



WSTĘP DO INFORMATYKI BRANŻOWEJ

Od kół zębatych do stanów kwantowych

Agenda

- ▶ Wstęp
- ▶ Historia
- ▶ Architektura komputerów
- ▶ Języki Programowania
 - Typy danych i tablice
 - Podstawowe algorytmy
 - Struktury danych
- ▶ Systemy Operacyjne
 - Wielozadaniowość
 - Procesy i Wątki
- ▶ Przyszłość

Wstęp

- ▶ Karol Gasiński

[http://www.linkedin.com/
kuktus@gmail.com](http://www.linkedin.com/kuktus@gmail.com)

- ▶ Graphic Software Engineer @ Intel

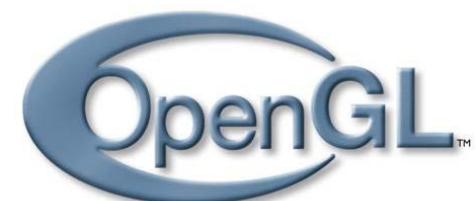
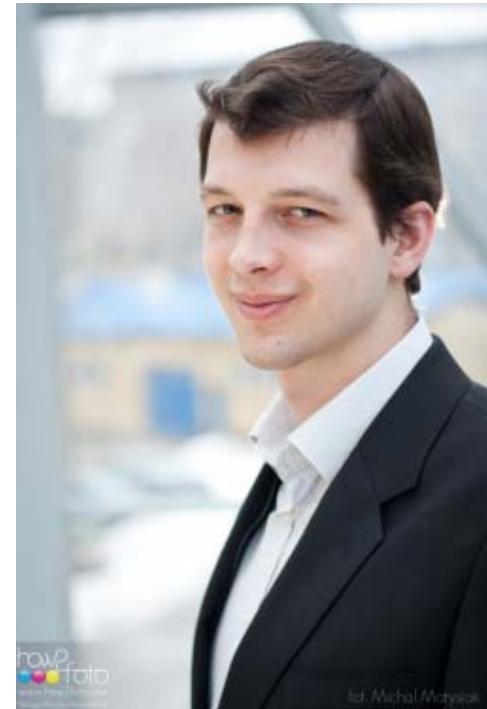
<http://www.intel.com/>

- ▶ Jako członek grupy KHRONOS,
uczestnicząc w pracach nad OpenGL

<http://www.opengl.org/>

- ▶ Projekty:

- Biblioteka graficzna dla Dos'a
- Wieloprocesowy System Operacyjny
- Wciąż rozwijany multiplatformowy silnik
(Android/BlackBerry/iOS/Windows)





WGK 2012 - zdjęcia i filmy w nowej jakości



Zarejestruj się

Sponsorki Złote



Internet Marketing



gry

WGK 2012 - zdjęcia i filmy w nowej jakości

opublikowane: 15.11.2012, przesy: Karol Kwaśkowski

Dostępny jest już komplet materiałów z minionego wydania WGK. Dostępne są one na wielu stronach internetowych organizatorów konferencji, do której należały m.in. delegaci, sprawi, edycja i nagrania video. Są one zawsze najnowszy możliwy moment w naszej galerii z kolei filmy najnowszych jest obejmująca naszym kanału YouTube (wraz z materiałami od Gamezilla.pl). Przy okazji mamy nadzieję, że Was ciekawostki związane z BlackBerry... [czytaj dalej]



Szczerze o WGK 2012 - okiem jednego z organizatorów

opublikowane: 09.10.2012, przesy: Karol Kwaśkowski

31 sierpnia - 2 września 2012. To były dni, byli dla mnie dniem święta. Jako dla jednego z organizatorów, były wyjątkiem mimo iż nie mieściły pracy. Jako, że dla jednego z uczestników, WGK była wyszukiwaną okazją, by skorzystać z wykładów, porozmawiać nim, zebrać się nad czym pracuje. Z obu perspektyw konferencję uważa za niebiańskie udane. [czytaj dalej]



Gamers Area - strefa dla graczy, wstęp wolny!

opublikowane: 19.08.2012, przesy: Karol Kwaśkowski

Konferencja zaczyna się już południe, ale my wciąż nie wycofaliśmy się z niezapomnianek. Keling, jest inicjatywa Gamers Area - specjalny obszar WGK, na który składają się wiele atrakcji dla graczy. W tym roku pojawiły się nowoczesne gry, edukacyjne, wciel się w przedstawiciela miedzianej brudzieli (Developera Śliewskiego, wiejskiej piałkarni). Zapraszamy wam również zebrać się jednym z kilku stawieniaków PlayStation 3. Koniecznie zajrzyjcie na Gamers Area, zaproście się znajomych! [czytaj dalej]

Archiwum nowości w

© 2010-2012 Conference Committee. All Rights Reserved. Design by Maciej Olskowski.

Wstęp

- ▶ Karol Gasiński
- ▶ Founder And Chairman @ WGK
 - <http://wgk.pgda.pl/>
 - 250+ uczestników
 - 25~30 wykładów
 - 2 edycje
 - 3 dni
- ▶ Prowadzę wykłady na:
 - Politechnika Gdańska
 - GameDay
 - WGK
 - IGK, ...
- ▶ Można mnie również spotkać na GDC San Francisco, GDC Europe, GamesCom, ...

Wstęp

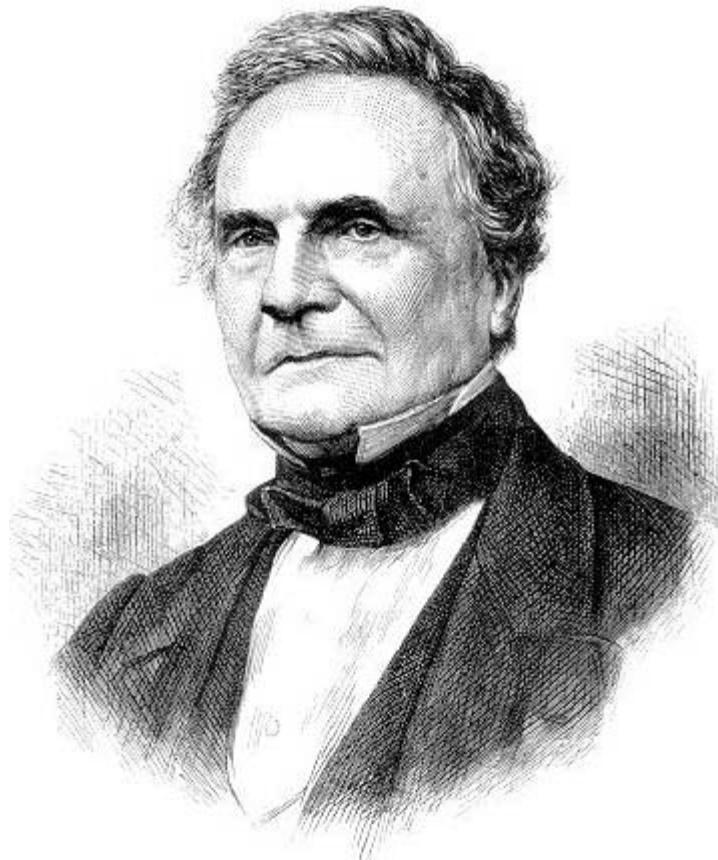
- ▶ Karol Gasiński
- ▶ Game Developer @ TBA
<http://mrkaktus.org/>
- ▶ Pracowałem nad mobilnymi wersjami:
 - *Medieval Total War*
 - *Pro Evolution Soccer*
 - *Silent Hill*
 - *Invaders Must Die!*
- ▶ Obecnie w developmencie:
 - *Invaders Must Die 2*
 - *Heroic Sci-Fi Title TBA*



Agenda

- ▶ Wstęp
- ▶ Historia
- ▶ Architektura komputerów
- ▶ Języki Programowania
 - Typy danych i tablice
 - Podstawowe algorytmy
 - Struktury danych
- ▶ Systemy Operacyjne
 - Wielozadaniowość
 - Procesy i Wątki
- ▶ Przyszłość

Historia

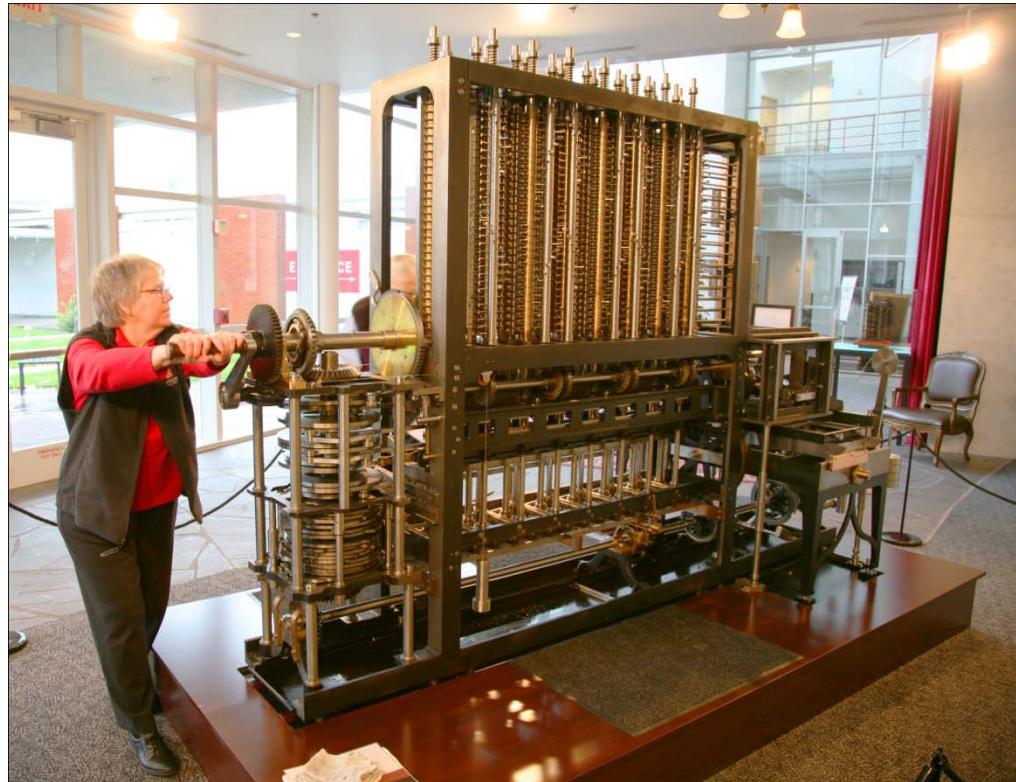


Charles Babbage, 1791–1871

Historia

- ▶ *Computer – człowiek wykonujący obliczenia*
- ▶ 1822 – *Maszyna różnicowa (projekt)*, miała wykonywać obliczenia do tablic matematycznych dla funkcji wielomianowych.
- ▶ 1837 – *Maszyna analityczna (projekt)*, pierwszy programowalny komputer ogólnego przeznaczenia. Miała być mechaniczną konstrukcją napędzaną przez silnik parowy.

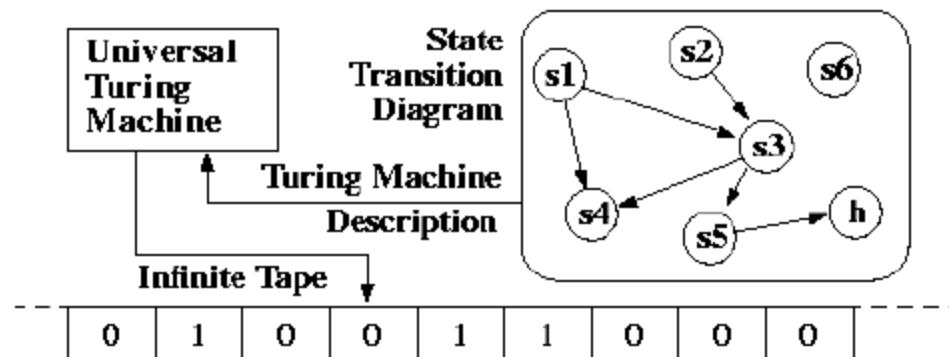
Historia



Maszyna różnicowa Babbage'a zbudowana przez
zespół z Londyńskiego Muzeum Nauki

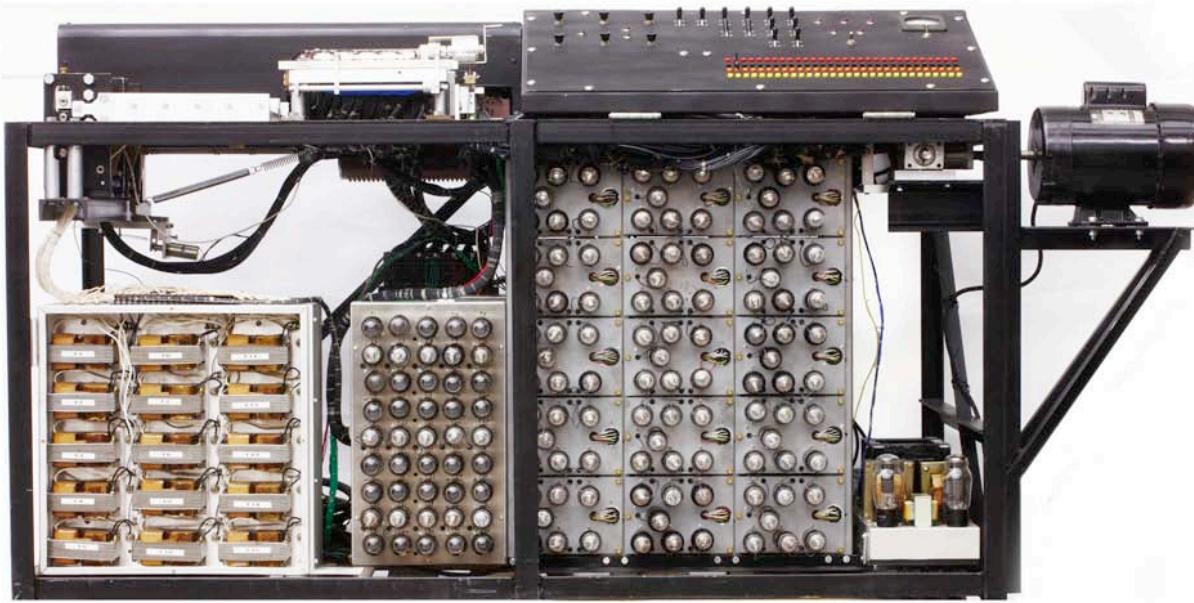
Historia

- ▶ 1930 – Vannevar Bush buduje *Analizator różniczkowy*, najbardziej skomplikowaną mechaniczną maszynę liczącą.
- ▶ 1936 – Alan Turing przedstawia opis *Maszyny Turinga*, pozwalającej sformalizować idee algorytmu i programu



