
Uniwersytet
w Siedlcach
Architektura
Systemów
Komputerowych
dr Marcin
Stępniak

ISA - instruction set architecture

- **Architektura zestawu instrukcji procesora** (ang. *instruction set architecture*, **ISA**), **model programowy procesora**:
  - ogólna definicja (specyfikacja) dotycząca organizacji, funkcjonalności i zasad działania procesora, widocznych z punktu widzenia programisty jako dostępne mechanizmy programowania.

ISA - instruction set architecture

- Na model programowy procesora składają się, między innymi:
  - lista rozkazów procesora
  - typy danych
  - dostępne tryby adresowania
  - zestaw rejestrów dostępnych dla programisty
  - zasady obsługi wyjątków i przerwań.

ISA - instruction set architecture

- Procesory posiadające ten sam model programowy są ze sobą kompatybilne, co oznacza, że mogą wykonywać te same programy i generować te same rezultaty.
- W początkowej historii procesorów model programowy procesora zależał od fizycznej implementacji procesora i niejednokrotnie całkowicie z niej wynikał.

- Obecnie tendencja jest odwrotna i stosuje się bardzo różne implementacje fizyczne (mikroarchitektury) pochodzące od różnych producentów, natomiast realizujące ten sam ISA.

ISA - instruction set architecture

- **ISA** to również abstrakcyjny model komputera, zwany również architekturą komputera.
- Realizacja ISA nazywana jest implementacją.

- ISA pozwala na wiele implementacji, które mogą różnić się wydajnością, rozmiarem fizycznym i kosztami (między innymi); ponieważ ISA służy jako interfejs między oprogramowaniem a sprzętem.

ISA - instruction set architecture

- Oprogramowanie napisane dla ISA może działać w różnych implementacjach tego samego ISA.

- Umożliwiło to łatwe osiągnięcie zgodności binarnej między różnymi generacjami komputerów oraz rozwój rodzin komputerów.
- Oba te rozwiązania pomogły obniżyć koszty komputerów i zwiększyć ich zastosowanie.

- Z tych powodów ISA jest jedną z najważniejszych abstrakcji w dzisiejszym informatyce.
- ISA definiuje wszystko, co programista w języku maszynowym musi wiedzieć, aby zaprogramować komputer.

CISC (*Complex Instruction Set Computing*)

- **CISC** (ang. *Complex Instruction Set Computing*) – typ architektury zestawu instrukcji procesora o następujących cechach:
  - występowanie złożonych, specjalistycznych rozkazów (instrukcji), które do wykonania wymagają od kilku do kilkunastu cykli zegara,
  - szeroka gama trybów adresowania,
  - przeciwnie niż w architekturze RISC, rozkazy mogą operować bezpośrednio na pamięci (zamiast przesłania wartości do rejestrów i operowania na nich),
  - powyższe założenia powodują, iż dekoder rozkazów jest skomplikowany.

CISC (*Complex Instruction Set Computing*)

- Istotą architektury CISC jest to, iż pojedynczy rozkaz mikroprocesora wykonuje kilka operacji niskiego poziomu, jak na przykład pobranie z pamięci, operację arytmetyczną i zapisanie do pamięci.
- Przed powstaniem procesorów RISC wielu komputerowych architektów próbowało zaprojektować zestawy rozkazów, które wspierałyby języki programowania wysokiego poziomu przez dostarczenie rozkazów wysokiego poziomu np. wywołania funkcji i zwrócenia jej wartości, instrukcje pętli czy kompleksowe tryby adresowania.
- Rezultatem tego były programy o mniejszym rozmiarze i z mniejszą ilością odwołań do drogiej i niewielkiej pamięci, co w tamtym czasie było istotne z punktu widzenia wydajności przy jednoczesnym dążeniu do obniżenia kosztów pojedynczego komputera.

CISC (*Complex Instruction Set Computing*)

- Przykłady rodzin procesorów o architekturze CISC to:
  - IBM System/360
  - VAX
  - PDP-11
  - x86

- Współczesne procesory zgodne z x86 produkowane przez firmy Intel, AMD i VIA przetwarzają rozkazy procesora x86 na proste mikropolecenia pracujące według idei RISC, często wykonujące się równolegle.

CISC (*Complex Instruction Set Computing*)

- Zanim podejście RISC stało się atrakcyjne, wielu architektów komputerowych próbowało wypełnić tak zwaną lukę semantyczną,

  tj. zaprojektować zestawy instrukcji, które bezpośrednio obsługują konstrukcje programowania wysokiego poziomu,

  takie jak wywołania procedur, sterowanie pętlami i złożone tryby adresowania, umożliwiając strukturę danych oraz dostępy do tablic, które można połączyć w pojedyncze instrukcje.
- Instrukcje są również zazwyczaj wysoko zakodowane w celu dalszego zwiększenia gęstości kodu.

CISC (*Complex Instruction Set Computing*)

- Kompaktowy charakter takich zestawów instrukcji skutkuje mniejszymi rozmiarami programów i mniejszą liczbą dostępów do pamięci głównej (które były często powolne i drogie),
- co w tamtych czasach (na początku lat 60. i później) skutkowało ogromnymi oszczędnościami na kosztach pamięci komputera i przechowywania dysków, ponieważ jak również szybsze wykonanie.
- Oznaczało to również dobrą produktywność programowania nawet w języku asemblerowym, ponieważ języki wysokiego poziomu, takie jak Fortran lub Algol, nie zawsze były dostępne lub odpowiednie.
- Rzeczywiście, mikroprocesory z tej kategorii są czasami wciąż programowane w języku asemblera dla pewnych typów krytycznych aplikacji.

