Języki programowania obiektowego

Nieobiektowe elementy języka C++

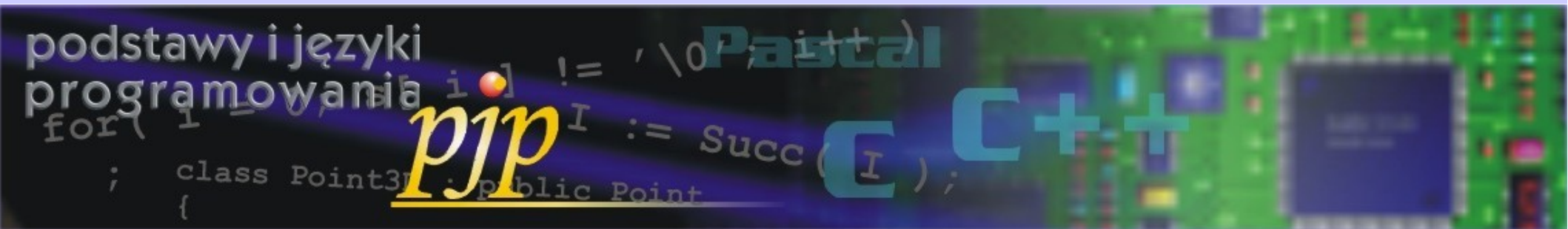
Roman Simiński

roman.siminski@us.edu.pl

www.programowanie.siminskionline.pl

Sprzęt jako platforma wykonawcza dla programów

C++ — wprowadzenie, właściwości, historia rozwoju, literatura, narzędzia dla programisty

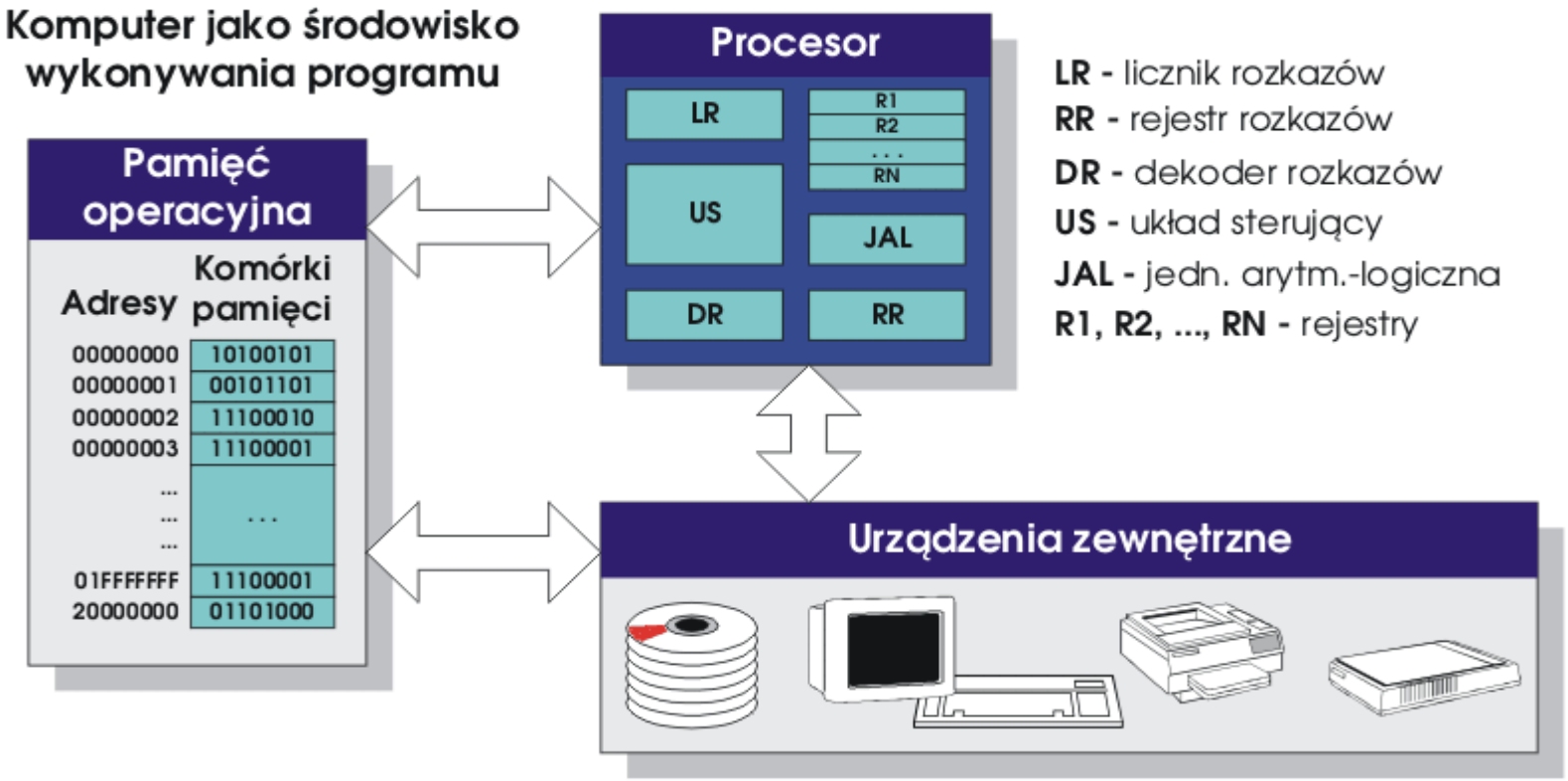


Od procesora do języka programowania

Czyli — co każdy programista wiedzieć powinien…

Sprzęt jako platforma wykonawcza dla programów

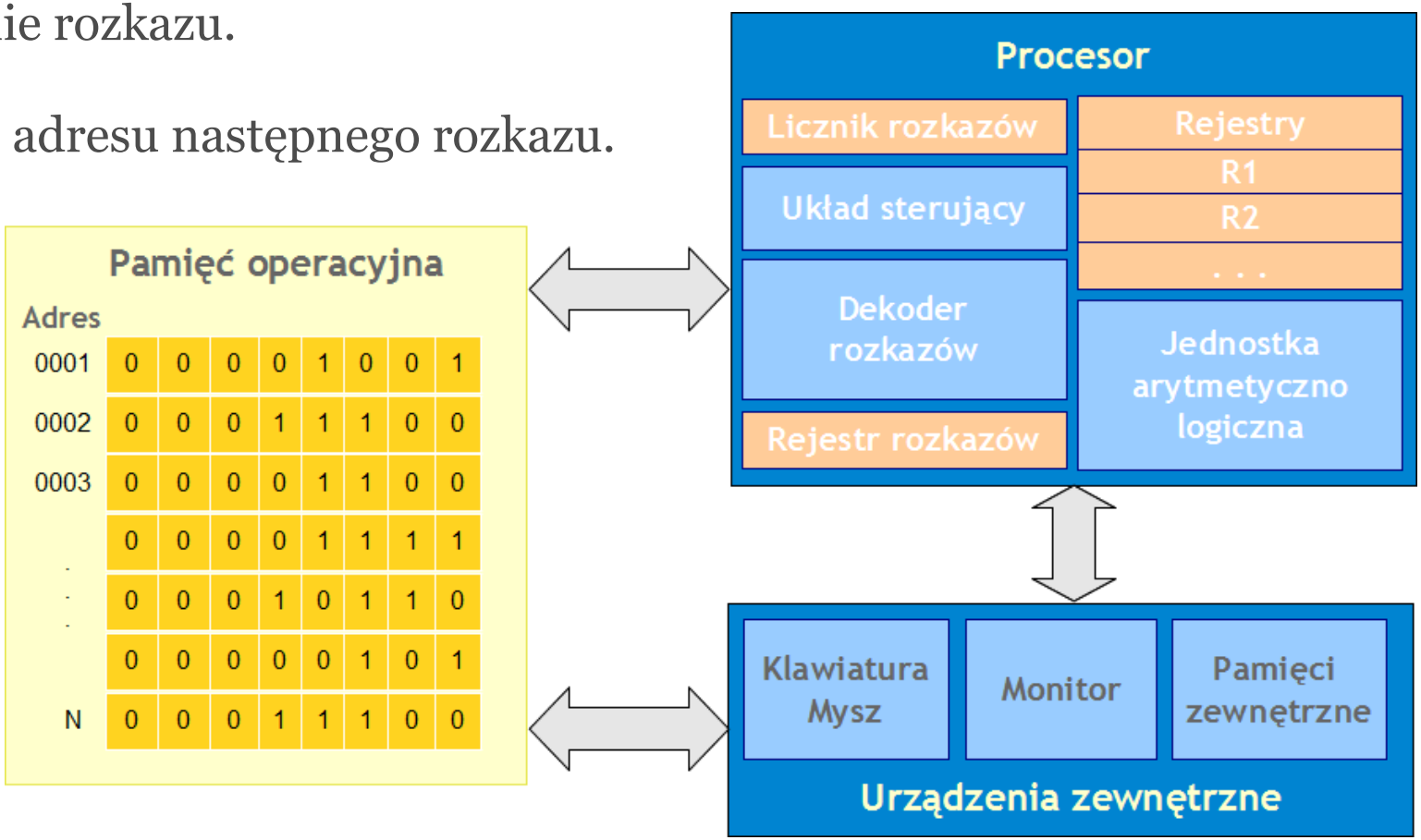
- Kod maszynowy to jedyna postać programu komputerowego, która jest zrozumiała dla procesora i może być przez niego wykonana.
- Procesor *pobiera*, *dekoduje* i *wykonuje* rozkazy *maszynowe*.
- Rozkazy pobierane są z ustalonych komórek *pamięci operacyjnej*, gdzie są zapisane w postaci binarnej.



Cykl rozkazowy procesora

Cykl rozkazowy procesora:

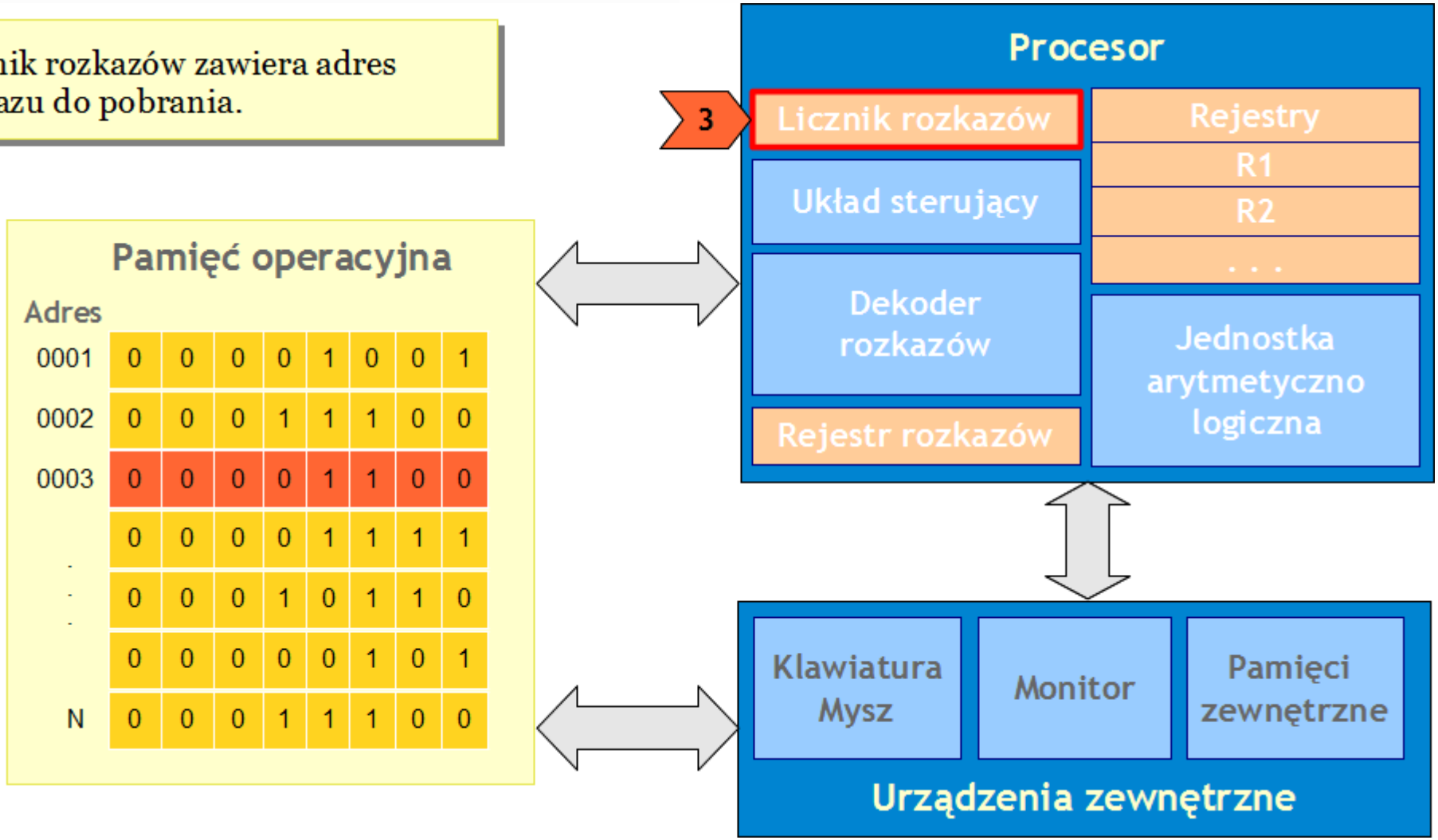
- Pobranie rozkazu zapisanego pod określonym adresem.
- Zdekodowanie rozkazu.
- Wykonanie rozkazu.
- Ustalenie adresu następnego rozkazu.



Cykl rozkazowy procesora

- Ustalenie adresu rozkazu — licznik rozkazów

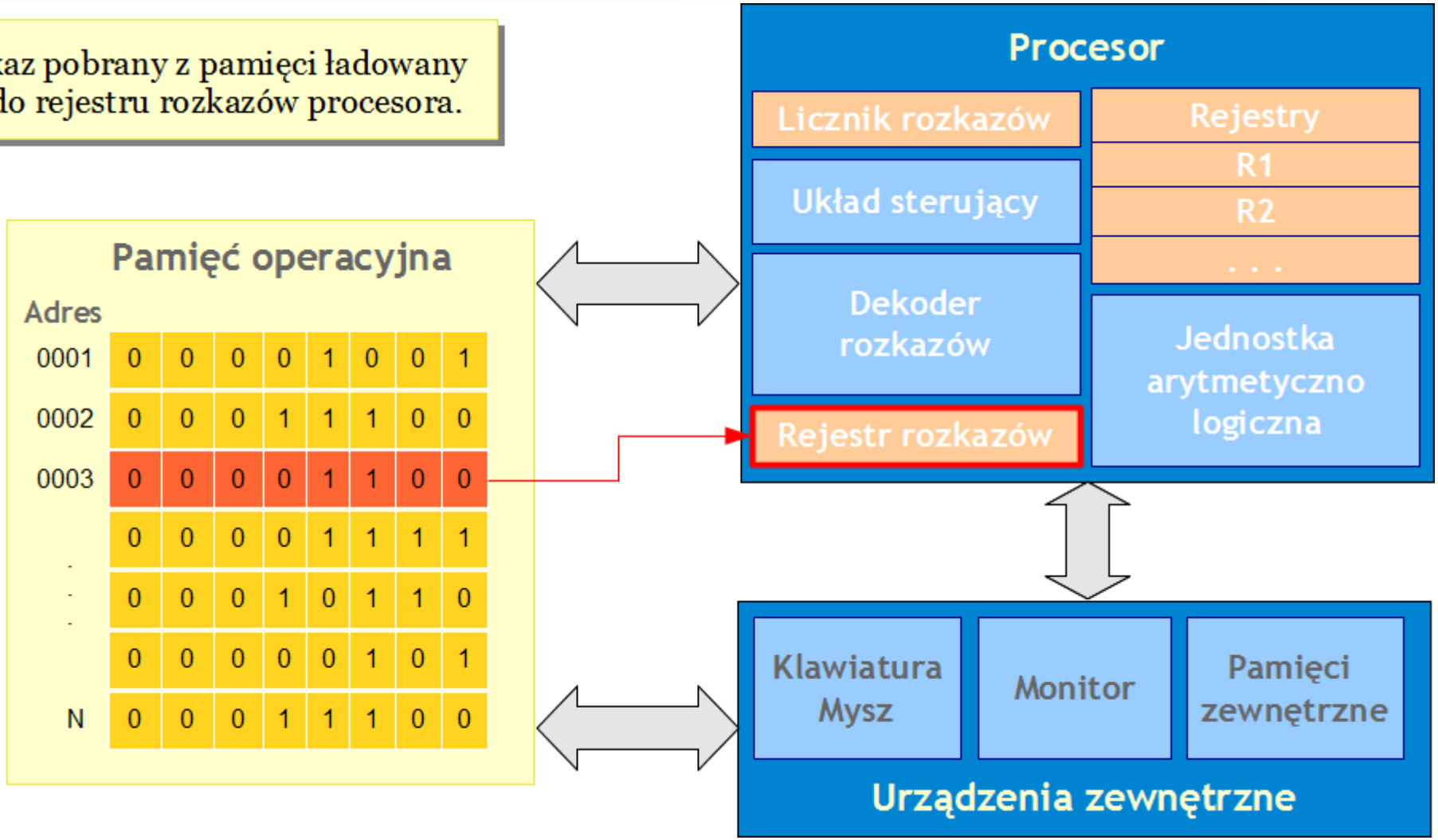
Licznik rozkazów zawiera adres rozkazu do pobrania.