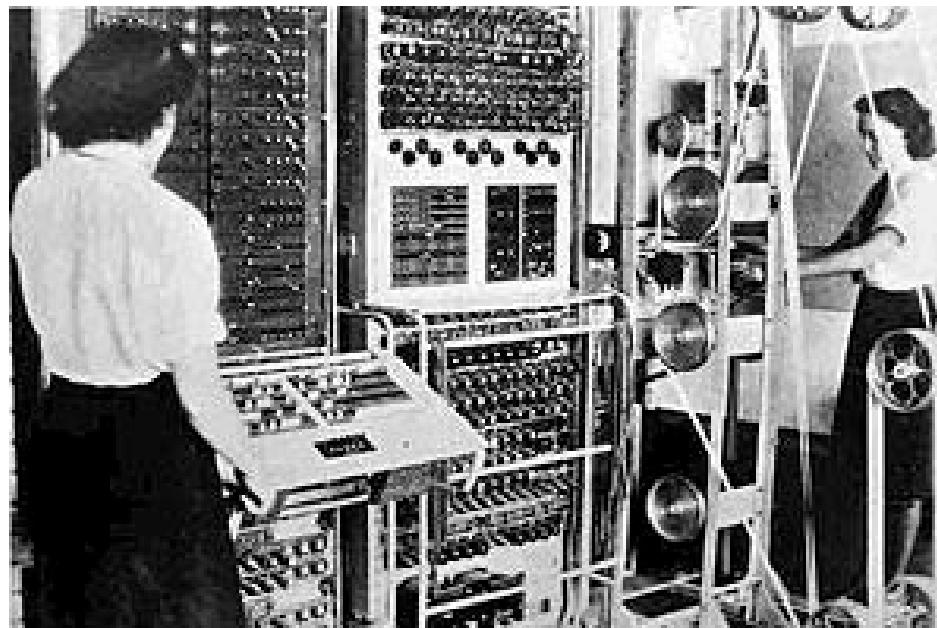
Historia

- ▶ 1937 – *Atanasoff-Berry Computer (ABC)* – pierwszy cyfrowy komputer (ale nie programowalny)



Historia

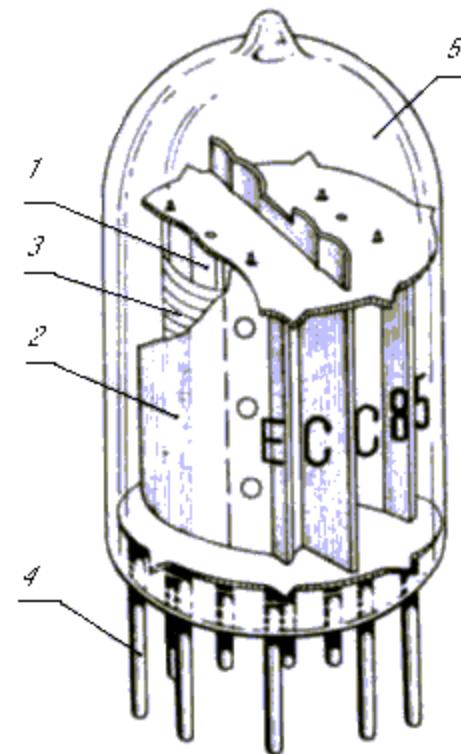
- ▶ 1943 – *Colossus* – pierwszy cyfrowy w pełni programowalny komputer. Był używany do złamania szyfru Lorenza podczas WWII.



Historia

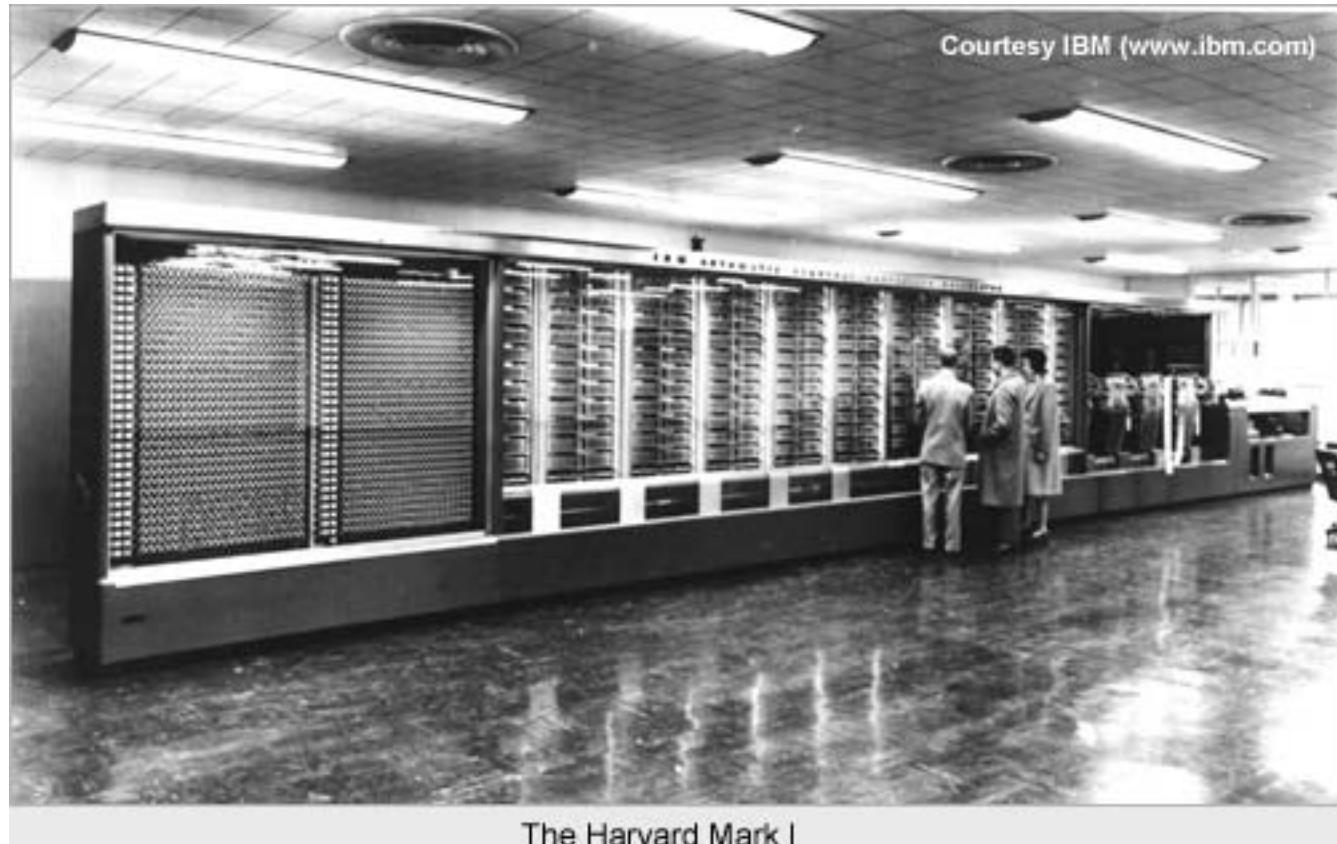
- ▶ Z początku do budowy komputerów używano mechanicznych przełączników elektromagnetycznych i lamp elektronowych.
- ▶ Lampy grzały się jednak i często przepalały, co sprawiało że były dość zawodnym rozwiązaniem.

- 1) Katoda
- 2) Anoda
- 3) Siatka
- 4) Nóżki
- 5) Bańka



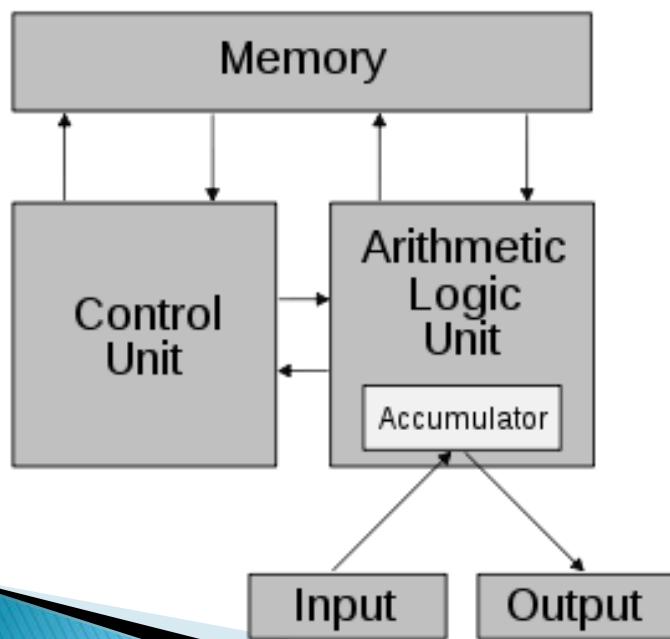
Historia

- ▶ 1944 – *Mark I* zbudowany we współpracy z IBM



Historia

- ▶ 1945 – *John Von Neumann* – proponuje nową architekturę komputera. *Architektura Von Neumanna* staje się podwaliną współczesnych komputerów.



Memory – przechowuje kod programu i jego dane. Każda komórka pamięci ma identyfikator zwany jej adresem.

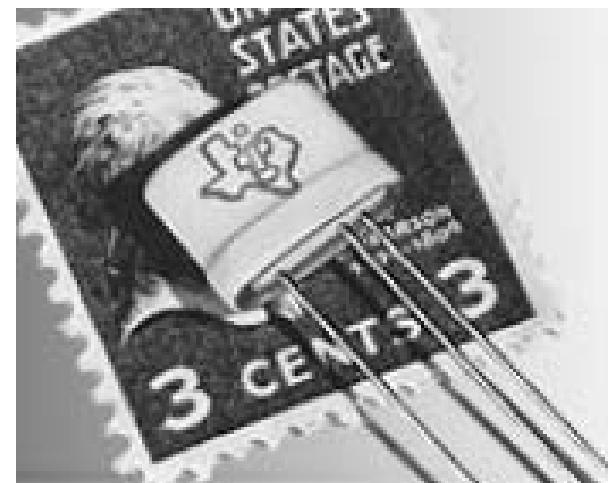
Control Unit – Odpowiedzialna za pobieranie instrukcji z pamięci i ich sekwencyjne przetwarzanie.

ALU – Wykonuje operacje arytmetyczne.

I/O – Urządzenia wejścia/wyjścia służą do interakcji z operatorem.

Historia

- ▶ 1947 – John Bardeen, William Shockley i Walter Brattain przeprowadzają pierwsze eksperymenty nad *tranzystorem*.



1954 – Pierwszy krzemowy tranzystor wyprodukowany przez Texas Instruments.

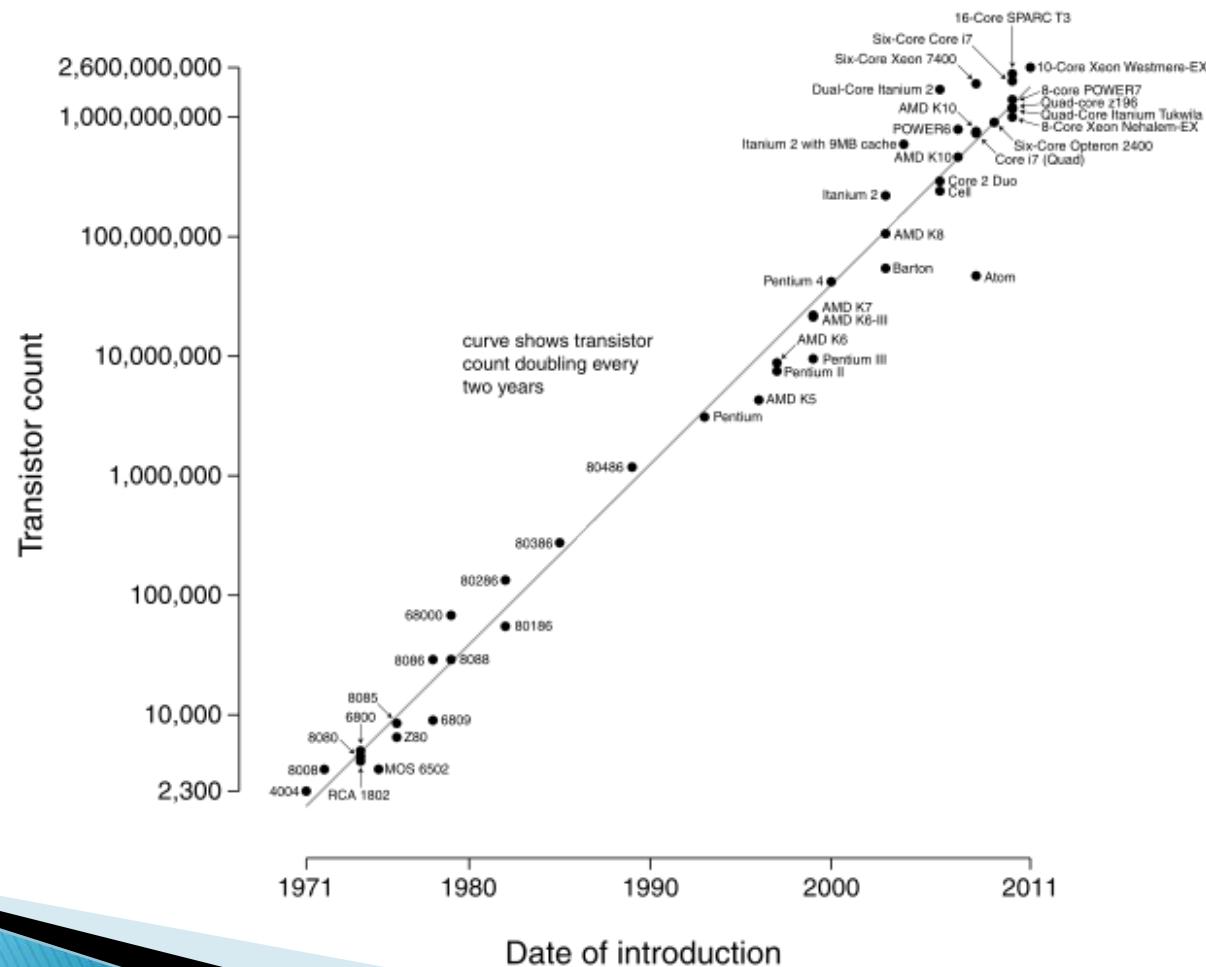
Historia

- ▶ 1968 – Gordon Moore i Robert Noyce zakładają firmę Int_el (od Integrated Electronics).
- ▶ 1970 – powstaje prawo Moore'a – co dwa lata ilość tranzystorów ma wzrastać dwukrotnie.



Historia

Microprocessor Transistor Counts 1971-2011 & Moore's Law



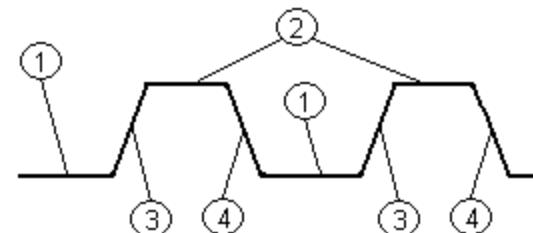
Agenda

- ▶ Wstęp
- ▶ Historia
- ▶ Architektura komputerów
- ▶ Języki Programowania
 - Typy danych i tablice
 - Podstawowe algorytmy
 - Struktury danych
- ▶ Systemy Operacyjne
 - Wielozadaniowość
 - Procesy i Wątki
- ▶ Przyszłość

Architektura komputera

- ▶ Podstawą działania komputerów jest zmiana napięcia sygnału.
- ▶ Napięcie niskie (np. 0.8V) odpowiada stanowi spoczynku, czyli „0”
- ▶ Napięcie wysokie (np. 1.2V) odpowiada stanowi wysokiemu, czyli „1” (napięcie potrzebne do pobudzenia innych układów)

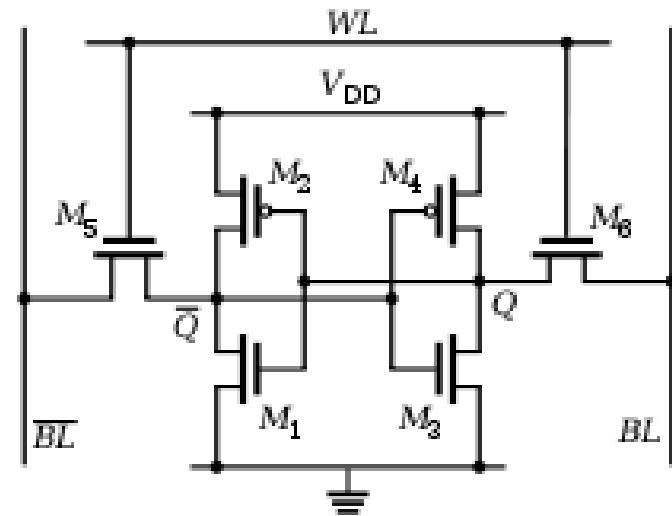
- 1) „0” – 0.8V brak pobudzenia układu
- 2) „1” – 1.2V pobudzenie układu
- 3) Zbocze narastające
- 4) Zbocze opadające



Patrz przedmiot „Układy Cyfrowe”

Architektura komputera

- ▶ Ponieważ istnieją tylko 2 stany – „0” i „1”, komputer jest maszyną binarną (dwójkową).
- ▶ Pojedynczy stan nazywany jest *bitem*.
- ▶ 4 bity tworzą *nibble*.
- ▶ Za podstawową jednostkę pamięci przyjęto się jednak *bajt*, czyli zbiór 8 bitów.



