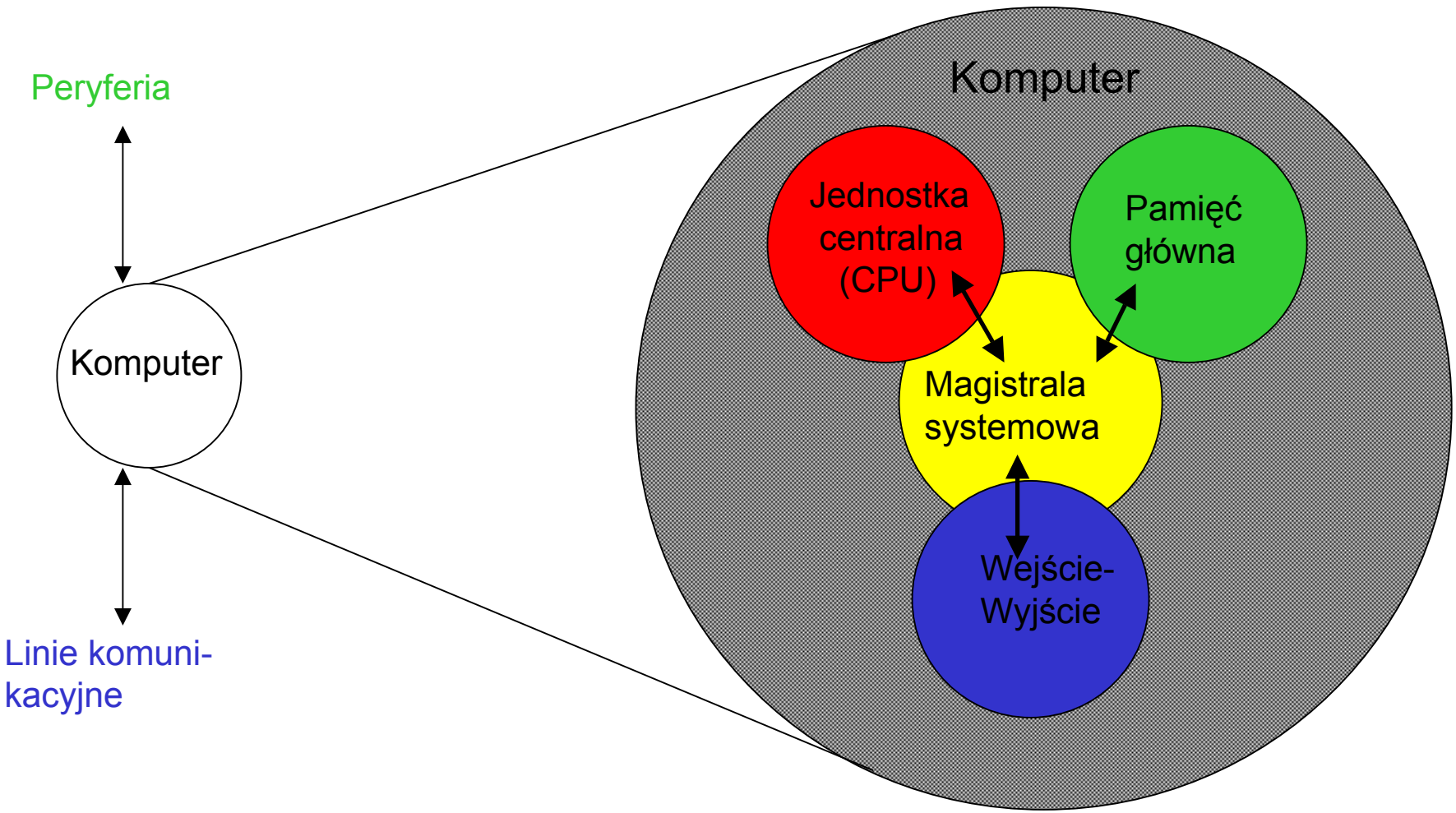
Zbigniew S. Szewczak
Podstawy Systemów Operacyjnych

Wykład 4

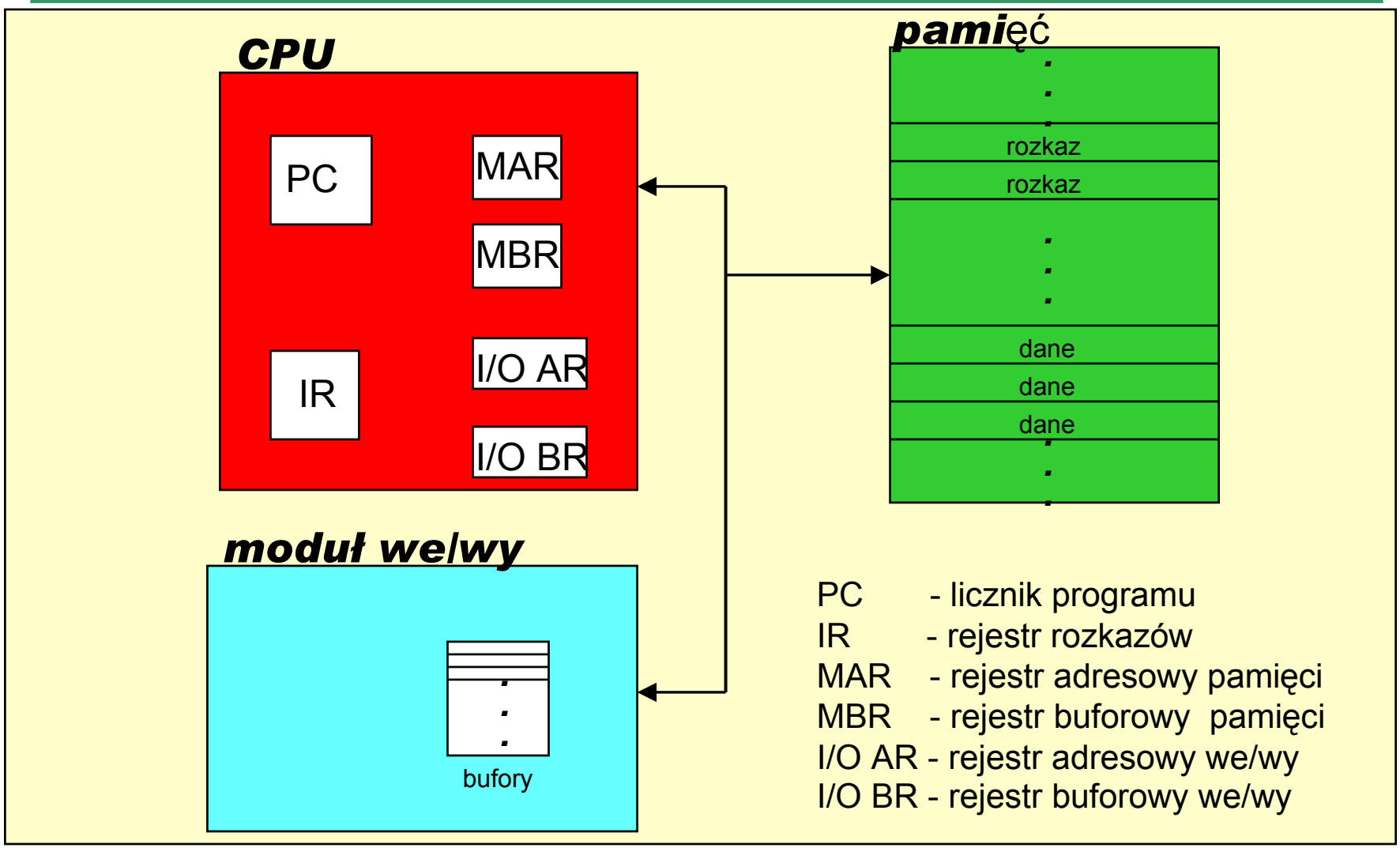
Działanie systemu komputerowego

Toruń, 2011

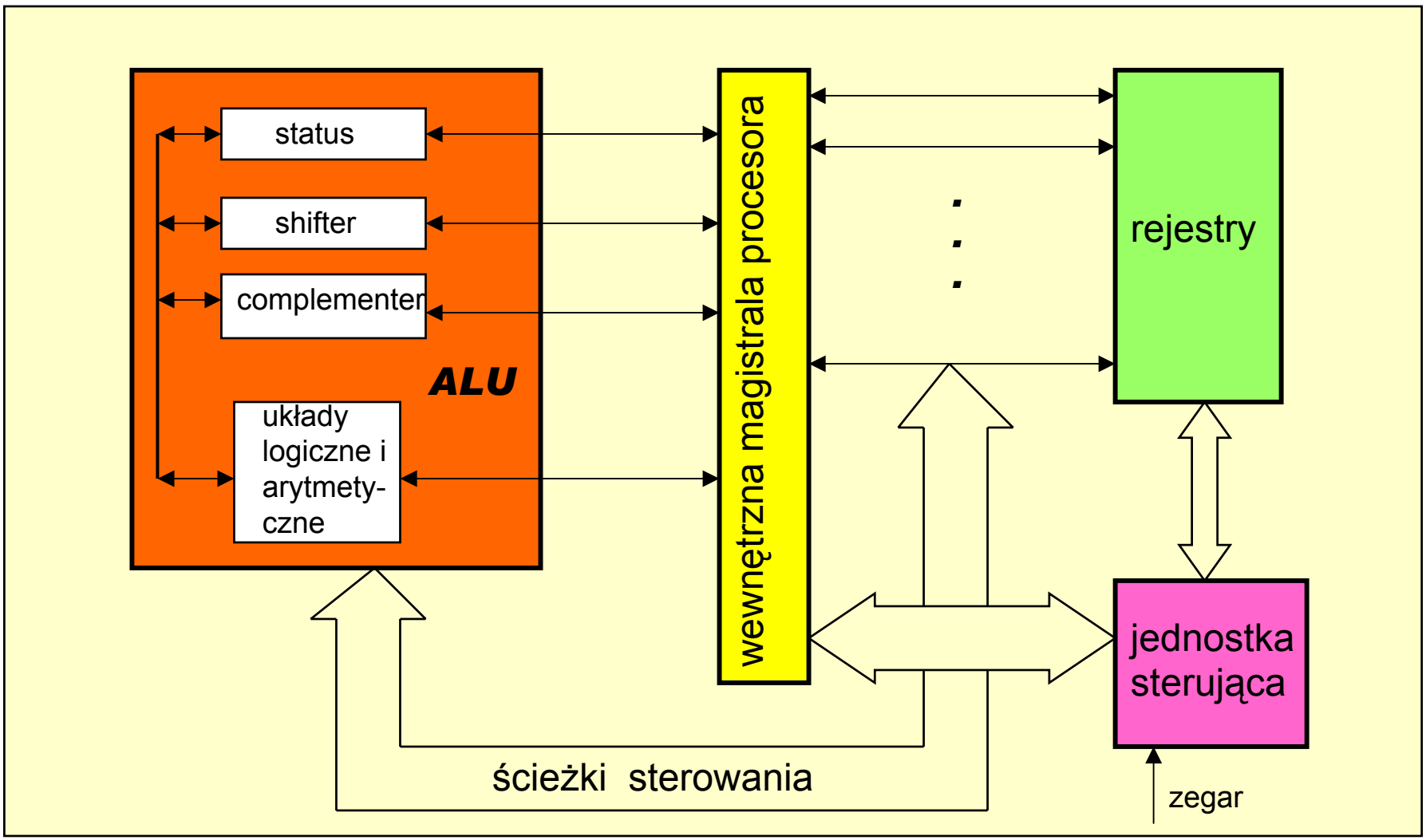
Działanie komputera



Struktura komputera



CPU - struktura wewnętrzna



Rejestry procesora

- Rejestry widoczne dla użytkownika (asembler)
  - Rejestry danych
    - rejestry do operacji zmiennoprzecinkowych i całkowitych
  - Rejestry adresów
    - rejestr indeksowy (dodanie indeksu do adresu)
    - wskaźnik segmentowy (segmentacja)
    - wskaźnik stosu (push, pop)
  - Rejestry znaczników (ang. condition codes)
- Rejestry sterowania i stanu (niewidoczne)
  - MAR, MBR, I/O AR, I/O BR, PC, IR, PSW

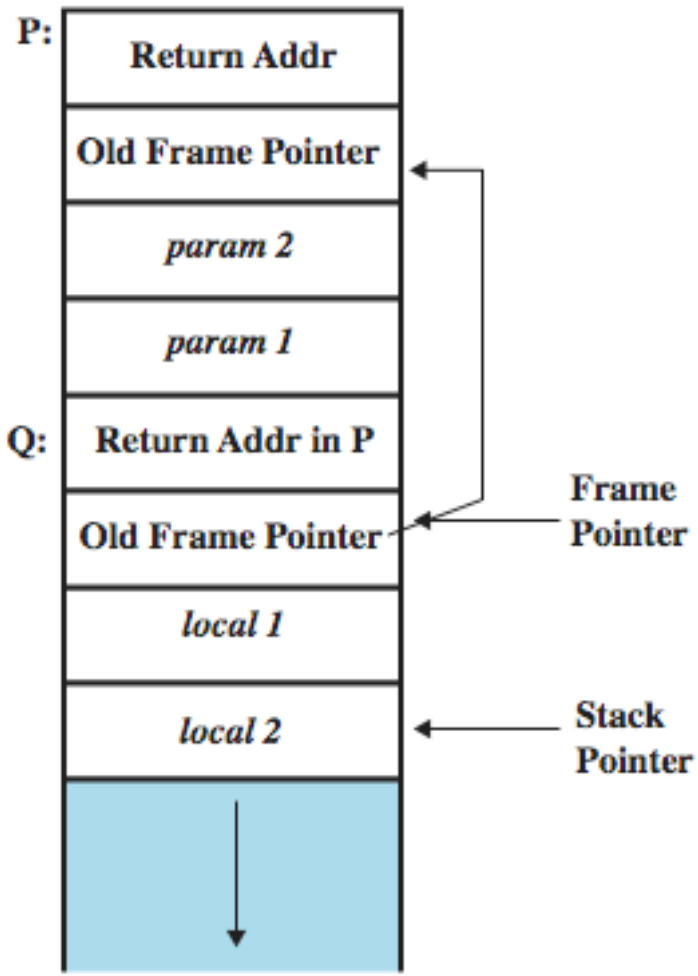
Rejestry Pentium

- Arytmetyka U2
- Ogólnego przeznaczenia – 8 rejestrów po 32b używanych przez wszystkie rozkazy (E{A|B|C|D}X, ESP, EBP, ESI, EDI)
  - EAX – akumulatorowy; EBX- bazowy, ECX- licznika, EDX – danych
    - {A|B|C|D}X – prawe 16b, {A|B|C|D}H – lewe 8b {A|B|C|D}X, {A|B|C|D}L – prawe 8b {A|B|C|D}X
  - ESP – wskaźnik (stack pointer), EBP - wskaźnik bazowy (base pointer), ESI- źródłowy, EDI -przeznaczeniowy
    - {S|D}I prawe 16b E{S|D}I
  - Prz. przeniesienie zawartości ESP do EBP, odjęcie 8 i zachowanie wyniku w ESP:
    8048375: 89 e5 mov ebp, esp
    8048377: 83ec 08 sub esp, 0x8
- EIP - wskaźnik rozkazu: adres bieżącego rozkazu
  - IP – prawe 16b