Cykl rozkazowy procesora

- Pobranie rozkazu z pamięci operacyjnej — rejestr rozkazów

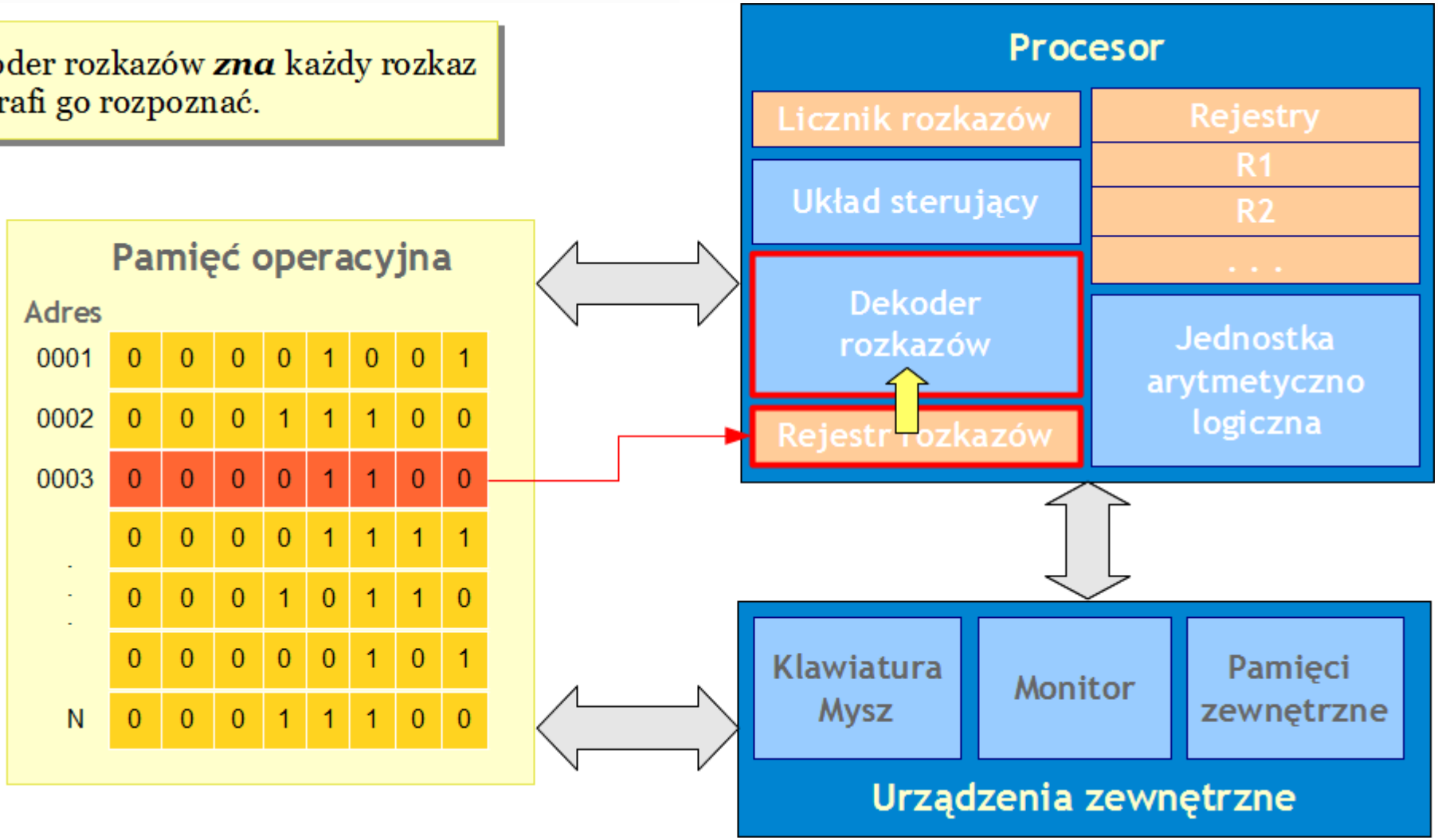
Rozkaz pobrany z pamięci ładowany jest do rejestru rozkazów procesora.



Cykl rozkazowy procesora

- Dekodowanie rozkazu pobranego z pamięci operacyjnej

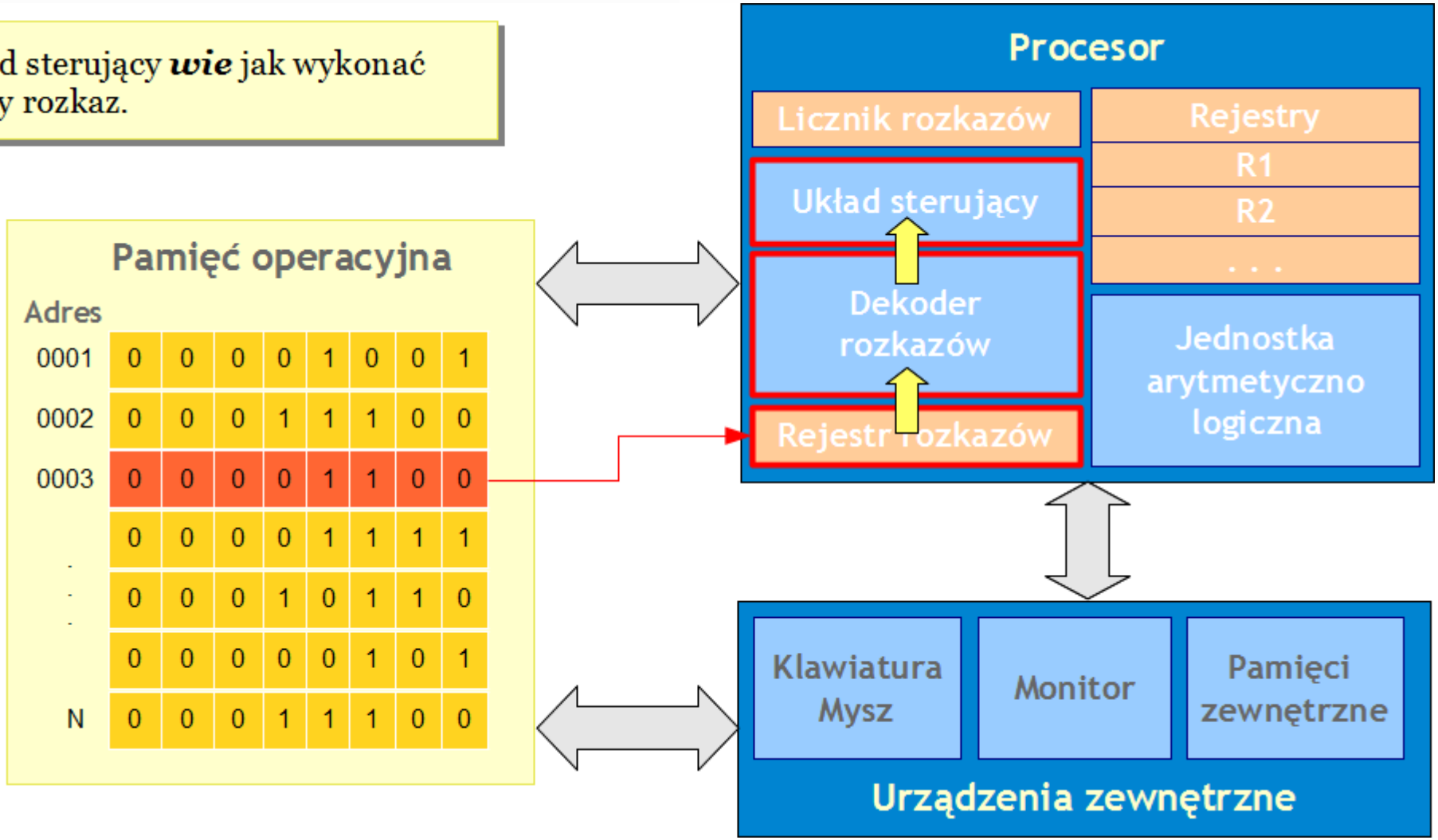
Dekoder rozkazów ***zna*** każdy rozkaz i potrafi go rozpoznać.



Cykl rozkazowy procesora

- Wykonanie rozkazu wg wskazówek układu sterującego procesora

Układ sterujący ***wie*** jak wykonać każdy rozkaz.

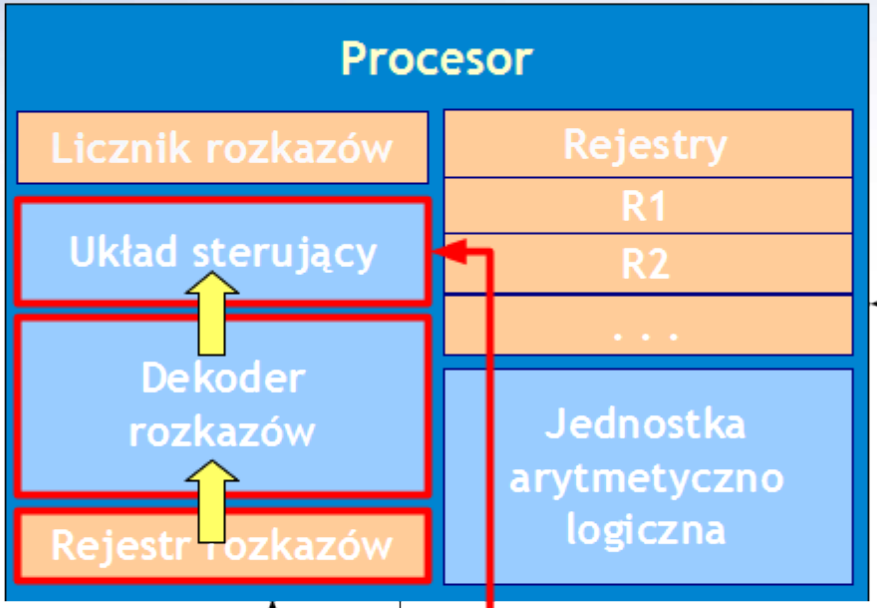


Rozkaz maszynowy jako ciąg mikrooperacji

- Każdy rozkaz składa się z szeregu drobnych akcji zwanych *mikrooperacjami*.
- Realizacja każdego rozkazu polega na wykonaniu *szeregu mikrooperacji w określonej kolejności*.
- Mikrooperacje są elementarnymi operacjami wykonywanymi przez procesor.
- Wykonanie rozkazu procesor dzieli na *cykle*, zwykle każdy cykl składa się z kilku *taktów*.
- Zwykle w każdym takcie wykonywana jest *jedna mikrooperacja* lub kilka mikrooperacji *niezależnych od siebie*.

Wykonaniem mikrooperacji kierują sygnały sterujące

Sygnały sterujące są zwykle doprowadzane bezpośrednio do odpowiednich układów scalonych