Pojedyncza komórka pamięci SRAM składająca się z 6 tranzystorów
(przechowuje 1 bit).

Architektura komputera

- ▶ Komputery których jednostki ALU wykonywały obliczenia na bajtach, nazywano 8 bitowymi.
- ▶ Przykładami takich komputerów są np.:
- ▶ Commodore 64, ZX Spectrum, Sinclair...



Montezumas Revenge
(którego nie dało się wygrać, ale gracze o tym nie wiedzieli ☺)

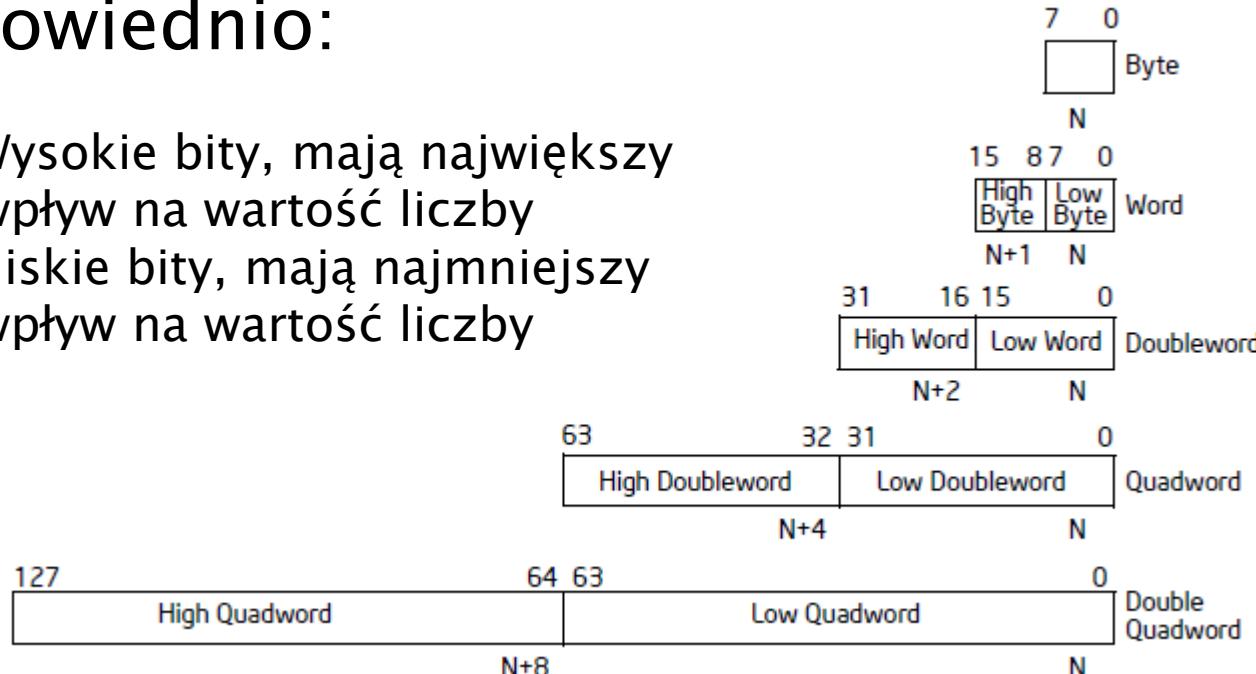
Więcej można wyczytać z listu Darka Drawsa do autora gry:
<http://idn.org.pl/users/lesz/legend/montezuma.html>

Architektura komputera

- ▶ Jednostki opisujące 16 bitowe, 32 bitowe, 64 bitowe i większe obszary danych nazywa się odpowiednio:

HI – Wysokie bity, mają największy wpływ na wartość liczby

LO – Niskie bity, mają najmniejszy wpływ na wartość liczby



Double Quadword jest też zwany OWORD - Octaword

Architektura komputera

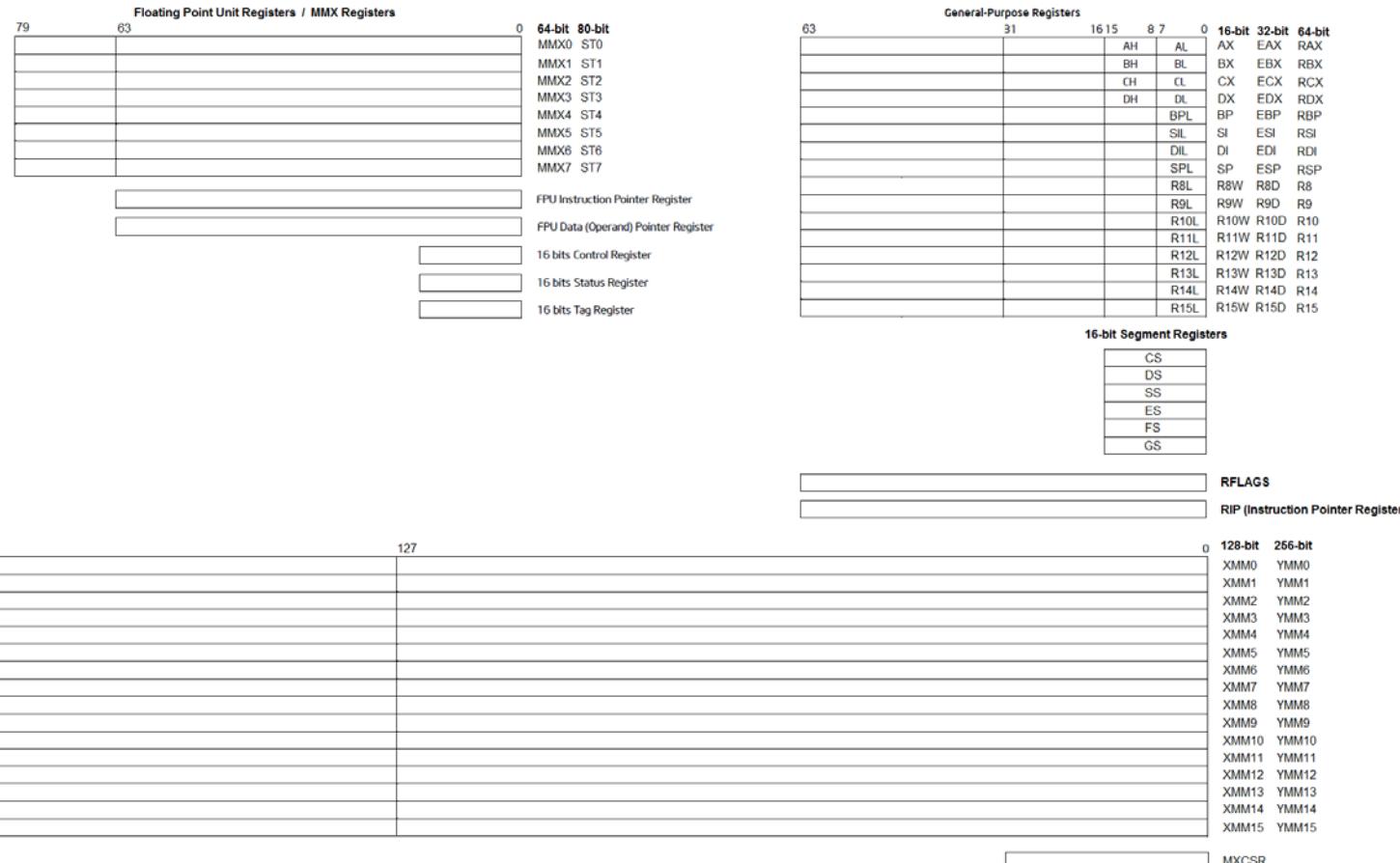
- ▶ Komputery z rodziny „x86”, są 32bitowe, czyli posiadają jednostki ALU operujące na 32bitowych rejestrach.

General-Purpose Registers				16-bit	32-bit
31	16 15	8 7	0		
		AH	AL	AX	EAX
		BH	BL	BX	EBX
		CH	CL	CX	ECX
		DH	DL	DX	EDX
		BP		EBP	
		SI		ESI	
		DI		EDI	
		SP		ESP	

- ▶ Rejestry przechowują tymczasowe wyniki operacji przed zapisaniem do pamięci.

Architektura komputera

- ▶ Obecnie w sprzedaży są już tylko komputery 64 bitowe. Aktualny zestaw rejestrów poniżej:



Architektura komputera

- Obecnie w sprzedaży są już tylko komputery 64 bitowe. Aktualny zestaw rejestrów poniżej:

Floating Point Unit Registers / MMX Registers		0 64-bit 80-bit		General Purpose Registers		0 16-bit 32-bit 4-bit	
79	63	MMX0	ST0	31	AH AL	AX EAX	RAX
		MMX1	ST1		BH BL	BX EBX	RBX
		MMX2	ST2		CH CL	CX ECX	RCX
		MMX3	ST3		DH DL	DX EDX	RDX
		MMX4	ST4		BPL	BP EBP	EBP
		MMX5	ST5		SIL	SI ESI	RSI
		MMX6	ST6		DIL	DI EDI	EDI
		MMX7	ST7		SPL	SP ESP	ESP
						R8L R8W R8D R8	
						R9L R9W R9D R9	
						R10L R10W R10D R10	
						R11L R11W R11D R11	
						R12L R12W R12D R12	
						R13L R13W R13D R13	
						R14L R14W R14D R14	
						R15L R15W R15D R15	
16-bit Segment Registers		16-bit		RFLAGS		RIP (Instruction Pointer Register)	
255	127	CS					
		DS					
		SS					
		ES					
		FS					
		GS					
128-bit 256-bit		0 128-bit 256-bit		MXCSR		MXCSR	
		XMM0	YMM0				
		XMM1	YMM1				
		XMM2	YMM2				
		XMM3	YMM3				
		XMM4	YMM4				
		XMM5	YMM5				
		XMM6	YMM6				
		XMM7	YMM7				
		XMM8	YMM8				
		XMM9	YMM9				
		XMM10	YMM10				
		XMM11	YMM11				
		XMM12	YMM12				
		XMM13	YMM13				
		XMM14	YMM14				
		XMM15	YMM15				

Architektura komputera

- ▶ Ludzie przyjęli system dziesiętny ze względu na ilość palców, np.:

$$1984 = (1 * 10^3) + (9 * 10^2) + (8 * 10^1) + (4 * 10^0)$$

- ▶ Komputery zmuszone są używać systemu dwójkowego:

$$\begin{aligned} 1984 &= 1024 + 512 + 256 + 128 + 64 = \\ &= (1 * 2^{10}) + (1 * 2^9) + (1 * 2^8) + (1 * 2^7) + (1 * 2^6) + (0 * 2^5) + \\ &\quad (0 * 2^4) + (0 * 2^3) + (0 * 2^2) + (0 * 2^1) + (0 * 2^0) = \\ &= 11111000000b \end{aligned}$$

- ▶ Jak widać liczba 1984 będzie przechowana w 16 bitowym rejestrze.

Architektura komputera

- ▶ Ponieważ zapis w systemie dwójkowym jest za długi, a zapis w systemie dziesiętnym nieadekwatny, popularny jest również system szesnastkowy:
- ▶ W systemie szesnastkowym dane opisujemy per nibble:

0000b - 0	0100b - 4	1000b - 8	1100 - C
0001b - 1	0101b - 5	1001b - 9	1101 - D
0010b - 2	0110b - 6	1010b - A	1110 - E
0011b - 3	0111b - 7	1011b - B	1111 - F

- ▶ Stosujemy prefix „0x” lub postfix „h” :

$$1984 = 11111000000b = 0x7F0 = 7F0h$$

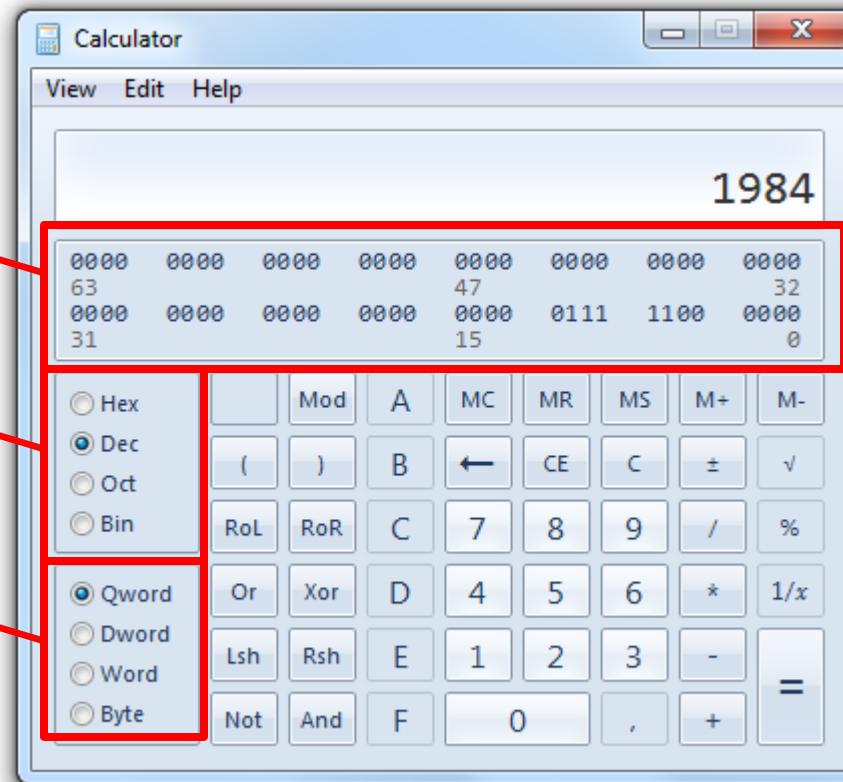
Architektura komputera

- ▶ W Windows 7 pomocny może okazać się kalkulator systemowy w trybie „Programmer”

Reprezentacja
Binarna w rejestrze

Aktualny system
liczbowy

Rozmiar rejestru



Architektura komputera

▶ Typy całkowite (*integer*)

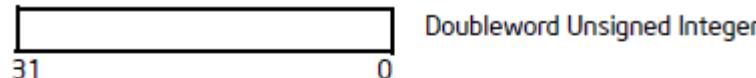
[0 ... 255]



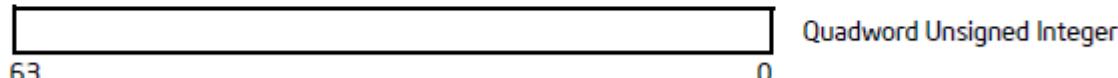
[0 ... $2^{16}-1$]



[0 ... $2^{32}-1$]



[0 ... $2^{64}-1$]



$$2^{16}-1 = 65.535$$

$$2^{32}-1 = 4.294.967.295$$

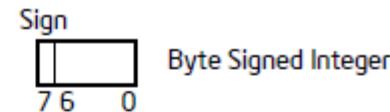
$$2^{64}-1 = 18.446.744.073.709.551.615$$

300.000.000.000 ← Szacowana ilość wszystkich gwiazd w naszej galaktyce.