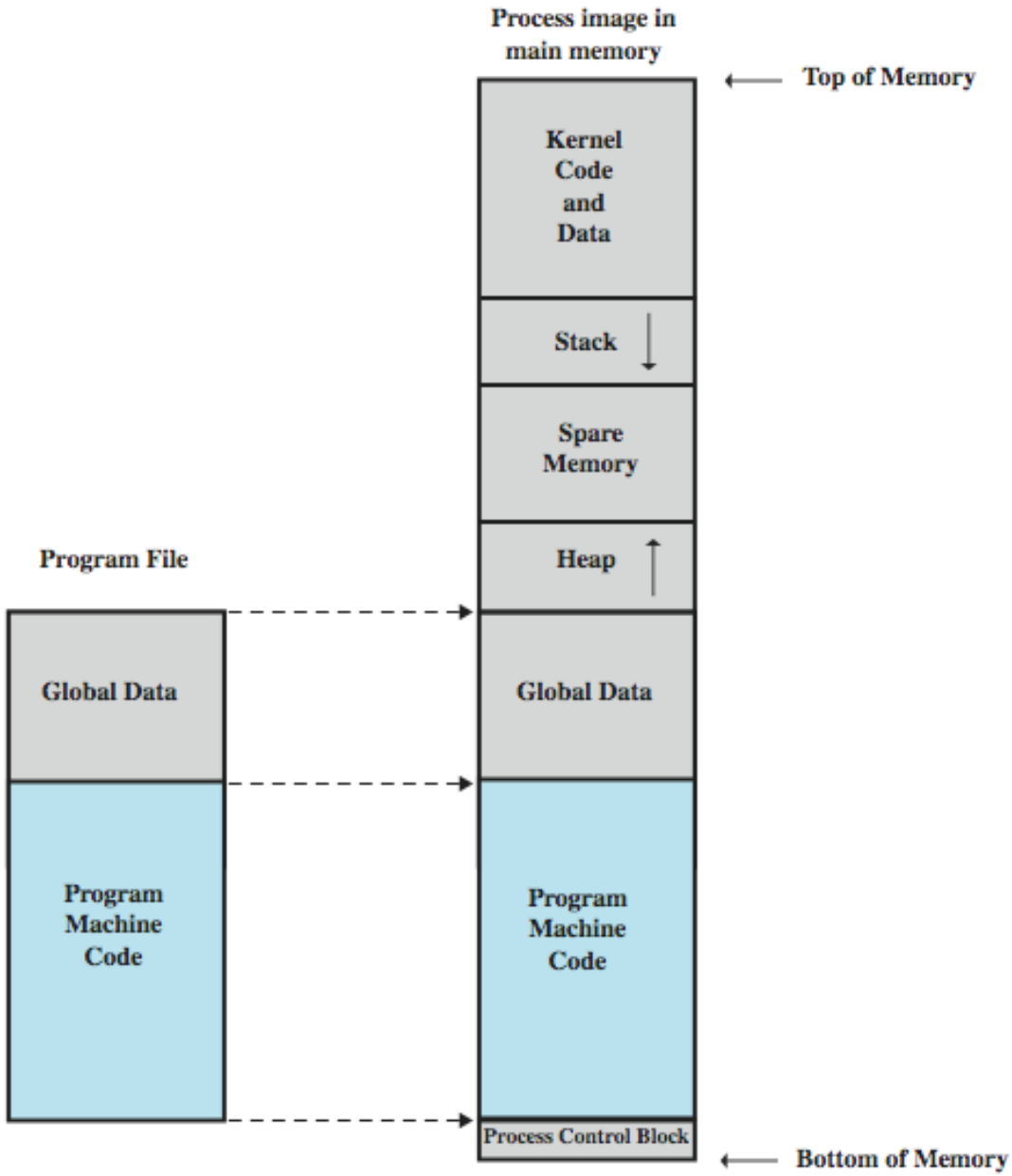
Rejestry Pentium (c.d.)

- Segmentowe – 6 rejestrów po 16b zawierających selektory segmentu (CS, SS, DS, ES, FS, GS)
  - segmenty text (kod), data i bss (zainicjalizowane i nie zmienne globalne), heap (sterta), stack (stos)
  - CS – adres początkowy segementu kodu programu (CS:IP-następny rozkaz)
  - SS – adres początkowy stosu programu (SS:SP – bieżące słowo na stosie)
  - DS – adres początkowy segmentu danych
- Znacznik stanu – rejestr EFLAGS, kody warunku i stan CPU
- Rejestry przeznaczone do współpracy z jednostką zmiennoprzecinkową
  - numeryczne – 8 po 80b rejestrów liczb zmiennopozycyjnych
  - sterowania – 16b rejestr sterujący pracą jednostki zmiennopozycyjnej
  - stanu – 16b rejestr opisujący bieżący stan jednostki zmiennopozycyjnej
  - słowo wyróżników – 16b rejestr umożliwiający sprawdzenie zawartości rejestru numerycznego

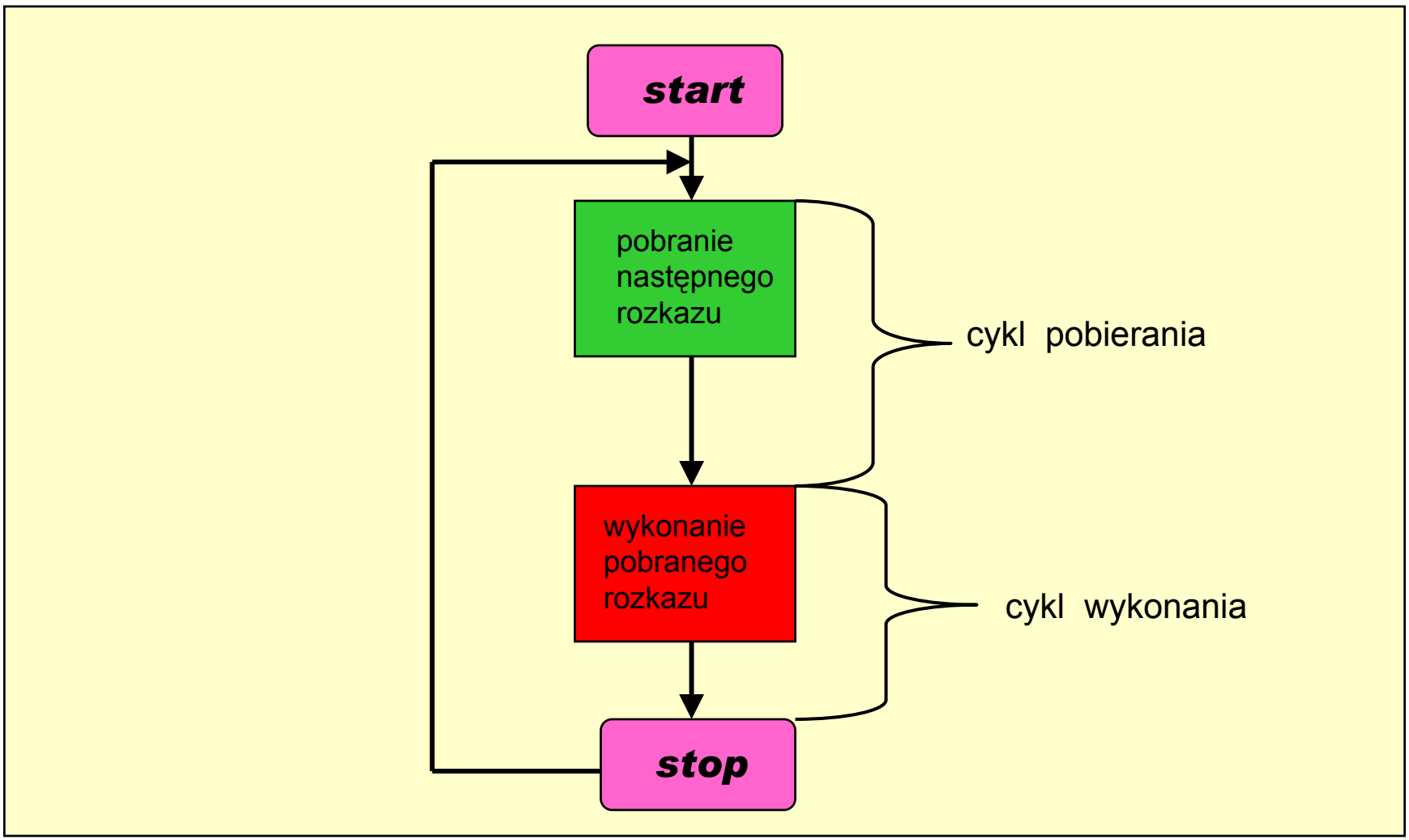
Ramki na stosie



Proces w pamięci



Podstawowy cykl rozkazu



Cykl pobierania

- licznik rozkazów (PC) zawiera adres następnego rozkazu do pobrania
- procesor pobiera rozkaz z pamięci z pod adresu wskazanego w PC
- zwiększa PC
  - o ile nie otrzyma innego polecenia
- rozkaz jest ładowany do rejestru rozkazu (IR)
- procesor interpretuje(dekoduje) rozkaz i przeprowadza wymagane działania

Cykl wykonywania

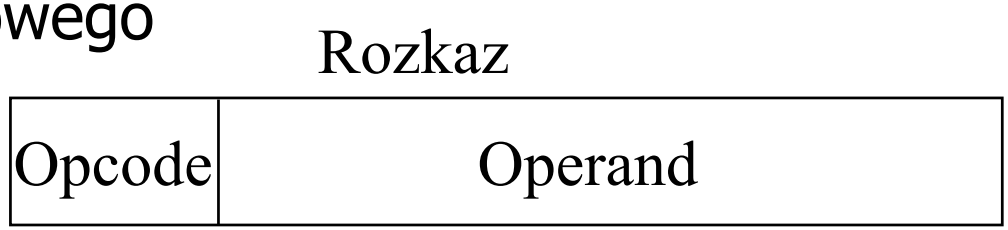
- procesor-pamięć
  - dane przenoszone między CPU i pamięcią główną
- procesor-we/wy
  - dane przenoszone między CPU i modułem we/wy
- przetwarzanie danych
  - wykonywanie opearcji logicznych i arytmetycznych na danych
- sterowanie
  - zmiana kolejności wykonania rozkazów (np. skok)
- kombinacja powyższych

Tryby adresowania

- natychmiastowy (ang. immediate)
- bezpośredni (ang. direct)
- pośredni (ang. indirect)
- rejestrowy (ang. register)
- rejestrowy pośredni (ang. register indirect)
- z przesunieciem (indeksowanie) (ang. displacement (indexed))
- stosowy (ang. stack)

Adresowanie natychmiastowe

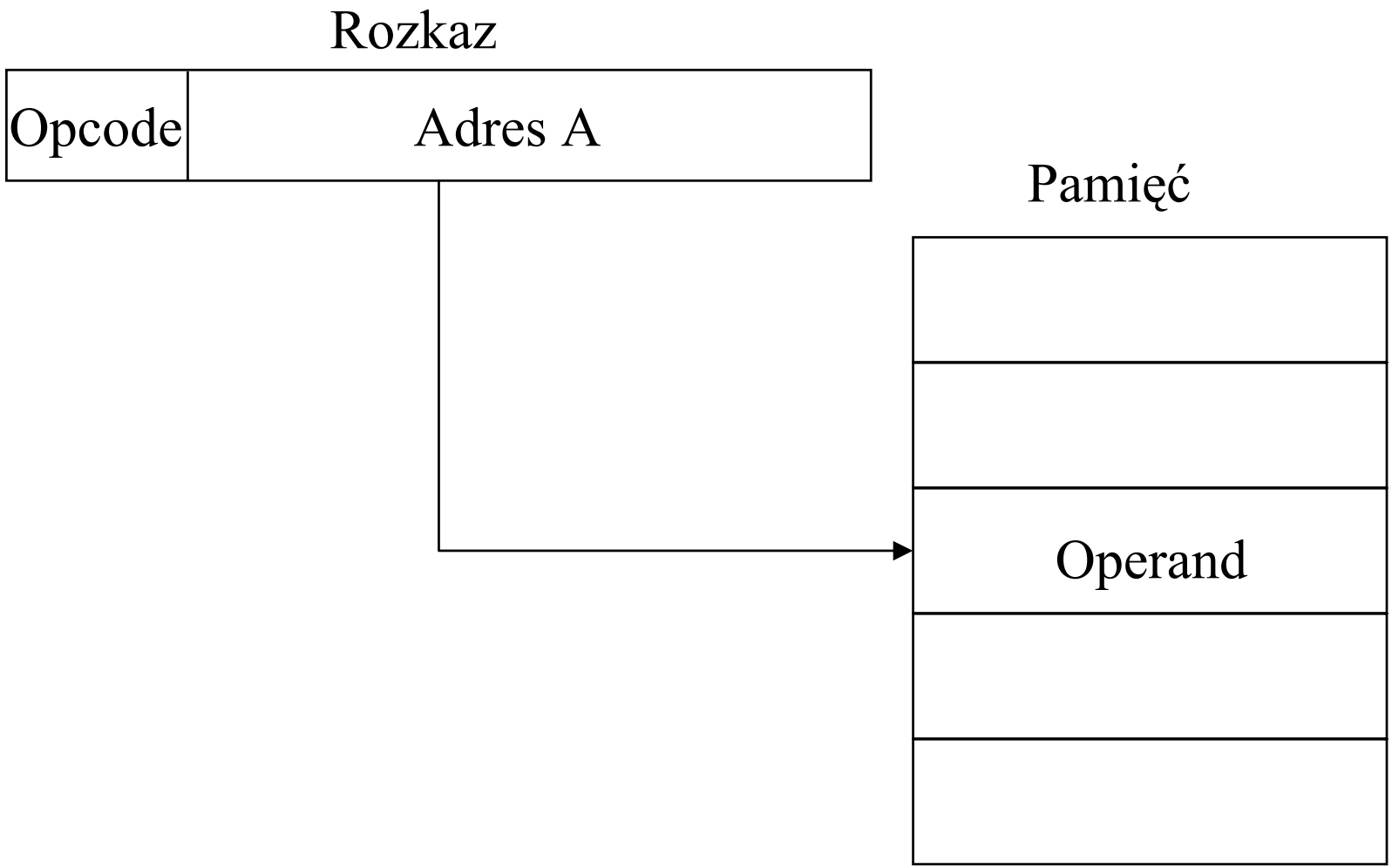
- Argument (ang. operand) jest częścią rozkazu
- A – zawartość pola adresowego w rozkazie
- Operand = A
  - np. ADD 5
  - dodaj 5 do zawartości akumulatora
  - 5 jest argumentem
- Nie ma odniesienia do pamięci w celu pobrania argumentu
- Zaoszczędzony jeden cykl pamięci
- Wielkość operandu ograniczona przez rozmiar pola adresowego



Adresowanie bezpośrednie

- Pole adresowe zawiera adres operandu
- EA - efektywny adres (ang. effective address) lokacji zawierający odniesiony argument
- EA = A
  - np. ADD A
  - dodaj zawartość komórki A do akumulatora
  - pod adresem A znajduje się operand
- Jedno odniesienie do pamięci
- Nie są potrzebne dodatkowe obliczenia
- Zakres adresacji ograniczony przez wielkość pola adresowego (słowo - opcode)