Pamięć operacyjna

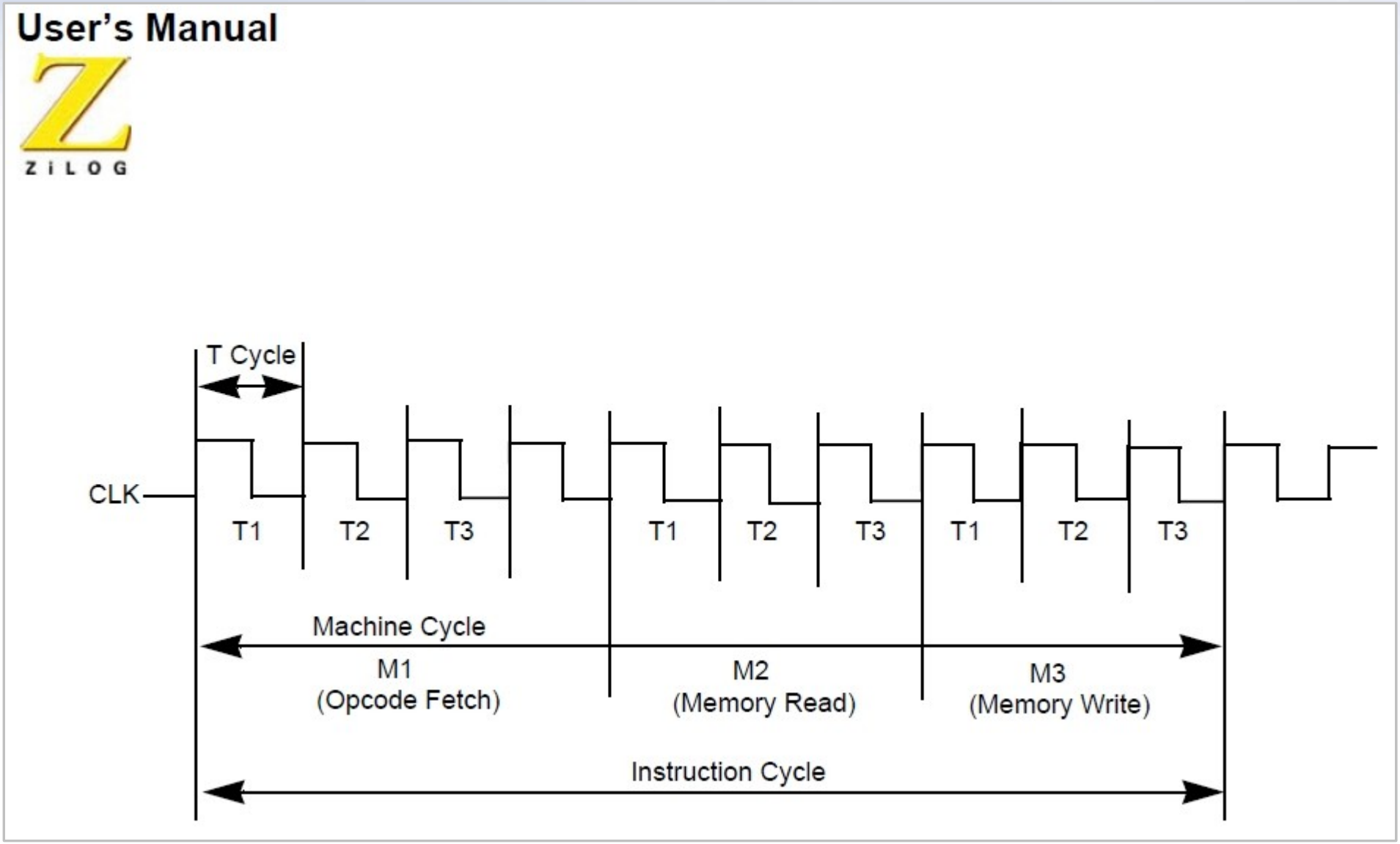
Urządzenia zewnętrzne

Magistrala danych

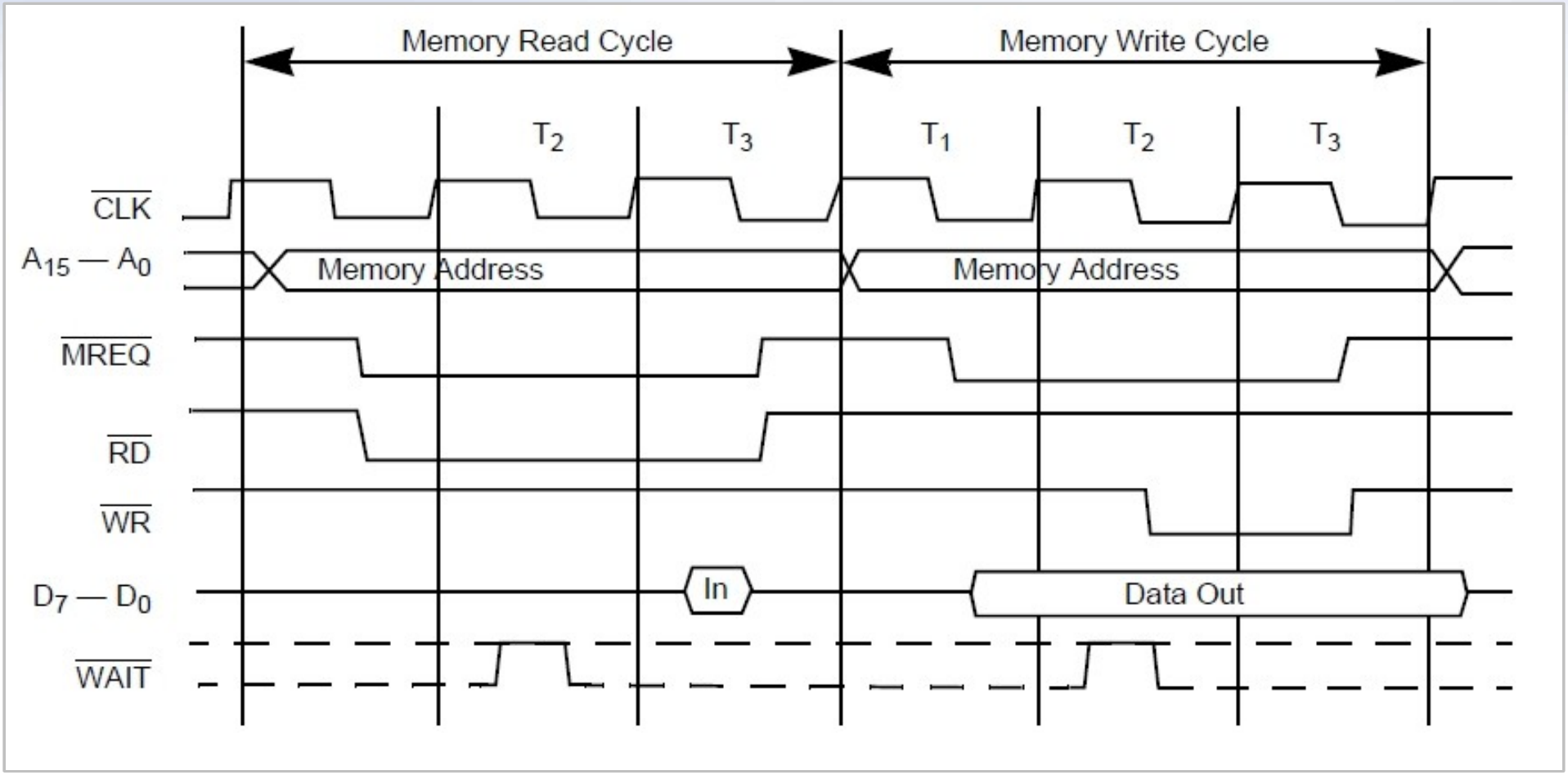
Magistrala adresowa

Magistrala sterująca

Wykresy czasowe sygnałów mikrosterujących — Z80



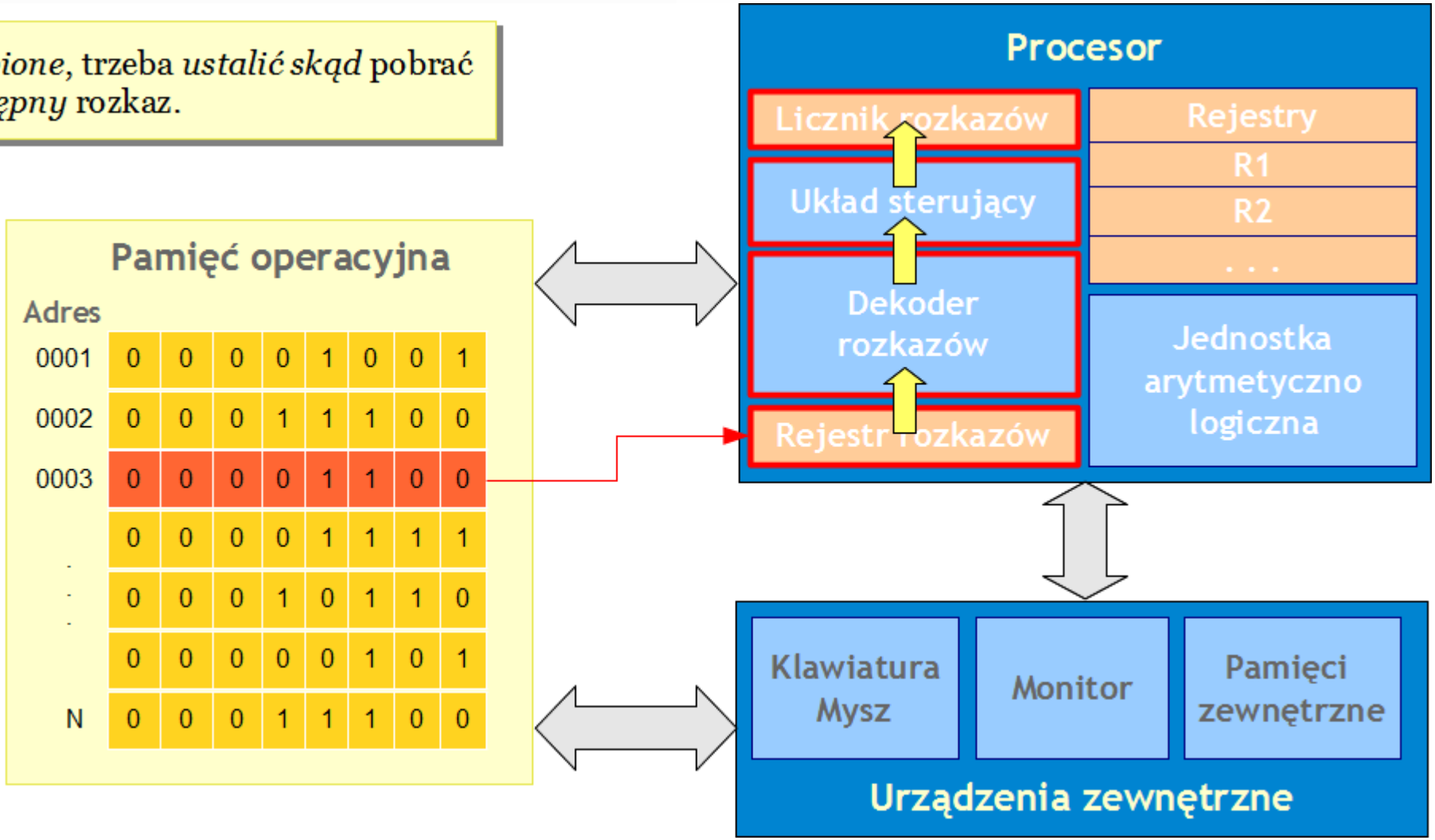
Wykresy czasowe Z80, kontakt z PaO



Wykresy czasowe Z80, kontakt z PaO

- Ustalenie adresu następnego rozkazu do wykonania

Zrobione, trzeba *ustalić skąd* pobrać *następny* rozkaz.

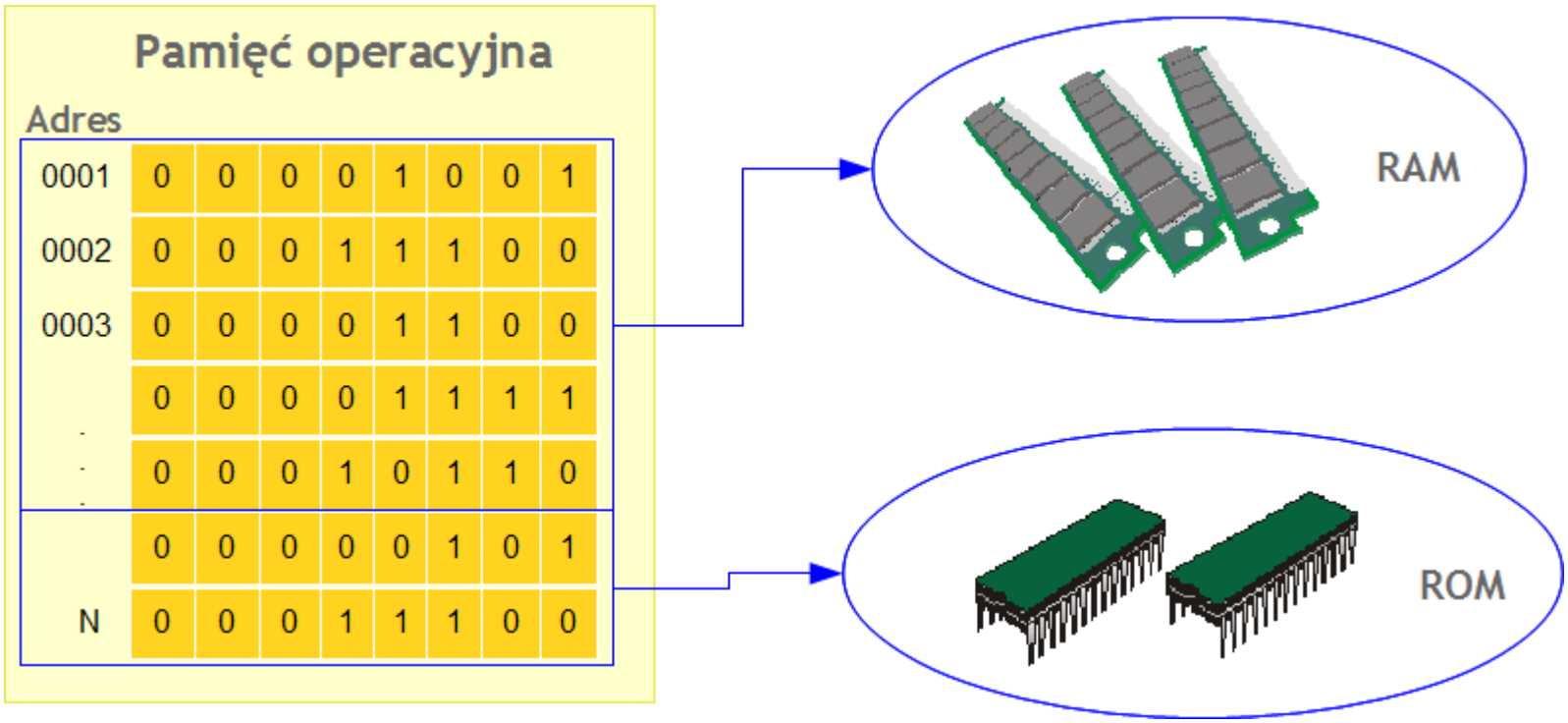


Co się dzieje po włączeniu komputera?

- Tuż po włączeniu komputera pamięć operacyjna zrealizowana w technologi RAM jest pusta.
- Procesor potrzebuje rozkazów maszynowych do swojego działania, może je odczytywać tylko z pamięci operacyjnej, a ta jest pusta… .
- Co teraz?

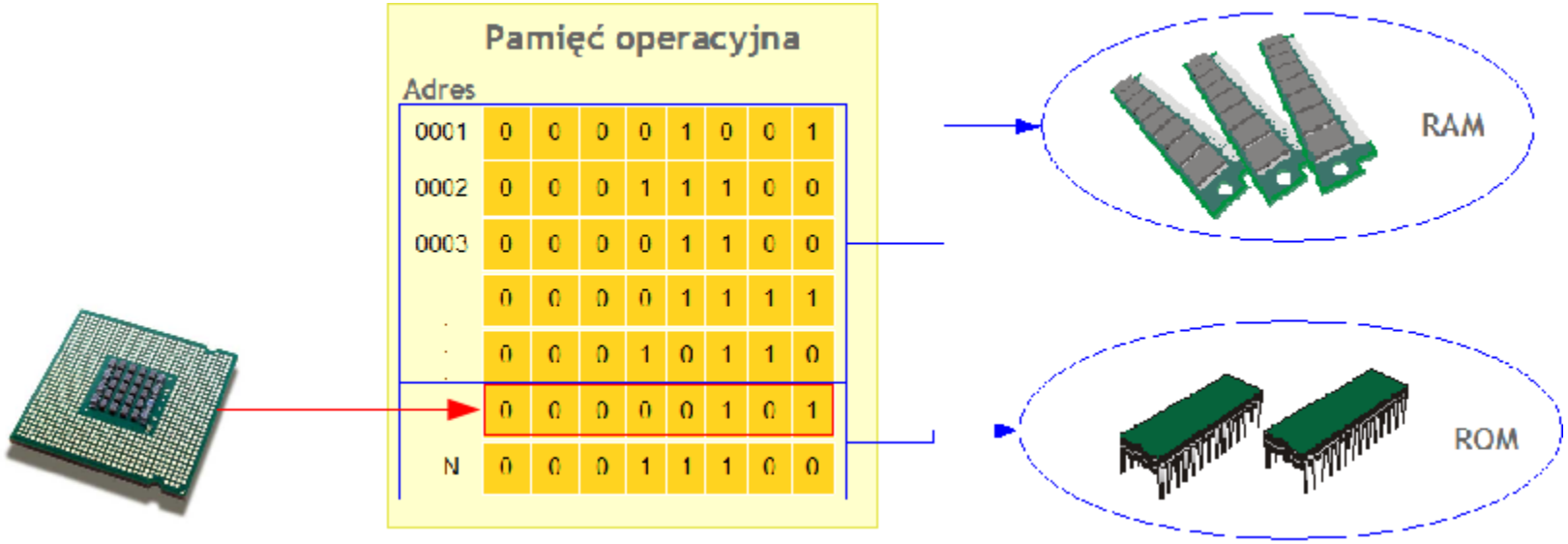
Co się dzieje po włączeniu komputera?

- Oprócz pamięci RAM są również różne rodzaje pamięci ROM.
- Obszar adresowy pamięci operacyjnej może być *nałożony* na oba te rodzaje pamięci.



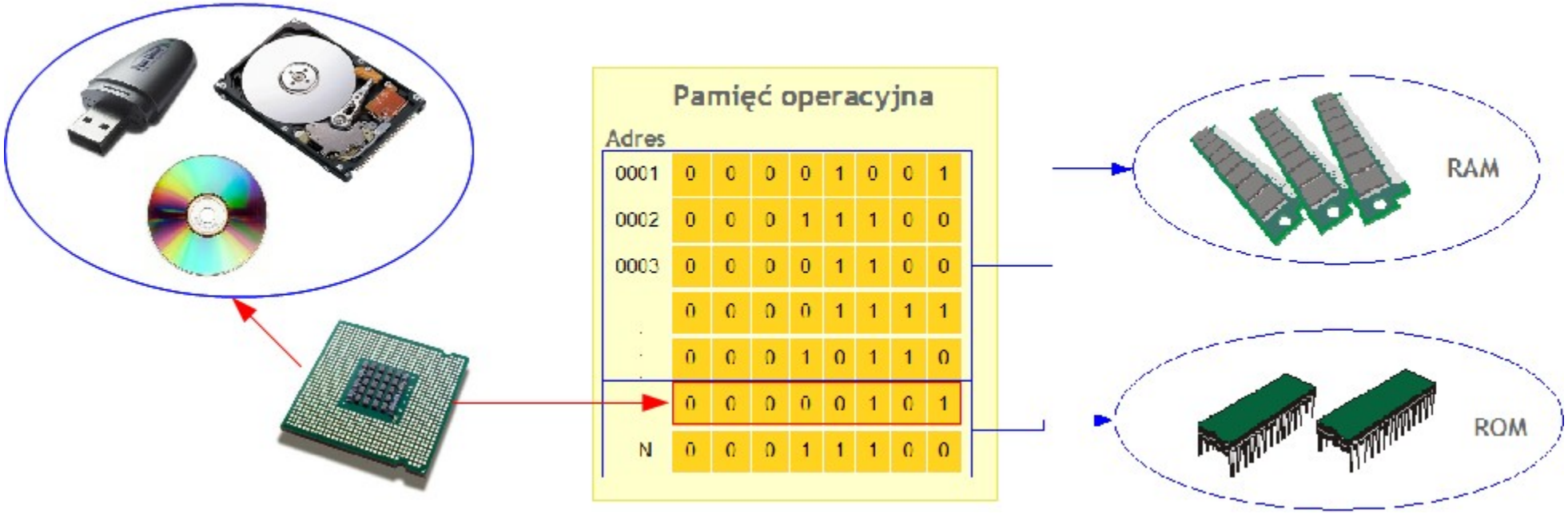
Co się dzieje po włączeniu komputera?

- Procesor po włączeniu zasilania rozpoczyna wykonywanie programu począwszy od pewnej, ściśle określonej komórki pamięci (ściśle określony adres).
- Komórka ta powinna być pobrana z pamięci stałej, o zawartości ustalonej np. przez producenta komputera.



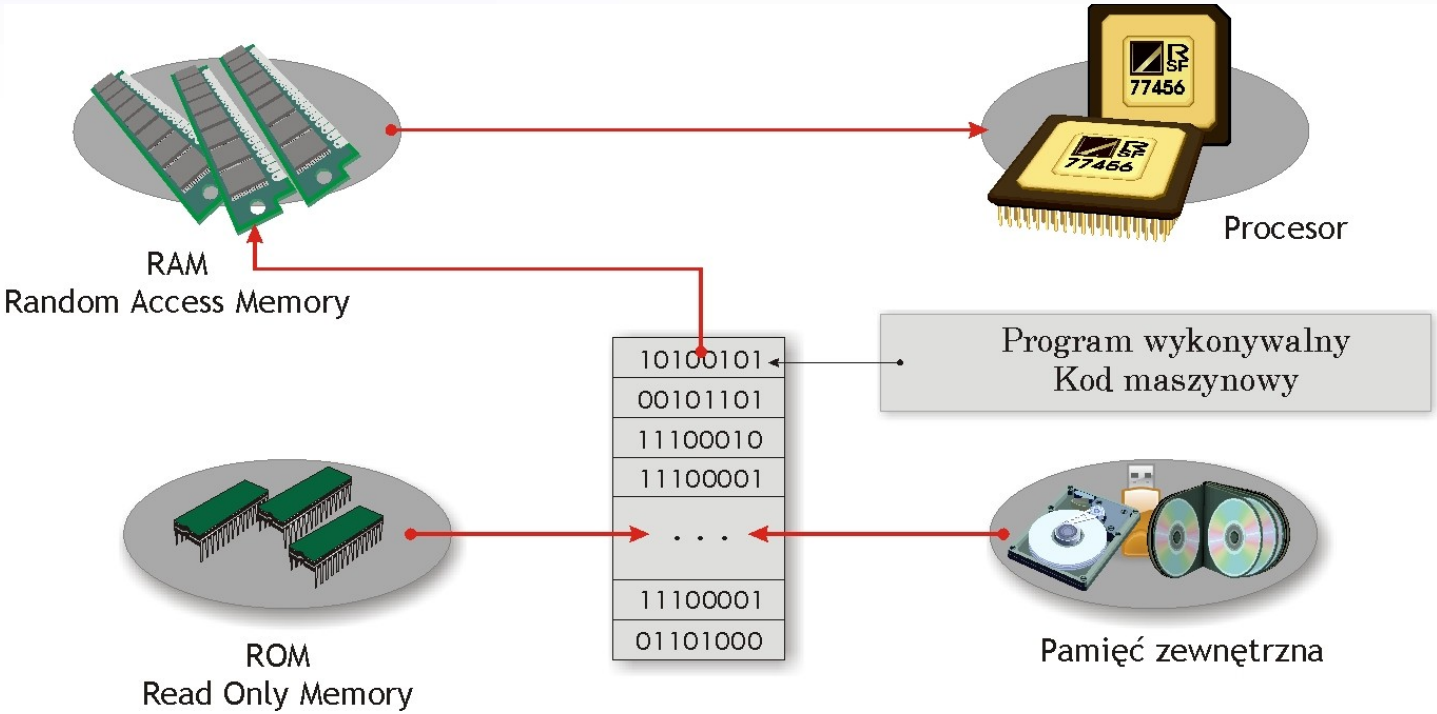
Co się dzieje po włączeniu komputera?

- Po zakończeniu programu z pamięci stałej, procesor rozpoczyna przeszukiwanie pamięci zewnętrznych.
- Poszukiwany jest główny program ładujący, którego zadaniem będzie wybór i załadowanie odpowiedniego systemu operacyjnego z pamięci zewnętrznej.