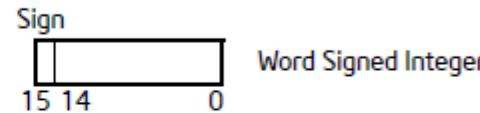
Architektura komputera

▶ Typy całkowite „ze znakiem”

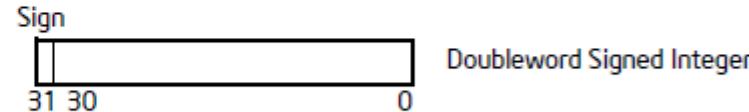
$[-128 \dots 127]$



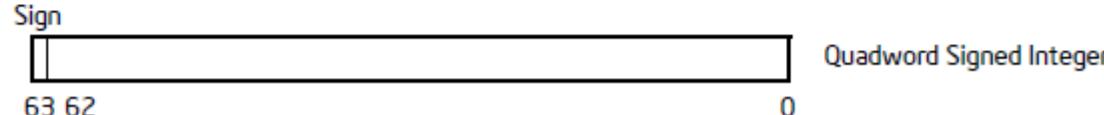
$[-32768 \dots 32767]$



$[-2^{31} \dots 2^{31}-1]$



$[-2^{63} \dots 2^{63}-1]$



▶ Ostatni, wysoki bit zapalony oznacza liczbę ujemną. W przyjętej arytmetyce brak -0 stąd -128 i „tylko” $+127$.

$$11111111b = -1$$

$$\dots$$

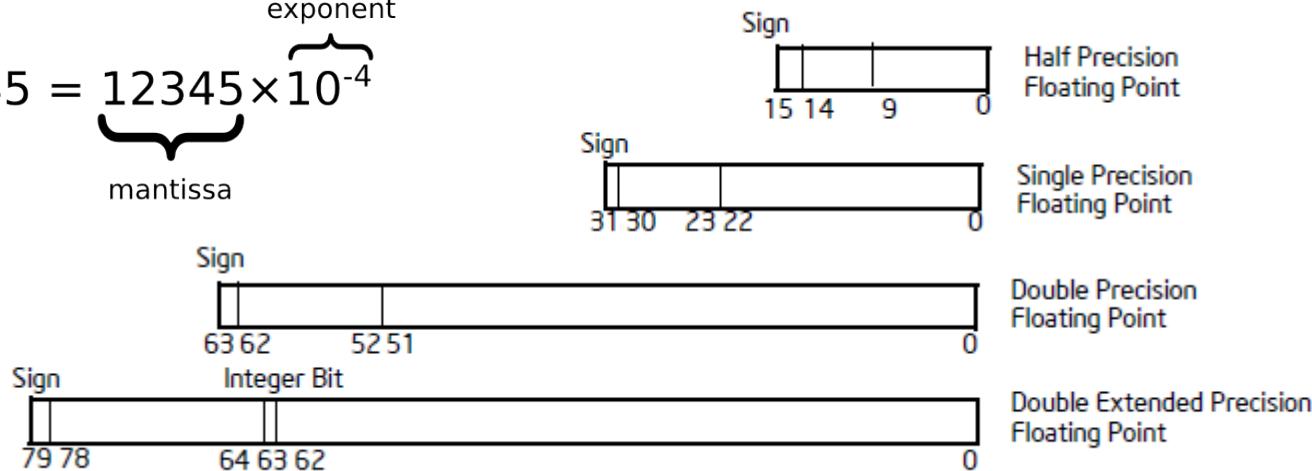
$$10000000b = -128$$

Architektura komputera

▶ Typy zmiennoprzecinkowe (*floating point*)

$$1.2345 = 12345 \times 10^{-4}$$

exponent
mantissa



Bity: Znak Exponent Mantysta

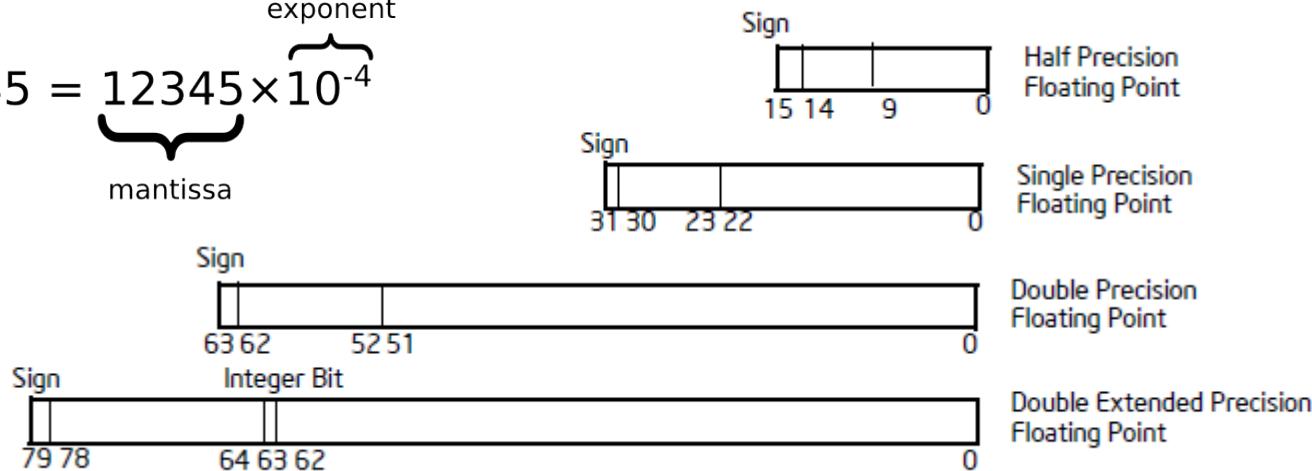
1	5	10	Half Precision
1	8	23	Single Precision
1	11	52	Double Precision
1	15	64	Double Extended Precision

Architektura komputera

▶ Typy zmiennoprzecinkowe (*floating point*)

$$1.2345 = 12345 \times 10^{-4}$$

exponent
mantissa



$$3.10 \times 10^{-5} \dots 6.50 \times 10^4$$

Half Precision

$$1.18 \times 10^{-38} \dots 3.40 \times 10^{38}$$

Single Precision

$$2.23 \times 10^{-308} \dots 1.79 \times 10^{308}$$

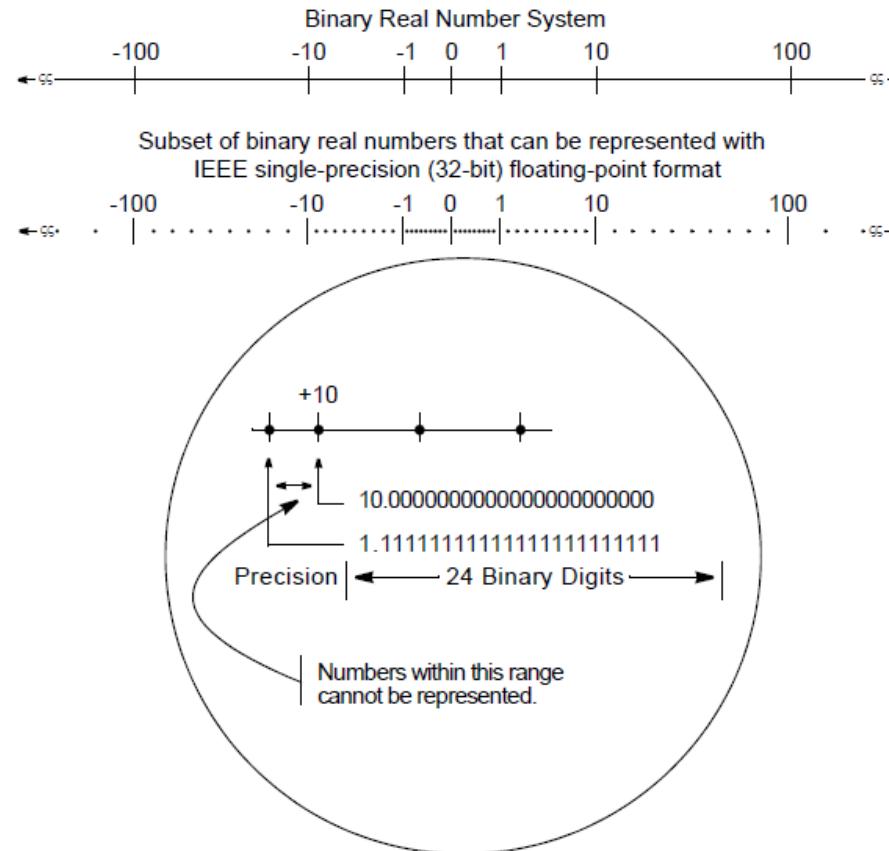
Double Precision

$$3.37 \times 10^{-4932} \dots 1.18 \times 10^{4932}$$

Double Extended Precision

Architektura komputera

▶ Typy zmiennoprzecinkowe (*floating point*)



Patrz przedmiot „Metody Numeryczne”

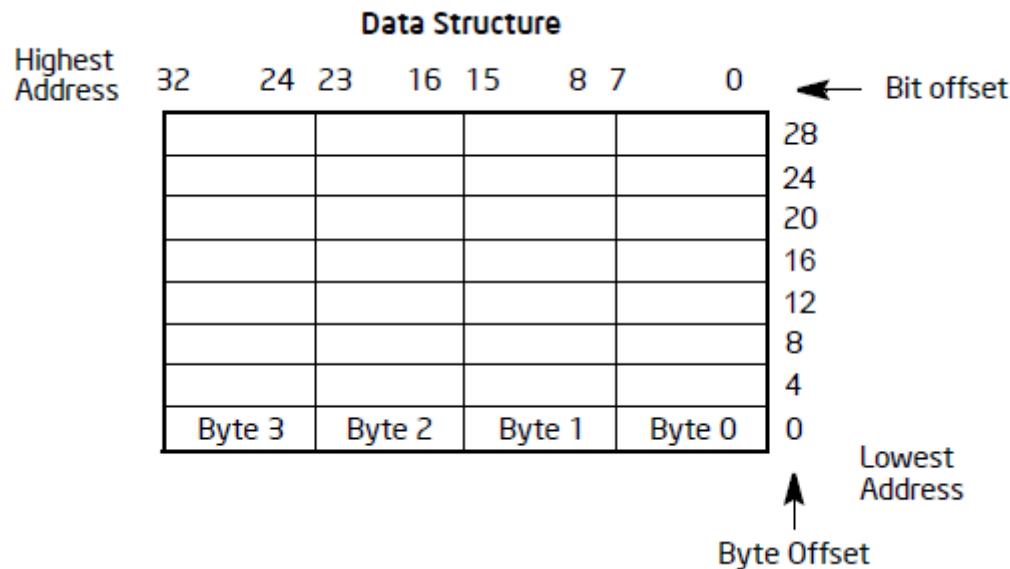
Architektura komputera

Wielokrotności bajtów						
Przedrostki dziesiętne (SI)			Przedrostki binarne (IEC 60027-2)			
Nazwa	Symbol	Mnożnik	Nazwa	Symbol	Mnożnik	
kilobajt	kB	$10^3=1000^1$	kibabajt	KiB	$2^{10}=1024^1$	
megabajt	MB	$10^6=1000^2$	mebibabajt	MiB	$2^{20}=1024^2$	
gigabajt	GB	$10^9=1000^3$	gibabajt	GiB	$2^{30}=1024^3$	
terabajt	TB	$10^{12}=1000^4$	tebibabajt	TiB	$2^{40}=1024^4$	
petabajt	PB	$10^{15}=1000^5$	pebibabajt	PiB	$2^{50}=1024^5$	
eksabajt	EB	$10^{18}=1000^6$	eksbibabajt	EiB	$2^{60}=1024^6$	
zettabajt	ZB	$10^{21}=1000^7$	zebibabajt	ZiB	$2^{70}=1024^7$	
jottabajt	YB	$10^{24}=1000^8$	jobabajt	YiB	$2^{80}=1024^8$	

- ▶ Jednostką ilości danych jest bajt.
- ▶ Przy czym w powszechnym użyciu stosuje się symbole dziesiętne mając na myśli wartości binarne, czyli:
 $1\text{ KB} = 1024\text{ B}$ etc.

Architektura komputera

- ▶ *RAM (Random Access Memory)* – pamięć operacyjna przechowująca programy i dane podczas ich przetwarzania.



- ▶ Pamięć jest ciągła. Odwołujemy się do niej za pomocą adresów komórek.

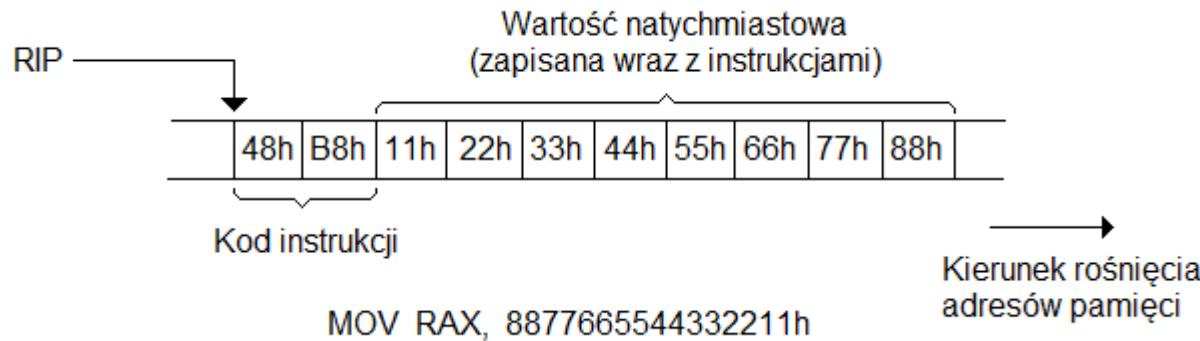
Agenda

- ▶ Wstęp
- ▶ Historia
- ▶ Architektura komputerów
- ▶ Języki Programowania
 - Typy danych i tablice
 - Podstawowe algorytmy
 - Struktury danych
- ▶ Systemy Operacyjne
 - Wielozadaniowość
 - Procesy i Wątki
- ▶ Przyszłość

Języki programowania

Kod Maszynowy

- ▶ Rejestr RIP wskazuje adres, w pamięci przechowującej program, spod którego procesor ma odczytać kolejną instrukcję do wykonania np.:



- ▶ Instrukcje mogą być różnej długości, a ich ciąg tworzy polecenia z których składa się program.