RISC *Reduced Instruction Set Computer*

- RISC lub komputer z ograniczonym zestawem instrukcji. to rodzaj architektury mikroprocesorowej, która wykorzystuje mały, wysoce zoptymalizowany zestaw instrukcji,

  a nie bardziej wyspecjalizowany zestaw instrukcji często spotykany w innych typach architektur jak CISC.

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/whatis/index.html

RISC *Reduced Instruction Set Computer*

- Pierwsze projekty RISC pochodziły z IBM, Stanford i UC-Berkeley pod koniec lat 70. i na początku 80. IBM 801, Stanford MIPS oraz Berkeley RISC 1 i 2 zostały zaprojektowane zgodnie z podobną filozofią, która stała się znana jako RISC.
- Niektóre cechy konstrukcyjne były charakterystyczne dla większości procesorów RISC:
  - czas wykonania jednego cyklu: procesory RISC mają CPI (zegar na instrukcję) jednego cyklu. Wynika to z optymalizacji każdej instrukcji na procesorze i techniki zwanej ;
  - pipelining: technika pozwalająca na jednoczesne wykonywanie części lub etapów instrukcji w celu wydajniejszego przetwarzania instrukcji;
  - duża liczba rejestrów: projektowanie RISC na ogół obejmuje większą liczbę rejestrów, aby zapobiec w dużej liczbie interakcji z pamięcią

RISC *Reduced Instruction Set Computer*

- Najpierw szczegółowo przyjrzymy się MIPS jako przykładowi wczesnej architektury RISC, aby lepiej zrozumieć cechy i projekt architektury RISC.
- Następnie przestudiujemy potokowanie, aby zobaczyć korzyści wydajnościowe takiej techniki.
- Dalej przyjrzymy się zaletom i wadom takiej architektury opartej na architekturze RISC w porównaniu z architekturami CISC.
- Na koniec omówimy niektóre z ostatnich osiągnięć i przyszłych kierunków rozwoju technologii procesorowej RISC w szczególności oraz technologii procesorowej jako całości.

RISC - MIPS

- Procesor MIPS (akronim od *Microprocessor without Interlocked Pipeline Stages*) został opracowany w ramach programu badawczego VLSI na Uniwersytecie Stanforda na początku lat 80-tych.
- Profesor John Hennessy, były rektor uniwersytetu, rozpoczął rozwój MIPS od zajęć z burzy mózgów dla doktorantów.
- Odczyty i sesje pomysłów pomogły w rozpoczęciu rozwoju procesora, który stał się jednym z pierwszych procesorów RISC, a IBM i Berkeley rozwijały procesory mniej więcej w tym samym czasie.

RISC - MIPS

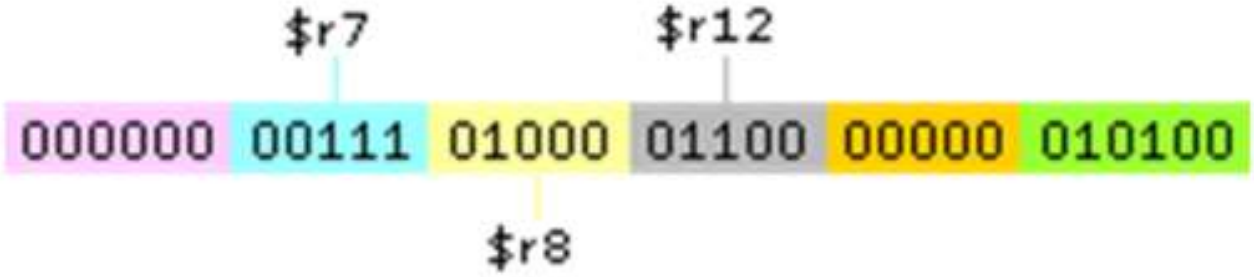
Architektura MIPS

- Grupa badawcza ze Stanford miała duże doświadczenie w kompilatorach, co doprowadziło ich do opracowania procesora, którego architektura reprezentowałaby obniżenie poziomu kompilatora do poziomu sprzętowego,
- w przeciwieństwie do podnoszenia sprzętu do poziomu oprogramowania, co miało miejsce wówczas od dawna w branży sprzętowej.
- W ten sposób procesor MIPS zaimplementował mniejszy, prostszy zestaw instrukcji.
- Każda z instrukcji zawartych w projekcie układu działała w jednym cyklu zegara.
- Procesor zastosował technikę zwaną potokowaniem, aby wydajniej przetwarzać instrukcje.