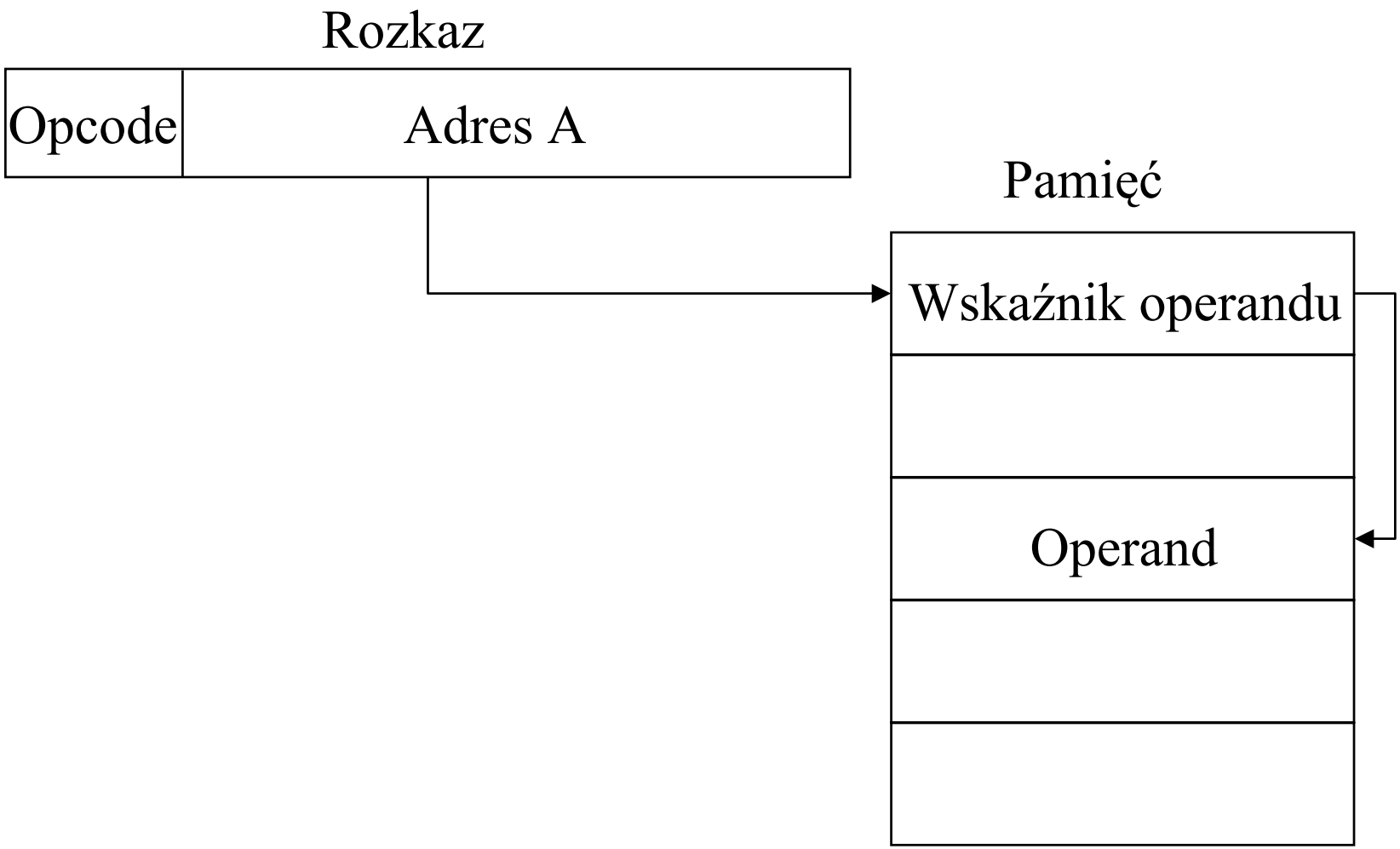
Adresowanie bezpośrednie (c.d.)



Adresowanie pośrednie

- Pole adresowe odnosi się do słowa w pamięci, które zawiera pełnej długości adres argumentu
- (X) – zawartość lokacji X (rejestr lub adres pamięci)
- EA = (A)
  - znajdź A, znajdź adres (A) pod którym jest operand
- np. ADD (A)
  - dodaj zawarość komórki pamięci wyznaczonej przez zawartość pod adresem A do akumulatora
- Większa przestrzeń adresowa: 2^n gdzie n = długość słowa
- Może być zagnieżdżone (ang. nested, multilevel, cascaded)
  - np. EA = (((A)))
    - Zad. Narysuj diagram
- Wiele odniesień do pamięci głównej w celu pobrania argumentu (dlatego wolne)

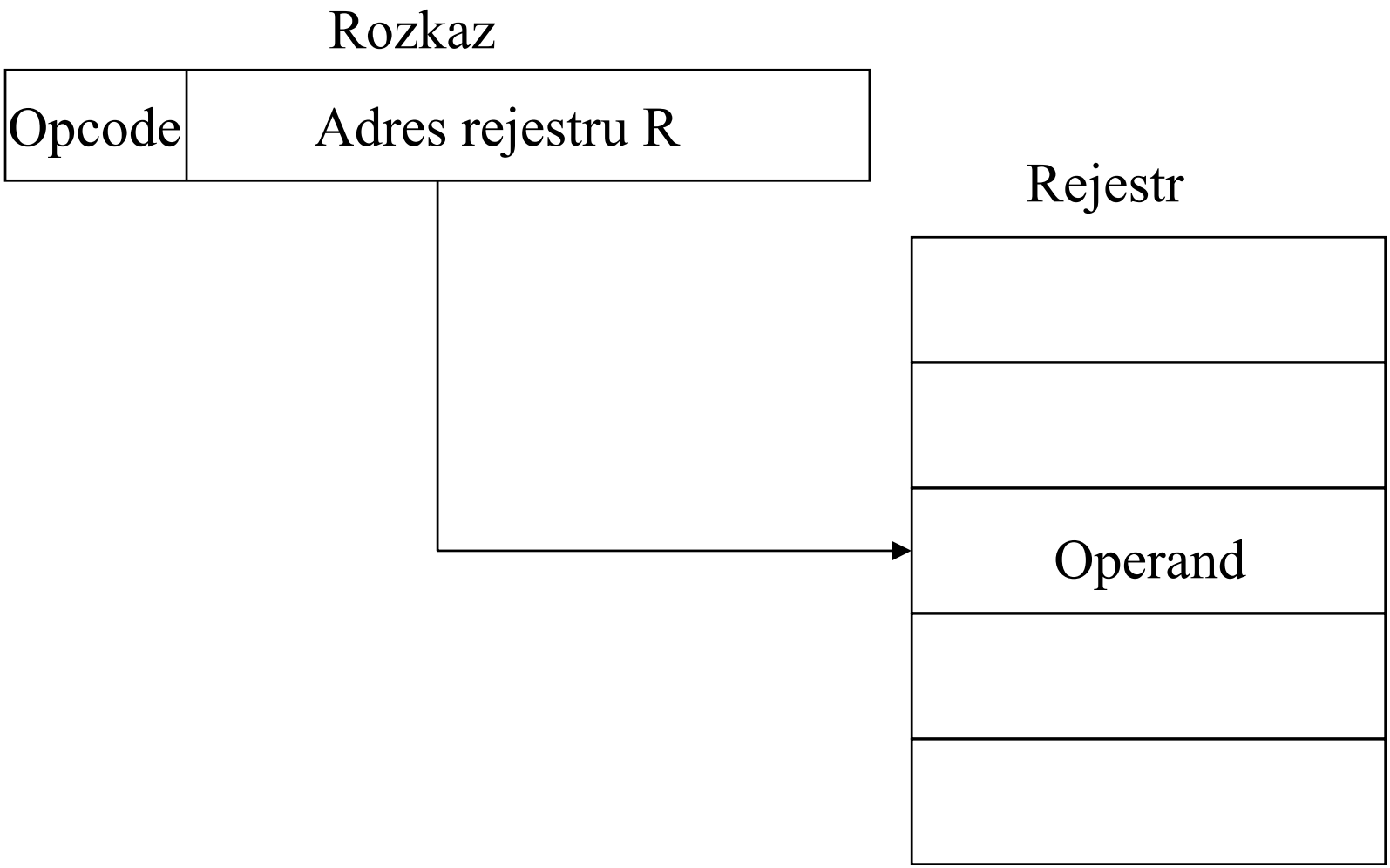
Adresowanie pośrednie (c.d.)



Adresowanie rejestrowe

- Operand znajduje się w rejestrze określonym w polu adresowym
- EA = R
- Ograniczona liczba rejestrów (32)
- Małe pole adresowe (zwykle 3 do 5 bitów)
  - krótsze rozkazy i czas pobrania z rejestru
- Ograniczona przestrzeń adresowa

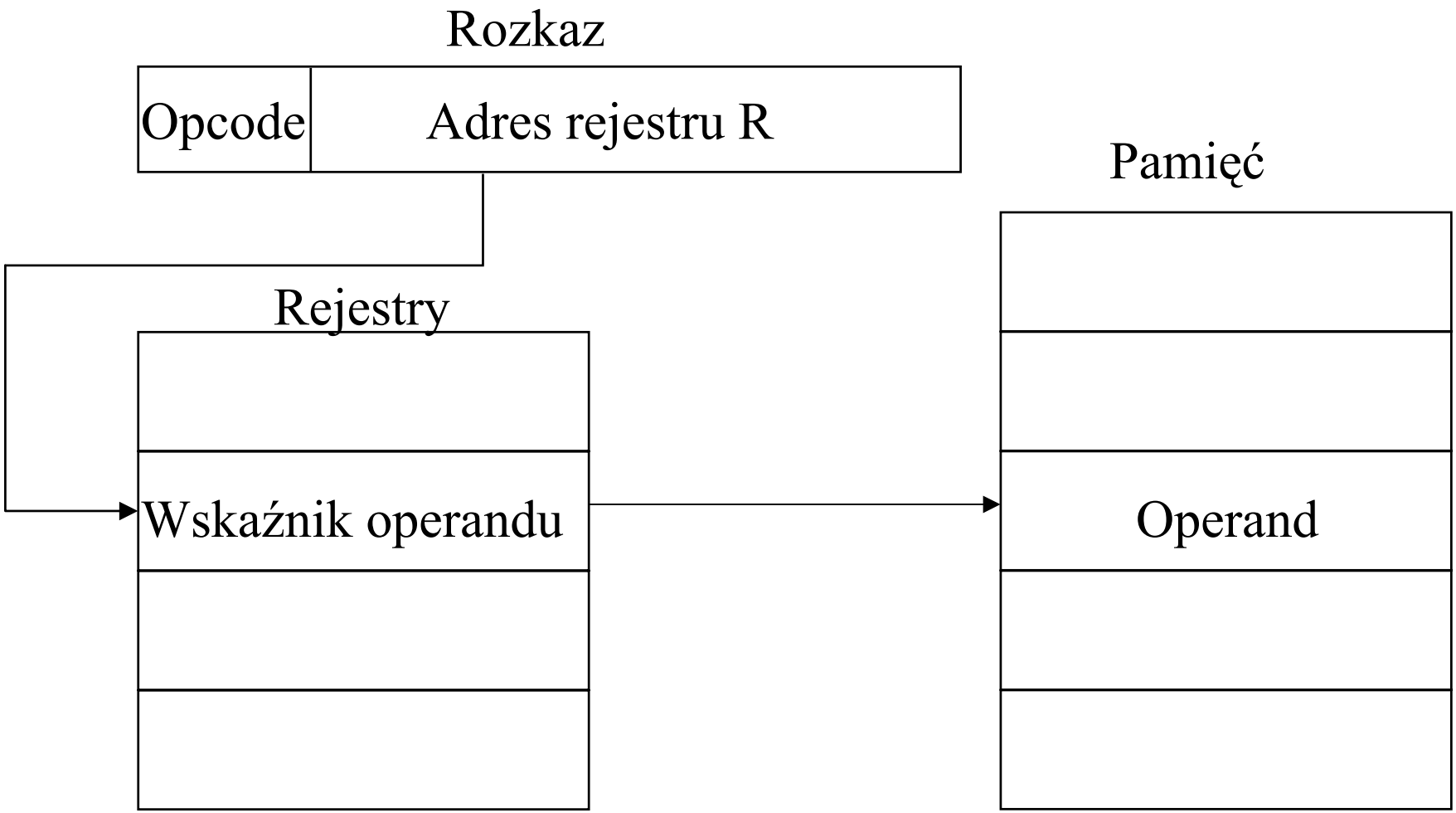
Adresowanie rejestrowe (c.d.)



Pośrednie adresowanie rejestrowe

- Adres w rejestrze odnosi się do słowa w pamięci, które zawiera adres operandu
- EA = (R)
- Operand jest w komórce pamięci wskazanej przez adres zawarty w rejestrze R
- Duża przestrzeń adresowa (2^n)
- O jedno odniesienie do pamięci mniej niż przy adresowaniu pośrednim

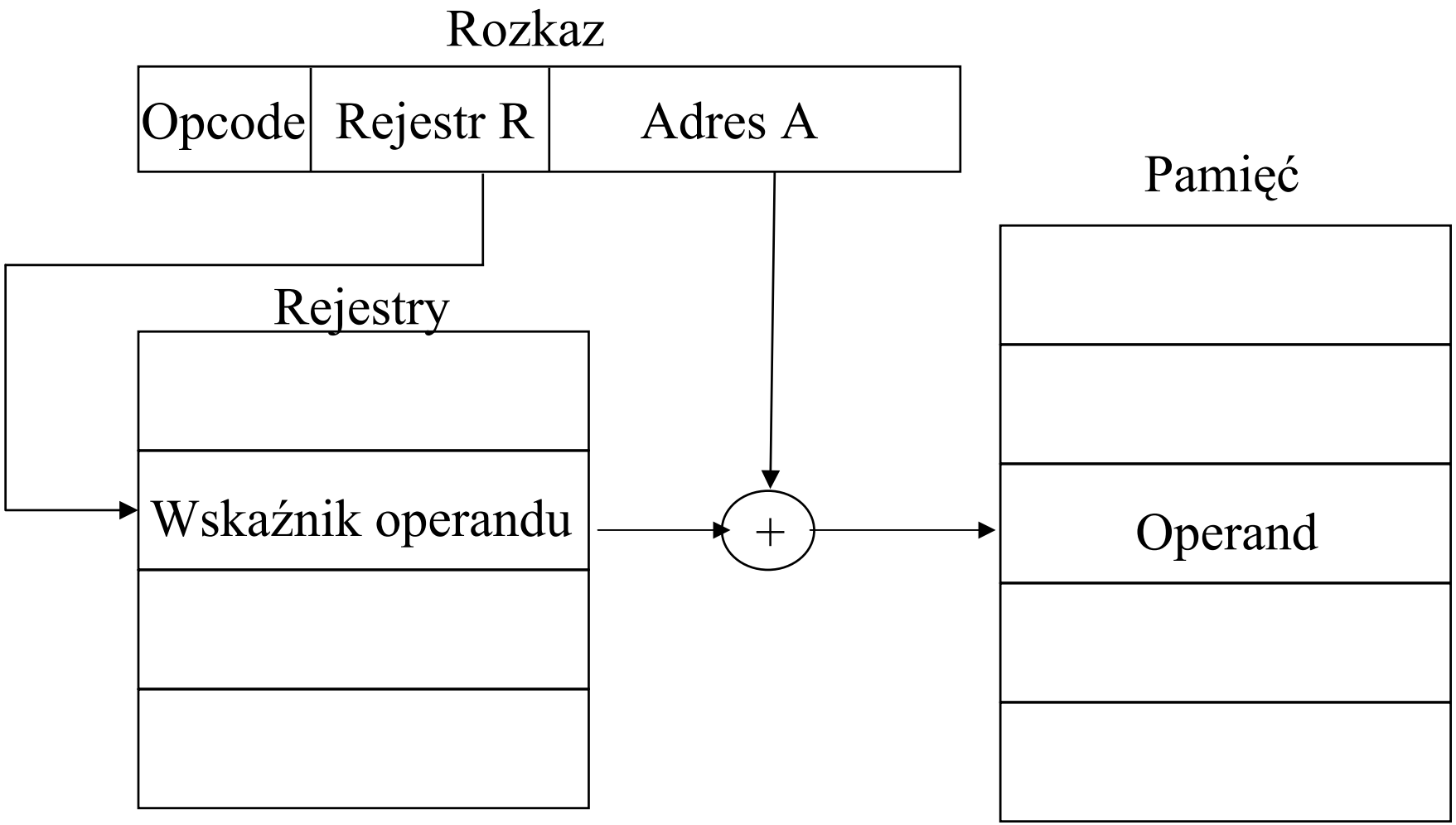
Pośrednie adresowanie rejestrowe (c.d.)