Języki programowania

Assemblery i Kompilatory

- ▶ Złożenie (*to assemble*) ciągu instrukcji zrozumiałych dla człowieka w ciąg bajtów tworzących program nazywamy komplikacją.
- ▶ Program dokonujący tego zadania z najprostszego opisu kodu nazywa się *assemblerem* (składaczem).
- ▶ Z tego też powodu kod maszynowy tworzący ciąg takich instrukcji jest potocznie również nazywany *assemblerem*.
- ▶ Zamiana kodu bardziej skomplikowanych języków programowania nazywana jest *kompilacją* .

Języki programowania

Języki kompilowalne

- ▶ Języki kompilowalne to takie których kod jest przetwarzany do kodu maszynowego i może być wykonany bezpośrednio przez procesor.
- ▶ Zalety:
 - Kod wykonywany bezpośrednio na procesorze, bardzo szybki
- ▶ Wady:
 - Każda zmiana w programie, wymaga jego rekompilacji przed ponownym uruchomieniem
- ▶ Przykłady:
 - Assembler, C, C++

Języki programowania

Języki kompilowalne – Assembler

- ▶ Przykładowy kod menedżera pamięci systemu operacyjnego:
- ▶ Programowanie w assemblerze jest żmudną pracą a kod jest wrażliwy na błędy.
- ▶ Dlatego w 1954 powstał *Fortran*.

```
; Utworzenie PT's i wpisów w nich

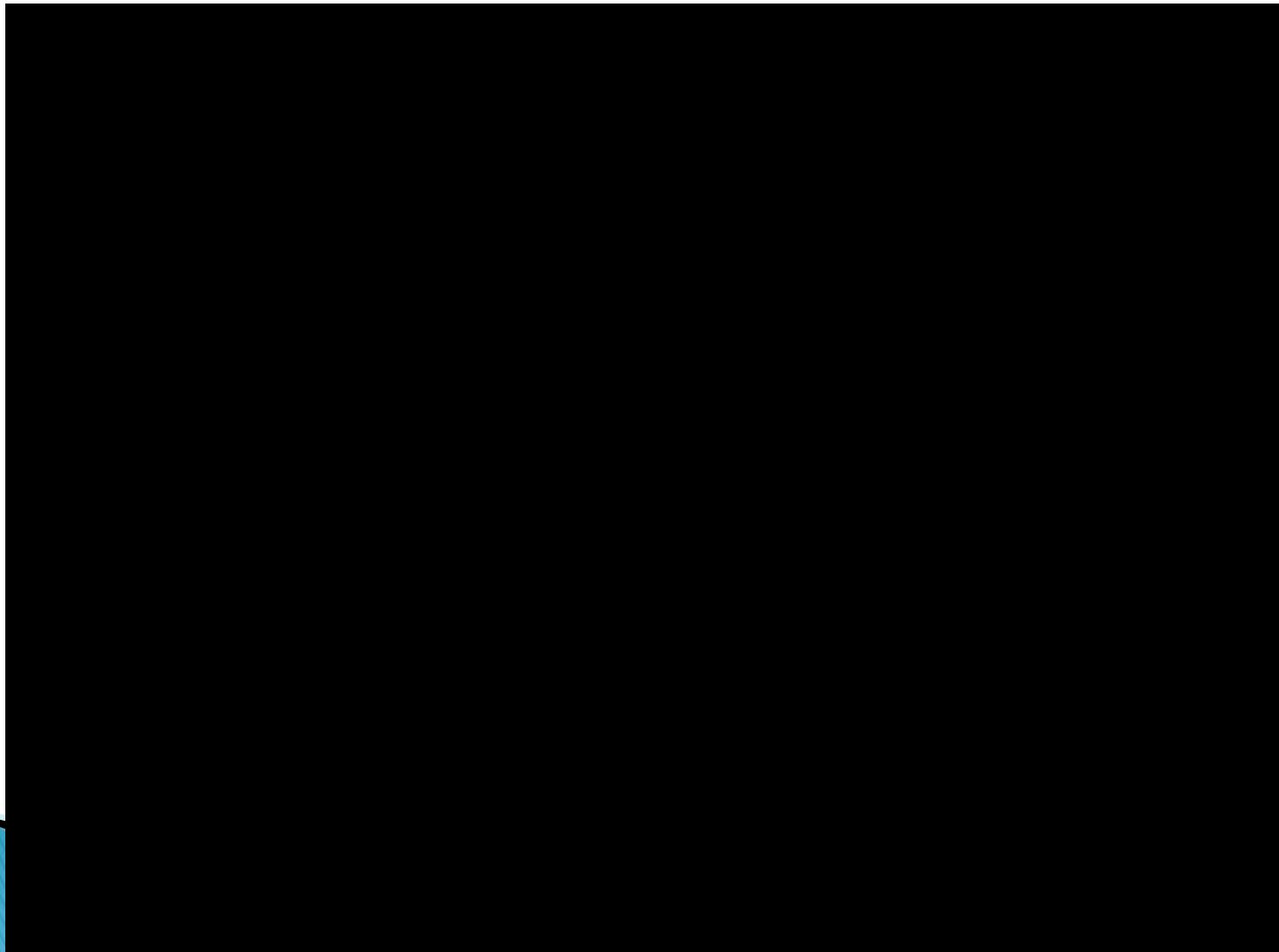
mov edi, [AdrPD]      ; EDI - Wskaz w PD
xor ecx, ecx          ; Globalny licznik ramek
mmip_p:
    mov eax, ecx      ;\ Jezeli numer aktualnej ramki
    and eax, 0x3FF    ; |jest ALIGN 1024 to trzeba
    jnz mmip_d3        ;/ zaalokowac nowa PT
    push edi           ;\Odklada wskaz w PD i
    push ecx           ;/licznik ramek
    call MM_AllocFrame ;\Alokuje nowa PT i uaktualnia
    mov esi, eax        ;/wskaz w PT
    mov edi, eax        ;\ Czysci zaalokowana ramke
    mov ecx, 1           ; |dla pełnego bezpieczeństwa.
    call MM_ClearFrames ;/
    pop ecx             ;\Przywraca wskaz w PD
    pop edi             ;/i licznik ramek
    mov eax, esi         ;\Pobiera adres nowej PT
    add eax, 3            ;/i tworzy z niego PDE
    mov [edi], eax        ;\Montuje PT w PD i przesuwa
    add edi, 4            ;/wskaz w PD na kolejne pole

mmip_d3:
    mov eax, ecx      ; Pobiera numer aktualnej ramki
    shl eax, 12         ;\Na podstawie numeru ramki
    add eax, 3            ;/tworzy jej PTE
    mov [esi], eax        ; Zapisuje PTE ramki
    add esi, 4            ; Przesuwa wskaz w PT
    inc ecx             ; Przesun numer na kolejna ramke
    cmp ecx, [Frames]   ;\Jezeli nie zainicjalowal wszystkich
    jb mmip_p            ;/ramek to zapetla
```

Historia języków programowania:
http://www.levenez.com/lang/lang_a4.pdf

Języki programowania

Języki kompilowalne – C / C++



Języki programowania

Języki kompilowalne - C / C++

- ▶ “Our rock stars aren’t like your rock stars” ☺



Ken Thompson i Dennis Ritchie,
twórcy języka programowania C.

Języki programowania

Języki kompilowalne - C / C++

- ▶ Język C został stworzony z myślą o pisaniu systemów operacyjnych na początku lat 70.
- ▶ Umożliwia on szczegółowe zarządzanie zasobami komputera jak pamięć czy procesor co sprawia, że programy w nim napisane są równie szybkie co te napisane w assemblerze.
- ▶ Jest to jeden z głównych powodów dla których jest on wybierany przez twórców gier komputerowych - pełna kontrola nad sprzętem.

Języki programowania

Języki kompilowalne - C / C++

- ▶ C wymagają określenia typów dla zmiennych których chcemy używać:

char – 8 bit Byte Signed Integer

short – 16 bit Word Signed Integer

int – 32 bit Doubleword Signed Integer

long long int – 64 bit Quadword Signed Integer

float – 32 bit Single Precision Floating Point

double – 64 bit Double Precision Floating Point

long double – 80 bit Extended Double Precision Floating Point

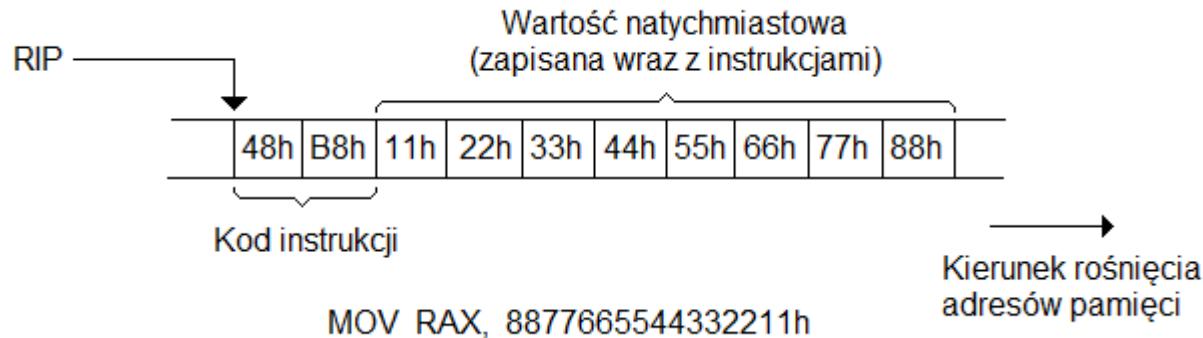
- ▶ Przedrostek „unsigned” oznacza zmienną przechowującą liczby całkowite bez znaku np.:

unsigned int – 32 bit Doubleword Integer

Języki programowania

Języki kompilowalne - C / C++

- ▶ Wracając do naszej instrukcji assemblerowej:



- ▶ Assembler:

`MOV RAX, 8877665544332211h`

- ▶ C++:

```
unsigned long long int zmienna;  
zmienna = 0x8877665544332211;
```

Języki programowania

Języki kompilowalne - C / C++

- ▶ Oznacza to jednak, że język C jest silnie typowany (*strongly typed*) i trzeba zwracać uwagę na wiele detali.

Weak Typed

```
a = 2
b = "2"
concatenate(a, b)      # Returns "22"
add(a, b)              # Returns 4
```

Strongly Typed

```
a = 2
b = "2"
concatenate(a, b)      # Type Error
add(a, b)              # Type Error
concatenate(str(a), b) # Returns "22"
add(a, int(b))         # Returns 4
```

Java Script, Perl, PHP, LUA

Action Script 3, C, C++, C#, Java

- ▶ Początkujący programiści wolą języki słabo typowane w których kompilator sam domyśla się typu danych (jeżeli jest to możliwe).

Języki programowania

IDE – Visual Studio

The screenshot shows the Microsoft Visual Studio interface with the title bar "Ngame4 - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, gDebugger, Team, BlackBerry, Data, Tools, WinGDB, Architecture, Test, Analyze, Window, Help. The toolbar has various icons for file operations. The Solution Explorer on the left lists the project "Ngame4" with two sub-projects: "Documents" and "External Dependencies". It contains files like axis.txt, codingStandard.txt, audio.h, context.h, log.h, log.inl, rendering.h, context.h, screen.cpp, screen.h, rendering.cpp, context.h, screen.cpp, strings.cpp, maintask.h, task.h, and maintask.cpp. The code editor in the center displays C++ code for a file named "storage.cpp". The code handles file reading across different platforms (Windows, BlackBerry, Android) using conditional compilation directives. The Output window at the bottom shows the build logs:

```
1 File(s) copied
0 File(s) copied
.\middleware\lib\debug\libzlibbbdbg.a
.\middleware\lib\debug\zlibdbg.lib
2 File(s) copied
FinalizeBuildStatus:
Deleting file "Device-Debug\Ngame4.unsuccessfulbuild".
Touching "Device-Debug\Ngame4.lastbuildstate".
Build succeeded.
2>Time Elapsed 00:00:36.49
===== Rebuild All: 2 succeeded, 0 failed, 0 skipped =====
```

The status bar at the bottom shows "Ready", "Ln 100", "Col 16", "Ch 16", "PL", "18:45", and the date "2012-11-19".

Języki programowania

IDE – Momentics

The screenshot shows the QNX Momentics IDE interface for developing C/C++ applications for BlackBerry Tablet OS. The main window displays the code editor for a file named `main.c`. The code implements a simple application loop that handles BPS events, specifically looking for a screen rotation event or a navigator exit event to terminate the application. The IDE also includes a Project Explorer, a Problems view, and a Welcome panel.

Project Explorer: Shows the project structure for "Sample".

Code Editor (main.c):

```
/*Signal BPS library that navigator orientation is not to be locked
if (BPS_SUCCESS != navigator_rotation_lock(false)) {
    fprintf(stderr, "navigator_rotation_lock failed\n");
    bbutil_terminate();
    screen_destroy_context(screen_ctxt);
    bps_shutdown();
    return 0;
}

while (!exit_application) {
    //Request and process all available BPS events
    bps_event_t *event = NULL;

    for(;;)
    {
        rc = bps_get_event(&event, 0);
        assert(rc == BPS_SUCCESS);

        if (event)
        {
            int domain = bps_event_get_domain(event);

            if (domain == screen_get_domain())
                handleScreenEvent(event);
            else
                if ((domain == navigator_get_domain())
                    && (NAVIGATOR_EXIT == bps_event_get_code(event)))
                    exit_application = 1;
        }
        else
            break;
    }
    render();

    //Stop requesting events from libscreen
    screen_stop_events(screen_ctxt);

    //Shut down BPS library for this process
    bps_shutdown();

    //Use utility code to terminate EGL setup
    bbutil_terminate();

    //Destroy libscreen context
    screen_destroy_context(screen_ctxt);
}
```

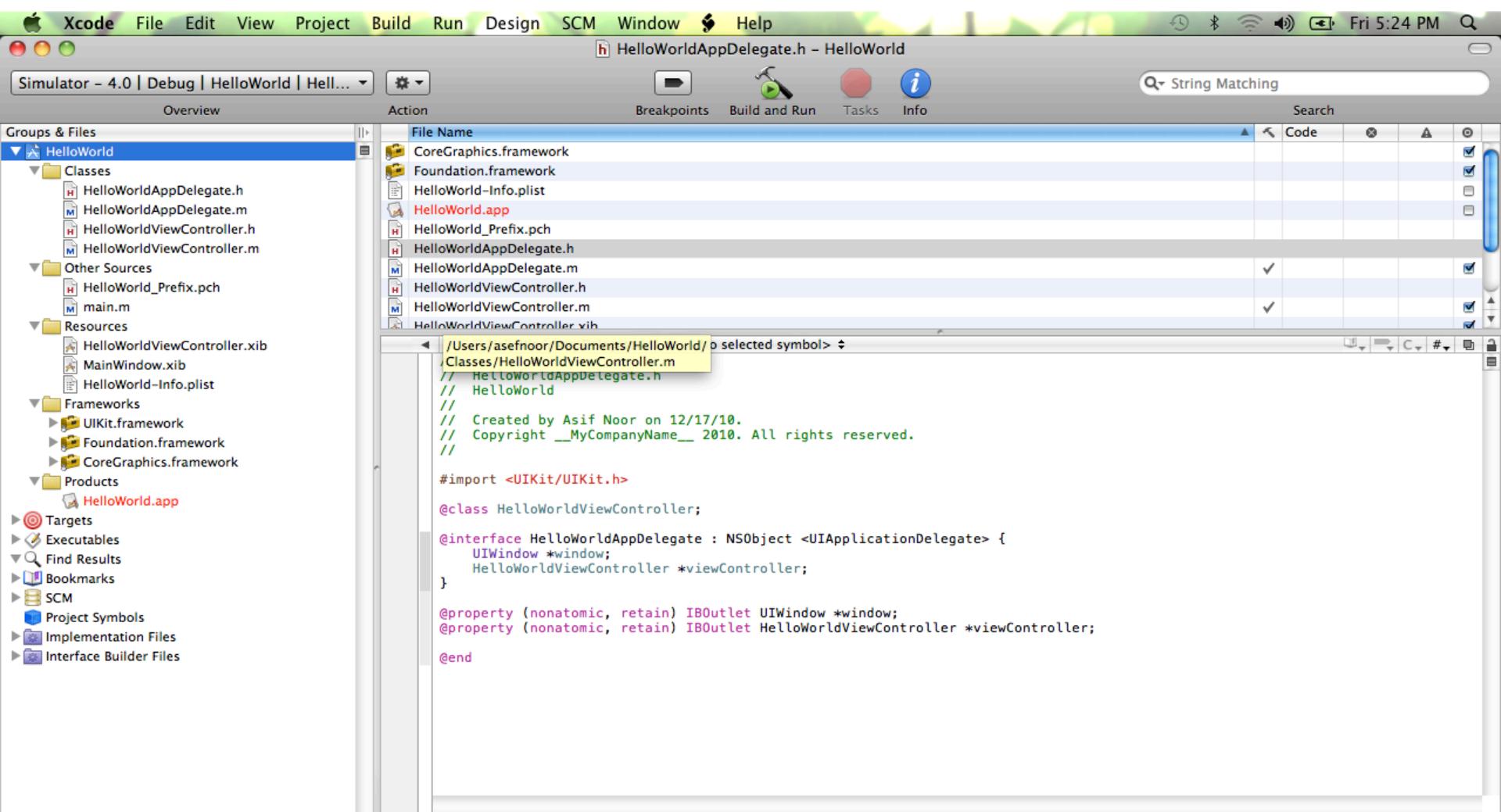
Problems View: Displays log messages related to the application's termination and OpenGL ES 2.0 module unload.