➢ Architektura MIPS

- MIPS używał 32 rejestrów, każdy o szerokości 32 bitów (wzorzec bitowy tego rozmiaru jest określany jako słowo).
- Zestaw instrukcji MIPS składa się z około 111 instrukcji, z których każda jest reprezentowana w 32 bitach. Przykład instrukcji MIPS znajduje

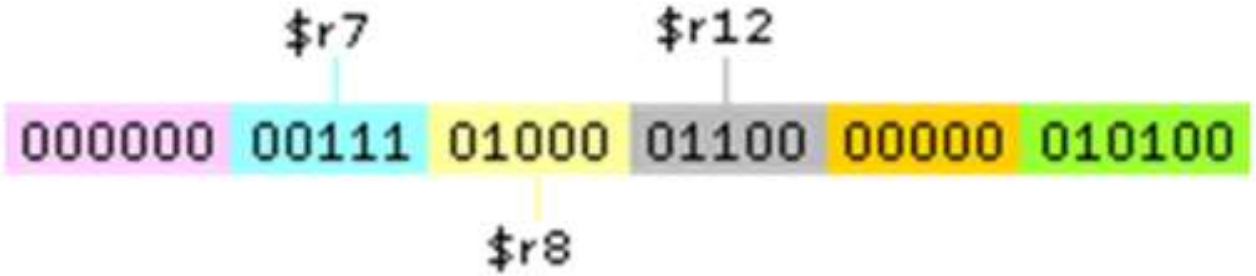
```
add $r12,
$r7, $r8
```



RISC - MIPS

- asembler (po lewej) i binarna (po prawej) to reprezentacja instrukcji dodawania MIPS.
- Instrukcja nakazuje procesorowi obliczenie sumy wartości w rejestrach 7 i 8 i zapisanie wyniku w rejestrze 12.
- Znaki dolara są używane do wskazania operacji na rejestrze.
- Kolorowa reprezentacja binarna po prawej stronie ilustruje 6 pól instrukcji MIPS.
- Procesor identyfikuje typ instrukcji za pomocą cyfr binarnych w pierwszym i ostatnim polu.
- W tym przypadku procesor rozpoznaje, że ta instrukcja jest dodawaniem od zera w jego pierwszym polu i 20 w jego ostatnim polu.
- Operandy są reprezentowane w niebieskich i żółtych polach, a żądana lokalizacja wyniku jest przedstawiona w czwartym (fioletowym) polu.
- Pomarańczowe pole reprezentuje kwotę przesunięcia, coś, co nie jest używane w operacji dodawania.

```
add $r12,
$r7, $r8
```



RISC - MIPS

Zestaw instrukcji składa się z różnych podstawowych instrukcji, w tym:

- 21 instrukcji arytmetycznych (+, -, *, /, %)
- 8 instrukcji logicznych (&, |, ~)
- 8 instrukcji manipulacji bitów
- 12 instrukcji porównawczych (>, <, =, >=, <=, ¬)
- 25 instrukcji rozgałęzień/skoków
- 15 instrukcji ładowania – pobierania z pamięci
- 10 instrukcji do zapisywania w pamięci
- 8 instrukcji do kopiowania
- 4 inne instrukcje

Listę podstawowych instrukcji MIPS można znaleźć tutaj. ftp://ftp.mkp.com/COD2e/Web_Extensions/survey.htm#I.3

RISC - MIPS

- **MIPS dzisiaj**
- Firma MIPS Computer Systems, Inc. została założona w 1984 roku w wyniku badań Stanford, w wyniku których powstał pierwszy chip MIPS.
- Została zakupiona przez Silicon Graphics, Inc. w 1992 roku i wydzielona jako MIPS Technologies, Inc. w 1998 roku.
- Obecnie MIPS zasila wiele urządzeń elektroniki użytkowej i innych urządzeń.
- Orócz John L. Hennessy oraz Chris Rowen należy wymienić też Skip Stritter, dawniej w Motorola, oraz John Moussouris poprzednio w IBM.

RISC - ARM

- **ARM** (ang. *Advanced RISC Machine*, pierwotnie *Acorn RISC Machine*) – rodzina architektur (modeli programowych) procesorów 32-bitowych oraz 64-bitowych, typu RISC.
- Różne wersje rdzeni ARM są szeroko stosowane w systemach wbudowanych i systemach o niskim poborze mocy, ze względu na ich energooszczędną architekturę.
- Procesory z architekturą ARM są jednymi z najczęściej stosowanych procesorów na świecie.
- Używa się ich między innymi w dyskach twardych, telefonach komórkowych, routerach, kalkulatorach, a nawet w zabawkach dziecięcych.

RISC - ARM

- **ARM** (ang. *Advanced RISC Machine*, pierwotnie *Acorn RISC Machine*) – rodzina architektur (modeli programowych) procesorów 32-bitowych oraz 64-bitowych, typu RISC.
- Obecnie zajmują one ponad 75% rynku 32-bitowych CPU dla systemów wbudowanych.
- Najpopularniejszym projektem ARM był rdzeń ARM7TDMI szeroko stosowany w telefonach komórkowych.
- Moc obliczeniowa architektury ARM umożliwia instalacje na procesorze systemu operacyjnego, z zaimplementowanymi mechanizmami wielowątkowości, z możliwością wykorzystania zawartego w systemie stosu TCP/IP czy systemu plików (np. FAT32).

RISC - ARM

- Projekt pierwszego procesora ARM w 1983 roku, jako projekt rozwojowy brytyjskiej Acorn Computers Ltd.
- Grupa inżynierów kierowana przez Rogera Wilsona i Steve'a Furbera rozpoczęła projektowanie jądra będącego ulepszoną wersją procesora MOS 6502 firmy MOS Technology.
- Acorn produkował w tym czasie komputery w oparciu o mikroprocesor MOS 6502, więc celem projektu było opracowanie nowego potężniejszego mikroprocesora programowalnego w podobny sposób.