Adresowanie z przesunięciem

- EA = A + (R)
- Pole adresowe zawiera dwie wartości
  - A = wartość bazowa
  - R = rejestr zawierający przesunięcie
  - lub odwrotnie
- Najczęstsze zastosowania
  - adresowanie względne (ang. relative addressing)
  - adresowanie z rejestrem podstawowym (ang. base-register addressing)
  - indeksowanie (ang. indexed addressing)

Adresowanie z przesunięciem (c.d.)



Adresowanie z przesunięciem (c.d.)

- adresowanie względne
  - R = licznik programu, PC
  - EA = A + (PC)
  - zgodność z zasadą lokalności odniesień (np. w pamięci podręcznej)
- adresowanie z rejestrem podstawowym
  - A zawiera przesunięcie a R adres bazowy
  - R może zawierać adres bezpośredni lub pośredni
  - np. rejestry segmentowe w 80x86
  - zgodność z zasadą lokalności odniesień

Adresowanie z przesunięciem (c.d.)

- indeksowanie
  - A = adres bazowy, R = przesuniecie
  - EA = A + R
  - adresowanie dobre dla tablic: EA = A + R; R++
  - autoindeksowanie: EA = A + (R); (R) <- (R) + 1
- kombinacje
  - indeksowanie wtórne (ang. postindex): EA = (A) + (R)
  - indeksowanie wstępne (ang. preindex): EA= (A + (R))

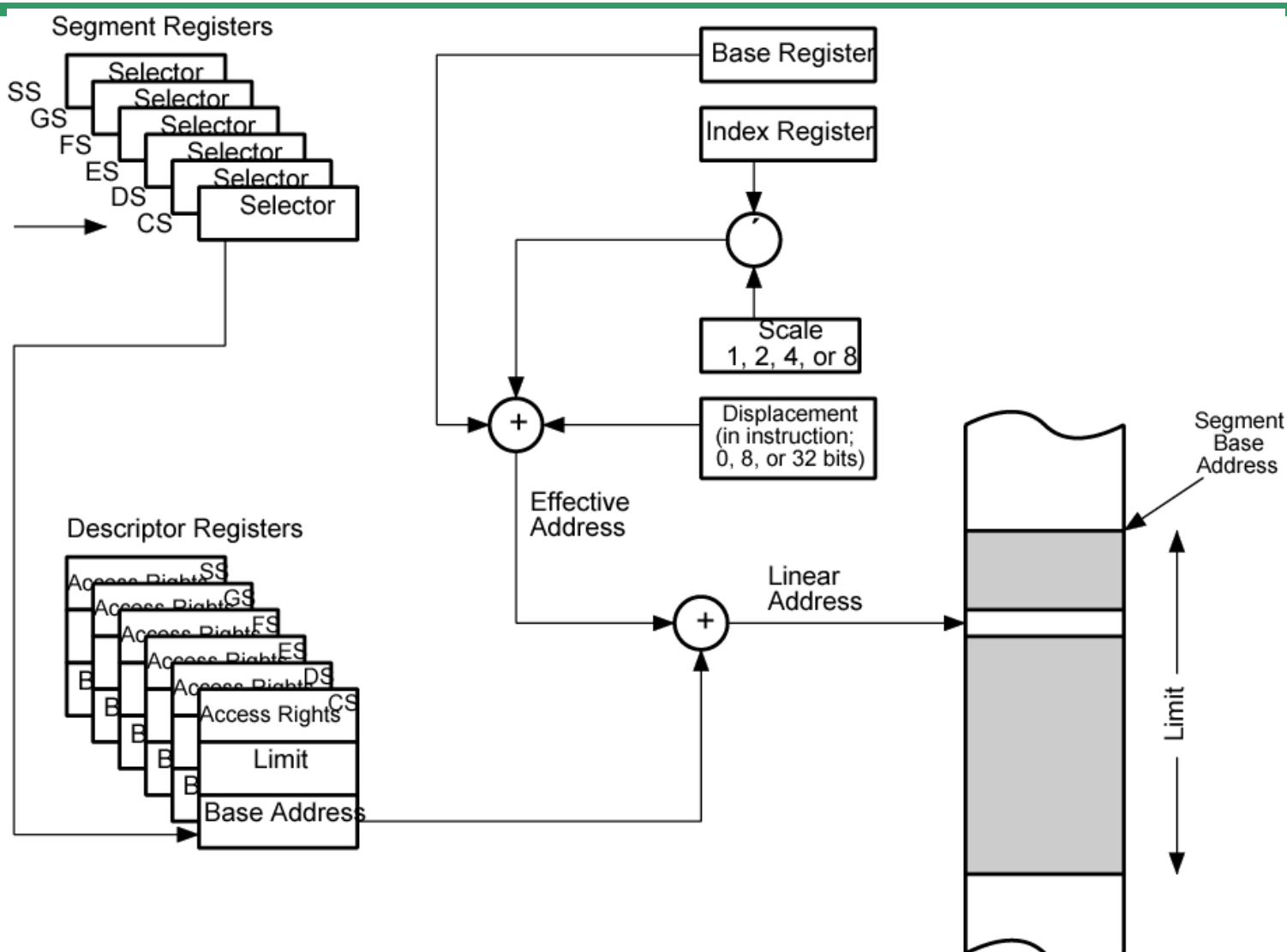
Adresowanie stosowe

- Operand znajduje się na wierzchołku stosu wskazanym przez rejestr
  - np. ADD
  - Usuń (POP) dwa wierzchołkowe elementy stosu i dodaj

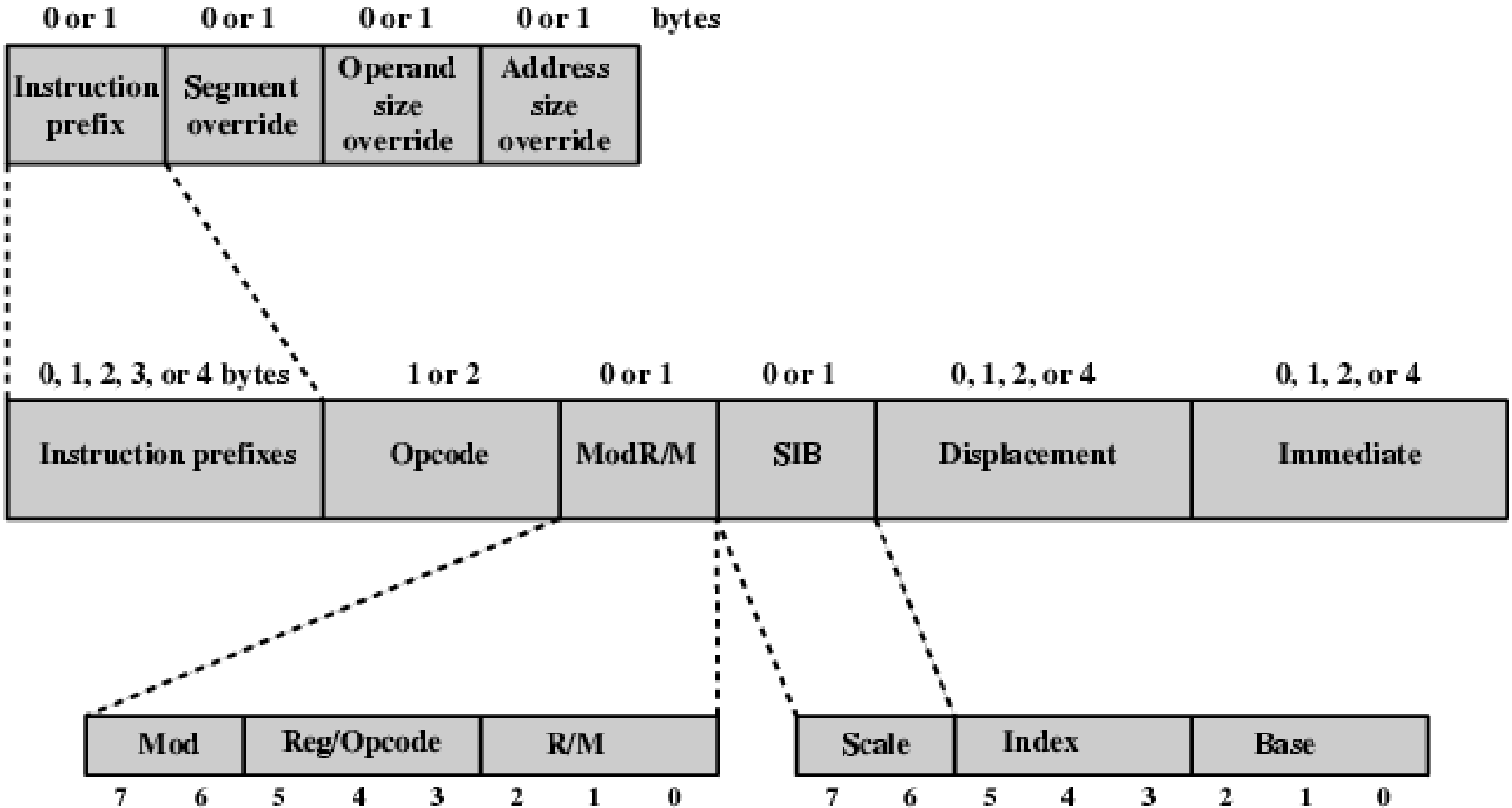
Tryby adresowania w Pentium

- Mechanizm translacji adresu tworzy adres wirtualny (efektywny), który jest adresem względnym w segmencie
  - adresu początkowego segmentu (SR) + przesunięcie = adres liniowy (LA)
  - LA przechodzi przez mechanizm translacji strony
  - SR – rejestr segmentowy, B – rejestr bazowy, I – rejestr indeksowy, S – czynnik skalowania
- Tryby adresowania
  - natychmiastowy: Operand = A
  - rejestrowy: LA = R
  - z przesunięciem: LA = (SR) + A
  - z rejestrem podstawowym: LA = (SR) + (B)
  - z rejestrem podstawowym i przesunieciem: LA = (SR) + (B) + A
  - skalowane indeksowanie z przesunieciem: LA = (SR) + (I)*S + A
  - z rejestrem podstawowym ze skalowanym indeksowaniem i z przesunięciem: LA = (SR) + (I)*S + (B) + A
  - względny: LA = (PC) + A

Obliczanie trybu adresowania w Pentium



Format rozkazu Pentium



Pentium - typy rozkazów

- przenoszenie danych
  - MOV dest,src ; kopiowanie z src do dest
  - LEA dest,src ; kopiowanie efektywnego adresu (load effective address) src do dest
  - PUSH src ; wstaw src na stos
  - POP dest ; zbierz słowo ze stosu do dest
- arytmetyczne: ADD, SUB, MUL, IDIV
  - ADD/SUB src, dest ; dodaj/odejmij src od dest i zachowaj w dest
- logiczne: AND, BTS, SHL, SHR
  - AND/OR/XOR ; operacje logiczne na src i dest – wynik w dest
  - CMP val1, val2 ; porównaj val1 i val2 – wynik w EFLAGS
- przekazywanie sterowania: JMP, CALL, INT, RET
  - JMP/JZ/JNZ addr ; skok /jeśli zero / nie zero pod addr
  - CALL addr ; wywołaj funkcję z pod addr
  - LEAVE ; wyczyść ramkę stosu przed powrotem z funkcji
  - RET ; powrót z funkcji
  - NOP ; no-operation

Pentium - typy rozkazów

- zwiększ/zmniejsz o 1: INC/DEC reg
- operacje łańcuchowe: MOVS
- wspieranie języka wysokiego poziomu: ENTER
- sterowanie za pomocą znaczników: STC
- rejestr segmentowy: LDS; ochrona: LSL
- sterowanie systemowe: HLT, WAIT
- zarządzanie pamięcią podręczną: INVD

- http://www.jegerlehner.ch/intel/

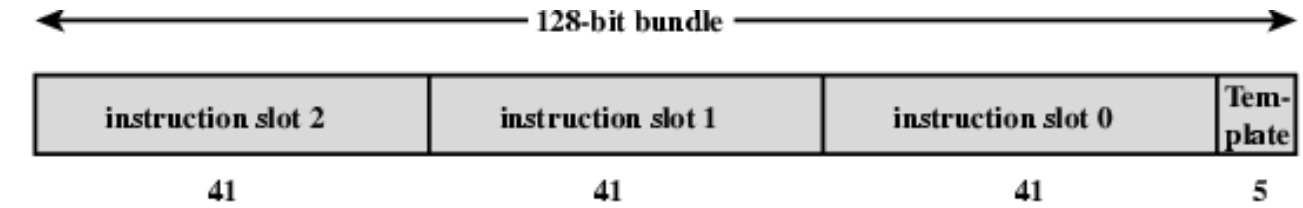
IA-64

- IA-32 – mała liczba rejestrów utrudnia generowanie przez kompilator efektywnego kodu maszynowego
  - uzależnienia typu WAR (write after read) – wyniki gdzieś trzeba zapisać
- IA-64
  - 128 64b-rejestrów ogólnych (pozwalają na redukowanie odwołań do pamięci), 128 82b-rejestrów zmiennopozycyjnych i graficznych, 64 1b-rejestrów predykcyjnych, 8 rejestrów skoku (architektura Itanium)
  - kompilator ma do dyspozycji więcej rejestrów (niż RISC) dlatego potrafi wygenerować bardziej efektywny kod maszynowy
  - jawne zrównoleglenie rozkazów, w skrócie EPIC (ang. Explicity Paralell Instruction Computing)
  - http://www.nasm.us/pub/nasm/releasebuilds/2.08.01/linux/ (-f elf64)
- Najważniejsze techniki IA-64
  - szeregowanie rozkazów i predykcja
  - spekulatywne ładowanie rozkazów z pamięci (tj. z programu) zanim będą one wykonywane przez procesor

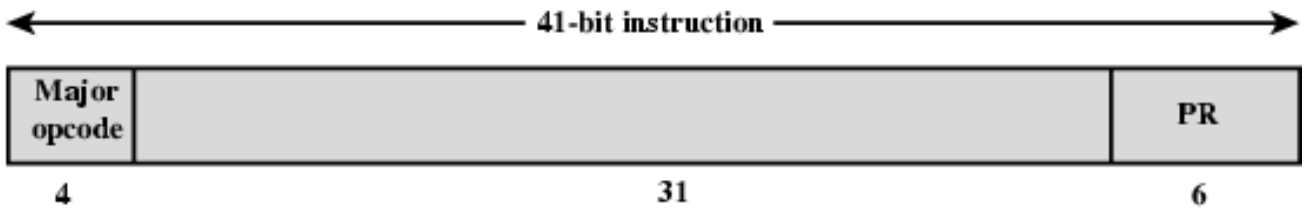
IA-64 – szeregowanie rozkazów

- Rozkazy procesora zorganizowane są w 128b paczki (wiązki) (ang. bundles)
  - każda paczka składa się z trzech 41b rozkazów zwanych sylabami (ang. syllables) i 5b szablonu (ang. template)
    - szablon zawiera infomację o jednostkach funkcjonalnych (wykonujących) używanych przez paczkę, których wykonanie może zostać zrównoleglone przez procesor
      - operacje arytmetyczno-logiczne (I), operacje zmiennopozycyjne (F), operacje pamięciowe (M), skoki (B)
      - procesor przegląda paczki w celu stwierdzenia, które z nich można zrównoleglić
      - kompilator może zmienić kolejność rozkazów (umieścić je w sąsiednich paczkach)
- 100 formatów rozkazów
  - 5b szablon paczki, 4b kod główny operacji, 10b modyfikacji
- Prz. typowy rozkaz arytmetyczno-logiczny: ADD
  - dodaje zawartość dwóch rejestrów i zapisuje wynik w trzecim rejestrze
  - grupa operacji, typ operacji, operandy rozkazu (3 rejestry), numer rejestru predykcyjnego