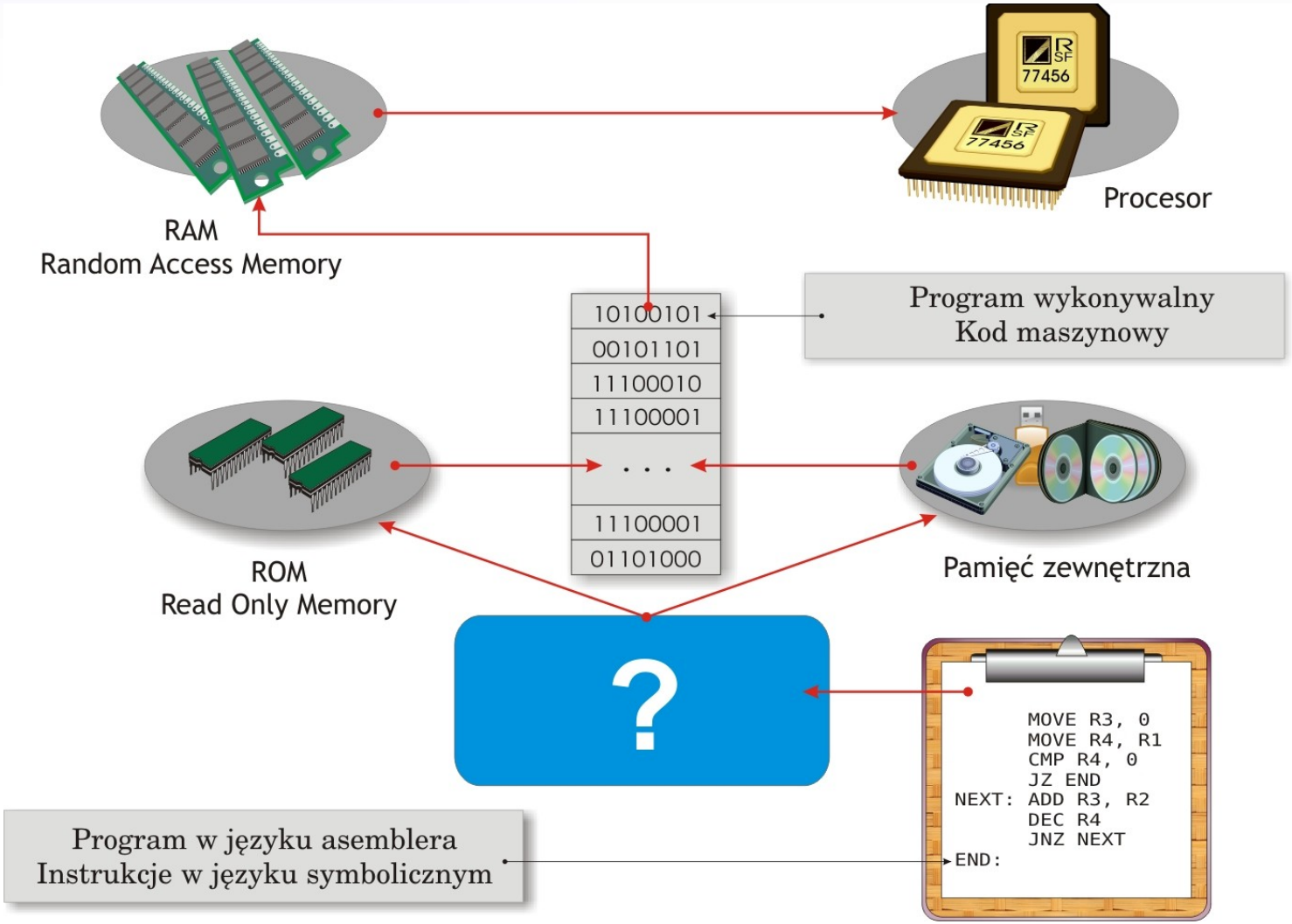
Od pliku wykonywalnego do programu w pamięci

- Procesor *pobiera*, *dekoduje* i *wykonuje* rozkazy pobrane z *pamięci operacyjnej*. Zwykle rozkazy te ładowane są z pliku wykonywalnego, zawierającego binarny obraz kodu maszynowego programu.



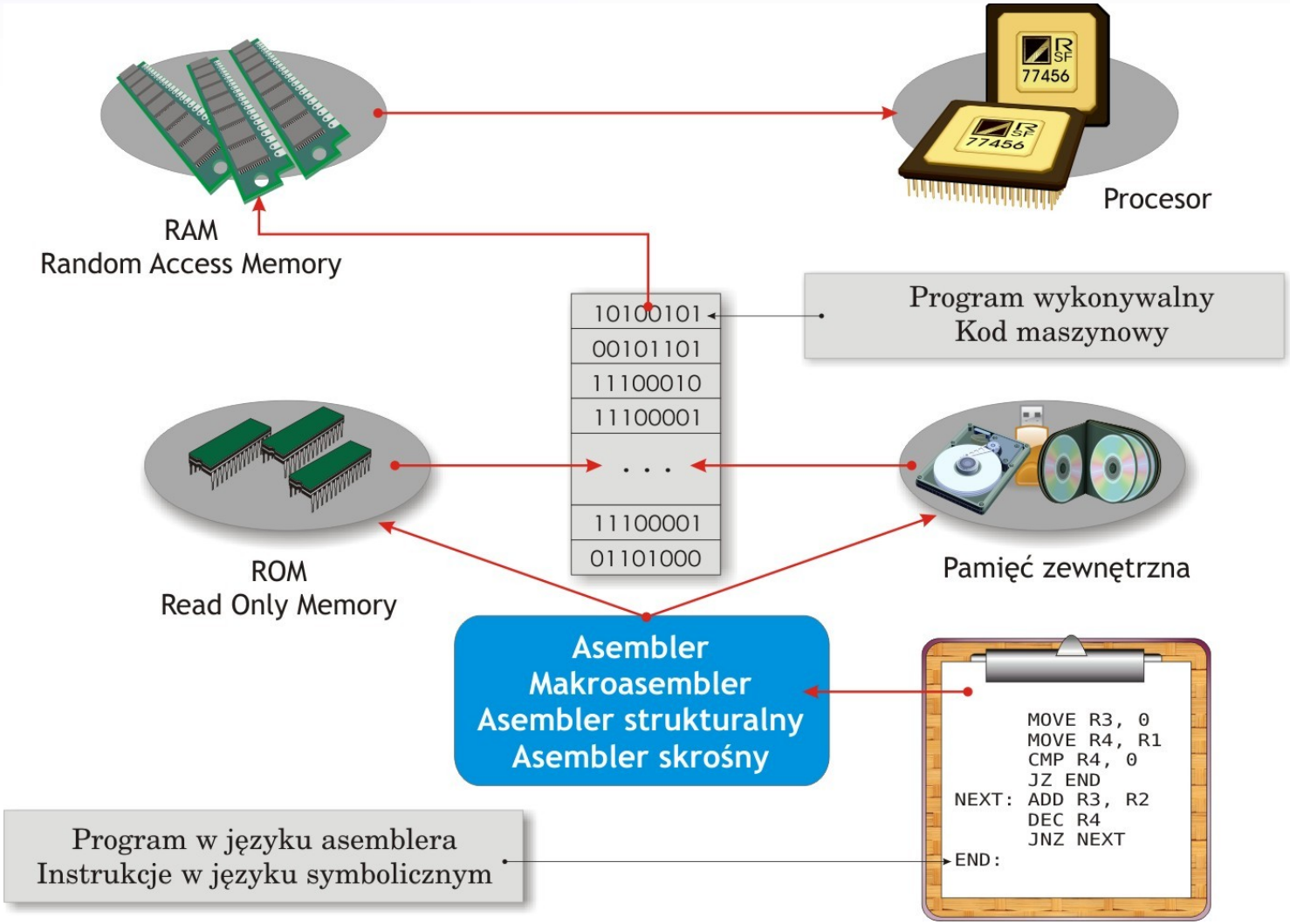
Programowanie w języku symbolicznym

- Historycznie pierwszymi językami programowania były *języki programowania symbolicznego*. Program w języku symbolicznym *nie jest zrozumiały* dla procesora.



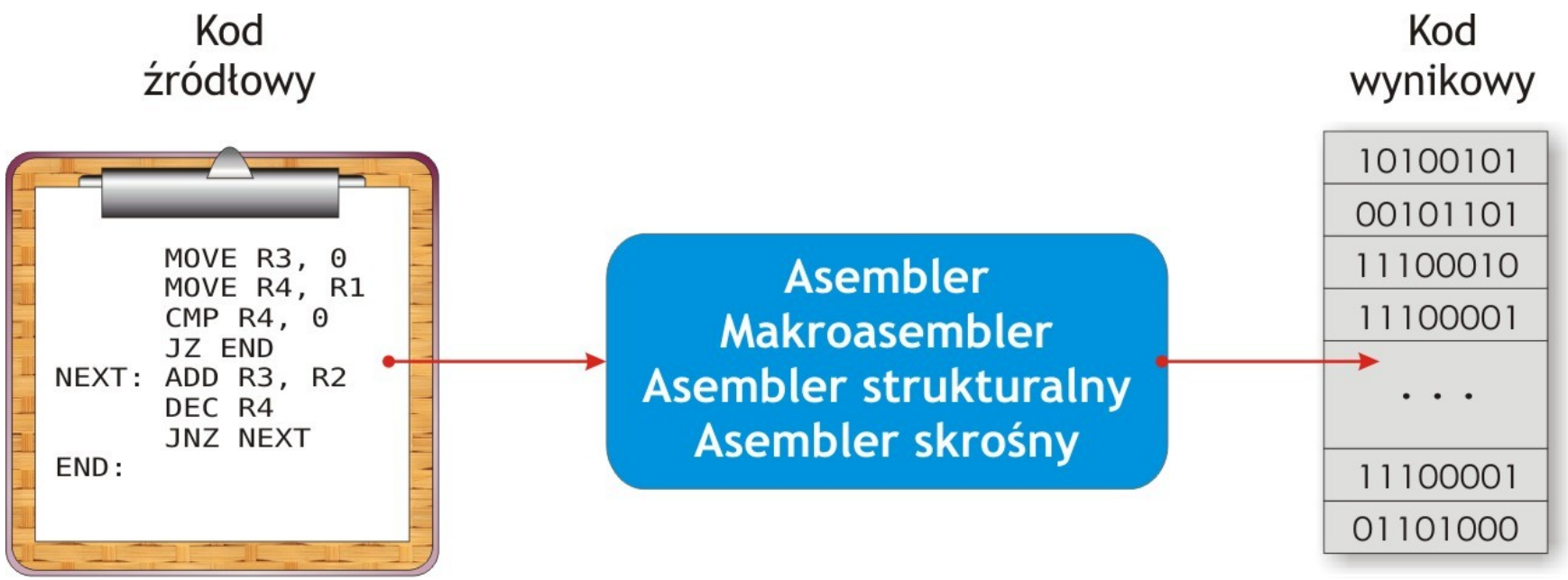
Programowanie w języku symbolicznym

- Program w języku symbolicznym musi zostać *przetłumaczony* na odpowiednie *rozkazy maszynowe*. Realizuje to *program translatora*, zwany ***asemblerem***.



Język symboliczny i asembler

- ***Język symboliczny*** (język *asemblera*) umożliwia pisanie programów z wykorzystaniem *symboli* (*mnemonik*, *mnemoników*) przypisanych poszczególnym rozkazom procesora.
- ***Mnemoniki*** to symboliczne *oznaczenia* (nazwy) *rozkazów* i *innych elementów*, np. rejestrów — **mov, move, ld, add, sub, out, ax, bc, si, di, ss, … .**
- ***Asembler*** to program dokonujący *tłumaczenia kodu źródłowego* programu w *języku symbolicznym* na *kod maszynowy* — *kod wynikowy*.



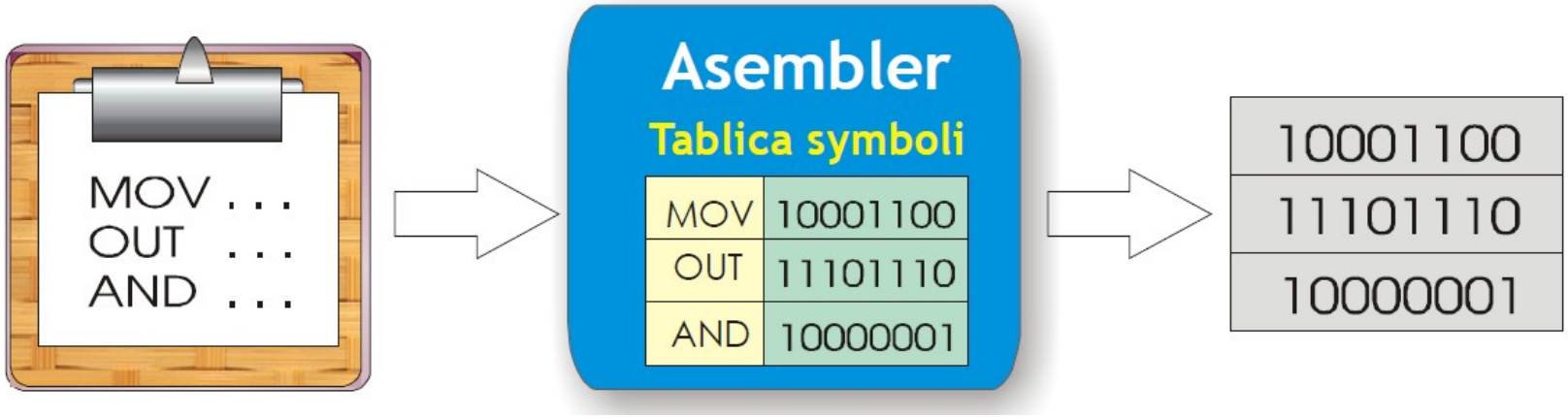
Asembler tłumaczy symbole na kod maszynowy

- Dla każdego procesora definiowana jest lista rozkazów, zawierająca binarne i mnemoniczne oznaczenia rozkazów:

Znaczenie rozkazu	Kod binarny	Mnemonika
Przesłanie bajtu pamięc—rejestr	10001100	MOV
Wyprowadzenie bajtu do układu wyjściowego	11101110	OUT
Mnożenie logiczne	10000001	AND
. . .	. . .	. . .

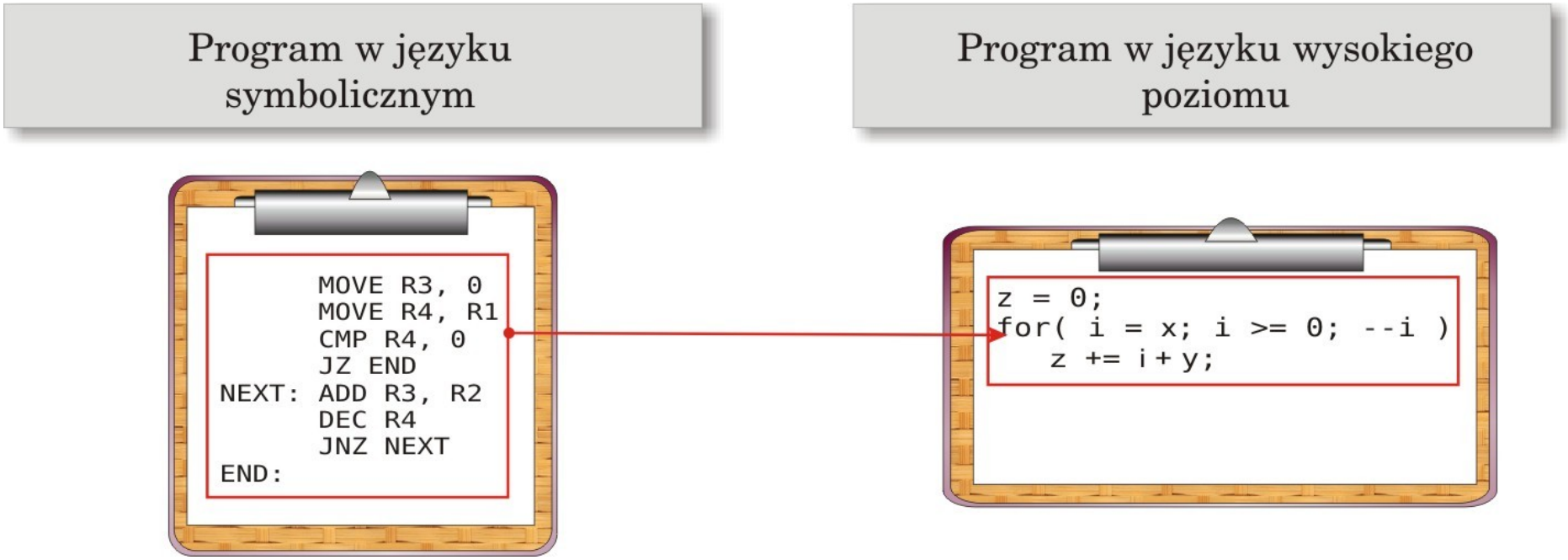
- Zwykle jedna instrukcja programu w języku symbolicznym odpowiada jednemu rozkazowi maszynowemu.