RISC - ARM

- Pierwsza wersja testowa, nazywana ARM1, opracowana została w 1985 roku, a rok później ukończono wersję produkcyjną ARM2.
- ARM2 wyposażony był w 32-bitową szynę danych, 26-bitową przestrzeń adresową oraz w szesnaście 32-bitowych rejestrów.
- Był to w tym czasie najprostszy szeroko stosowany 32-bitowy mikroprocessor, zawierający tylko 30 tysięcy tranzystorów.
- Prostota wynikała głównie z braku mikrokodu i, jak w większości procesorów w tym czasie, braku *cache*.
- ARM2 miał z tego powodu bardzo niski pobór mocy i jednocześnie szybkość przetwarzania większą od procesora Intel 80286.
- Następna wersja ARM3 produkowana była z 4 KB cache, co jeszcze bardziej poprawiło wydajność.

RISC - ARM

- W późnych latach osiemdziesiątych XX [illegible] Computer rozpoczęło współpracę z Acorn Computers w projektowaniu nowszej wersji jądra ARM.
- Projekt był na tyle istotny, że Acorn wydzielił grupę projektową tworząc w 1990 roku Advanced RISC Machines (ARM Ltd.).
- Wynikiem tej współpracy był procesor ARM6, udostępniony w roku 1990.
- Apple użył opartego na ARM6 procesora ARM610 w palmtopie (PDA) o nazwie Apple Newton.

RISC - ARM

- Jądro procesora ARM6 zawiera około 35 tysięcy tranzystorów i jest tylko niewiele większe od jądra ARM2 (30 tysięcy tranzystorów).
- Dzięki swej prostocie jądro ARM może być łączone z dodatkowymi blokami funkcjonalnymi, tworząc w jednej obudowie, mikroprocesor dostosowany do konkretnych wymagań.
- Jest to możliwe, gdyż podstawą działalności ARM Ltd. jest sprzedaż licencji na zaprojektowane jądra.
- Dzięki temu powstały także mikrokontrolery oparte na architekturze ARM.

RISC - ARM

- DEC zakupiło licencję na architekturę ARM i na jej podstawie zaprojektowało procesor StrongARM.
- Przy częstotliwości 233 MHz procesor ten pobierał tylko 1 W mocy (najnowsze wersje StrongARM pobierają znacznie mniej).
- Projekt ten został następnie przejęty przez Intel, na podstawie umowy między przedsiębiorstwami.
- Dla Intel była to szansa na zastąpienie przestarzałej architektury i960 nową architekturą StrongARM.
- Na podstawie StrongARM Intel zaprojektował bardzo wydajny mikroprocesor o nazwie XScale.

RISC - ARM

- Zgodnie z założeniami architektury RISC, rozkazy procesorów ARM są tak skonstruowane, aby wykonywały jedną określoną operację i były przetwarzane w jednym cyklu maszynowym.
- Interesującą zmianą w stosunku do innych architektur jest użycie 4-bitowego kodu warunkowego na początku każdej instrukcji.
- Dzięki temu każda instrukcja może być wykonana warunkowo.
- Ogranicza to przestrzeń dostępną, na przykład, dla instrukcji przeniesień w pamięci, ale z drugiej strony nie ma potrzeby stosowania instrukcji rozgałęzień dla kodu zawierającego wiele prostych instrukcji warunkowych.

RISC - ARM

Klasycznym przykładem jest implementacja algorytmu Euklidesa wyznaczania największego wspólnego dzielnika.

Funkcja w języku C wyglądająca tak:

```
while (i != j)
{
    if (i > j)
        i -= j;
    else
        j -= i;
}
```

W asemblerze procesora ARM będzie miała postać następującej pętli:

```
loop    CMP    Ri, Rj       ; porównaj i z j, ustawiając flagi warunkowe:
                            ;   - "GT" dla (i > j)
                            ;   - "LT" dla (i < j)
                            ;   - "NE" dla (i != j)
        SUBGT  Ri, Ri, Rj   ; jeśli "GT" (większa niż ang. greater than),
wykonaj i := i - j;
        SUBLT  Rj, Rj, Ri   ; jeśli "LT" (mniejsza niż ang. less than),
wykonaj j := j - i;
        BNE    loop         ; jeśli "NE" (nie równe ang. not equal),
wykonaj skok do etykiety 'loop'
```

RISC - ARM

- Inną unikatową cechą zestawu instrukcji procesora ARM jest łączenie operacji przesunięcia i obrotu bitów w rejestrze z instrukcjami arytmetycznymi, logicznymi, czy też przesłania danych z rejestru do rejestru.
- Dzięki temu wyrażenie języka C „a += (j << 2);” może zostać przetłumaczone przez kompilator w pojedynczą instrukcję asemblera.
- Przedstawione cechy powodują, że typowy program zawiera mniej linii kodu niż w przypadku innych procesorów RISC.
- W rezultacie jest mniejsza liczba operacji pobrania/zapisania argumentów instrukcji, więc potokowość jest bardziej efektywna.
- Pomimo że procesory ARM są taktowane zegarem o stosunkowo niskiej częstotliwości są konkurencyjne w stosunku do znacznie bardziej złożonych procesorów.