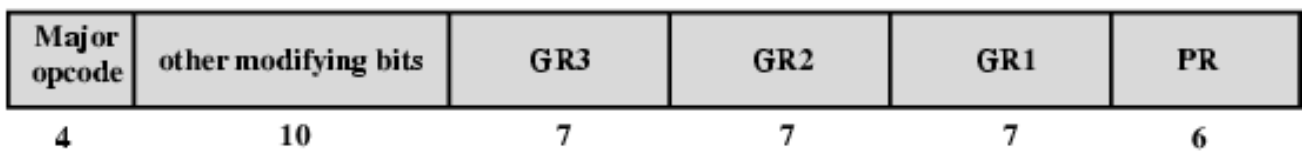
Formaty rozkazu IA-64



(a) IA-64 bundle



(b) General IA-64 instruction format



(c) Typical IA-64 instruction format

PR = Predicate register
GR = General or floating-point register

IA-64 – predykcja

- Predykcja pozwala na redukcję skoków warunkowych
- Instrukcja if języka C
  - if (R1 = = 0)
    R2 = R3
- Tradycyjna (IA-32) sekwencja asemblerowa
  - CMP R1, 0
    BNE L1
    MOV R2, R3
    L1: ....
- Rozkaz predykcyjny (uwarunkowany)
  - CMOVZ R2, R3, R1
  - Rozkaz CMOVZ bada zawartość rejestru R1 i jeżeli jest on równy 0 to następuje kopiowanie zawartości rejestru R3 do R2, w przeciwnym razie NOOP

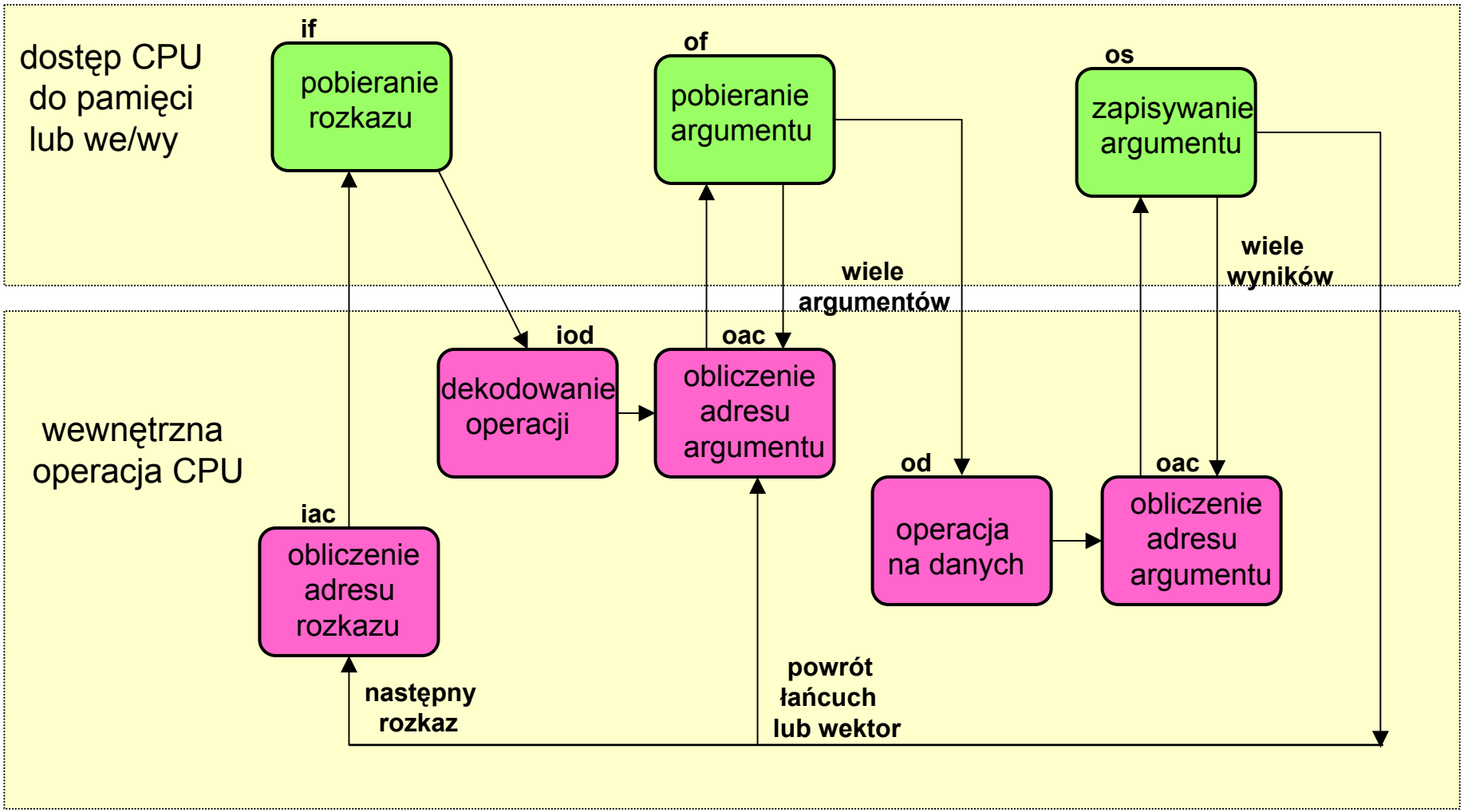
Stany cyklu rozkazu (1)

- obliczanie adresu rozkazu (iac - instruction address calculation) - adres następnej instrukcji
- pobieranie rozkazu (if - instruction fetch) - wczytanie z pamięci do CPU
- dekodowanie operacji rozkazu (iod - instruction operation decoding)
- obliczanie adresu argumentu (oac - operand address calculation)

Stany cyklu rozkazu (2)

- pobieranie argumentu (of - operand fetch)
- operacja na danych (do - data operation) - operacja na danych
- przechowanie argumentu (os - operand store) - zapisanie wyniku

Graf stanów cyklu rozkazu



We/wy

- Procesor inicjuje operacje we/wy oraz może wymieniać dane z modułem we/wy podobnie jak z pamięcią główna

- Wymiana danych pomiędzy we/wy i pamięcią może też mieć miejsce bez udziału CPU (DMA),
  - procesor zrzuca odpowiedzialność za we/wy na moduł we/wy

Przerwania (ang. Interrupts)

- Mechanizm za pomocą, którego inne moduły (np. we/wy) mogą przerwać normalne przetwarzanie danych przez procesor
- programowe
  - np. przepełnienie arytmetyczne, dzielenie przez 0
- zegarowe
  - generowane przez wewnętrzny zegar procesora
  - umożliwia wywłaszczanie w systemie wielozadaniowym
- we/wy - od modułu we/wy
- sprzętowe
  - np. błąd parzystości pamięci

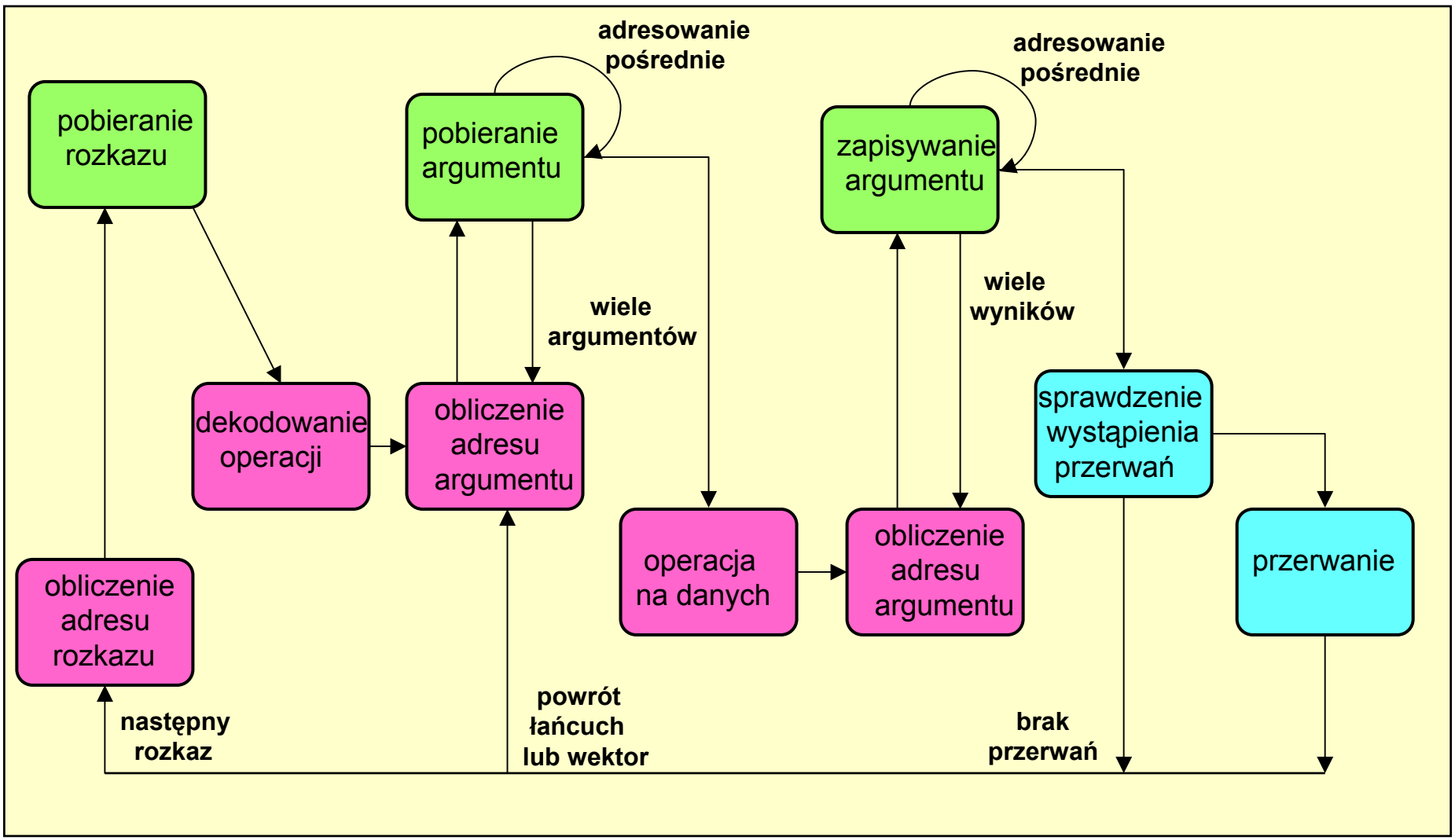
Idea stosu

- Mechanizm do sterowania wykonaniem wywołań procedur
- Struktura danych – lista typu LIFO (ang. last-in-first-out)
  - wskaźnik stosu -wierzchołek stosu
    - PUSH zwiększa wskaźnik o 1, POP zmniejsza wskaźnik o 1
  - podstawa stosu - adres pierwszego bloku stosu
  - granica stosu – adres ostatniego bloku stosu
- Stos rośnie od najwyższego adresu do najniższego
- Na stosie możemy umieszczać adresy powrotu wraz zawartościami rejestrów (parametrami)
  - PUSH – umieszczenie nowego elementu na wierzchołku stosu
  - operacja dwuargumentowa: wykonanie operacji na dwu wierzchołkowych elementach stosu następnie ich usunięcie i umieszczenie wyniku operacji na wierzchołku stosu

Cykl przerwania

- do cyklu rozkazu dodaje się cykl przerwania
- procesor sprawdza czy nastąpiło przerwanie
  - obecność sygnału przerwania
- jeśli nie ma przerwania pobiera nowy rozkaz
- jeśli następuje przerwanie to procesor:
  - zawiesza wykonanie bieżącego programu
  - zachowuje kontekst bieżącego programu
    - np. przechowanie PC, PSW i rejestrów na stosie systemowym
  - ustawia licznik rozkazów (PC) na adres programu obsługi przerwań (ang. interrupt handler routine)
  - obsługuje przerwanie
  - odtwarza kontekst i kontynuuje przerwany program

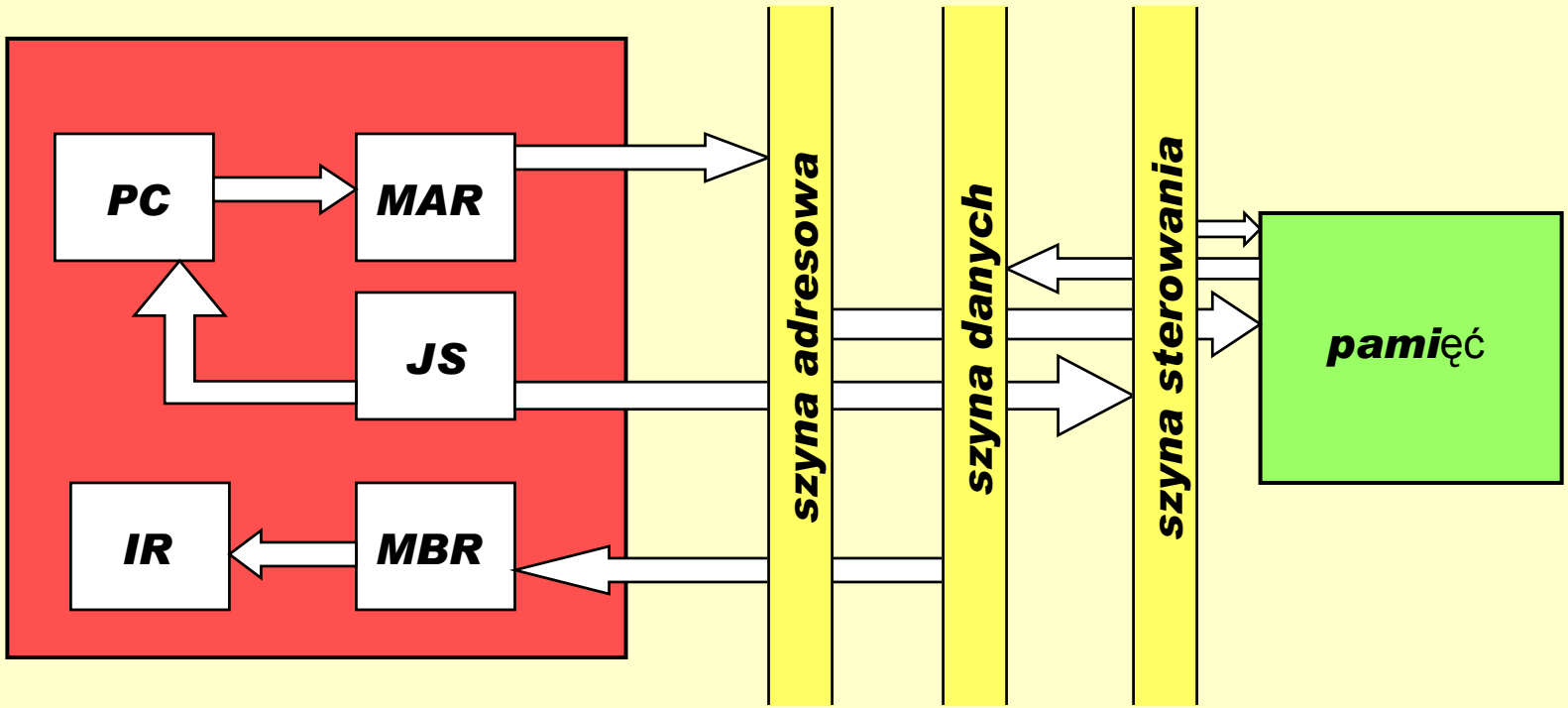
Graf stanów cyklu rozkazu z przerwaniami