Welcome Panel: Provides links to various IDE features and information about the BlackBerry Native SDK for Tablet OS.

System Tray: Shows standard system icons for battery, signal, and time.

Języki programowania

IDE - XCode



Języki programowania

Języki skryptowe

- ▶ Języki skryptowe to takie, których kod zwany też skryptem, uruchamiany jest na procesorze za pośrednictwem innego programu w *maszynie wirtualnej*.
- ▶ Zalety:
 - Zmiany w skrypcie nie wymagają rekompilacji.
 - Możliwe jest edytowanie skryptów „w locie”.
- ▶ Wady:
 - Skrypty są wolniejsze w wykonaniu niż kod natywny z powodu interpretacji w czasie rzeczywistym.

Języki programowania

Języki skryptowe – LUA

- ▶ LUA jest najpopularniejszym językiem skryptowym stosowanym w GameDevie.
- ▶ Maszyny wirtualne LUA są implementowane w wielu silnikach do tworzenia gier.
- ▶ Innymi językami skryptowymi są np.:
 - Quake C, UnrealScript, GameMonkey, AngelScript, Python...
- ▶ LUA jest językiem luźno typowanym więc możemy po prostu napisać:

```
zmienna = 0x8877665544332211
```

Języki programowania

Języki skryptowe – LUA

- ▶ W LUA istnieje osiem abstrakcyjnych typów danych:
 - *nil* – nic, zwracany np. w przypadku błędu, brak wyniku
 - *boolean* – zmienna boolowska, *true* lub *false*
 - *number* – liczba dowolnego typu
 - *string* – ciąg znaków
 - *function* – funkcja
 - *userdata* – wskaźnik na obszar pamięci (C)
 - *thread* – wątek
 - *table* – tablica danych

Języki programowania

Języki skryptowe – LUA

- ▶ Najprostrzy program w LUA:

```
print(„Hello World”)
```

- ▶ Nie musimy deklarować żadnych nagłówków, ani miejsca rozpoczęcia programu
- ▶ Skrypt LUA jest interpretowany linijka po linijce i gdy zostanie znalezione polecenie do wykonania, zostanie ono wykonane

„Hello World” w kilkudziesięciu językach programowania:
http://en.wikipedia.org/wiki/Hello_world_program_examples

Języki programowania

Języki skryptowe – LUA – Komentarze

- ▶ Czasami przydaje się skomentować jakiś bardziej skomplikowany fragment kodu
- ▶ Komentarze są przydatne tylko dla programisty i są pomijane podczas interpretacji skryptu:

-- To jest komentarz zajmujący jedną linię

--[[-- To jest dłuższy komentarz który szczegółowo opisuje zasadę działania programu i wymaga kilku linii --]]--

Języki programowania

Języki skryptowe – LUA – Instrukcje warunkowe

- ▶ Programy składają się w większości z różnych warunków które wykonują się lub też nie w zależności od wyników poprzednich operacji.
- ▶ Najprostrza instrukcja warunkowa to *if*:

```
a = 5
```

```
b = 3
```

```
if a ~= b then
```

```
    print(„A jest rozne od B”)
```

```
end
```

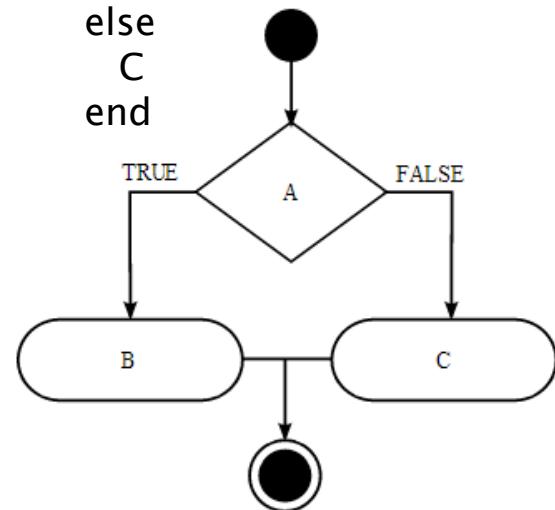
Języki programowania

Języki skryptowe – LUA – Instrukcje warunkowe

▶ Instrukcja warunkowa *if-else*:

```
a = 5  
b = 3  
if a ~= b then  
    print(„A jest rozne od B”)  
else  
    print(„A jest rowne B”)  
end
```

```
if A then  
    B  
else  
    C  
end
```



▶ Operatory porównania:

$==$ równość	$\sim=$ nierówność	not negacja
--------------	--------------------	----------------------

\leq mniejszy lub równy	\geq większy lub równy
$<$ mniejszy	$>$ większy

Języki programowania

Języki skryptowe – LUA – Operatory

▶ Operatory matematyczne:

= przypisanie

+ suma

- różnica

* mnożenie

/ dzielenie

^ potęgowanie (Shift+6)

% modulo, np:

```
print(14 % 5) -- Zwraca 4
```

.. Konkatenacja „I”..”am”..”Legend” da w rezultacie „IamLegend”

Długość, np:

```
tablica = {"do", "re", "me"}
```

```
print("Mam "..#tablica.." rzeczy") -- Mam 3 rzeczy
```

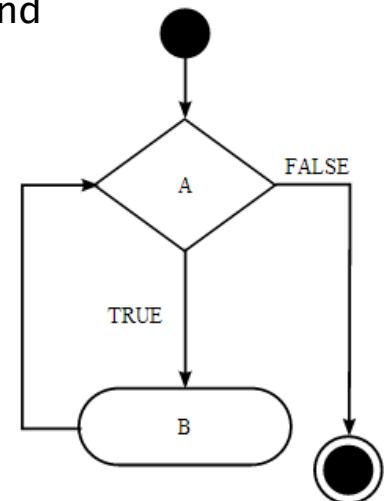
Języki programowania

Języki skryptowe – LUA – Instrukcje pętli

▶ Instrukcja warunkowa *while - do* :

```
a = 0  
while a < 2 do  
    print(„hello!”)  
    a = a + 1  
end
```

while A do
 B
end



▶ Operatory porównania:

$==$ równość	$\sim=$ nierówność	not negacja
--------------	--------------------	----------------------

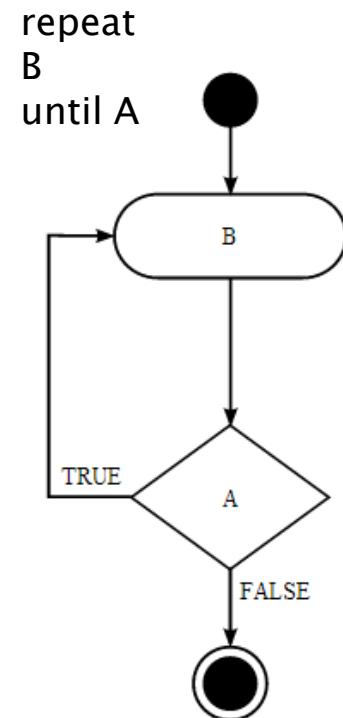
\leq mniejszy lub równy	\geq większy lub równy
$<$ mniejszy	$>$ większy

Języki programowania

Języki skryptowe – LUA – Instrukcje pętli

▶ Instrukcja warunkowa *repeat – until* :

```
a = 0
repeat
    print(„hello!”) ← Chcemy wykonać ciało
    a = a + 1           pętli minimum raz.
    if ( a == 4 )
        break
until a < 5
```



- ▶ Wykonanie aktualnej pętli można przerwać z dowolnego miejsca w jej wnętrzu za pomocą polecenia *break*.

Języki programowania

Języki skryptowe – LUA – Instrukcja pętli

▶ Instrukcja pętli *for*:

```
for i=1, 10, 2 do  
    print(i)  
end
```

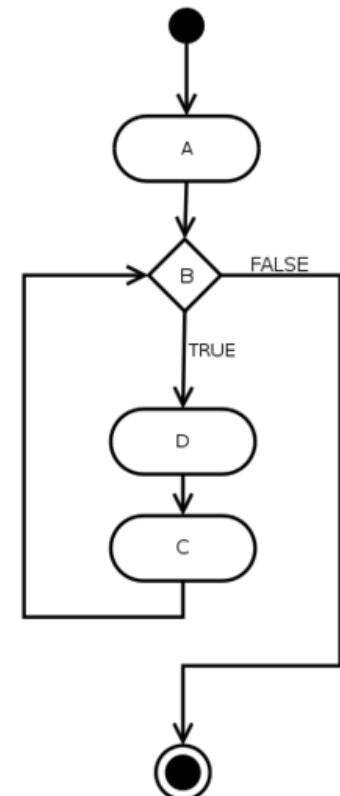
▶ Wynik:

```
1  
2  
5  
7  
9
```

▶ Instrukcja for ma składnie:

```
for number, maximum (number < maximum), step
```

```
for(A;B;C)  
D;
```



Języki programowania

Języki skryptowe – LUA -Tablice

- ▶ Możemy też deklarować tablice danych:

```
tablica = {}  
table.insert( tablica, 7 )  
table.insert( tablica, 5 )  
table.sort( tablica )  
table.foreach( tablica, print )
```

- ▶ Tablice są indeksowane od 1 .. N

```
print( tablica[0] ) -- Rezultat: nil
```

- ▶ Do operacji na tablicach potrzebujemy odpowiednich poleceń

Języki programowania

Języki skryptowe – LUA – Funkcje

- ▶ Bardziej złożone programy składają się z funkcji, które grupują kod w logiczne bloki.
- ▶ Funkcje mogą:
 - przyjmować parametry
 - zwracać wyniki
 - wywoływać siebie oraz inne funkcje nawzajem
- ▶ Przykład funkcji:

```
function nazwa_funkcji(nazwa_parametru)  
    ...  
    return zwracany_wynik  
end
```

```
wynik = nazwa_funkcji(5)
```

Języki programowania

Języki skryptowe – LUA – Zmienne lokalne

- ▶ Zmienne lokalne istnieją tylko w ramach aktualnego bloku instrukcji, np. w funkcji:

```
function hello( imie )  
    local nazwisko = " Lock"  
    print( "Hello "..imie..nazwisko )
```

```
end
```

```
hello( „John” )
```

- ▶ Rezultat:

```
Hello John Lock
```

Języki programowania

Języki skryptowe – LUA – Biblioteki

- ▶ Zbiór funkcji nazywamy biblioteką.
- ▶ Biblioteki w LUA można tworzyć zapisując zbiór funkcji w pliku z rozszerzeniem „lua”.
- ▶ Bibliotekę taką można załadować w następujący sposób:

```
loaded_chunk = assert(loadfile(„biblioteka.lua”))  
loaded_chunk() -- Inicjacja funkcji z biblioteki  
foo(2, 3)      -- Wywołanie funkcji z biblioteki
```

Języki programowania

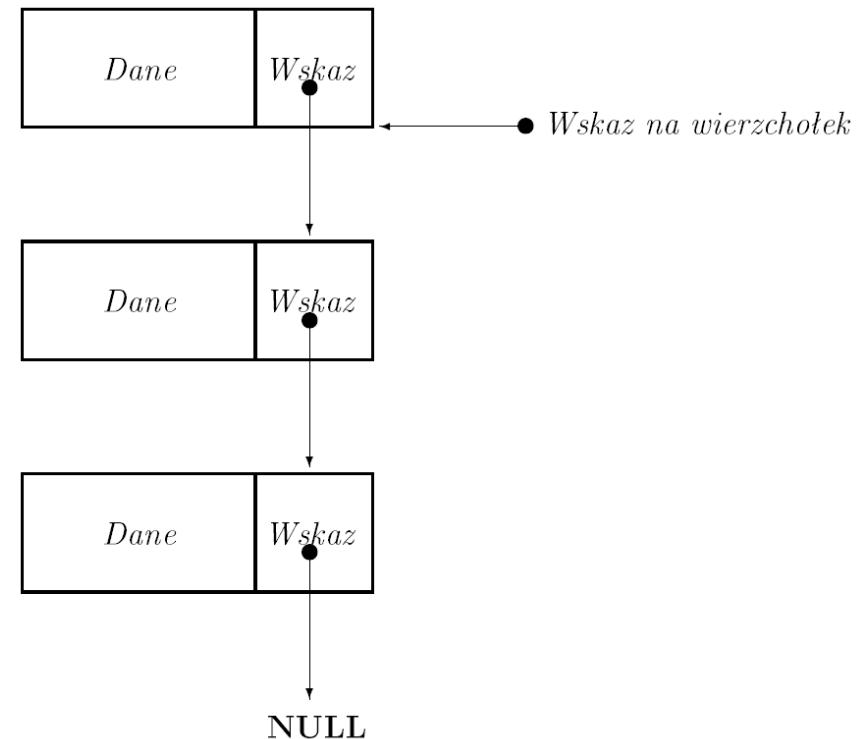
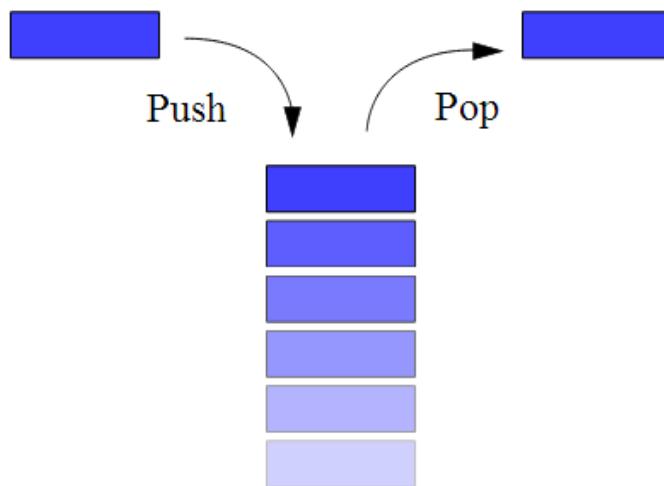
Języki skryptowe – LUA – Biblioteki

- ▶ Poza funkcjami zadeklarowanymi przez programistę, istnieją gotowe biblioteki dostarczane w ramach maszyny wirtualnej.
- ▶ Zestawy te można podzielić na dwie kategorie:
 - Standardowe funkcje maszyny wirtualnej: print, math, loadstring, string, table ...
 - Funkcje interfejsowe silnika udostępniające jego wewnętrzne mechanizmy.