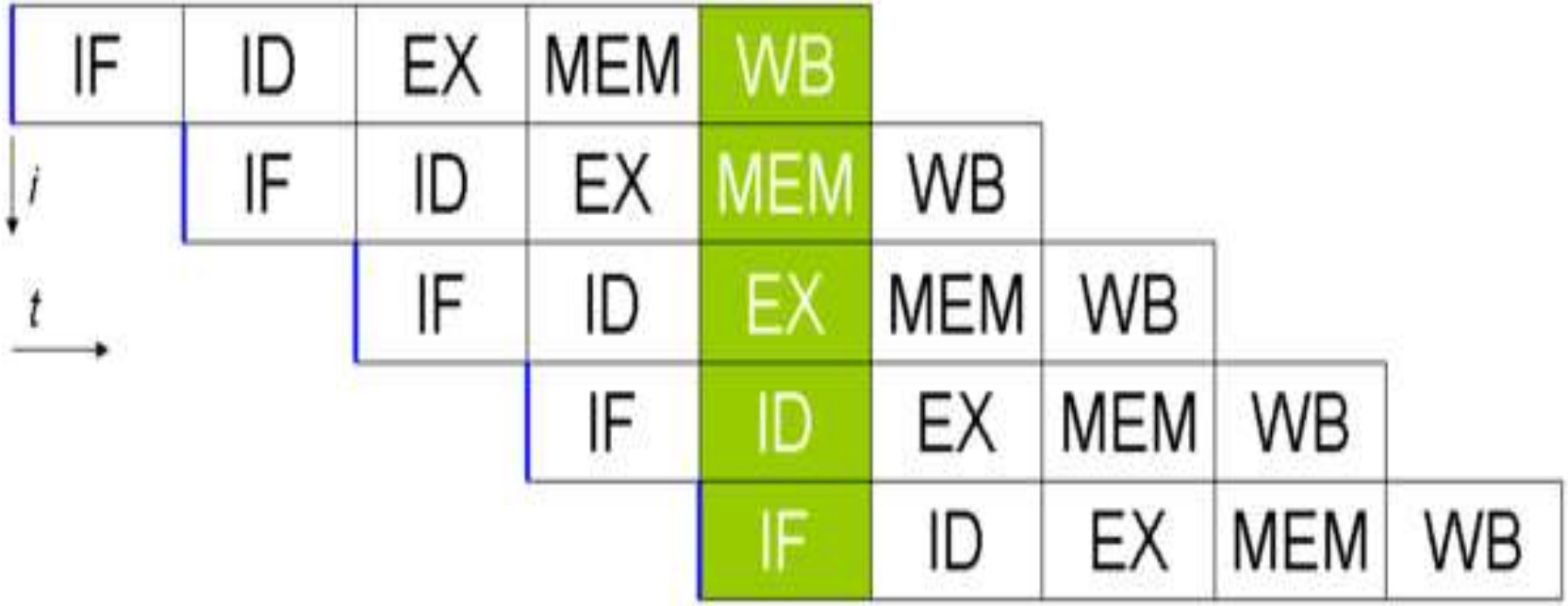
RISC - potokowanie (pipelininig)

- Pipelining, standardowa funkcja procesorów RISC, przypomina linię montażową.
- Ponieważ procesor pracuje na różnych etapach instrukcji w tym samym czasie, więcej instrukcji może być wykonanych w krótszym czasie.

Potokowość

- **Potokowość** – technika budowy procesorów polegająca na podziale logiki procesora odpowiedzialnej za proces wykonywania programu (rozkazów procesora) na specjalizowane grupy w taki sposób, aby każda z grup wykonywała część pracy związanej z wykonaniem rozkazu.
- Grupy te są połączone sekwencyjnie – w potok (ang. *pipe*) – i wykonują pracę równocześnie, pobierając dane od poprzedniego elementu w sekwencji.
- W każdej z tych grup rozkaz jest na innym stadium wykonania.
- Można to porównać do taśmy produkcyjnej. W uproszczeniu potok wykonania instrukcji procesora może wyglądać następująco:

Potokowość



- Uproszczony schemat potokowości. Części rozkazów oznaczone na zielono wykonywane są równocześnie
- pobranie instrukcji z pamięci – ang. *instruction fetch* (IF)
- zdekodowanie instrukcji – ang. *instruction decode* (ID)
- wykonanie instrukcji – ang. *execute* (EX)
- dostęp do pamięci – ang. *memory access* (MEM)
- zapisanie wyników działania instrukcji – ang. *store; write back* (WB).

Potokowość

- W powyższym pięciostopniowym potoku przejście przez wszystkie stopnie potoku (wykonanie jednej instrukcji) zabiera co najmniej pięć cykli zegarowych.
- Jednak ze względu na równoczesną pracę wszystkich stopni potoku, jednocześnie wykonywanych jest pięć rozkazów procesora, każdy w innym stadium wykonania.
- Oznacza to, że taki procesor w każdym cyklu zegara rozpoczyna i kończy wykonanie jednej instrukcji i statystycznie wykonuje rozkaz w jednym cyklu zegara.
- Każdy ze stopni potoku wykonuje mniej pracy w porównaniu do pojedynczej logiki, dzięki czemu może wykonać ją szybciej – z większą częstotliwością – tak więc dodatkowe zwiększenie liczby stopni umożliwia osiągnięcie coraz wyższych częstotliwości pracy.