Przykład - cykl pobierania

- Zależny od architektury CPU - założenia: MAR (rejestr adresowy), MBR (rejestr buforowy), PC (licznik programu), IR (rejestr rozkazu)
  - licznik PC zawiera adres rozkazu do pobrania
  - zawartość PC jest przenoszona do MAR
  - adres z MAR jest umieszczony na szynie adresowwej
  - jednostka sterująca zgłasza zapotrzebowanie na odczyt z pamięci
  - wynik umieszczony zostaje na szynie danych
  - dane te zostają skopiowane do MBR i IR
  - w tym czasie następuje przygotowanie do pobrania następnego rozkazu t.j.: PC:=PC+1

Schemat - cykl pobierania

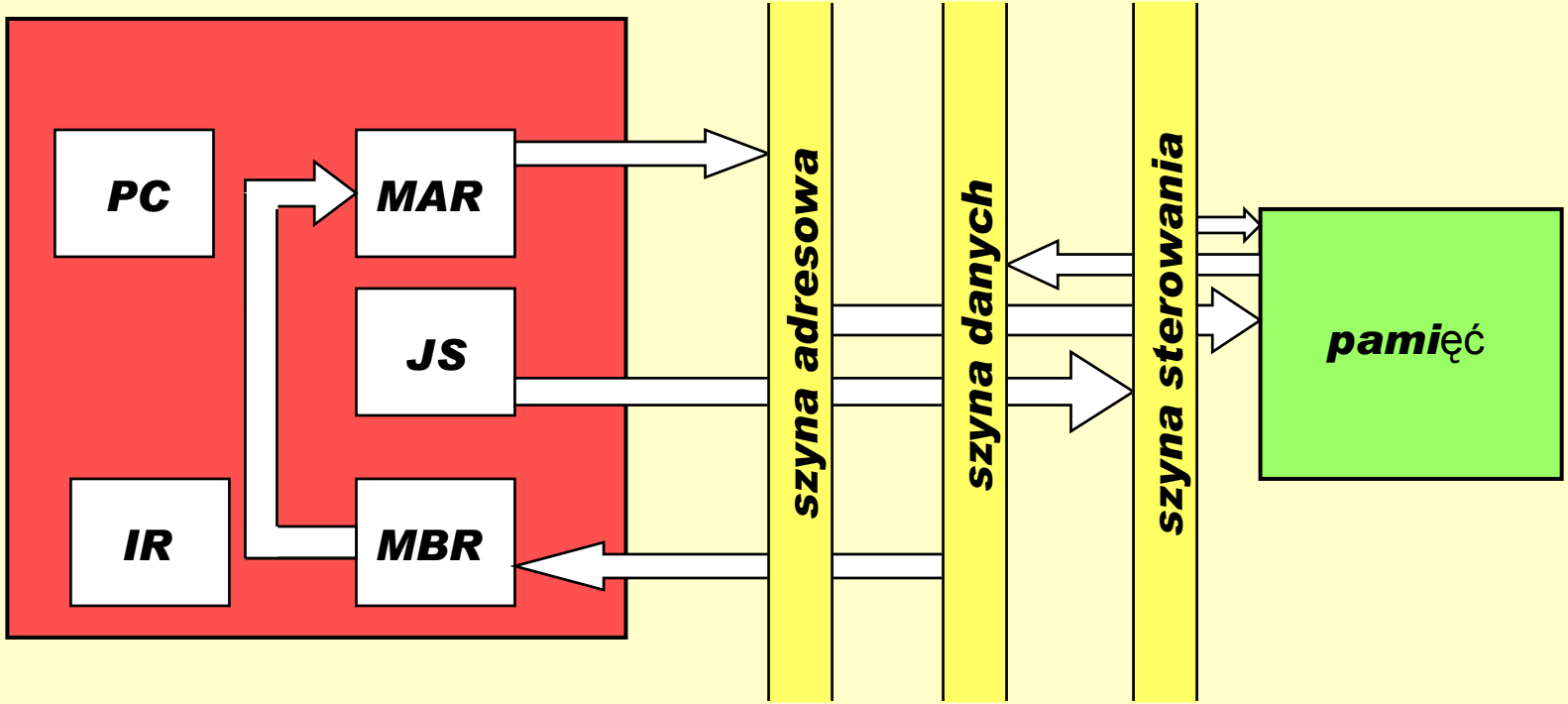


PC - licznik programu
IR - rejestr rozkazów
MAR - rejestr adresowy pamięci
MBR - rejestr buforowy pamięci
JS - jednostka sterująca

Przykład - cykl pośredni

- Gdy cykl pobierania jest zakończony jednostka sterująca bada typ adresowania rozkazu w IR
- Jeśli rozkaz w IR oznacza wykorzystanie adresowania pośredniego to ma miejsce cykl pośredni
  - bity z MBR zawierające odniesienie do adresu są kopiowane do MAR
  - jednostka sterująca zgłasza zapotrzebowanie na odczyt z pamięci
  - wynik (zawartość adresu w MAR) jest wpisany do MBR

Schemat - cykl rozkazu pośredni



PC - licznik programu
IR - rejestr rozkazów
MAR - rejestr adresowy pamięci
MBR - rejestr buforowy pamięci
JS - jednostka sterująca

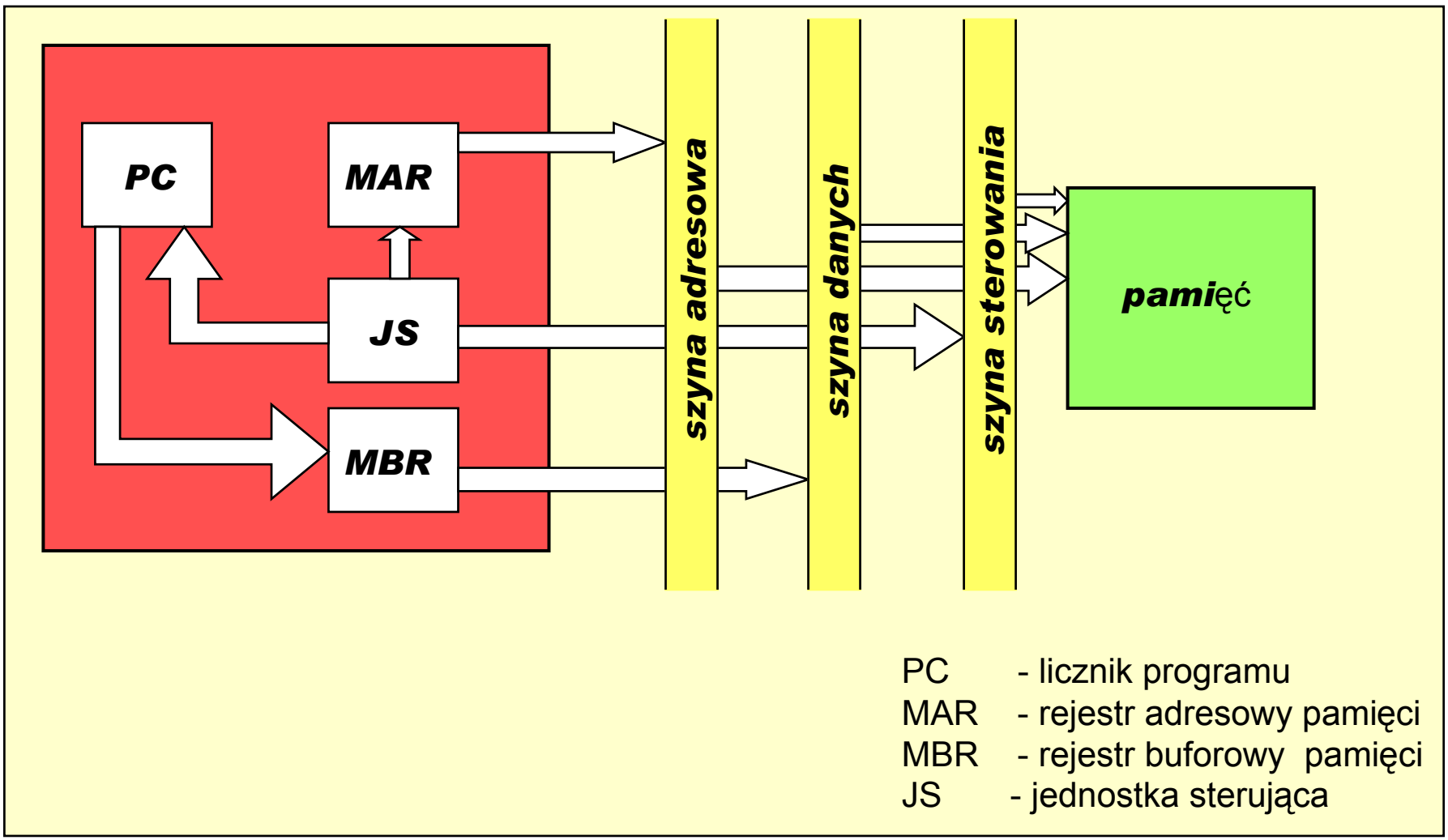
Przykład - cykl wykonania rozkazu

- Cykl wykonania rozkazu może przyjmować różnorodne postaci albowiem zależy od tego, który z całej listy rozkazów maszynowych znalazł się w IR
- Cykl rozkazu może więc realizować kombinacje
  - transmisję pomiędzy rejestrami
  - transmisję do/z pamięci
  - transmisję z/do modułu we/wy
  - wywołanie ALU

Przykład - cykl przerwania

- PC jest kopiowany do MBR
- jednostka sterująca ładuje do MAR zawartość specjalnej, zarezerwowanej do tego celu lokalizacji pamięci (np. wskaźnik stosu)
- MBR zostaje zapisany w pamięci
- do PC jest ładowany adres procedury przerwania (ang. interrupt handling routine)
- przejście do trybu pobierania
  - pobierany jest następny rozkaz (t.j. pierwszy rozkaz procedury obsługującej przerwanie)

Schemat - cykl przerwania

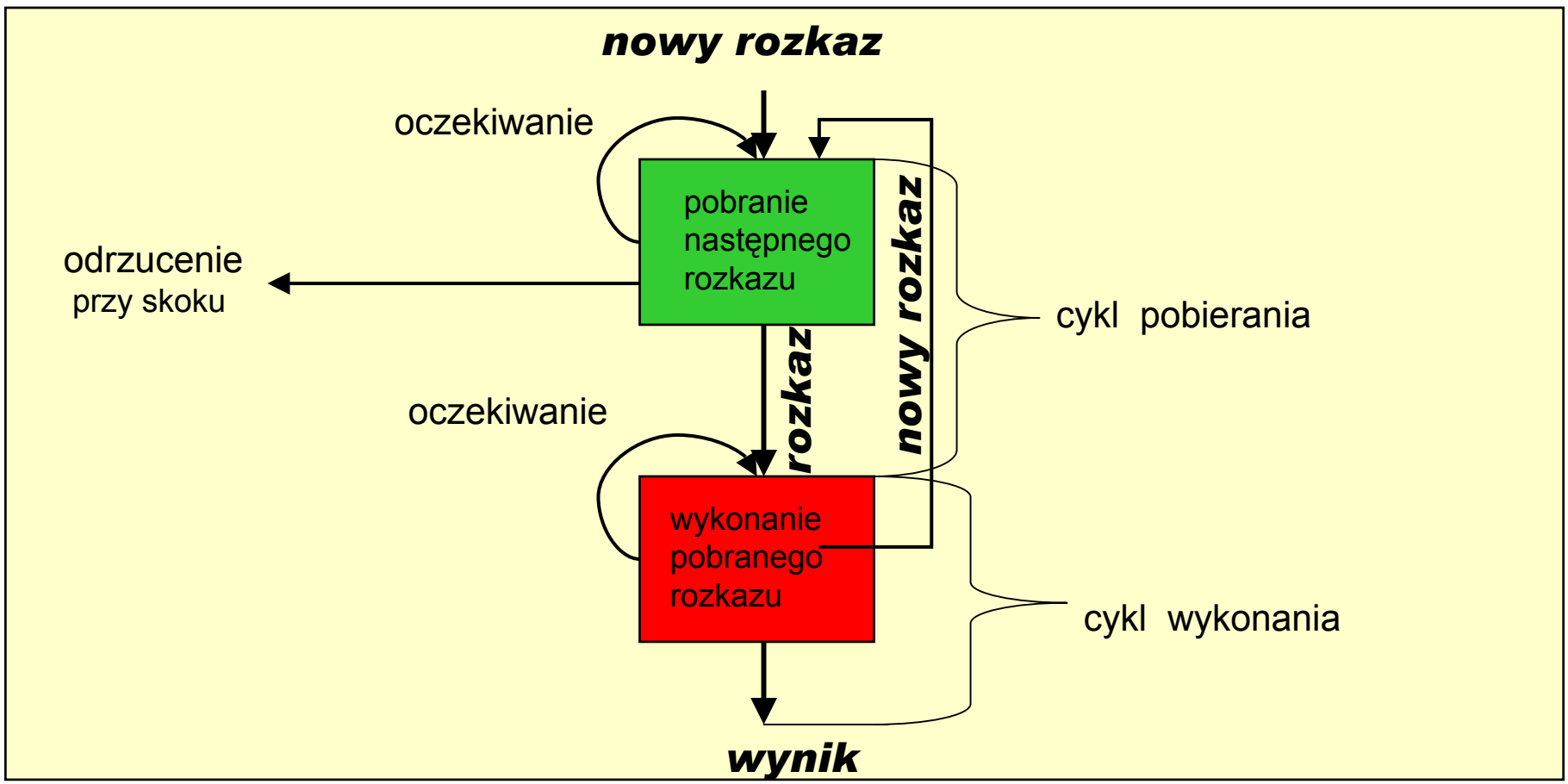


Wieloprogramowanie

- Przerwanie zostało obsłużone
- Procesor ma więcej niż dwa programy
- Kolejność wykonania programów zależy od względnych priorytetów wykonania tych programów oraz od tego czy czekają one na zakończenie we/wy
- Po zakończeniu obsługi przerwania sterowanie może nie zostać przekazane do przerwanego programu lecz do programu, który ma wyższy priorytet wykonania