- Asembler w trakcie tłumaczenia wykorzystuje *tablice symboli*:



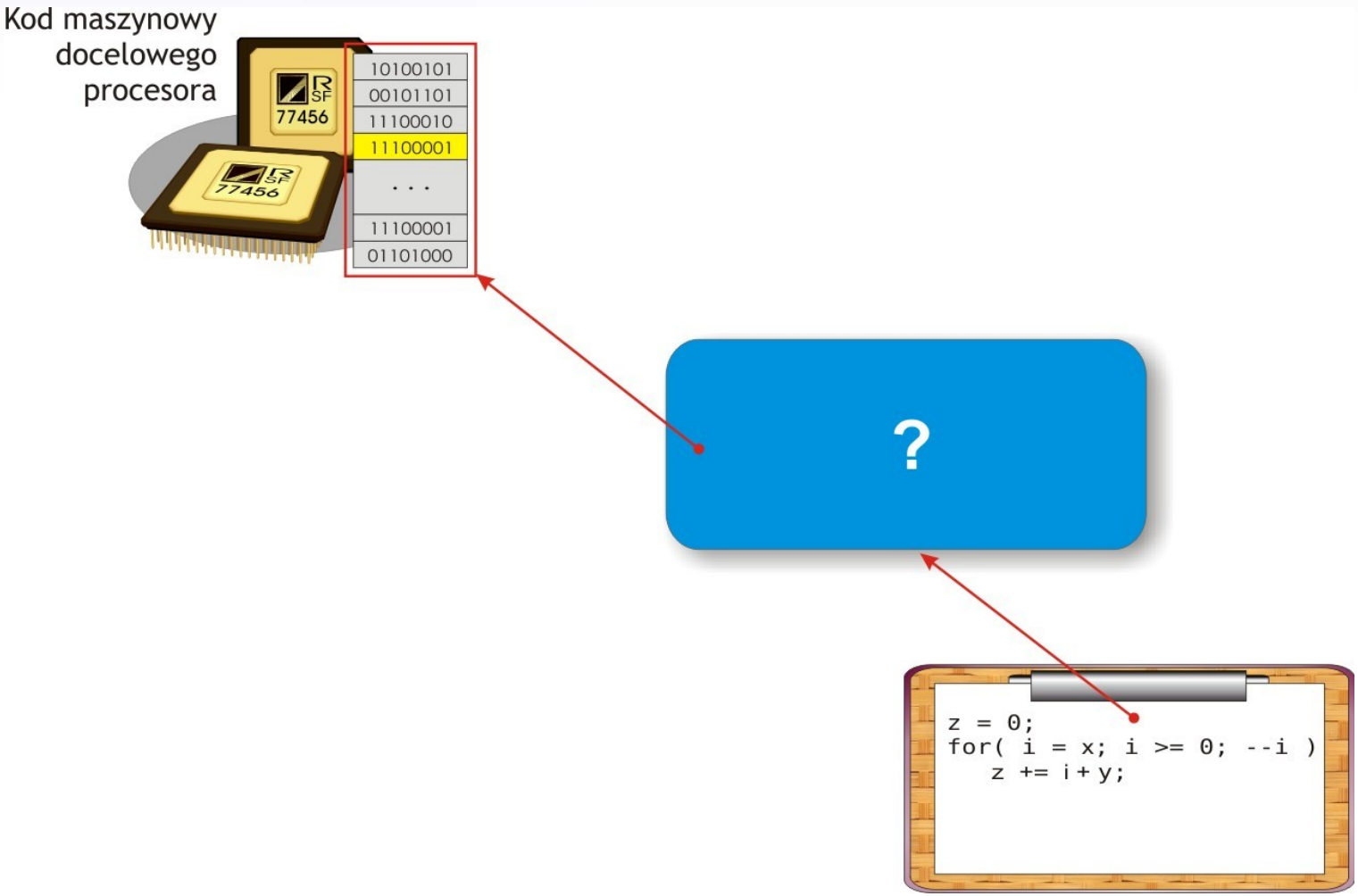
Języki wysokiego poziomu a kod maszynowy

- ***Język wysokiego poziomu*** — niezależny od platformy sprzętowej i systemowej język programowania, pozwalający programiście skoncentrować się na logice rozwiązywanego problem.
- Notacja typowych języków wysokiego poziomu to połączenie *elementów języka angielskiego* z notacją wywodzącą się z *matematyki*.
- Jedna instrukcja programu w języku wysokiego poziomu odpowiada zwykle wielu rozkazom maszynowym.



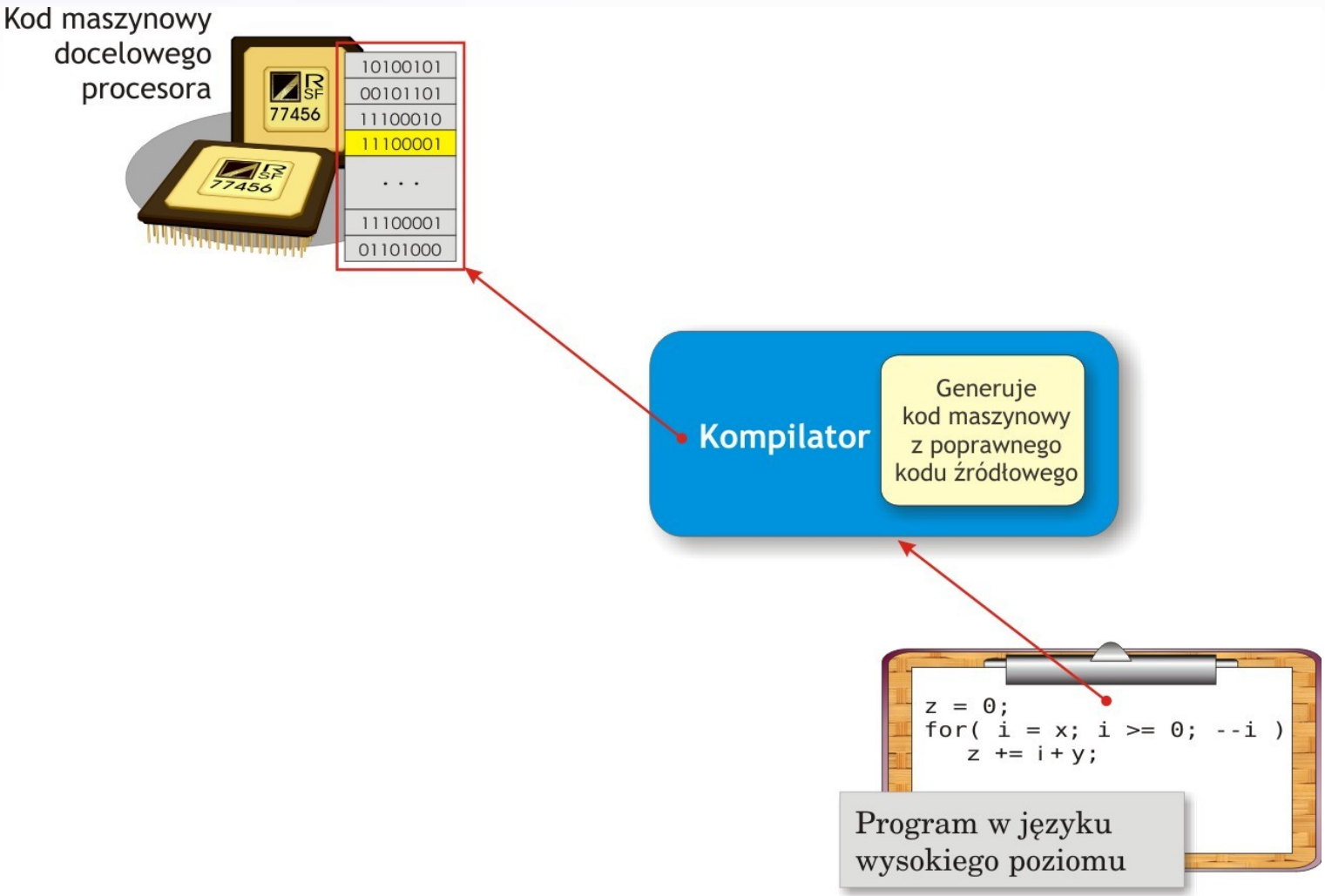
Translacja programu w języku wysokiego poziomu

- Program w języku wysokiego poziomu jest (a przynajmniej powinien być) *zrozumiały* dla *programisty* ale *nie jest zrozumiały* dla *procesora*.



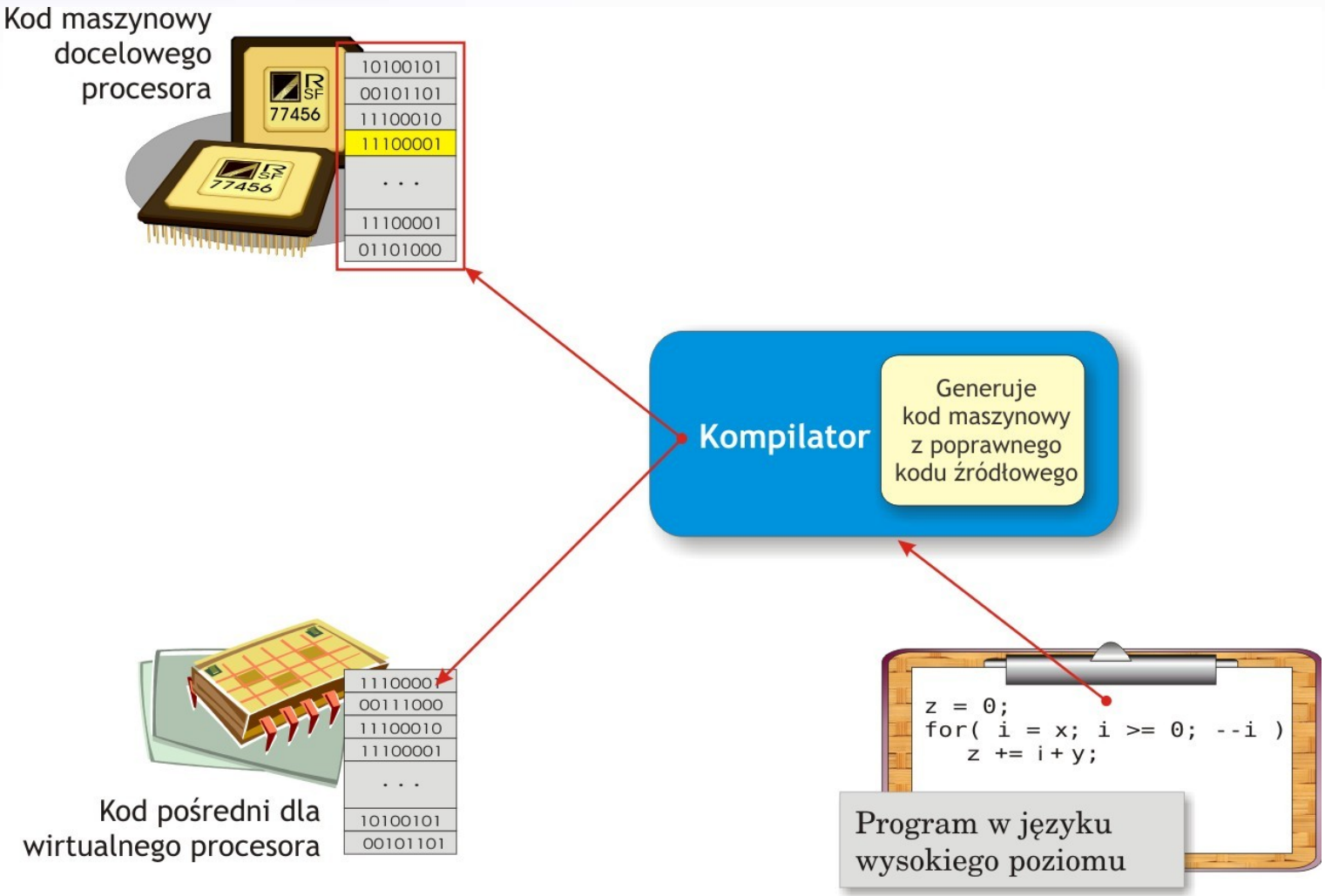
Translacja programu w języku wysokiego poziomu

- Translacja do postaci kodu maszynowego konkretnego procesora realizowana jest przez *kompilator*.



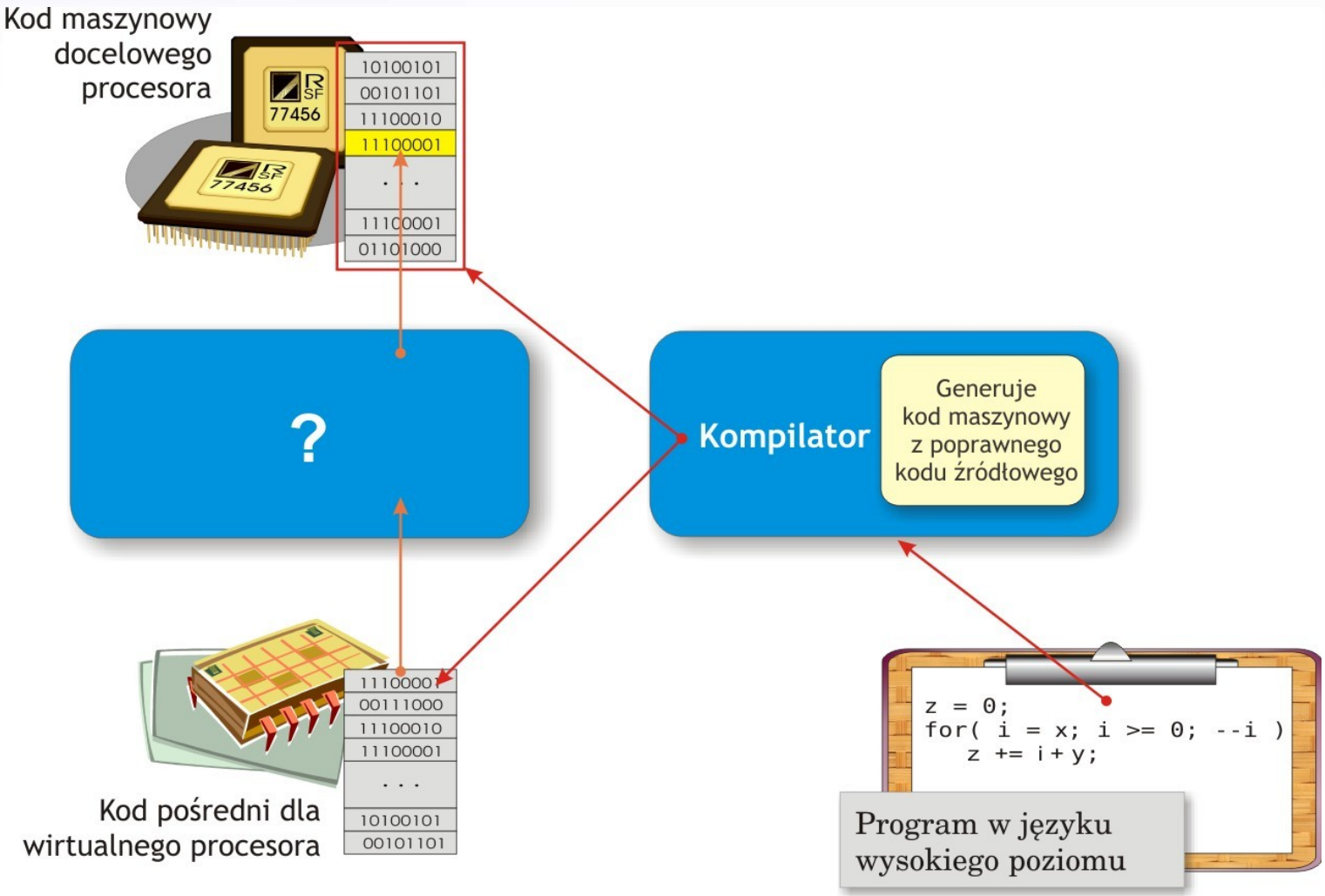
Translacja programu w języku wysokiego poziomu

- Kompilator może dokonać translacji do kodu „maszynowego” procesora wirtualnego — taki kod nazywa się *kodem pośrednim*.



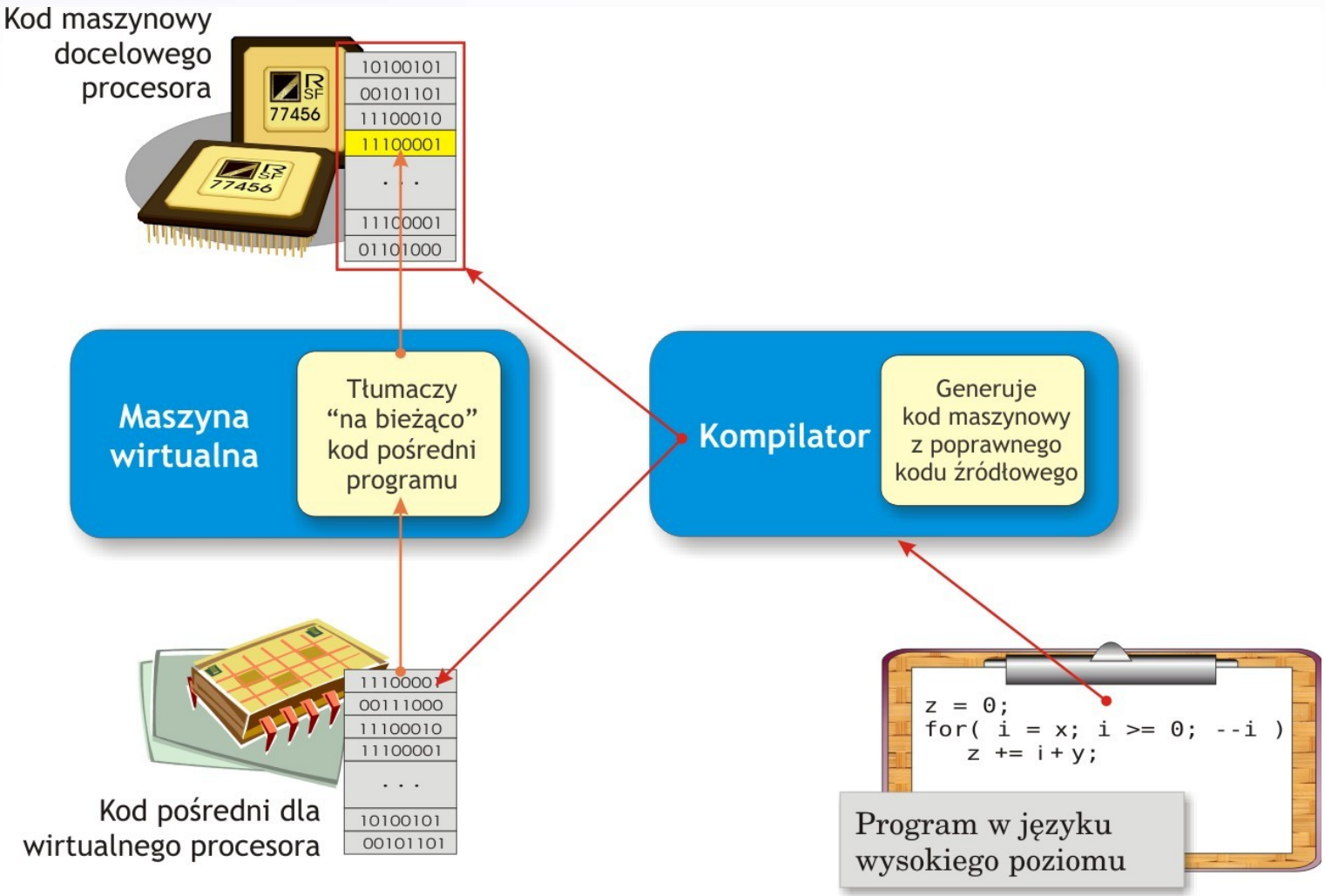
Translacja programu w języku wysokiego poziomu

- Kod pośredni *nie jest kodem maszynowym* konkretnego procesora, jakim cudem kod pośredni może *działać*?