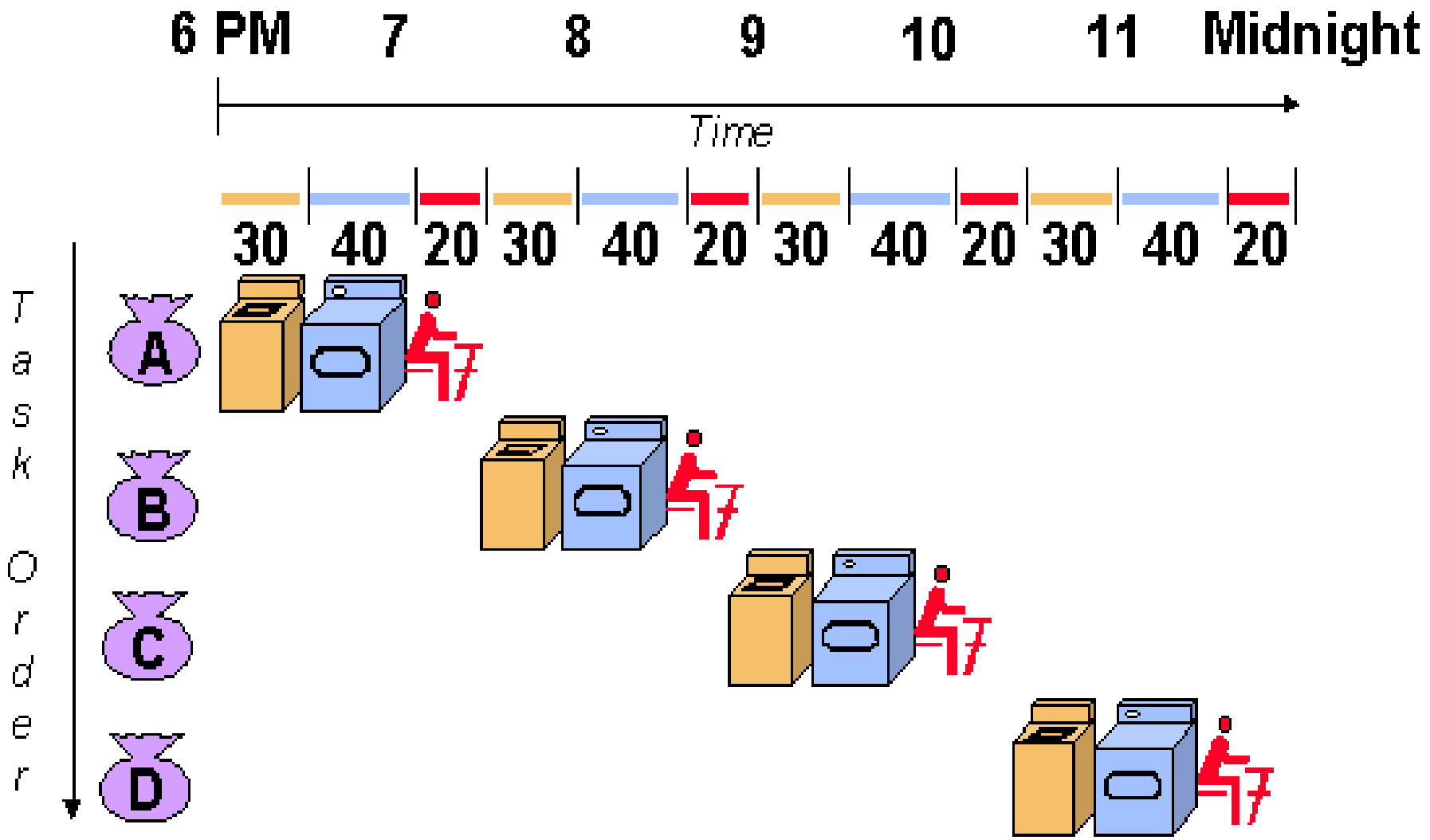
Potokowość

- Podstawowym mankamentem techniki potoku są rozkazy skoku, powodujące w najgorszym wypadku potrzebę przeczyszczenia całego potoku i wycofania rozkazów, które następowały zaraz po instrukcji skoku i rozpoczęcie zapełniania potoku od początku od adresu, do którego następował skok.
- Taki rozkaz skoku może powodować ogromne opóźnienia w wykonywaniu programu – tym większe, im większa jest długość potoku.
- Dodatkowo szacuje się, że dla modelu programowego x86 taki skok występuje co kilkanaście rozkazów.
- Z tego powodu niektóre architektury programowe (np. SPARC) zakładały zawsze wykonanie jednego lub większej liczby rozkazów następujących po rozkazie skoku, tak zwany skok opóźniony.
- Stosuje się także skomplikowane metody predykcji skoku lub metody programowania bez użycia skoków.

RISC - potokowanie (pipelininig)

- **Przykład: pranie.**
- Załóżmy, że są cztery ładunki brudnego do prania, które trzeba wyprać, wysuszyć i złożyć.
- Pierwszy ładunek mogliśmy włożyć do pralki na 30 minut, suszyć przez 40 minut, a następnie złożyć ubrania przez 20 minut.
- Następnie weź drugi ładunek i upierz, wysusz, złóż i powtórz dla trzeciego i czwartego ładunku.
- Przypuśćmy, że zaczęliśmy o 18:00 i pracowaliśmy tak wydajnie, jak to możliwe, nadal robilibyśmy pranie do północy.