Potokowe przetwarzanie rozkazów

- ang. instruction pipelining



Potokowe przetwarzanie rozkazów

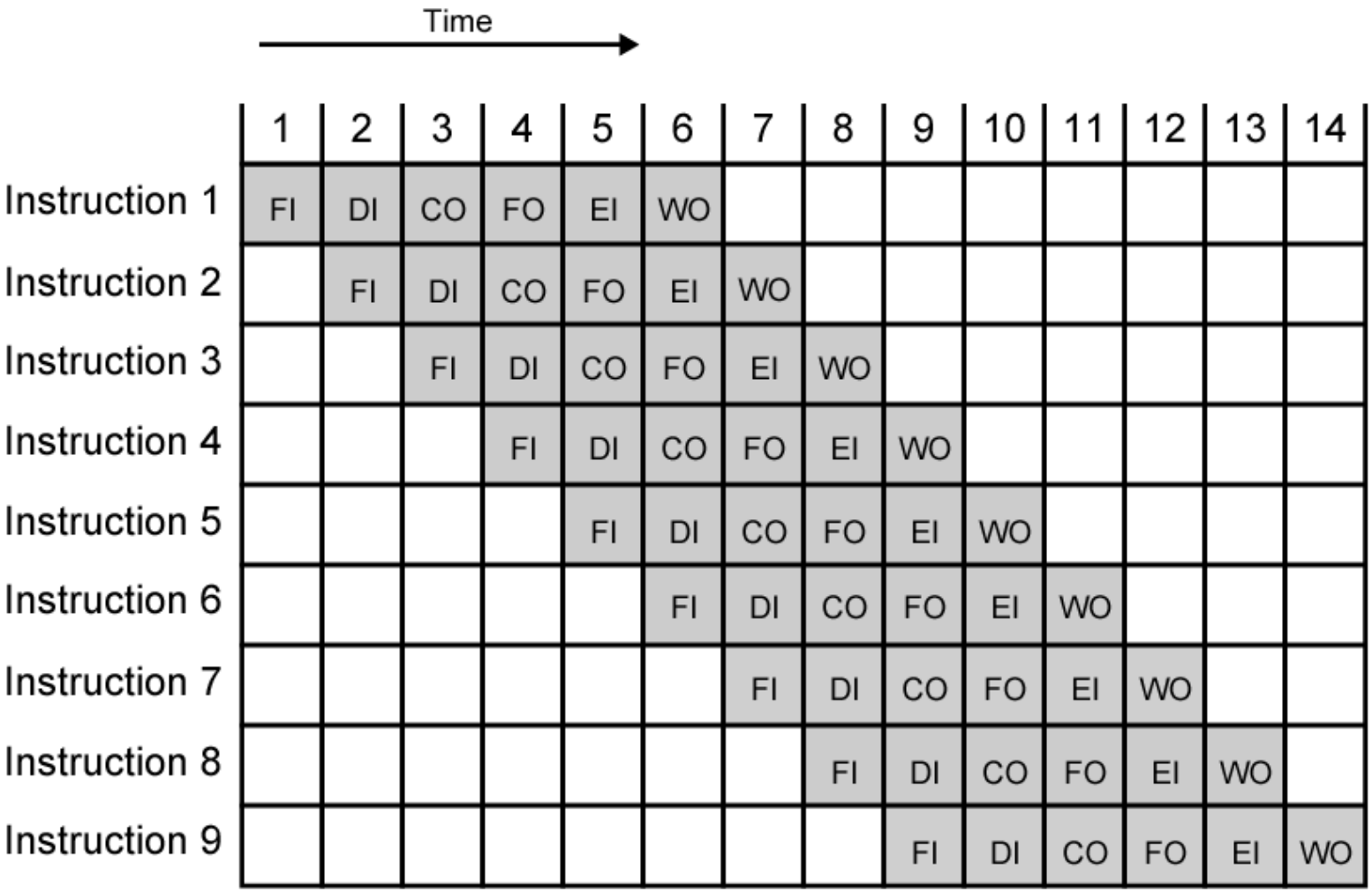
- Czas wykonywania rozkazu jest zwykle większy od czasu pobierania a więc musi być zwłoka wynikająca z zajętości bufora
- Rozkaz skoku warunkowego powoduje, że adres następnego rozkazu jest nieprzywidywalny
- Gdy rozkaz skoku warunkowego przechodzi z etapu pobierania do wykonywania możemy pobrać również rozkaz następny
  - jeśli jednak skok nastąpi to musi on zostać usunięty
- Aby uzyskać większe przyspieszenie potok musi być przetwarzany w większej liczbie etapów

6-etapowa dekompozycja rozkazu

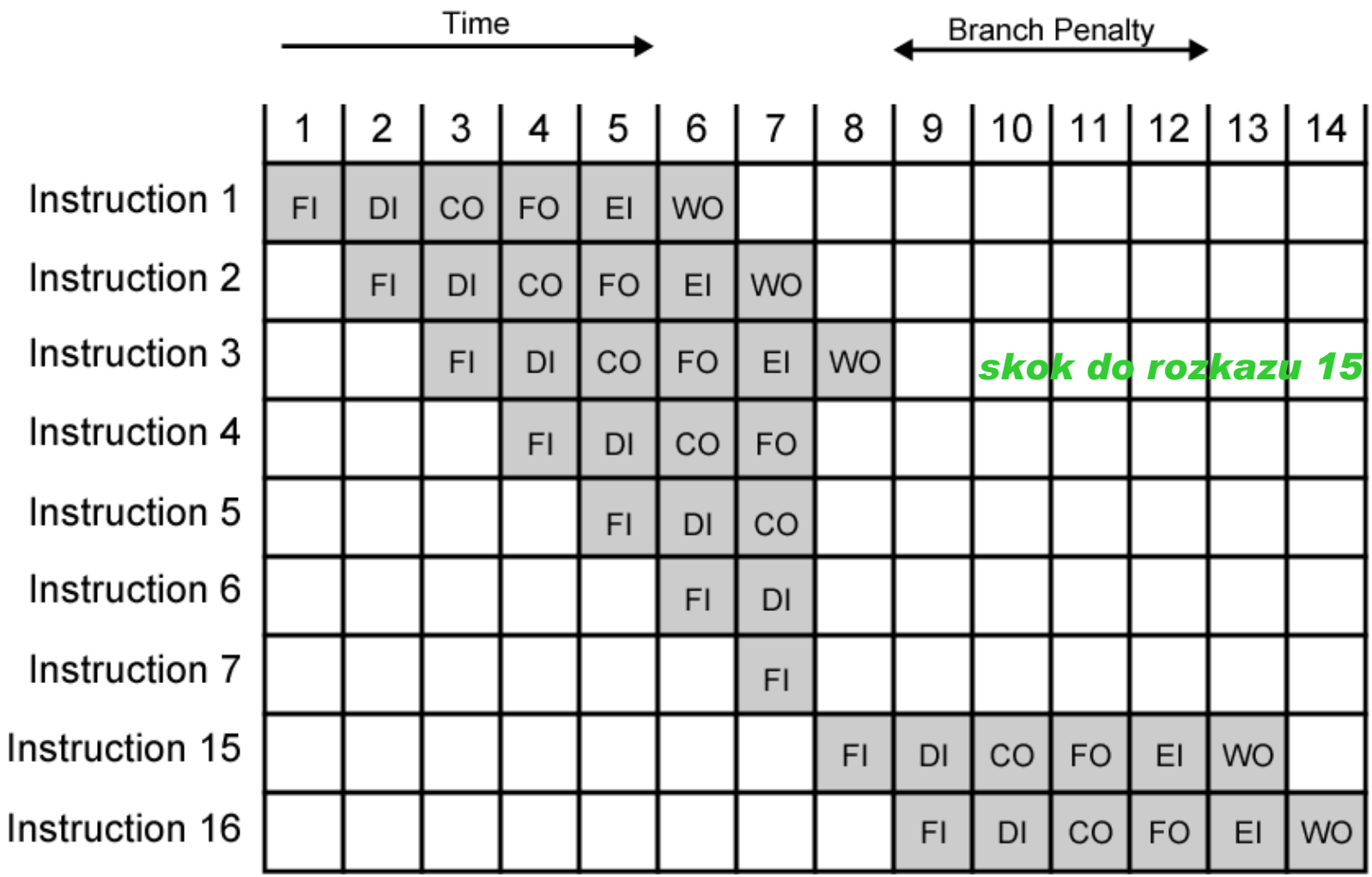
- Pobranie rozkazu (ang. Fetch Instruction)
- Dekodowanie rozkazu (ang. Decode Instruction)
- Obliczanie argumentów (ang. Calculate Operands)
- Pobieranie argumentów (ang. Fetch Operands)
- Wykonanie rozkazu (ang. Execute Instructions)
- Zapisanie wyników (ang. Write Operands)
  - rozkaz LOAD nie wymaga etapu (WO)
  - zrównoleglone etapy (FI), (FO), (WO) mogą powodować konflikty przy dostępie do pamięci

- Aby poprawić wydajność należy zrównoleglić te etapy
  - Prz. zmniejszenie czasu wykonywania 9 rozkazów z =9*6=54 jednostek do 14

Przebieg czasowy przetwarzania potoku rozkazów

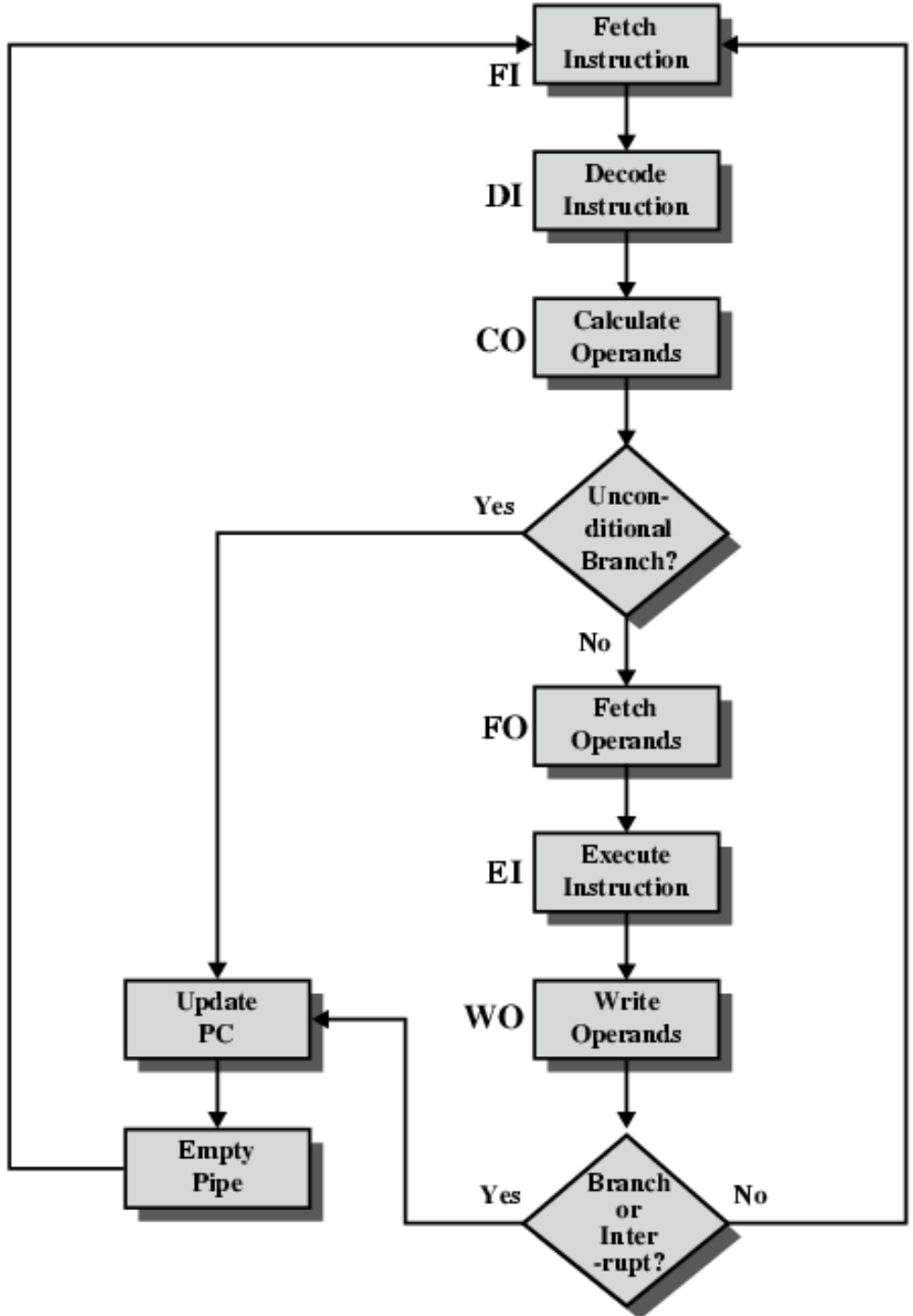


Wpływ rozgałęzienia warunkowego na potok rozkazu



Potok 6-etapowy

Schemat potoku z uwzględnieniem rozgałezień i przerwań



Ochrona sprzętowa

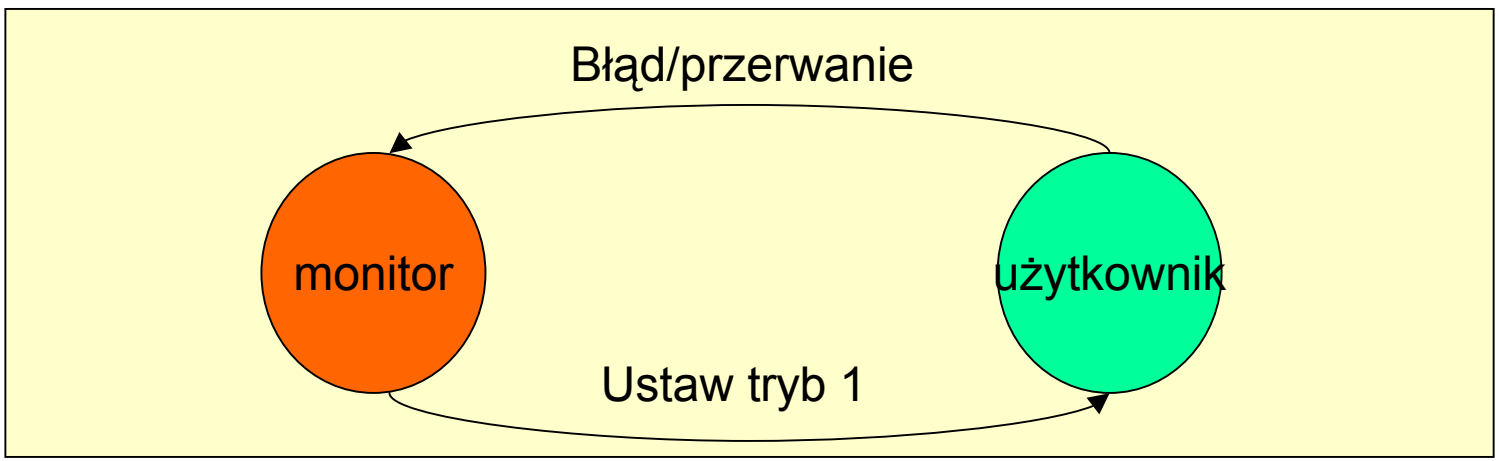
- Dualny tryb operacji (ang. dual-mode operation)
- Ochrona wejścia-wyjścia
- Ochrona pamięci
- Ochrona jednostki centralnej

Dualny tryb operacji

- System wielozadaniowy musi chronić system operacyjny oraz wykonywane programy przed każdym niewłaściwie działającym programem
- Należy wyposażyć sprzęt w środki pozwalające na rozróżnienie przynajmniej dwóch trybów pracy:
  - tryb użytkownika (ang. user mode) - wykonanie na odpowiedzialność użytkownika
  - tryb monitora (ang. monitor mode) = tryb nadzorcy (ang. supervisor mode) = tryb systemu (ang. system mode) = tryb uprzywilejowany (ang. privileged mode) - wykonanie na odpowiedzialność systemu operacyjnego

Dualny tryb operacji (c.d.)