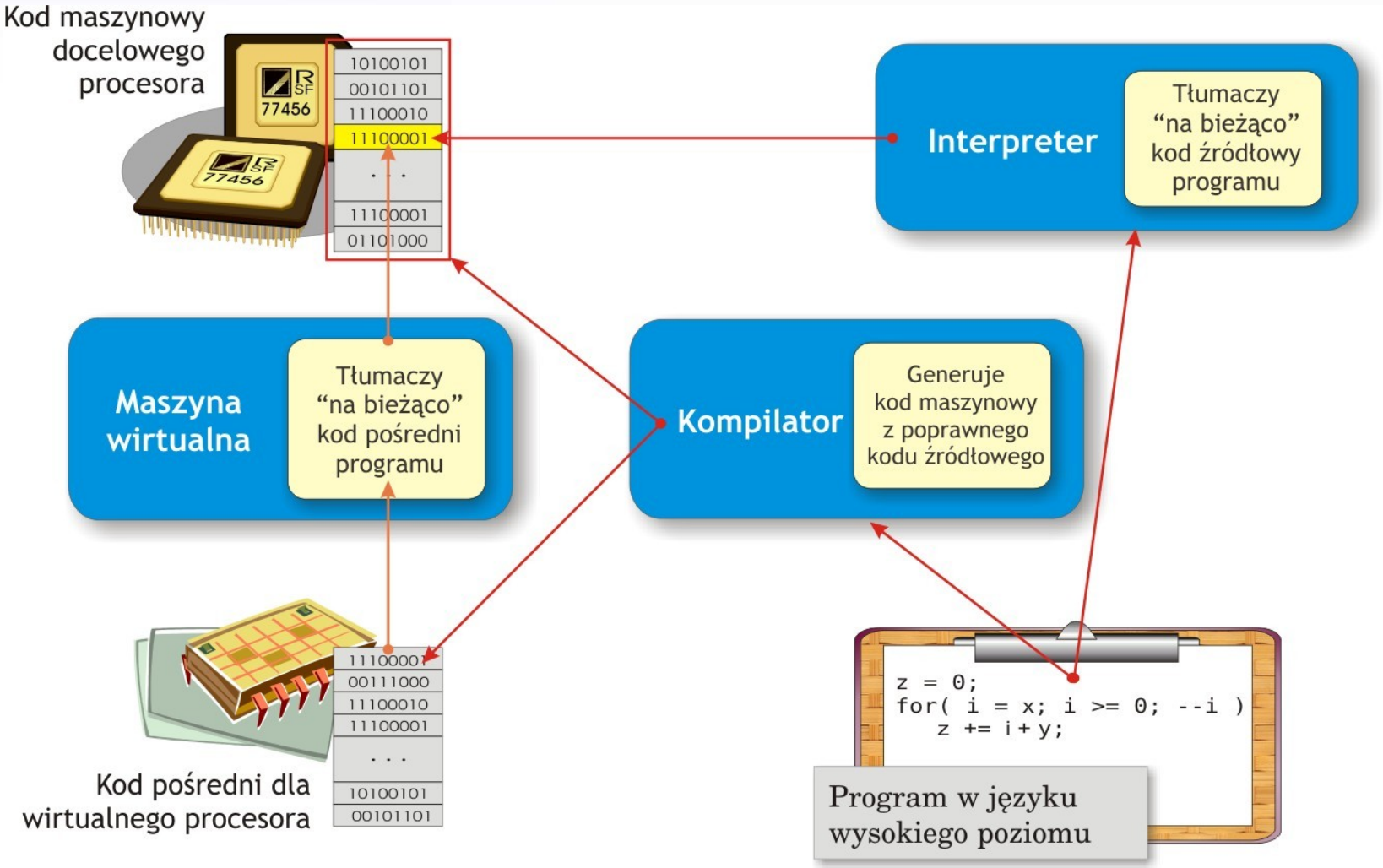
Translacja programu w języku wysokiego poziomu

- Kod pośredni może być na bieżąco tłumaczony i podsyłany do wykonania procesorowi — realizuje to *interpreter kodu pośredniego*, aktualnie przybierający postać *maszyny wirtualnej*.



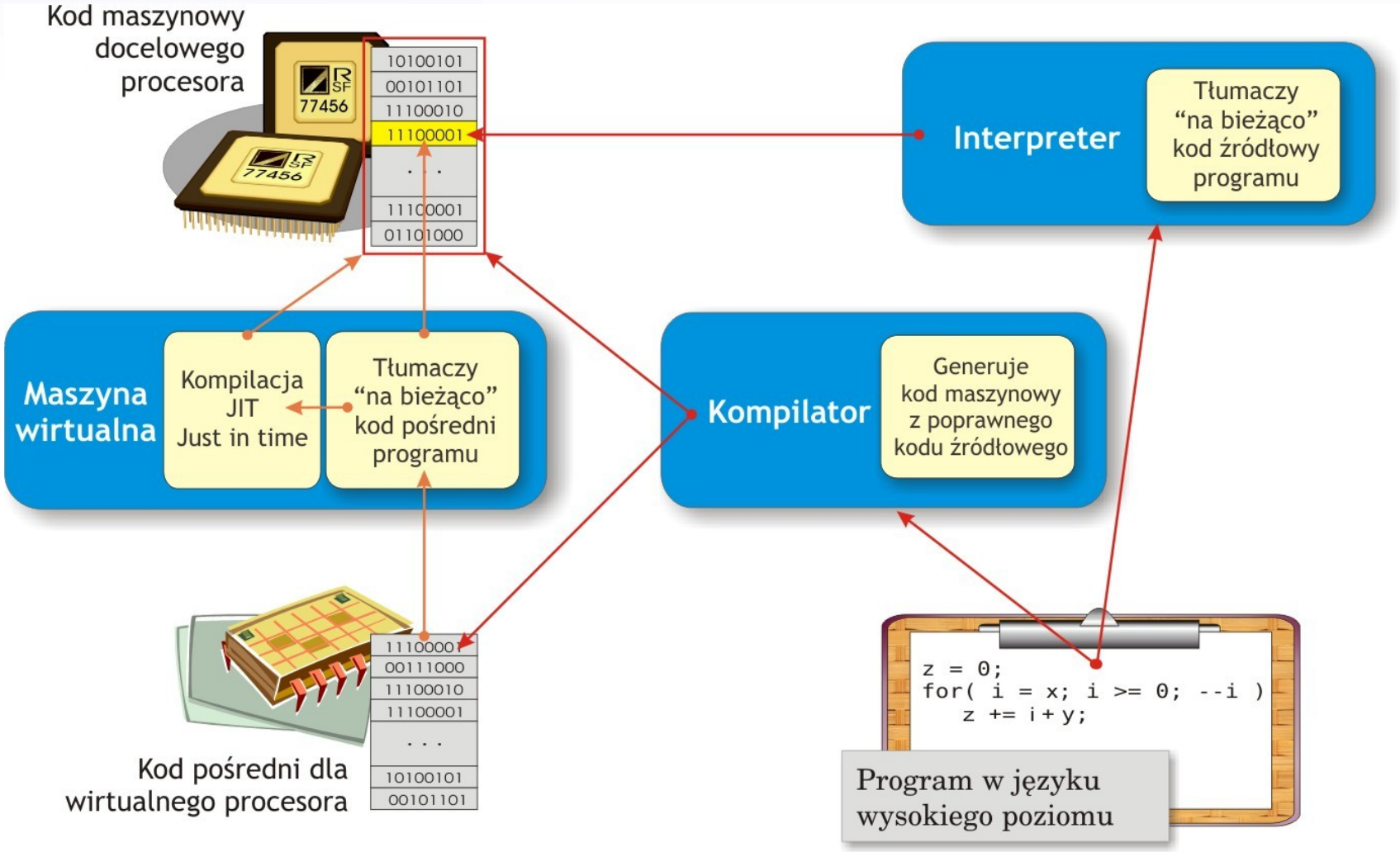
Translacja programu w języku wysokiego poziomu

- Inny rodzaj translacji to *interpretacja* programu źródłowego, realizowana jest przez translator zwany *interpreterem*.



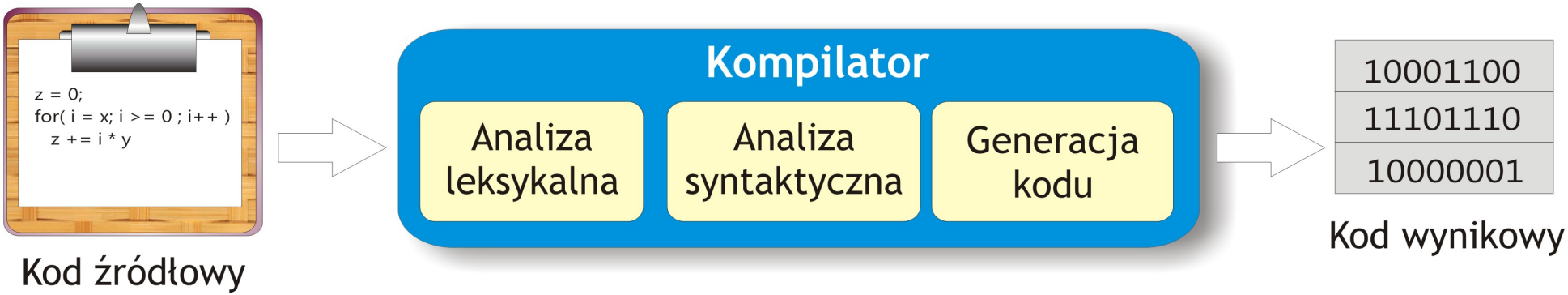
Translacja programu w języku wysokiego poziomu

- Współczesne maszyny wirtualne dokonują *kompilacji kodu pośredniego* do *kodu maszynowego* — ang. *just in time compiling*.



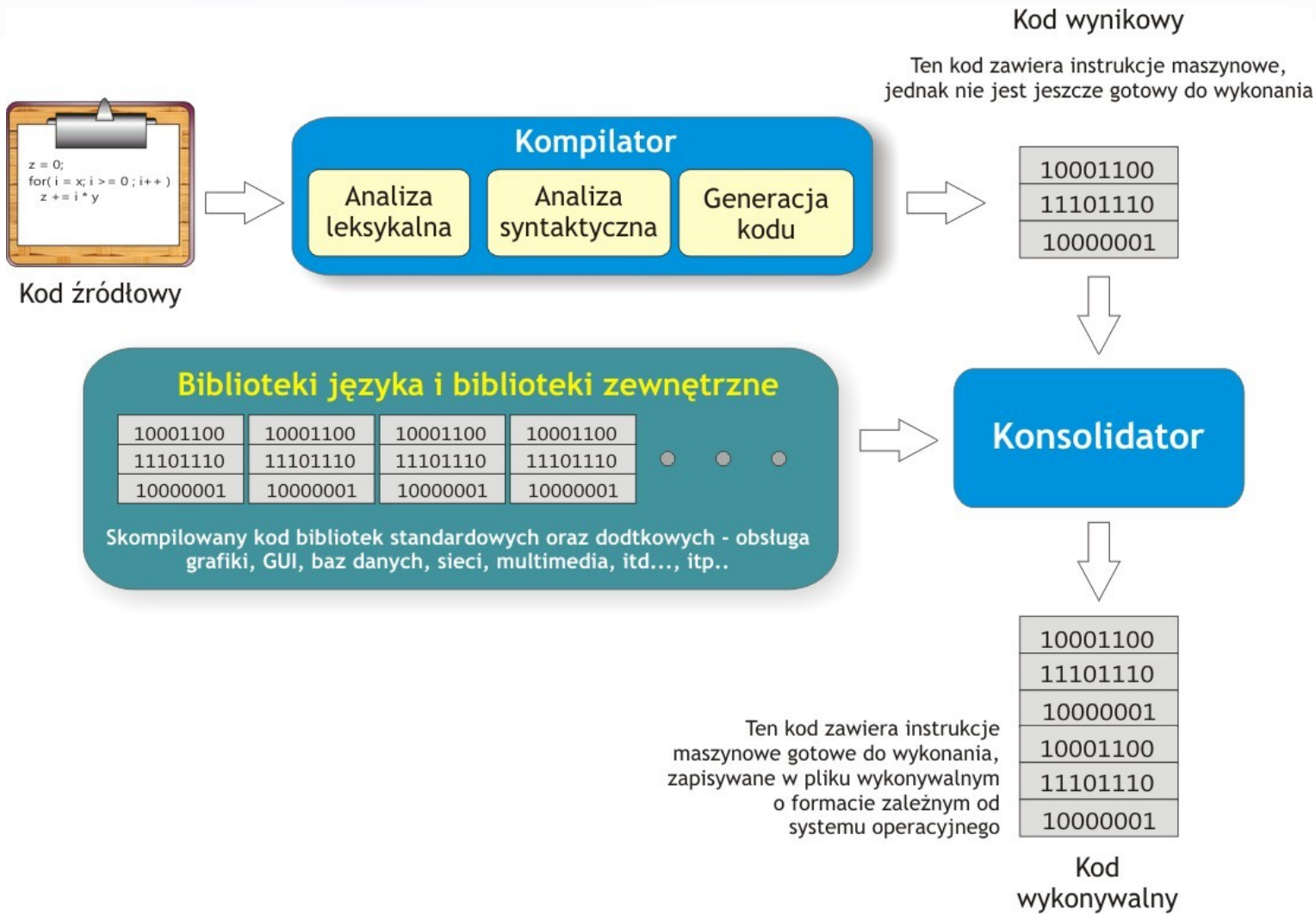
Kompilacja i jej etapy

- ***Kompilator*** (ang. *compiler*) to program dokonujący tłumaczenia programu napisanego w języku wysokiego poziomu na kod maszynowy konkretnego, rzeczywistego lub wirtualnego procesora.
- Kompilacja programu obejmuje szereg faz, najistotniejsze to *analiza leksykalna*, *syntaktyczna* oraz *generacja kodu wynikowego*.
- Praktycznie w trakcie kompilacji wykonuje się jeszcze wiele dodatkowych elementów, takich jak np. *optymalizacja kodu wynikowego*.



Konsolidacja programu

- ***Konsolidator*** (ang. *linker*) to program łączący w *jedną całość* wszystkie, uprzednio skompilowane elementy programu, tworząc odpowiedni kod *wykonywalny*.



Języki programowania niskiego poziomu

Zalety:

- szybki i relatywnie nieduży kod wynikowy,
- dobre i efektywne wykorzystanie sprzętu,
- implementacja programów dedykowanych i specyficznych.

Wady:

- ograniczenie do listy rozkazów danego procesora,
- wymagana znajomość struktury i specyfiki działania procesora i jego otoczenia,
- ręczne zarządzanie danymi, stosem, uciążliwe kodowanie,
- trudne kodowanie rozwiązań problemów złożonych.

Programista myśli kategoriami *sprzętu* a nie *dziedziny problemu*, dużo uwagi poświęca na *kodowanie*, często kosztem myślenia o *rozwiązaniu problemu*.

Języki programowania wysokiego poziomu

Zalety:

- zwykle duża niezależność od platformy sprzętowo-systemowej,
- elastyczność i łatwość programowania,
- wielość technik, metod i paradygmatów programowania,
- wsparcie dla budowania dużych i złożonych programów.