RISC - potokowanie (pipelininig)



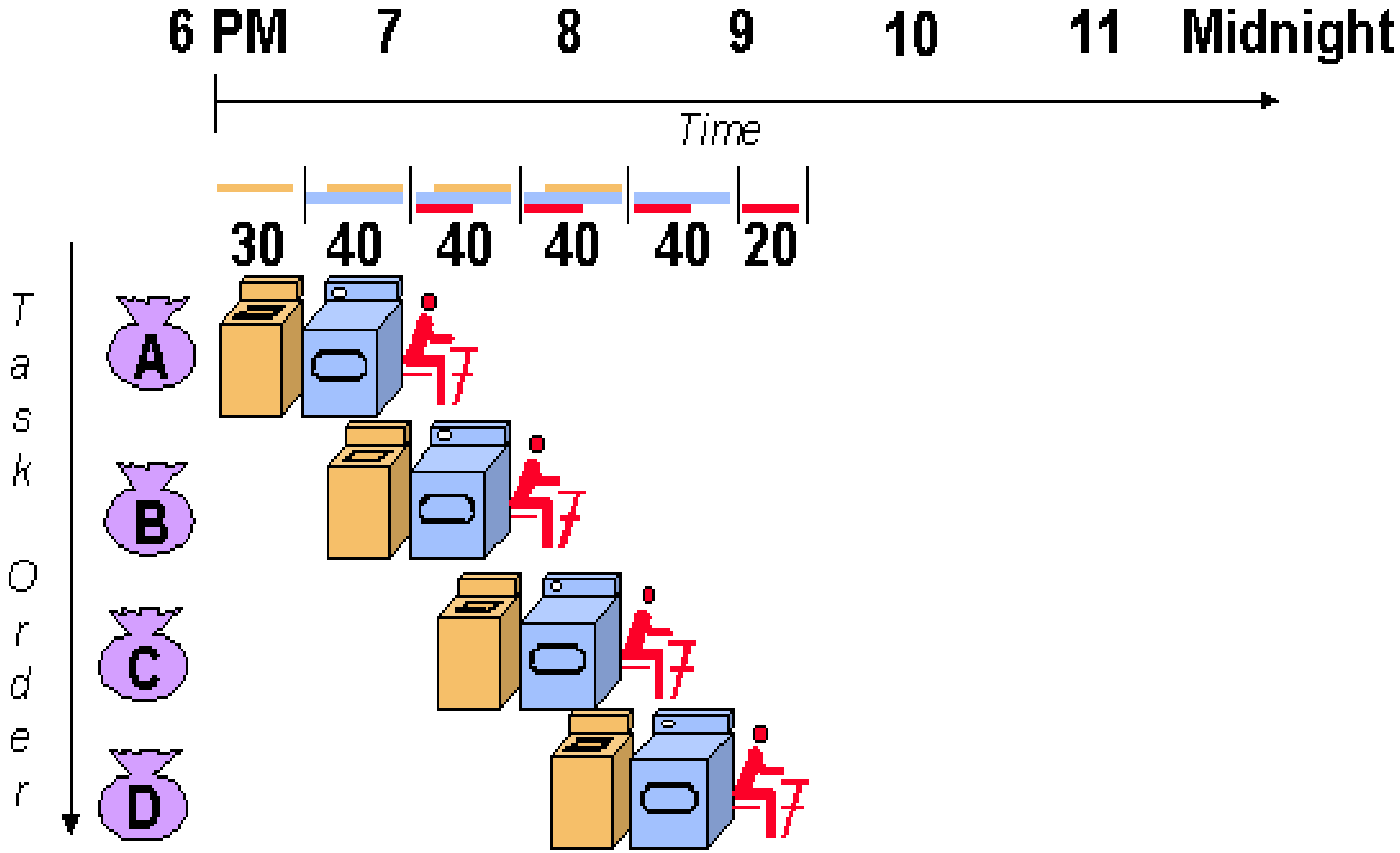
Source: http://www.ece.arizona.edu/~ece462/Lec03-pipe/

RISC - potokowanie (pipelininig)

- Jednak mądrzejszym podejściem do problemu byłoby włożenie drugiego ładunku brudnego prania do pralki po tym, jak pierwszy był już czysty i szczęśliwie wirował w suszarce.
- Następnie, podczas gdy pierwszy ładunek był składany, drugi ładunek wysychał, a trzeci ładunek można było dodać do procesu prania.
- Przy tej metodzie pranie byłoby gotowe do godziny 9:30 PM (tj, 21:30)

Source: http://www.ece.arizona.edu/~ece462/Lec03-pipe/

RISC - potokowanie (pipelininig)



Source http://www.ece.arizona.edu/~ece462/Lec03-pipe/

RISC - potokowanie (pipelininig)

Potoki (pipelines) RISC

- Potok procesora RISC działa w bardzo podobny sposób, chociaż etapy potoku są różne.
- Chociaż różne procesory mają różną liczbę kroków, są to zasadniczo odmiany tych pięciu, używane w procesorze MIPS R3000:
  - pobrać instrukcje z pamięci
  - odczytaj rejestry i dekoduj instrukcję
  - wykonaj instrukcję lub oblicz adres
  - dostęp do operandu w pamięci danych
  - zapisz wynik do rejestru

RISC - potokowanie (pipelininig)