Struktury danych

Stos

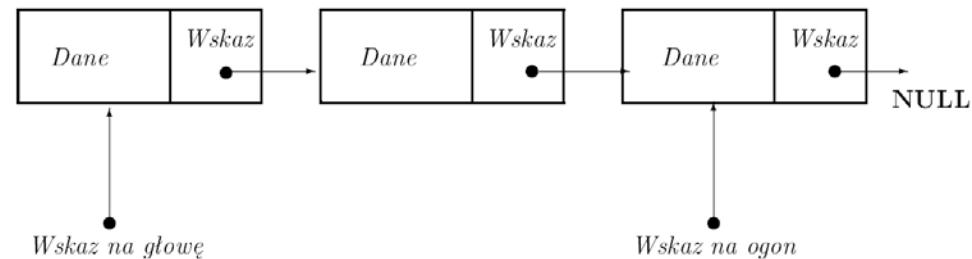
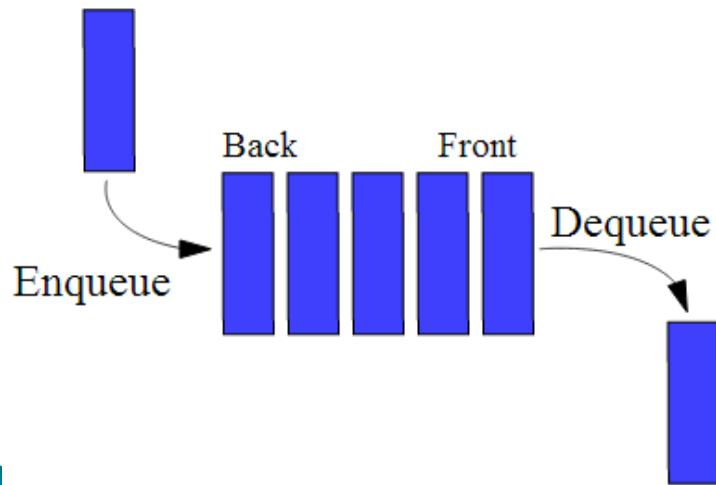
- ▶ Stos – struktura danych LIFO (*Last In First Out*). Możemy odkładać elementy na stos a następnie je z niego pobierać w odwrotnej kolejności.



Struktury danych

Kolejka

- ▶ Kolejka jednokierunkowa – działa na zasadzie FIFO (*First In First Out*). Kolejki działają jak bufor.
- ▶ Odkładamy do nich elementy asynchronicznie w stusunku do ich pobierania.



Struktury danych

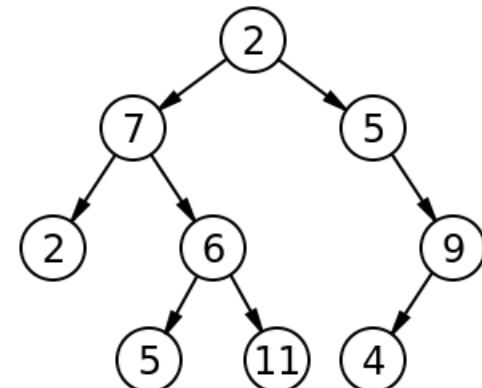
Kolejka dwukierunkowa

- ▶ Kolejki jednokierunkowe dla zaoszczędzenia pamięci, w każdym elemencie posiadają tylko wskaz na kolejny element.
- ▶ Oznacza to że nie można się „cofać”.
- ▶ Kolejki dwukierunkowe pozwalają na iteracje po jej elementach w obu kierunkach co pozwala usunąć element z ich środka.
- ▶ W kolejkach jednokierunowych jest to niemożliwe.

Struktury danych

Drzewa

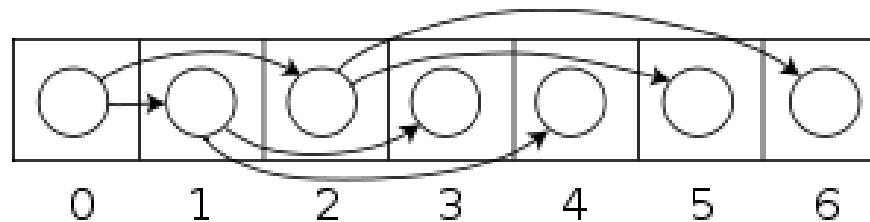
- ▶ Drzewa to struktury starające się przechowywać dane w miarze równomiernym rozkładzie ułatwiającym wyszukiwanie odpowiednich elementów.
- ▶ Drzewa binarne posiadają 2 elementy potomne.
- ▶ Po prawej stronie przykład nieuporządkowanego drzewa binarnego.



Struktury danych

Drzewa

- ▶ Drzewa podobnie jak inne dynamiczne struktury danych są przechowywane w pamięci która jest liniowa.
- ▶ Poniżej przykład przechowywania drzewa binarnego:



Agenda

- ▶ Wstęp
- ▶ Historia
- ▶ Architektura komputerów
- ▶ Języki Programowania
 - Typy danych i tablice
 - Podstawowe algorytmy
 - Struktury danych
- ▶ Systemy Operacyjne
 - Wielozadaniowość
 - Procesy i Wątki
- ▶ Przyszłość

Systemy Operacyjne

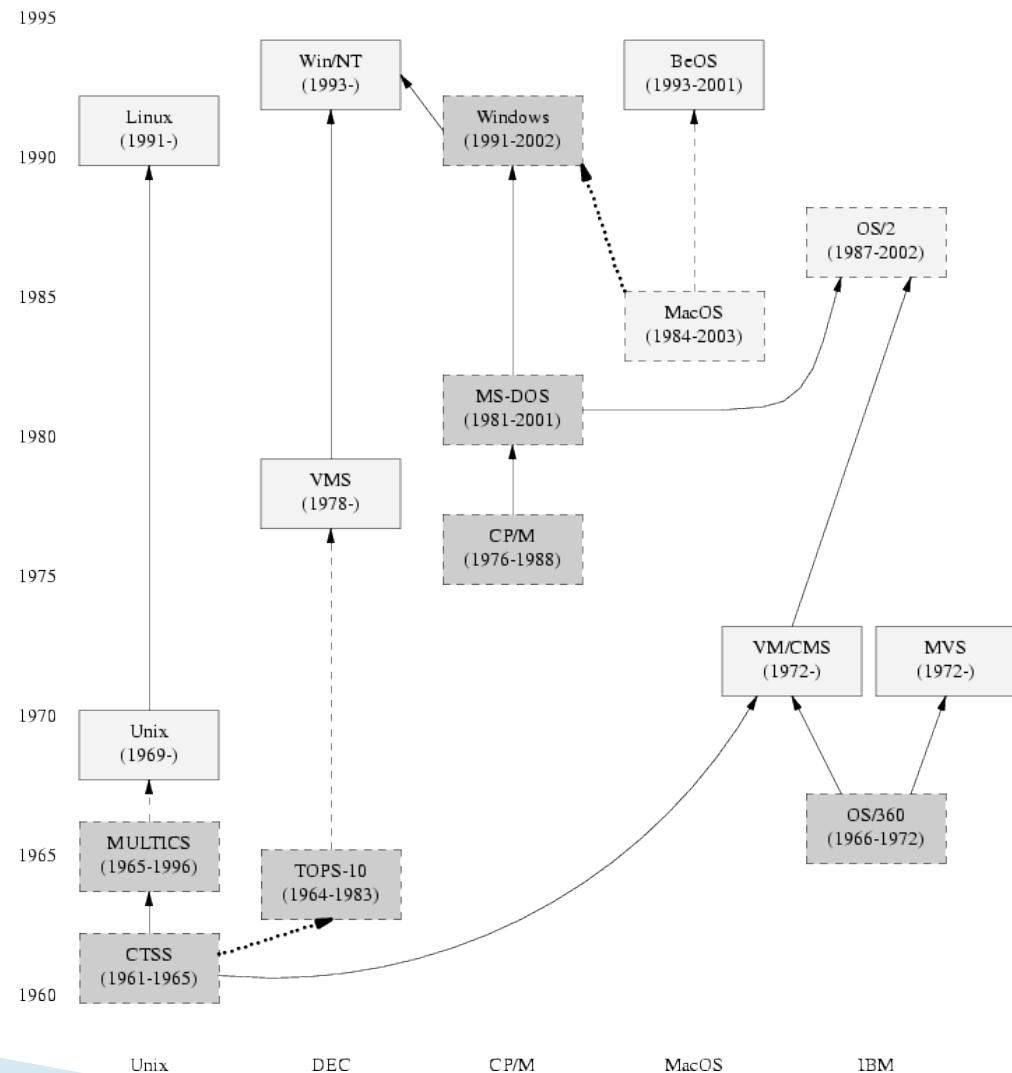
- ▶ Z początku programiści komputerów musieli wszystko napisać samemu, od podstaw: funkcje wyświetlające tekst na ekranie, drukujące go na drukarce, wykonujące matematyczne obliczenia etc.
- ▶ Komputery uruchamiano podając im taki uniwersalny program do wykonania.
- ▶ Z czasem zauważono że duża część programów odpowiedzialna za czynności pomocnicze lub standardowe jest podobna.

Systemy Operacyjne

- ▶ Zestawy tych funkcji zaczęto wydzielać do bibliotek i standaryzować tak aby programiści mogli ich używać na starcie nie tracąc czasu na odkrywaniu koła na nowo.
- ▶ Zdecydowano się również stworzyć zbiór programów pomocniczych, które można by uruchamiać naprzemiennie z tak zwanego wiersza poleceń.
- ▶ Tak powstał zarys pierwszych Systemów Operacyjnych.

Systemy Operacyjne

- ▶ Pierwszy OS powstał w 1961 roku.
- ▶ Z niego zaś wyewoluowały inne rodziny systemów operacyjnych jak UNIX, VMS, CP/M, MacOS czy inne.



Systemy Operacyjne

- ▶ Pierwsze Systemy Operacyjne pozwalały uruchamiać jeden program na raz.
- ▶ Po uruchomieniu dostawał on pełną kontrolę nad zasobami komputera (podobnie jak ma to miejsce w konsolach do gier).

```
Welcome to FreeDOS

CuteMouse v1.9.1 alpha 1 (FreeDOS)
Installed at PS/2 port
C:>>ver

FreeCom version 0.82 pl 3 XMS_Swap [Dec 10 2003 06:49:21]

C:>>dir
Volume in drive C is FREEDOS_C95
Volume Serial Number is 0E4F-19EB
Directory of C:\

FDOS              <DIR>  08-26-84  6:23p
AUTOEXEC.BAT        435   08-26-84  6:24p
BOOTSECT.BIN        512   08-26-84  6:23p
COMMAND.COM       93,963  08-26-84  6:24p
CONFIG.SYS         881   08-26-84  6:24p
FDOSBOOT.BIN        512   08-26-84  6:24p
KERNEL.SYS       45,815  04-17-84  9:19p
                           6 file(s)    142,038 bytes
                           1 dir(s)   1,064,517,632 bytes free

C:>>_
```

Systemy Operacyjne

Wielozadaniowość

- ▶ Zauważono jednak, że programy nie zawsze wykorzystują pełną moc komputera, czekają na odczyt danych z dysku, lub wykonują coś tylko co określony odstęp czasu.
- ▶ Aby nie marnować drogocennego czasu ówczesnych komputerów zaprojektowano nowe systemy operacyjne – wieloprocesowe.
- ▶ Program gdy zostaje uruchomiony nazywany *procesem* .

Systemy Operacyjne

Wielozadaniowość

- ▶ W standardowej architekturze x86, pamięć jest adresowana 32 bitowymi adresami.
- ▶ Oznacza to że maksymalny rozmiar pamięci wynosi w niej 4GB.
- ▶ W systemie Windows, część pamięci zajmowana jest przez system operacyjny.
- ▶ Domyslnie pierwsze 2GB pamięci należą do aplikacji, a ostatnie 2GB do systemu operacyjnego (w Windows XP).
- ▶ Pamięć tę można opcjonalnie rozszerzyć do 3GB, ale wymaga to ręcznych zmian ustawień.

Systemy Operacyjne

Wielozadaniowość

- ▶ Jeżeli proces mojej gry ma 2GB pamięci dla siebie a Windows pozostałe 2GB, gdzie znajdują się inne procesy?
- ▶ Systemy wieloprocesowe zarządzają wszystkimi uruchomionymi naraz procesami.
- ▶ Każdemu z nich przydzielają kwant czasu procesora podczas którego proces działa.
- ▶ W tym czasie wszystkie inne procesy są zawieszone.
- ▶ Procesor jest przydzielany średnio 100 razy na sekundę, co sprawia wrażenie że procesy działają równocześnie.

Systemy Operacyjne

Procesy i Wątki

- ▶ Każdy proces działa w tak zwanym *sandboxie*. Podczas swojej pracy widzi on tylko siebie i system operacyjny w pamięci.
- ▶ Obszar pamięci który dla procesu wydaje się ciągły, w rzeczywistości jest rozrzucony po całej pamięci operacyjnej i przemieszany z innymi obszarami używanymi przez inne procesy.
- ▶ Każde odwołanie do pamięci przechodzi w procesorze przez tablice translacji które tłumaczą go na rzeczywisty adres w pamięci komputera.

Systemy Operacyjne

Procesy i Wątki

- ▶ Mechanizm taki nazywamy stronicowaniem ponieważ pamięć podzielona jest na strony (zazwyczaj o rozmiarze 4KB).
- ▶ Gdy następuje przełączenie procesu na aktywny, system operacyjny podmienia tablice translacji używane przez procesor.
- ▶ Teraz nowy aktywny proces widzi swoją pamięć jako ciągłą i jedyną.
- ▶ Nadmiarowe strony odkładane są na dysk.

Mój artykuł o menedżerach pamięci i stronicowaniu:

http://www.binboy.org/os/articles/126/OSDB_Memory_Manager_Cz_1.html

(niestety zmieniono layout strony i posypało się formatowanie, a ilustracje zniknęły :/)

Systemy Operacyjne

Procesy i Wątki

- ▶ Ponieważ przełączenie procesu jest dość wolne (trzeba przełączyć dużo więcej stanów), powstała idea *wątków*.
- ▶ Program podczas pracy nazywany *procesem* istnieje w swoim *sandboxie*. Można nazwać go głównym *wątkiem 0*. Program może jednak uruchomić więcej wątków, które również będą działać w jego sandboxie i będą miały swobony dostęp do całej pamięci procesu i innych zasobów.
- ▶ W ten sposób OS może przełączać się między wątkami prawie natychmiastowo.

Systemy Operacyjne

Procesy i Wątki

- ▶ System Operacyjny który obsługuje wątki nazywamy wielowątkowym i wieloprocesowym.
- ▶ Aplikację która działa używając wielu wątków również nazywamy wielowątkową.
- ▶ Wielowątkowość jest bardzo skomplikowanym zagadnieniem.
- ▶ Programy działające sekwencyjnie nagle mogą działać asynchronicznie i różnej kolejności przetwarzając te same dane.
Sytuacja taka bardzo komplikuje poprawną implementację algorytmów.