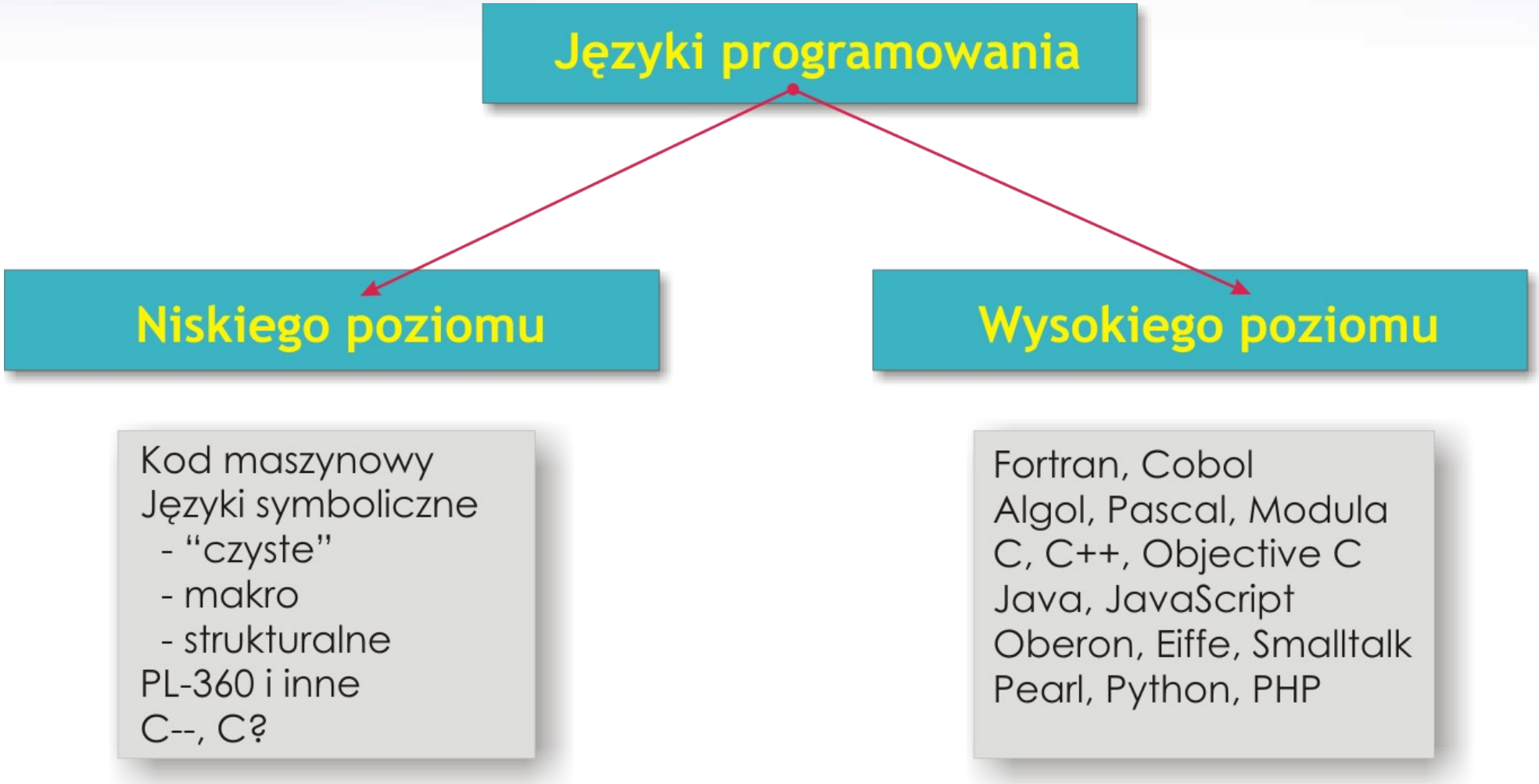
Wady:

- czasami wolny i duży kod wynikowy,
- kłopotliwe programowanie fragmentów “bliskich sprzętu”,
- zależność od kompilatora, bibliotek i narzędzi wspomagających.

Język wysokiego poziomu pozwala programiście skupić się na *rozwiązywaniu* potencjalnie *złożonego problemu*, a nie na *kodowaniu* podstawowych operacji.

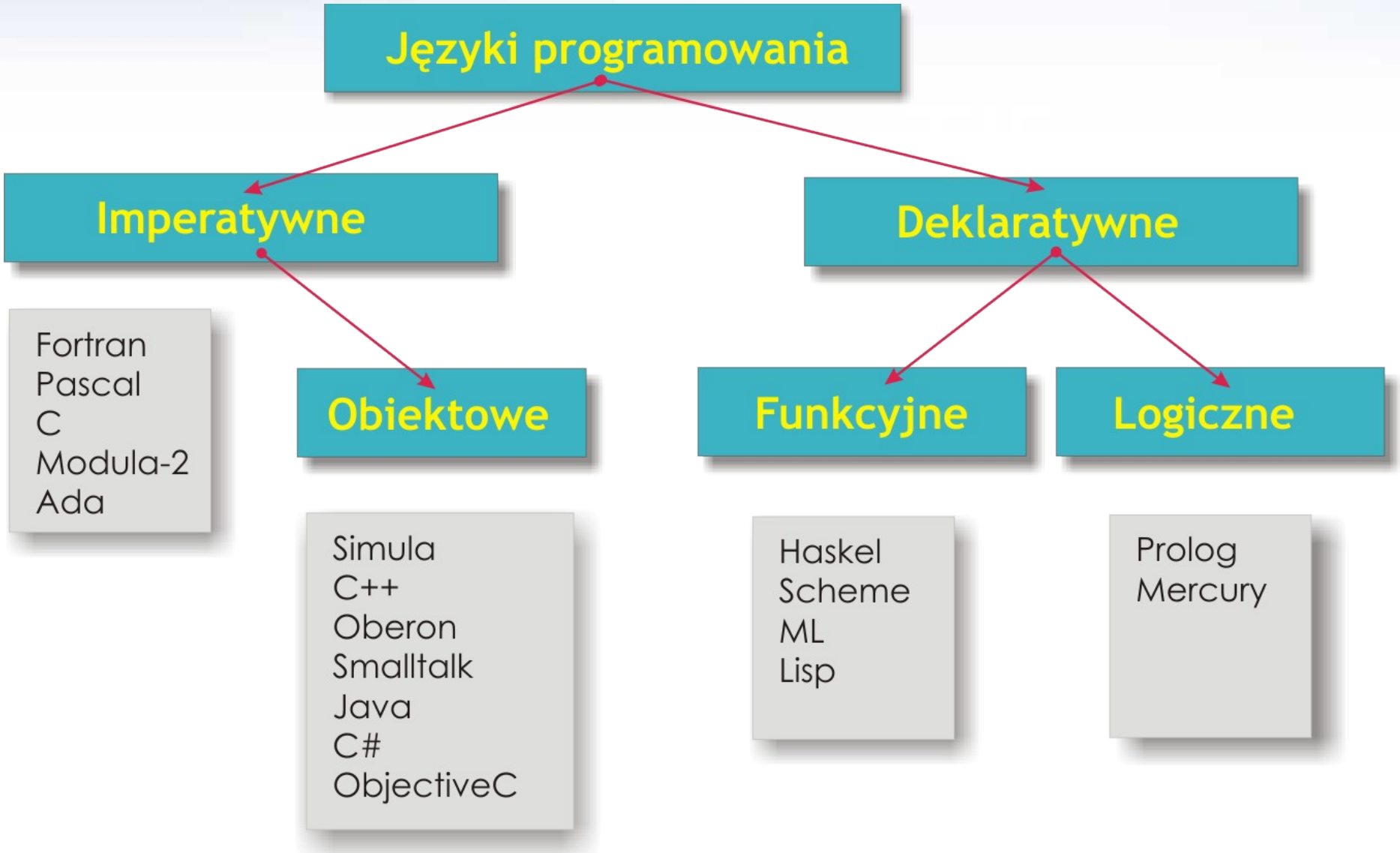
Podział języków programowania

- Według stopnia “bliskości” w stosunku do sprzętu.



Podział języków programowania

- Według *paradygmatów* programowania



Język C i język C++

Właściwości ogólne, geneza, historia, rozwój, narzędzia

Kompilator Javy vs. Kompilator C/C++

- W przypadku języka Java wykreowano marketingowy slogan:

  WORA — Write Once, Run Anywhere lub *WORE — Write Once, Run Everywhere*

- Oznacza to, że ***kod wynikowy jest przenośny***, i ***powinien uruchomić się*** wszędzie tam, gdzie istnieje odpowiednia maszyna wirtualna JVM.

- *WORE* najczęściej w praktyce oznacza:

  Write Once, Debug Everywhere

- W przypadku języków C/C++ sparafrazowano powyższe stwierdzenie:

  WOCA — Write Once, Compile Anywhere

Języki C i C++

- To języki kompilowane do kodu maszynowego konkretnego procesora.
- Kod wynikowy ***z definicji nie jest przenośny pomiędzy*** platformami sprzętowymi i systemowymi (choć istnieją narzędzia kompilujące do kodu pośredniego JVM).
- Kod źródłowy ***jest potencjalnie przenośny***, o ile zadba o to programista i pozwolą na to stosowane biblioteki.
- Uwaga — istnieją normatywne zalecenia jak pisać przenośny kod źródłowy — to POSIX oraz Single UNIX Specification, ukierunkowane oczywiście na systemy Unix'owe.
- Programy pisane w C++ są przenośne na poziomie kodu źródłowego, o ile są *zgodne ze standardami* i wykorzystują *przenośne biblioteki*.
- W praktyce uzyskanie przenośności — pisanie programów wieloplatformowych — jest *zadaniem niełatwym*.

C vs. C++ vs. Java

- Całkiem uzasadnione porównania samochodowe...

Język C++ jako rozszerzenie C

- Język C++ powstał jako obiektowe rozszerzenie języka C.
- C i C++ są jednak *różnymi językami*, w zakresie elementów wspólnych występują *subtelne* lecz *istotne różnice*.
- Język C bazuje na *koncepcji programowania strukturalnego*, program dzieli się na *modułu* a te na *podprogramy zwane funkcjami*. Język ten nie posiada mechanizmów programowania obiektowego.
- Język C++ ma charakter hybrydowy — można z jego wykorzystaniem programować *strukturalnie* i/lub *obiektowo*. Programista ma pełną swobodę wyboru stylu programowania.
- C++ to nie jedyny, obiektowy następca języka C, istnieją języki Objective-C (natywny język dla platformy IOS) oraz D. Jednak C++ jest — póki co — najpopularniejszy.

Kto rządzi językami C i C++?

- Nie ma „firmy-właściciela” języka C i C++, opracowaniem norm języka zajmują się zespoły fachowców opracowujące standardy dla ANSI i ISO.
- Języki te wywodzą się z laboratoriów badawczych firm telekomunikacyjnych Bell i AT&T, przy czym nie były ani nie są oficjalnie przez te firmy moderowane.
- Istnieje wiele kompilatorów języka C++, występują jako produkty płatne (np. Microsoft, Borland/Embarcadero), jak również w postaci produktów free oraz open source (np. *gcc*).
- Praktycznie — jeżeli istnieje jakaś komputerowa platforma sprzętowa lub systemowa, można się spodziewać, że istnieje dla niej przynajmniej kompilator języka C.
- A to zwykle pozwala na stworzenie narzędzia dla C++, bowiem kod C++ może być tłumaczony na C i kompilowany dalej zwykłym kompilatorem C — tak działał pierwszy „kompilator” języka C++ o nazwie *cfront*.

Geneza języka C — prehistoria

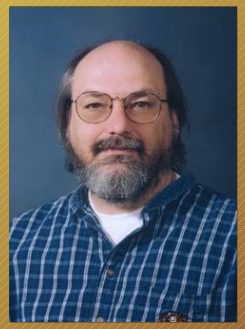
- Początki języka C są związane z wczesną fazą rozwoju systemu *Unix*.
- Prace nad protoplastą tego systemu zostały rozpoczęte pod koniec lat sześćdziesiątych w laboratoriach telekomunikacyjnej firmy Bell – BTL – wspólnie z General Electric Company oraz MIT.
- W ramach projektu powstać miał wielodostępny system operacyjny *Multics – Multiplexed Information and Computing Service*.
- Mimo iż system *Multics* działa, prace przerwano w 1969 roku, programiści z BTL szukają innych projektów. Ciągle chcą napisać uniwersalny system operacyjny. Jednak w BTL nie ma dla takiego projektu dobrej atmosfery.

http://www.multicians.org/

Geneza języka C — ludzie

Kenneth Thompson

Urodzony w 1943 roku w Nowym Orleanie. Otrzymuje tytuł magistra Uniwersytetu Kalifornijskiego w 1966, jest entuzjastą elektroniki, jednak to informatyka staje się jego pasją.



Dennis M. Ritchie

Urodzony w 1941 w Bronxville (N.Y.), w 1963 kończy Harvard z dyplo-mem fizyka, pięć lat później broni doktorat z matematyki. Zainteresowany teoretycznymi podstawami informatyki

Brian Kernighan

Urodzony w 1942 w Toronto (Ontario), w roku 1964 kończy Uniwersytet w Toronto, w roku 1969 broni doktorat z elektroniki na Uniwersytecie w Princeton.

Wszyscy trzej spotykają się w laboratorium badawczym firmy Bell, pierwsi dwaj pracują w zespole tworzącym system *Multics*.

Geneza języka C — nudy i gra

- Informatycy z firmy *Bell*, wycofani z projektu *Multics* nudzą się.
- Ken Thompson rozwija grę *Space Travel*, początkowo napisaną na komputerze GE Honeywell 635.
- Thompson reanimuje porzucony gdzieś w laboratorium firmy *Bell* komputer PDP-7.
- Thompsom używa asemblera skrośnego i przenosi kod binarny na komputer DEC PDP-7 używając taśmy papierowej.
- Komputer DEC PDP-7 ma w 1968 roku pamięć 8K 18-sto bitowych słów.

Komputer PDP-7

Geneza języka C — od PDP 7 do PDP 11

- Praca nad rozwojem gry Space Travel wymaga napisania systemu operacyjnego dla PDP-7, Ken Thompson wspólnie z Denisem Ritchie rozpoczynają pracę nad tym systemem.
- Powstaje pierwsza wersja systemu nazwanego *Unix* — jego nazwa jest kalamburem nazwy *Multics*.
- Komputer PDP-7 jest przestarzały. Zespół podejmuje próbę zdobycia komputera PDP-11.
- Pod koniec lat 60-tych PDP-11 kosztuje 65 000 $!

Geneza języka C — PDP 11!

- Ostatecznie zespół otrzymuje komputer w lecie 1970r., jednak prace nad przeniesieniem nań systemu Unix startują dopiero w grudniu, po doposażeniu go w dysk.

- Oficjalnie prace poświęcone są opracowaniu systemu dla wydziału patentowego BTL.

Geneza języka C — przenosiny oprogramowania

Denis Ritchie i Ken Thompson przenoszą system Unix na z PDP-7 na PDP-11 za pośrednictwem terminalu telegraficznego.

Geneza języka C — wykluwanie się języka

- Ken Thompson potrzebuje nowego języka programowania do tworzenia oprogramowania systemowego.
- Istnieje już jego własny język o nazwie B, stworzony przez Thompson'a.
- Język B miał sporo ograniczeń, co spowodowało rozpoczęcie przez Ritchie'go w roku 1971 prac nad rozszerzeniem tego języka. Doprowadziło to do powstania dialektu NB (ang. new B).
- Nazwa NB nie oddaje w pełni różnicy pomiędzy tym językiem a językiem B.
- Nowy język otrzymuje nazwę C, w roku 1973 powstaje specyfikacja języka C oraz odpowiednie narzędzia – rozpoczynają się prace nad przepisanie jądra *Unix*'a dla komputera PDP-11.
- W 1978 roku Dennis Ritchie i Brian Kernighan publikują książkę *Język programowania C* (oryginał ang. *The C Programming Language*).

Geneza języka C — standaryzacja

- Ostatecznie w lecie 1983 roku ANSI powołało komitet X3J11, którego celem było opracowanie standardu języka C. Komitet ten opracował raport (ANSI 89) w końcu roku 1989, standard ten został zaakceptowany przez ISO jako ISO/IEC 9899-1990.
- Standaryzacja przyczynia się do poprawy przenośności kodu źródłowego. Kompilatory języka C dostępne są na każdej, szanującej się platformie sprzętowej. Pośrednio ułatwia to przenoszenie systemu Unix i powstawanie jego dedykowanych wersji.
- Powstają nowe języki bazujące na C, np.: Objective C, C++.
- Po określeniu standardu języka C, jego specyfikacja przez dłuższy czas nie zmienia się. W tym samym czasie specyfikacja języka C++ ciągle ewoluuje.

Geneza języka C — standaryzacja, następcy

- Rozwój języka C++ pośrednio wpływa na modyfikację standardu C.
- Powstaje standard C99, opublikowany przez ISO w 1999 r. Standard ten zostaje przyjęty przez ANSI w marcu 2000 r.
- Większość zmian standardu C99 zbliża język C do C++. Z tego powodu wielu producentów kompilatorów nie jest zainteresowanych implementacją C99, preferując zamiast niego programowania w C++ bez wykorzystania obiektów.
- Z pnia języka C i C++ wyrosły również: Java, JavaScript, PHP, C#, i wiele innych.

Geneza języka C++ — autor

Bjarne Stroustrup

Urodzony w 1943 roku w Nowym Orleanie. Otrzymuje tytuł magistra Uniwersytetu Kalifornijskiego w 1966, jest entuzjastą elektroniki, jednak to informatyka staje się jego pasją. Urodzony w 1950 w Aarhus (Dania), skończył studia w swym rodzinnym mieście, doktorat obronił na Uniwersytecie w Cambridge. Od 1979 roku związany z centrum badawczym firmy Bell.

Aktualnie pracuje jako profesor w Texas A&M University, utrzymując jednocześnie swoje związki z firmą AT&T jako członek zespołu badawczego laboratorium tej firmy.
Twórca języka C++, autor książek:

- *Język programowania C++* (ang. *The C++ Programing Language*), trzy edycje + edycja specjalna;
- *Projektowanie i ewolucja C++* (ang. *The Design and Evolution of C++*).

Geneza języka C++ — prehistoria

- Bjarnie Sroustrup, pracujący w Centrum Badań Komputerowych Laboratorium firmy Bell (ang. Computing Science Research Center of Bell Labs) prowadzi badania symulacyjne związane z sieciami komputerowymi.
- Język Simula, idealny do symulacji pod względem funkcjonalnym, jest za mało efektywny. Stroustrup potrzebuje nowego, obiektowego języka programowania łączącego funkcjonalność języka Simula z efektywnością języka C.
- Bjarne Stroustrup projektuje język programowania łączący w sobie cechy języków Simuli i C.
- Język zostaje nazwany C z klasami (ang. *C with Classes*). Wersja ta nie zawiera jeszcze znanych aktualnie z C++ rozbudowanych mechanizmów obiektowych.
- Rick Mascitti proponuje nazwę C++ dla nowego języka. Nazwa sugeruje ewolucyjna naturę C++ w stosunku do języka C.

Geneza języka C++ — standaryzacja

- W połowie lat osiemdziesiątych XX w. język C++ zostaje zaprezentowany publicznie z możliwością wykorzystania poza Laboratorium Bell.
- Koniec lat osiemdziesiątych — Publikacja pierwszej specyfikacji C++ w postaci książki „The Annotated C++ Reference Manual” autorstwa Stroustrupa i Ellisa, znanej jako ARM. Rozpoczyna się proces standaryzacji języka.
- Grudzień 1997, oficjalna premiera pełnej wersji standardu ANSI C++.
- Obowiązuje standard ISO/IEC 14882:1998 (Standard for the C++ Programming Language) z drobnymi poprawkami zatwierdzonymi w 2003 r. (ISO/IEC 14882:2003).
- Ogłoszenie nowego standardu (tzw. C++0x) planowane było na 2009, ostatecznie standard ten oficjalnie opublikowano w roku 2011.