- Bit trybu (ang. mode bit) w sprzęcie komputerowym wskazujący na bieżący tryb pracy : system (0), użytkownik (1)
- Wystąpienie błędu: przejście w tryb 0



- Rozkazy uprzywilejowane (ang. privileged) - tryb systemu

Ochrona wejścia-wyjścia

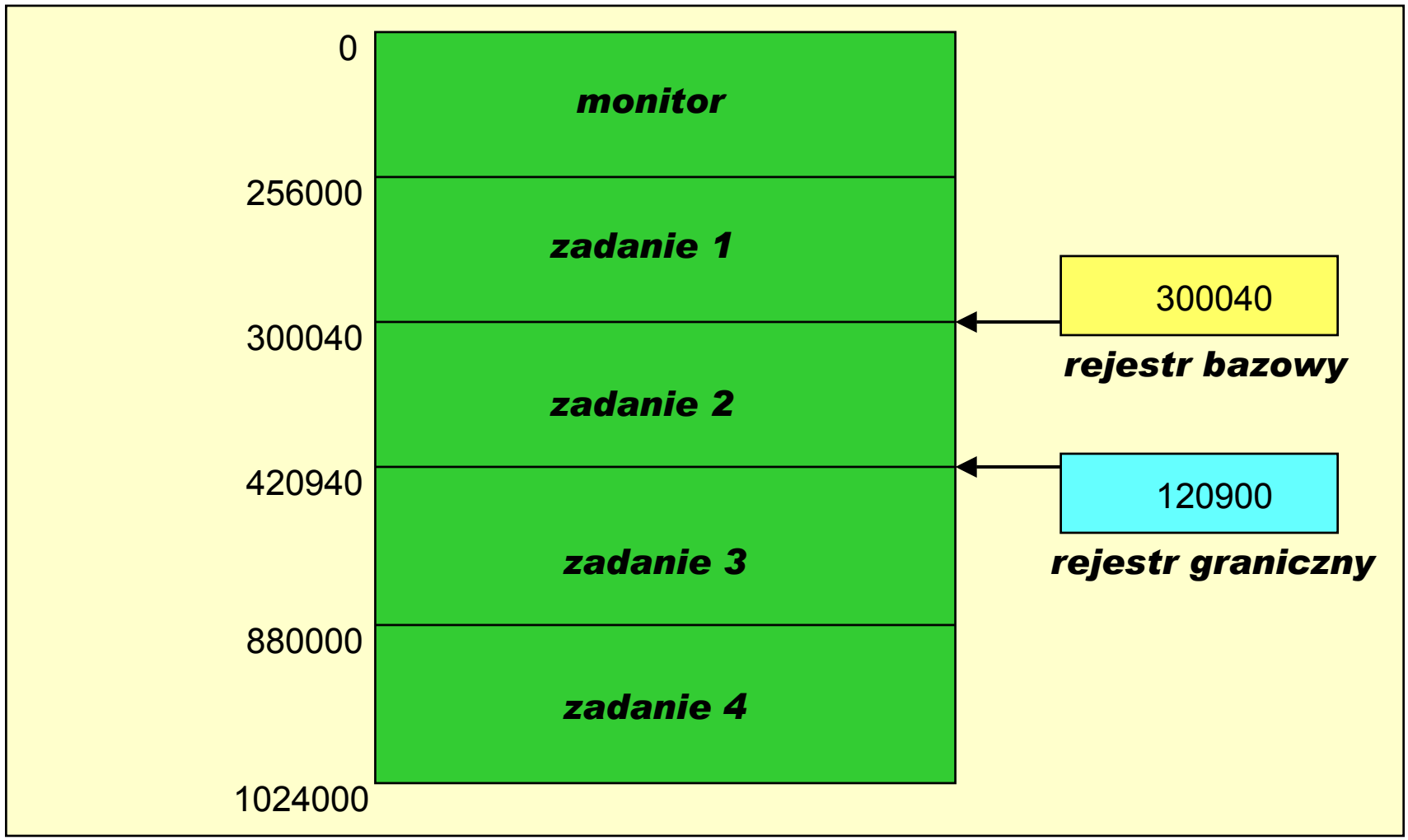
- Wszystkie instrukcje wejścia-wyjścia są uprzywilejowane

- Ochrona we/wy musi zapewniać, że użytkownik nigdy nie uzyska kontroli nad komputerem w trybie monitora (np. program użytkownika w czasie swego wykonania zapamiętuje nowy adres w wektorze przerwań we/wy)

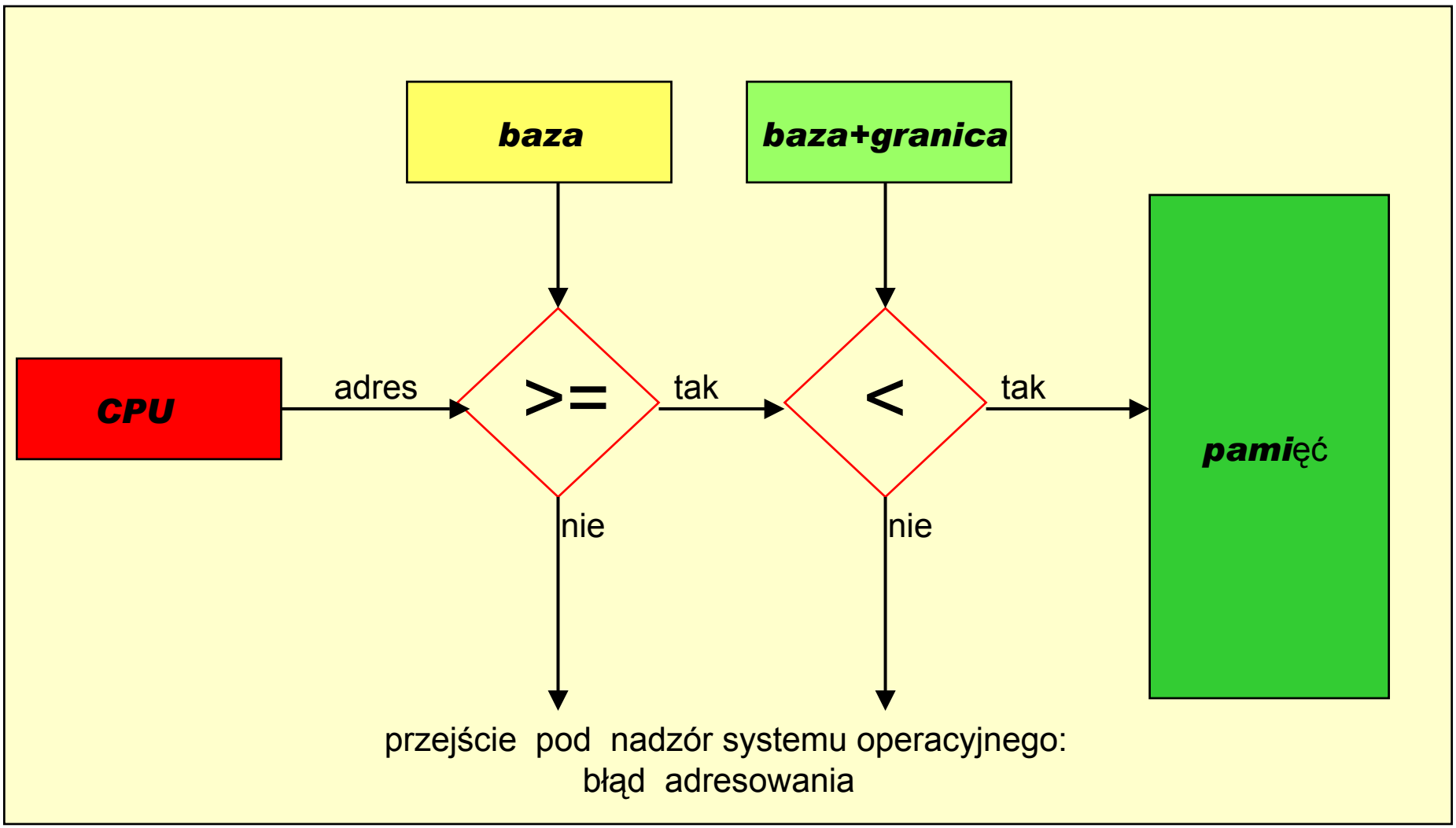
Ochrona pamięci

- Musi zapewniać ochronę pamięci (ang. memory protection) przynajmniej dla wektora przerwań oraz systemowych procedur ich obsługi
- Aby mieć ochronę pamięci musimy mieć dwa rejestry, które określają zakres dozwolonych adresów dostępu dla programu:
  - rejestr bazowy (ang. base) - pierwszy dozwolony adres
  - rejestr graniczny (ang. limit) - rozmiar dozwolonego obszaru
- Pamięć poza zdefiniowanym zakresem jest chroniona

Rejestr bazowy i graniczny definiują przestrzeń adresową



Sprzętowa ochrona adresów



Sprzętowa ochrona adresów

- sprzęt jednostki centralnej porównuje każdy adresu wygenerowany w trybie użytkownika z zawartością rejestrów bazowego i granicznego
- zwartości rejestru bazowego i granicznego mogą być załadowane jedynie w trybie monitora (load jest instrukcją uprzywilejowaną)
- przerwanie protekcja pamięci

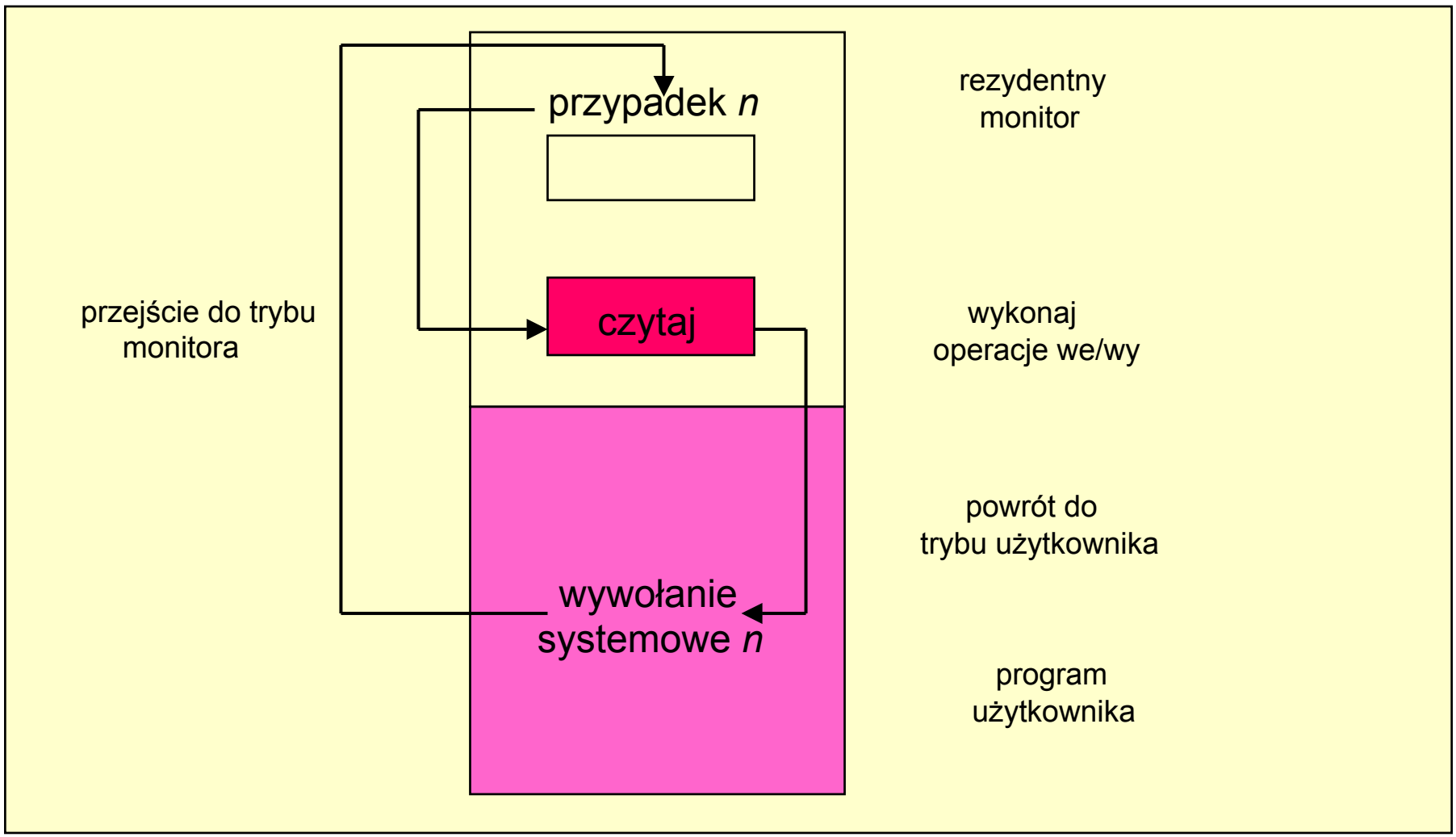
Ochrona CPU

- Zegar jest ustawiany przez system operacyjny przed przekazaniem sterowania do programu użytkownika
  - w trakcie wykonywania programu zegar jest zmniejszany
  - jeśli zegar osiągnie 0 generowane jest przerwanie
- ładuj zegar jest rozkazem uprzywilejowanym
- zegar może być wykorzystany do realizacji podziału czasu (ang. time sharing) - przerwanie zegarowe następuje po wykorzystaniu kwantu czasu przez proces

Ogólna architektura systemu

- Jak dla danej uprzywilejowanej instrukcji we/wy program użytkowy może ją wykonać?
- wywołanie systemowe (ang. system call) to sposób na zamówienie przez proces działania systemu operacyjnego
  - zwykle przejście do określonej komórki w wektorze przerwań
  - sterowanie zostaje przekazane do podprogramu obsługi przerwania w trybie monitora
  - system sprawdza poprawność wywołania systemowego, wykonuje zlecenie i oddaje sterowanie do nastepnej instrukcji po wywołaniu systemowym

Użycie odwołania do systemu



Start systemu - Booting

- Rozruch systemu (ang. booting) - mały fragment kodu, przechowywany w ROM (firmware), określany jako program rozruchowy (ang. bootstrap program) lub elementarny program ładujący (ang. bootstrap loader)
  - niektóre systemy komputerowe (PDA, komórki, konsole) posiadają cały system operacyjny w ROMie
  - problem upgrade'u systemu rozwiązuje się zastępując ROM EPROMem
- Program ładujący jest w stanie zlokalizować kod jądra systemu, wprowadzić go do pamięci i rozpocząć jego wykonanie
  - dwuetapowy program ładujący sprowadza do pamięci bardziej złożony program ładujący, który powoduje załadowanie jądra systemu
    - zwykle z pierwszego bloku na systemowym dysku (logicznym) (ang. boot block)

Start jądra systemu - Unix

- Pierwszy sektor na dysku (ang. Master Boot record, MBR) zawiera program boot, który zostaje wczytany do pamięci
- Uruchomiony zostaje program boot, który
  - relokuje się aby zwolnić początkowe adresy pamięci na jądro systemu
  - czyta katalog root na dysku
  - wczytuje jądro systemu
  - przekazuje sterowanie jądru systemu
    - asemblerowy kod inicjujący jądra systemu

Start systemu - Windows 2000

- Pierwszy sektor na dysku (ang. Master Boot Record, MBR) zawiera program boot, który zostaje wczytany do pamięci
- Uruchomiony zostaje program boot, który:
  - relokuje się aby zwolnić początkowe adresy pamięci na jądro systemu
  - czyta katalog root na dysku
  - wczytuje program ntldr
  - przekazuje sterowanie programowi ntldr:
    - czyta plik konfiguracyjny Boot.ini
    - wczytuje pliki: hal.dll, ntoskrnl.exe, bootvid.dll
    - wczytuje drivery (myszy,...)
    - przekazuje sterowanie programowi ntoskrnl.exe