajt pod najmniejszym adresem
- Big Endian – najbardziej znaczący bajt pod najmniejszym adresem



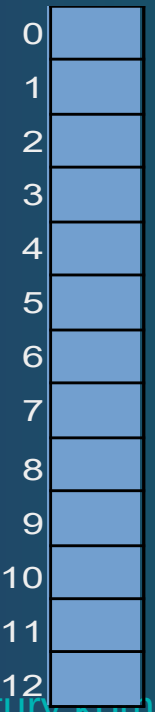
Wektory i tablice



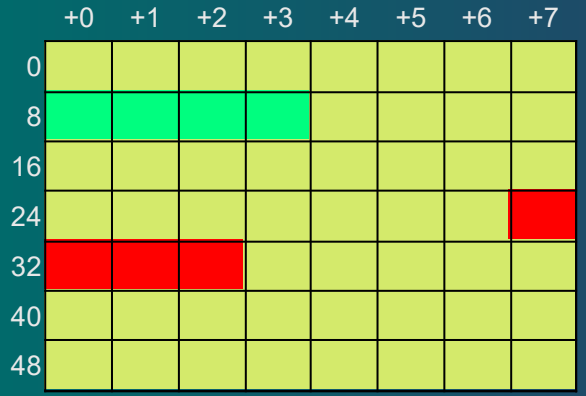
- Elementy wektora (tablicy 1-wymiarowej) zajmują kolejne lokacje pamięci
  - Pierwszy element wektora znajduje się pod najniższym adresem – adresem wektora
  - Element zajmuje tyle bajtów, ile wynosi wartość operatora sizeof() - z uwzględnieniem wyrównania
- Wyrównanie wektora jest takie, jak wyrównanie jego pojedynczego elementu
- Tablica n-wymiarowa jest wektorem tablic (n-1)-wymiarowych

Logiczna i fizyczna organizacja pamięci

- Logiczna – wektor bajtów



- Fizyczna - „dwuwymiarowa” - słowa 32-, 64- lub 128-bitowe
- Możliwy równoczesny dostęp do dowolnej grupy bajtów w obrębie jednego słowa



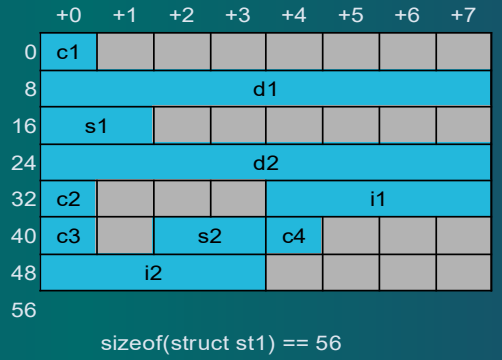
Wyrównanie naturalne

- Każda dana powinna być położona pod adresem podzielnym przez jej długość
  - daje to gwarancję najszybszego dostępu
- Wymuszone w nowszych architekturach
- Podnosi wydajność w architekturach, w których nie jest wymuszone
  - Obecnie wymuszone przez kompilatory nawet jeśli sprzęt nie narzuca konieczności wyrównania
- Wyrównanie struktur
  - Kompilator musi zachować kolejność pól struktury
  - Struktury muszą być wyrównane zgodnie z wyrównaniem najdłuższego typu danych występującego w strukturze
  - Operator sizeof zwraca odstęp pomiędzy początkami dwóch kolejnych struktur danego typu

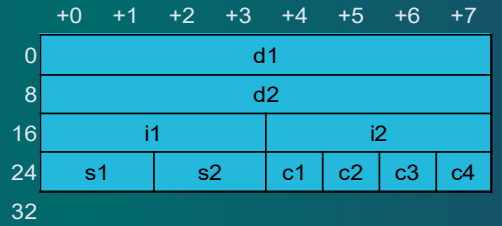
Kolejność pól struktury a zajętość pamięci

```
struct st1 {
     char c1;
     double d1;
     short int s1;
     double d2;
     char c2;
     int i1;
     char c3;
     short int s2;
     char c4;
     int i2;
} e1;
```

```
struct st2 {
     double d1, d2;
     int i1, i2;
     short int s1, s2;
     char c1, c2, c3, c4;
} e2;
```



sizeof(struct st1) == 56



sizeof(struct st2) == 32

Program, zadanie, wątek (1)