Systemy Operacyjne

Mobilne Systemy

- ▶ Mobilne systemy operacyjne również przeszły długą drogę ewolucji.



symbian
OS

Palm
MeeGo

 Windows
Phone


ANDROID

iOS 6

 **BlackBerry**®

Systemy Operacyjne

Mobilne Systemy

- ▶ Równocześnie do klasycznych systemów operacyjnych w ostatnim dziesięcioleciu zaczęły się rozwijać mobilne systemy operacyjne.
- ▶ Głównie wywodzą się one od odmian i klonów Unix'a takich jak Linux czy QNX.
- ▶ Większość systemów takich jak Android czy iOS nie udostępnia pełnego środowiska wielowątkowego w klasycznym tego słowa rozumieniu.

Agenda

- ▶ Wstęp
- ▶ Historia
- ▶ Architektura komputerów
- ▶ Języki Programowania
 - Typy danych i tablice
 - Podstawowe algorytmy
 - Struktury danych
- ▶ Systemy Operacyjne
 - Wielozadaniowość
 - Procesy i Wątki
- ▶ Przyszłość

Przyszłość branży

Świat gier AAA

„Developing a AAA game is one of the most expensive enterprises humans can undertake, outside of building battleships, launching space vehicles or making movies”

To stwierdzenie nie jest już nawet prawdziwe, robienie gier stało się jeszcze trudniejsze...

Wystąpienie Johna Carmacka w NASA

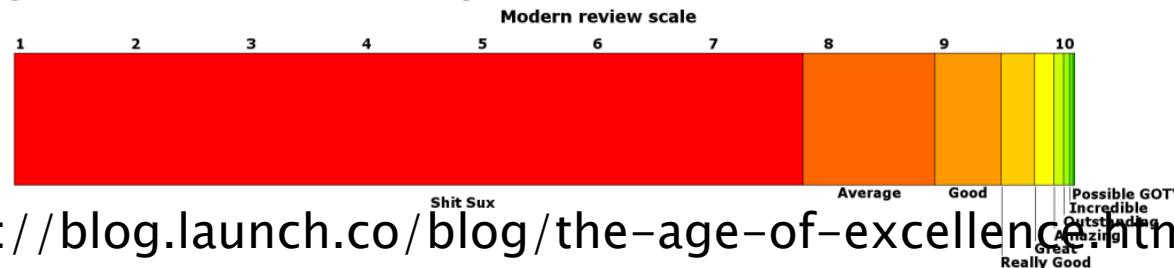
<http://www.youtube.com/watch?v=VcWRc1wK3gM>

Przyszłość branży

Świat gier AAA

Games market is very demanding now:

- Players want only top quality products
- Metacritic rating below 75% is treated as mixed
- Average games are not taken into notice and fail
- There are very high expectations from the beginning and through whole time of game production cycle



<http://blog.launch.co/blog/the-age-of-excellence.html>

All of this created genre of big AAA games which:

- Are very expensive in production (budget from \$40 to \$100 millions)
- Are repetitive and not innovative to minimize risk of failure
- For „smaller” studios they mean be or not to be in the business

Przyszłość branży

Świat gier AAA

- ▶ You can hear from different people in game industry:
- ▶ „*The state of AAA is chaos, bordering on impending doom*”
„*AAA development is a high-risk, narrow margins game with only a very few winners, and too many players*”
„*It's a mosh pit*”
„*It's chaos.*”
„*It's falling apart*”
„*No one is profitable*”
- ▶ And there is a lot of truth in it.

Przyszłość branży Świat gier AAA



Budżet: \$40 milionów

Dochód: \$56 milionów

Sprzedanych kopii: 3.5 milionów!

Trzeba było sprzedać 3 miliony kopii aby uzyskać jakikolwiek zysk!

Przyszłość branży

Świat gier AAA

Budżet:

~\$105 million!

Sprzedanych:

~3 million!



Przyszłość branży

Rozłam między AAA a Mobile

- ▶ **Big AAA games** - The instability, state of the art cutting edge graphics and technique, huge budgets, small or none revenue, development teams starting from 100 people, 4-8 years in production, small market, high entry cost for boxed release...
 - ▶ ...Nothing in the middle can survive...
- ▶ ...small teams of 2-15 people, 6-12 month production cycles, barely none entry cost, digital distribution, cheap development, gameplay more important than graphics, well known graphic standards, quickly evolving HW capabilities – **Mobile games**.



Infinity Blade

Przyszłość branży

Mobilne platformy nowym PC

- ▶ Because development of AAA titles for PC / XBOX / PS is so demanding, game industry is strongly shifting towards mobile development which was clearly visible this year:
 - Mobile (Smartphones and Tablets), Casual, Social and Online Games lectures dominated on most of game industry conferences.
 - Unreal Engine 4 is targeting consoles and mobile devices, while Unreal Engine 3 was targeting PS and consoles.
 - OpenGL ES 3.0 was followed by OpenGL 4.3 during their reveal on SIGGRAPH.
 - First AAA mobile games released, for e.g. Infinity Blade series.

Przyszłość branży

Mobilne platformy nowym PC

Mobile games industry is also evolving much faster than Console one, because of fast production cycle of new devices in comparison to current generation of consoles which is more than **8 years old.**

As a result we are observing customer shift from Laptops to Tablets and Handhelds similar to what we have seen some time ago with shift from desktop PC's to Laptops.

Przyszłość branży

Mobilne platformy nowym PC

- ▶ Mobile HW is evolving extremely fast:
- High end mobile configuration RAM grown from 512 MB to 2GB in about one year.
- Internal storage grown from 16GB up to 64GB.
- Up to 4GB of game content in Android Market.
- Introduction of OpenGL ES 3.0!
- Higher resolutions up to 2048x1536!



Przyszłość branży

Mobilne platformy nowym PC

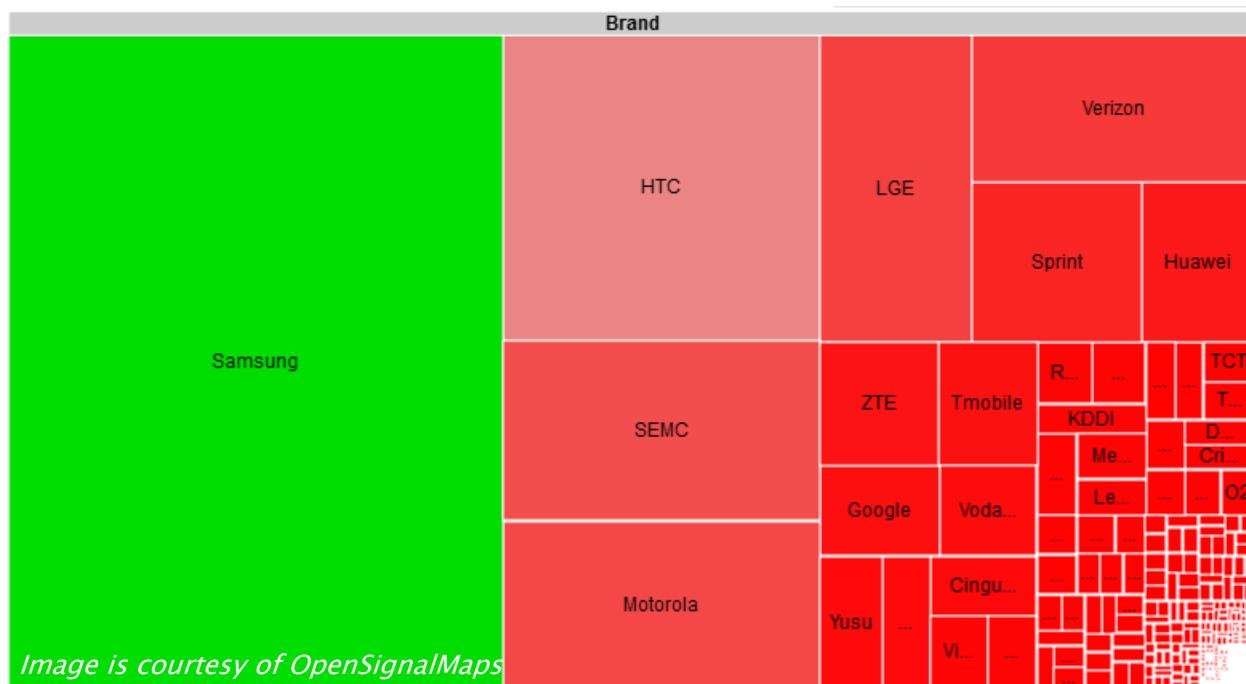
- ▶ This can be rendered in 2048 x 1536 resolution:



Przyszłość branży

Mobilne platformy nowym PC – Fragmentacja

Mobile market is evolving so fast, that it is already becoming similar to PC market in variety of different configurations, brands and HW capabilities:

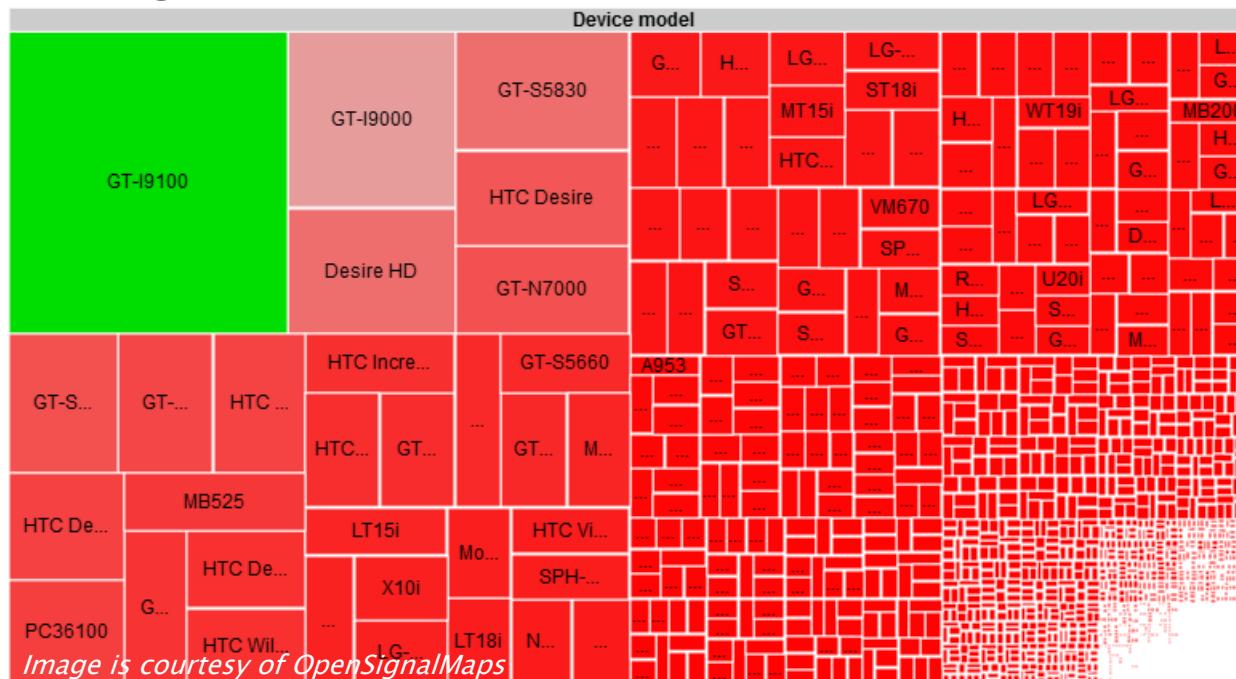


Android market share of different brands

Przyszłość branży

Mobilne platformy nowym PC – Fragmentacja

All this leads to huge fragmentation known from PC market. Below is shown popularity of different device models using Android:



Przyszłość branży

Mobilne platformy nowym PC – Fragmentacja

Mobile market is fragmented in huge amount of ways:

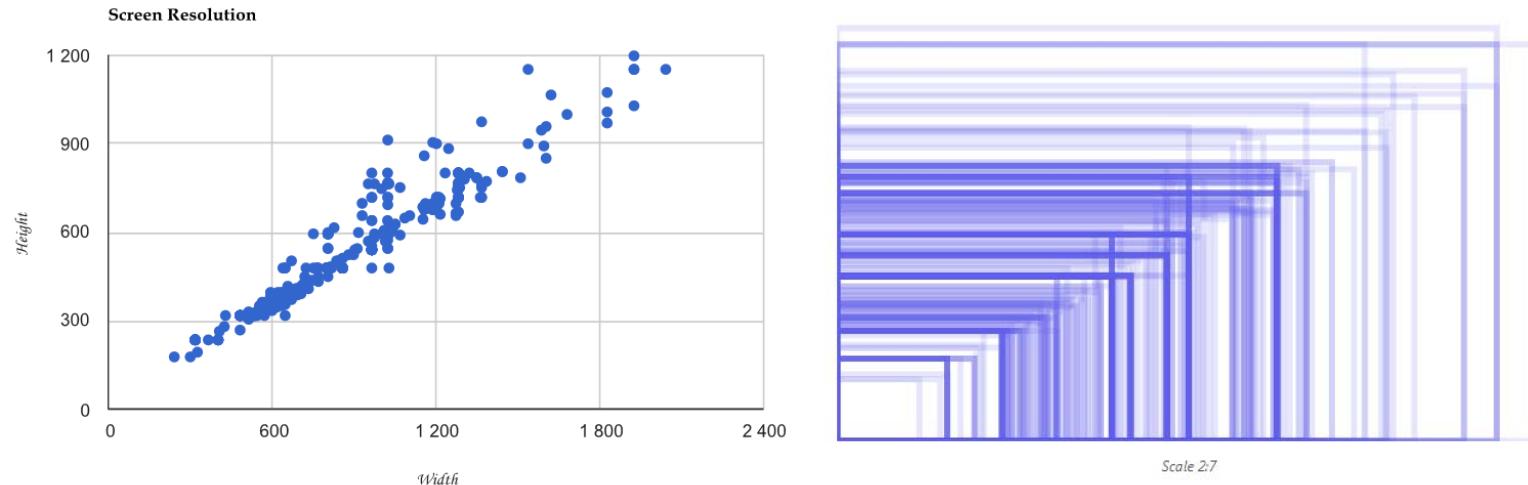


Image is courtesy of OpenSignalMaps

Screen resolutions for different mobile devices.
Most common aspect ratio is 5:3.

Przyszłość branży

Mobilne platformy nowym PC

Growth of mobile market also means changes in development and monetization models:

Old model:

- Develop once, fix through patches or DLC.
- Biggest income in launch period (so called launch window).

New models:

- Develop and give up for free (F2P) on launch, then earn on ingame micro-transactions
- Develop and sell, then fix, change and upgrade continuously (Social). Re-sell again with each new addon, upgrade and promotion.

Objective: Clear the area before entering Gunshop

II



LIFE 
\$ 376

10 m

COLT M4



Dead Trigger

Przyszłość branży

Mobilne platformy nowym PC

- ▶ We will see huge changes in Games Industry in upcoming years. Market is changing fast, and we need to be able to follow it!
- ▶ Shift to Mobile devices will be much faster and more intense than previous shift from PC to Laptops.
- ▶ PC graphics will always push boundaries, but Mobile graphics will gain more visibility!
- ▶ Mobile market explosion means fragmentation and a lot of confusion for developers!