Literatura — język C

- Brian W. Kernigham, Dennis M. Richie, *Język ANSI C*, Wydawnictwo Naukowo–Techniczne, Warszawa, 2003.

  Klasyka!

- Clovis L. Tondo, Scott E. Gimpel, *Język ANSI C, ćwiczenia i rozwiązania*, Wydawnictwo Naukowo–Techniczne, Warszawa, 2003.

  Kopalnia sztuczek!

- Adam Sapek, *Wgłąb języka C*, Helion, Gliwice, 1993.

  Stare ale jare!

- Stephen Prata, *Szkoła programowania. Język C*, 2006, Helion.

  Może być tą jedyną :) *Gorąco polecam*!

Literatura — język C++

- S. Prata, *Szkoła programowania. Język C++*, 2006, Helion.

  Może być tą jedyną :) Gorąco polecam!

- J. Liberty, *C++ dla każdego*, Helion, 2002.

  Jeżeli programowanie to nie Twoja pasja, to to książka dla Ciebie ;)

- S.B. Lippman, J. Lajoie, *Podstawy języka C++*, WNT, 2003.

  Świetne, ale trzeba już coś o C++ wiedzieć.

- B. Stroustrup, *Język C++*, WNT, 2002.

  Dobre lecz trudne.

- B. Eckel, *Thinking in C++ 2nd Edition*, dostępne via Internet:

  `http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html`

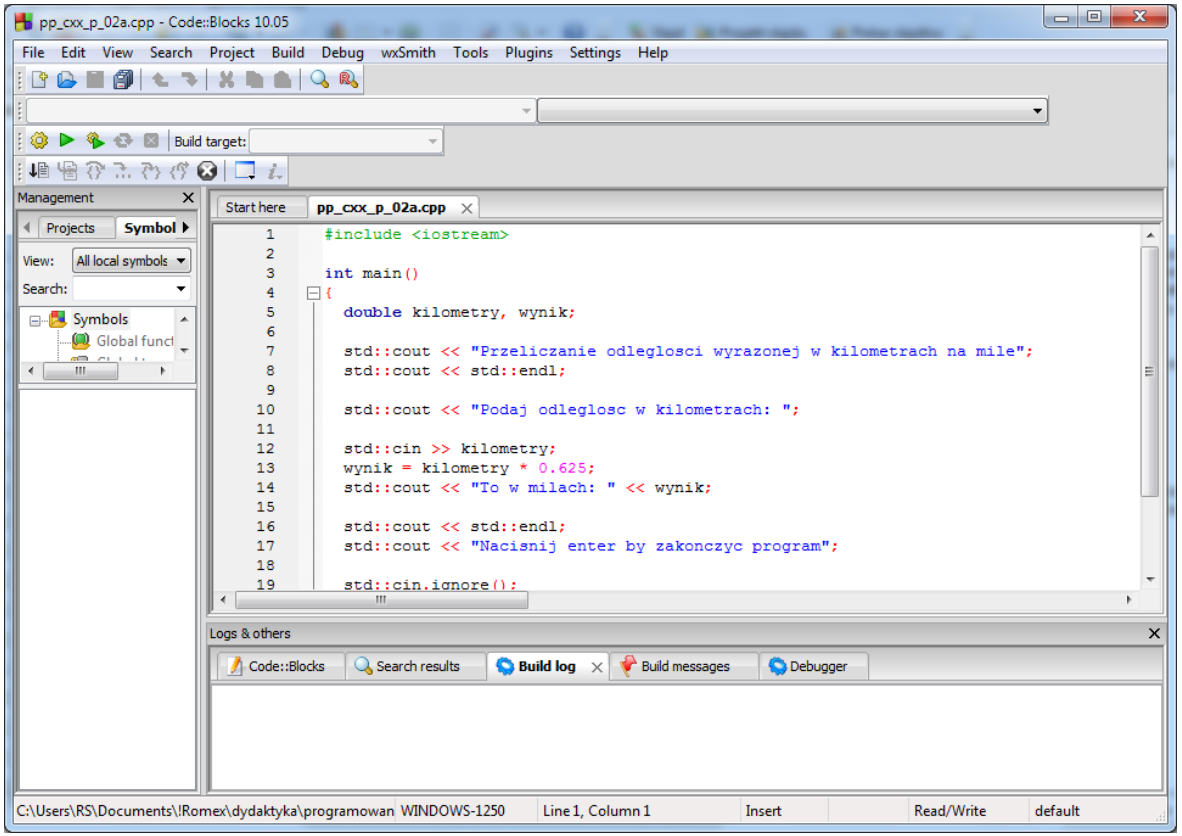
  Lektura na wiele tygodni.

Narzędzia — CodeBlocks

- Zintegrowane, wieloplatformowe środowisko do tworzenia programów w językach C i C++.
- Dostępne nieodpłatnie: `http://www.codeblocks.org/`

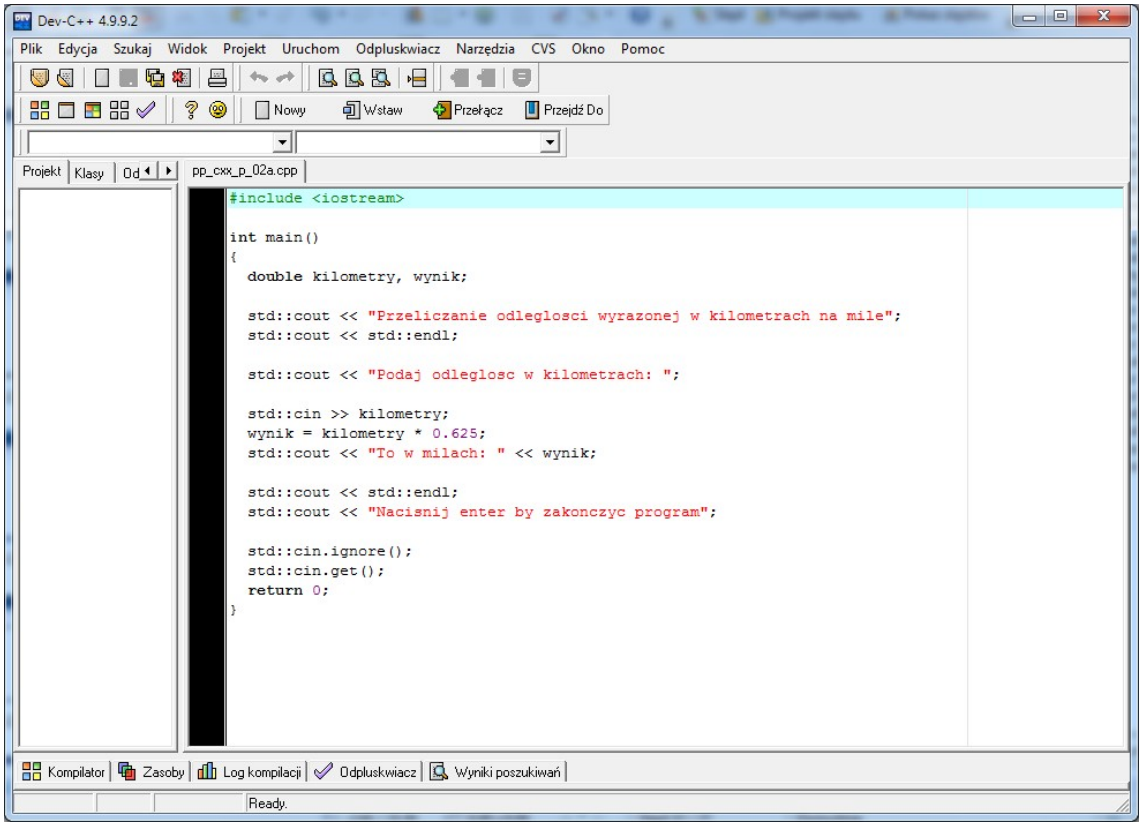
Gorąco polecam!

- Dostępna wersja zintegrowana z kompilatorem *gcc*.



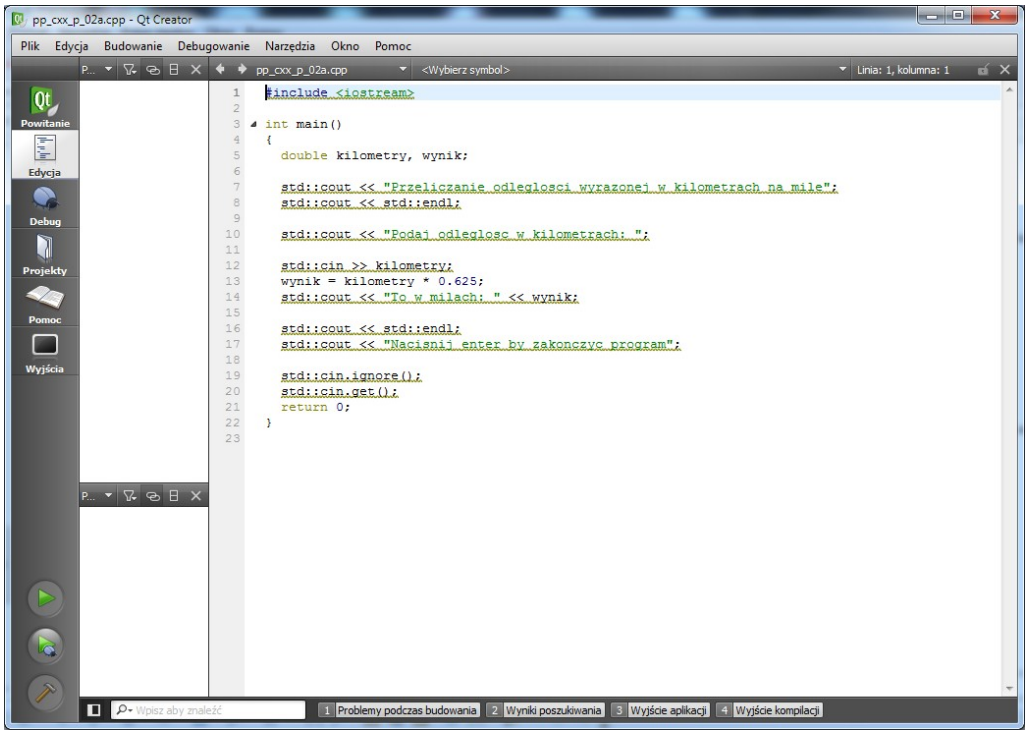
Narzędzia — DevC++

- Zintegrowane środowisko do tworzenia programów w językach C i C++ dla systemów z rodziny Windows. Rozwój pakietu stoi pod znakiem zapytania.
- Dostępne nieodpłatnie: `http://www.bloodshed.net/devcpp.html`
- Dostępna wersja zintegrowana z kompilatorem *gcc*.



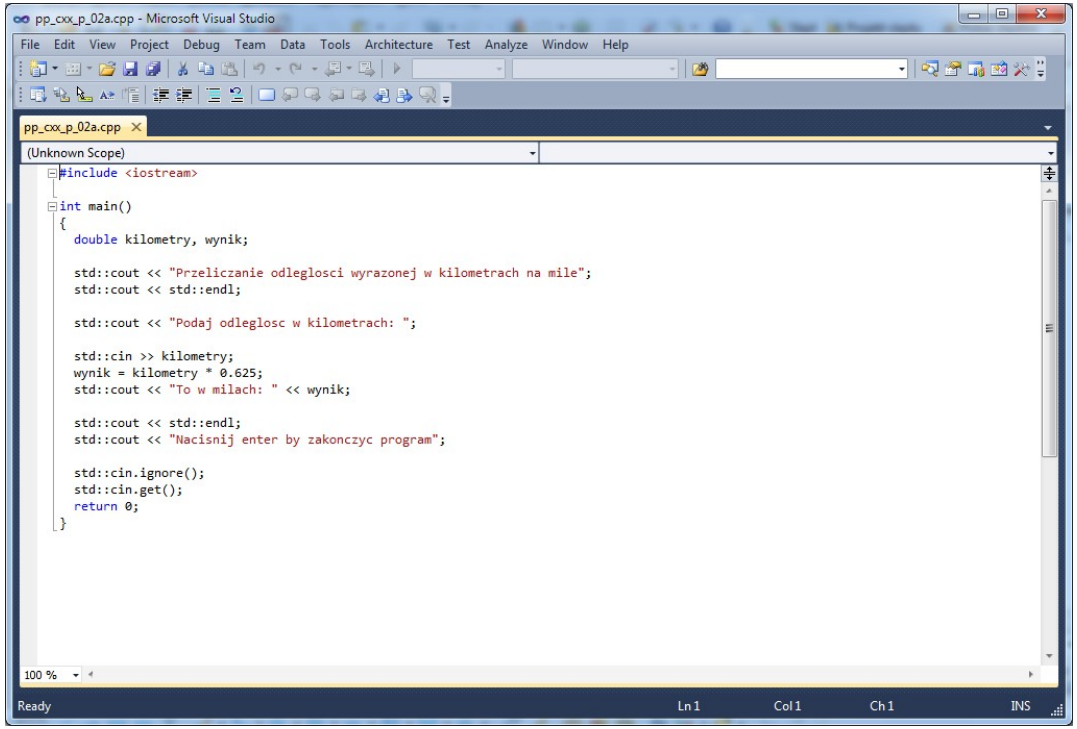
Narzędzia — QtCreator

- Zintegrowane, wieloplatformowe środowisko do tworzenia programów w językach C i C++.
- Aktualnie pakiet firmowany i rozwijany pod patronatem i kontrolą firmy NOKIA.
- Dostępne komercyjnie oraz nieodpłatnie: `http://qt.nokia.com/`
- Wersja zintegrowana z kompilatorem *gcc*.



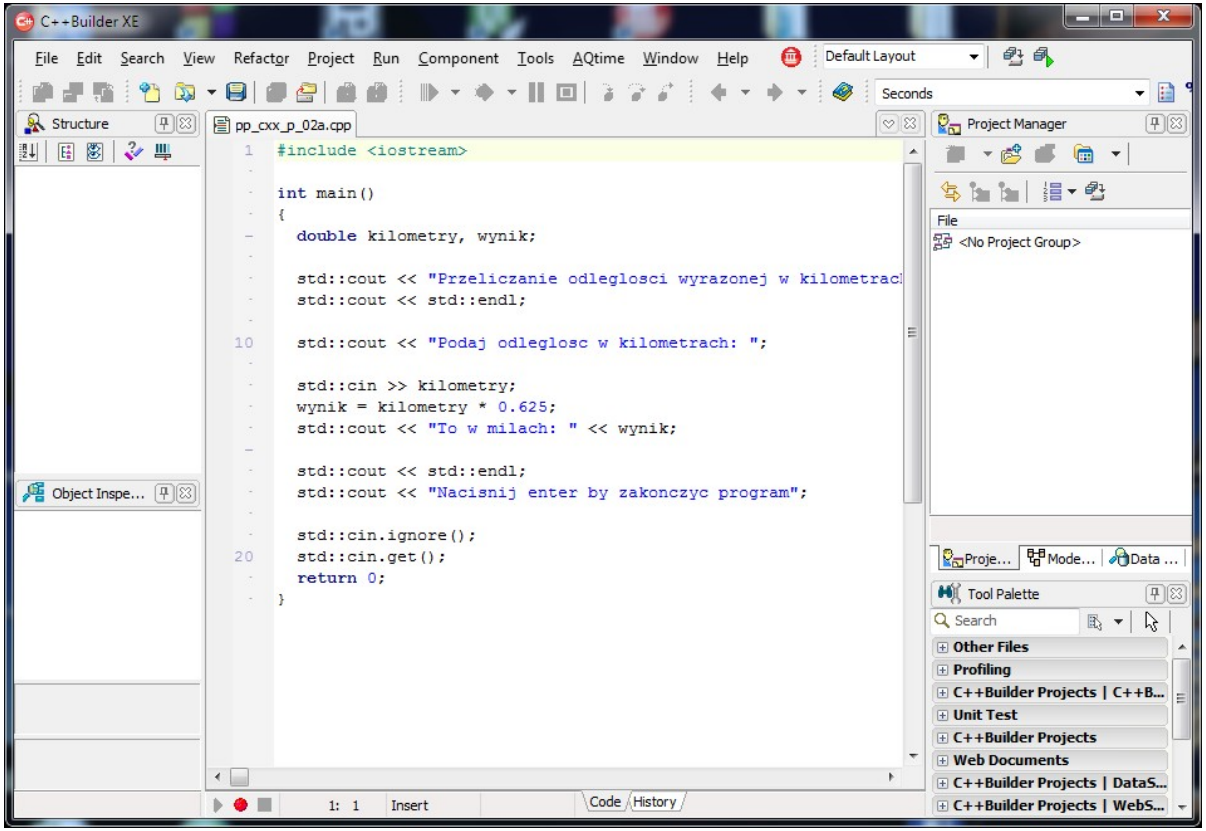
Narzędzia — VisualC++ z VisualStudio

- Zintegrowane środowisko do tworzenia programów w językach C i C++ dla systemów z rodziny Windows.
- Element VisualStudio, integracja z C#, VisualBasic.
- Dostępne nieodpłatnie dla studentów, płatne dla zastosowań komercyjnych.
- Bazuje na platformie .NET.



Narzędzia — C++ Builder

- Zintegrowane środowisko do tworzenia programów w językach C i C++ dla systemów z rodziny Windows.
- Element RADStudio, integracja z Delphi, produkt Borland/Embarcadero.
- Dostępne na warunkach komercyjnych.



Dziękuję za uwagę

Pytania? Polemiki?

Teraz, albo:

roman.siminski@us.edu.pl