- Program – statyczny zapis algorytmu i związanych z nim danych
  - W systemach jednozadaniowych/jednowątkowych pojęcia programu, zadania i wątku są równoważne
- Zadanie/proces (task/process) – instancja programu w systemie wielozadaniowym
  - Proces ma własną, prywatną pamięć, zawierającą jego kod i dane
  - W systemie może działać wiele procesów, w tym również wiele instancji tego samego programu, z których każda ma oddzielną pamięć
  - W systemach wielozadaniowych jednowątkowych proces=wątek

Program, zadanie, wątek (2)

- Wątek (thread) – instancja wykonania w obrębie procesu/zadania
  - Wątki zadania mają wspólną pamięć, nie chronioną przed innymi wątkami
    - kod, dane statyczne, sterta
  - każdy wątek ma własny stos
  - wątek może mieć również własne dane statyczne
  - W prostszych komputerach bez sprzętowego zarządzania pamięcią nie ma zadań (rozdzielności pamięci), ale mogą istnieć wątki (np. w systemie FreeRTOS)

Program w języku wysokiego poziomu

```
int x, y;                    // zmienne globalne
char *p;

int myadd(int a, int b)
{
     int r;                  // zmienna lokalna
     r = a + b;
     return r;               // wartość r staje się wartością funkcji
}

void cleanup(void)
{
     free(p);                // dealokacja zmiennej dynamicznej
}

int main(void)
{
     p = malloc(1000);       // alokacja zmiennej dynamicznej
     x = myadd(3, 20);       // wywołanie funkcji z przekazaniem argumentow, zapis wyniku
     cleanup();
}
```

Wymagania języka wysokiego poziomu

- Klasy pamięci (sekcje)
  - Odpowiadają różnym klasom obiektów niezbędnych do działania programu
- Operacje na danych
  - Arytmetyczne
  - Logiczne
  - Inne
- Dostęp do danych
  - Indeksowanie wektorów/tablic
  - wskaźniki/referencje
- Przekazywanie sterowania
  - Wywołania procedur i powroty
  - Przekazywanie parametrów
  - Zmienne lokalne procedur

Kod (sekcja TEXT)

- Statyczny, obecny w pamięci przez cały czas życia programu
- Stały rozmiar
- Tylko do odczytu
- Może zawierać:
  - instrukcje programu
  - stałe, np.
    - tablice adresów dla instrukcji *switch*
    - Literały

Dane statyczne (STATIC)

- Czas życia równy czasowi życia programu
- Stały rozmiar
- Mogą być podzielone na obszary - podsekcje
  - Stałe (.rodata) – tylko do odczytu
  - Zmienne zainicjowane (.data) – możliwy odczyt i zapis, nadane wartości początkowe
  - Zmienne niezainicjowane (.bss) – bez wartości początkowych
    - nazwa sekcji - „BSS” - pochodzi od *Block Started by Symbol*
    - Współczesne standardy implementacji języków programowania mogą gwarantować zerowanie sekcji BSS przed rozpoczęciem wykonania programu
- Tworzone (deklarowane/definiowane) przy użyciu dyrektywy asemblera

Dane dynamiczne automatyczne

- Argumenty i zmienne lokalne procedur
- Tworzone i usuwane w czasie działania programu – zmienny rozmiar
- Kolejność usuwania zawsze odwrotna do kolejności tworzenia
- Tworzą stos (*stack*)
- Alokowane i dealokowane przez instrukcje programu
  - Najczęściej poprzez operacje na wierzchołku stosu

Dane dynamiczne kontrolowane

- Tworzone i usuwane jawnie przez programistę (*malloc/free*, *new/dispose*)
- Czas życia nie związany z czasem życia i zagłębianiem procedur, nie wynika ze struktury wywołań
- Kolejność tworzenia i usuwania – dowolna
- Tworzą stertę (*heap*)
- Alokowane i dealokowane przez procedury biblioteki standardowej języka programowania, korzystające z systemowych funkcji alokacji pamięci

Symboliczny zapis instrukcji procesora

- Instrukcje są zapisane w pamięci komputera w postaci słów binarnych
- Zapis instrukcji określa akcję, którą procesor ma wykonać (operację) oraz dane, na których ma operować – argumenty
- Instrukcje procesora można zapisać w postaci czytelnej dla człowieka
- Język symbolicznego zapisu instrukcji jest nazywamy językiem asemblerowym lub – w skrócie – asemblerem
- Procesory o różnych modelach programowych mają różne języki asemblerowe
- Zapis instrukcji w języku asemblerowym składa się z mnemonicznej nazwy instrukcji oraz listy jej argumentów

Operacje potrzebne do implementacji języka wysokiego poziomu

- Kopiowanie danych (przesłania)
  - Ładowanie z pamięci do rejestru
  - Składowanie z rejestru do pamięci
  - Przesłania z konwersją formatów (rozmiarów)
- Operacje arytmetyczne i logiczne
  - +, -, *, /, &, |, ^, ~, itd..
- Rozejścia warunkowe, pętle
- Przekazywanie sterowania pomiędzy procedurami
  - Operacje na stosie – przekazywanie argumentów
  - Wywołania i powroty z procedur
  - Odbiór wartości funkcji

Procedury - wywołanie i powrót

- Kończąc wykonanie procedury należy przekazać sterowanie do instrukcji położonej bezpośrednio za instrukcją wywołującą procedurę
- Wywołanie procedury następuje poprzez instrukcję skoku ze śladem
  - skocz zapamiętując bieżącą wartość PC (wskazującą na następną instrukcję po instrukcji skoku)
- Zakończenie procedury (powrót) - instrukcja powrotu według śladu
  - skocz do adresu zapamiętanego przez instrukcję skoku ze śladem

Stos

- Struktura danych używana do przekazywania sterowania pomiędzy procedurami programu
  - Parametry wywołania
    - Umieszczane na stosie przez procedurę wołającą
  - Ślad powrotu
    - Pozostawiany na stosie przez instrukcję wywołania procedury
  - Zmienne lokalne
    - Tworzone przez procedurę wywołaną na początku jej wykoanania

Wyjątek - definicja

Zdarzenie w systemie komputerowym
wymagające przerwania wykonania bieżącej sekwencji instrukcji
i przekazania sterowania do systemu operacyjnego

Podział wyjątków

- Asynchroniczne – nie wynikające bezpośrednio z wykonywanych instrukcji lub niemożliwe do powiązania z instrukcją
  - Przerwania (interrupts) – obsługiwane w zależności od priorytetu procesora i przerwania (obsługa może nastąpić po jakimś czasie od wygenerowania)
  - (w niektórych architekturach) błędy nienaprawialne
- Synchroniczne – wynikające z wykonania instrukcji, obsługiwane natychmiast po wygenerowaniu
  - Pułapki (traps)
  - Błędy (errors)
    - Naprawialne (faults)
    - Nienaprawialne (aborts)

Przerwania

- Powstają poza procesorem
  - za wyjątkiem asynchronicznego przerwania programowego
- Asynchroniczne względem wykonywanego strumienia instrukcji
  - ich wystąpienie nie jest bezpośrednim wynikiem wykonania konkretnej instrukcji
- Służą do sygnalizacji zdarzeń istotnych dla systemu operacyjnego
  - zmiana kontekstu urządzeń zewnętrznych, np.
    - wciśnięcie klawisza na klawiaturze, przesunięcie myszy
    - nadejście pakietu z sieci lokalnej
    - zakończenie transmisji danych do/z pamięci masowej
  - upłynięcie określonego odcinka czasu
    - timer systemowy służący do periodycznego przełączania procesów w systemie wieloprocesowym
    - „budzenie” procesu o określonym czasie

Priorytet procesora i przerwania

- Procesor rozpoczyna obsługę przerwania, gdy priorytet oczekującego przerwania jest wyższy od aktualnego priorytetu procesora.
  - Wyjątki synchroniczne są obsługiwane niezależnie od priorytetu
  - Priorytet procesora jest przechowywany w jednym z rejestrów systemowych, zwykle w rejestrze stanu
  - Przy wejściu w obsługę przerwania priorytet procesora jest ustawiany na równy priorytetowi przyjętego przerwania
- Liczba poziomów priorytetowych:
  - „jednopoziomowy” system przerwań - dwa lub trzy poziomy:
    - Wątku (niższy) – procesor przyjmuje przerwania
    - Przerwania (wyższy) – procesor nie przyjmuje przerwań
    - (przerwania niemaskowalnego – najwyższy)
  - Wielopoziomowy system przerwań – poziom wątku i >1 poziom przerwań
    - Priorytety przerwań wynikają z pilności ich programowej obsługi

Priorytety przerwań

- Przerwania mogą mieć przypisane różnych poziomów priorytetów
  - przerwania od szybkich urządzeń – wyższe poziomy
  - przerwania od wolnych urządzeń – niższe poziomy
- priorytet procesora może być zmieniany programowo tylko przez oprogramowanie systemowe
  - wykonanie niektórych procedur jądra wymaga zablokowania przerwań (sekcje krytyczne)
- Podczas wykonywania kodu aplikacji procesor ma najniższy priorytet

Ankieta

https://plbe.nobleprog.com/open-tef/2455