- Jeśli spojrzysz wstecz na schemat potoku w pralni, zauważysz, że chociaż pralka kończy pracę za pół godziny, suszarka zajmuje dodatkowe dziesięć minut, a zatem mokre ubrania muszą czekać dziesięć minut, aż suszarka się zwolni.
- Zatem długość potoku zależy od długości najdłuższego kroku. Ponieważ instrukcje RISC są prostsze niż te używane w procesorach pre-RISC (obecnie nazywanych CISC lub Complex Instruction Set Computer), bardziej sprzyjają potokom.
- Podczas gdy instrukcje CISC różnią się długością, wszystkie instrukcje RISC mają tę samą długość i można je pobrać w jednej operacji.
- W idealnym przypadku każdy z etapów potoku procesora RISC powinien trwać 1 cykl zegara, tak aby procesor kończył instrukcję w każdym cyklu zegara i uśredniał jeden cykl na instrukcję (CPI).

RISC - potokowanie (pipelininig)

Problemy z potokami

- W praktyce jednak procesory RISC działają w więcej niż jednym cyklu na instrukcję.
- Procesor może czasami zatrzymywać się powodu zależności danych i instrukcji skoku.

- Zależność od danych występuje, gdy instrukcja zależy od wyników poprzedniej instrukcji.
- Instrukcja może wymagać danych w rejestrze, które nie zostały jeszcze zapisane, ponieważ jest to zadanie poprzedzającej instrukcji, która nie osiągnęła jeszcze tego kroku w potoku.

RISC - potokowanie (pipelininig)

Przykład:

- add $r3, $r2, $r1
  add $r5, $r4, $r3
  dalej więcej instrukcji niezależnych od dwóch pierwszych

- pierwsza instrukcja: dodaj zawartości rejestrów r1 i r2 i zapisz wynik w rejestrze r3.
- Druga, dodaj r3 i r4 i zapisaz sumę w r5.
- Umieszczamy ten zestaw instrukcji w potoku.
- Gdy druga instrukcja jest w drugim etapie, procesor będzie próbował odczytać r3 i r4 z rejestrów.

RISC - potokowanie (pipelininig)

Przykład:

- Pamiętaj jednak, że pierwsza instrukcja jest tylko o krok przed drugą, więc dodawana jest zawartość r1 i r2, ale wynik nie został jeszcze zapisany do rejestru r3.
- Dlatego druga instrukcja nie może czytać z rejestru r3, ponieważ nie została jeszcze zapisana i musi czekać, aż dane, których potrzebuje, zostaną zapisane.
- W konsekwencji potok zostaje zatrzymany i pewna liczba pustych instrukcji (znanych jako bąbelki) trafia do potoku.
- Zależność danych wpływa na długie potoki bardziej niż na krótsze, ponieważ potrzeba więcej czasu, zanim instrukcja osiągnie końcowy etap zapisu do rejestru w długim potoku.

RISC - potokowanie (pipelininig)

- Rozwiązaniem MIPS dla tego problemu jest zmiana kolejności kodu.
- Jeśli, jak w powyższym przykładzie, poniższe instrukcje nie mają nic wspólnego z pierwszymi dwoma, kod może zostać przeorganizowany tak, aby te instrukcje były wykonywane pomiędzy dwoma zależnymi instrukcjami, a potok mógł działać wydajnie.
- Zadanie zmiany kolejności kodu jest generalnie pozostawione kompilatorowi, który rozpoznaje zależności danych i stara się zminimalizować przerwy w wydajności.

RISC - potokowanie (pipelininig)

- Instrukcje rozgałęzienia to te, które mówią procesorowi, aby podjął decyzję o tym, jaka kolejna instrukcja ma zostać wykonana, na podstawie wyników innej instrukcji.
- Instrukcje rozgałęzienia mogą być kłopotliwe w potoku, jeśli rozgałęzienie jest uzależnione od wyników instrukcji, która nie zakończyła jeszcze swojej ścieżki w potoku.

RISC - potokowanie (pipelininig)

- **Przykład:**

```
Loop :
add $r3, $r2, $r1
sub $r6, $r5, $r4
beq $r3, $r6, Loop
```

- dodaj r1 i r2 i umieścił wynik w r3
- odjmij r4 od r5, przechowując różnicę w r6.
- beq oznacza skok na warunek (rozgałęzienie) , jeśli r3 i r6 są równe, procesor powinien skoczyc do etykiety „Loop”. W przeciwnym razie powinien przejść do następnej instrukcji.
- W tym przykładzie procesor nie może podjąć decyzji, którą gałąź wybrać, ponieważ do rejestrów r3 i r6 nie zostały jeszcze zapisane wartości z poprzednich instrukcji.

RISC - potokowanie (pipelininig)

- Procesor może się zawiesić, ale bardziej wyrafinowaną metodą radzenia sobie z instrukcjami rozgałęzień jest przewidywanie rozgałęzień.
- Procesor zgaduje, którą ścieżkę wybrać - jeśli się pomyli, to wszystko, co zostało zapisane w rejestrach, musi zostać wyczyszczone, a potok musi zostać uruchomiony ponownie z poprawną instrukcją.
- Niektóre metody przewidywania gałęzi zależą od zachowania stereotypowego.
- Gałęzie skierowane do tyłu (pętle) zajmują około 90% czasu, ponieważ gałęzie skierowane do tyłu często znajdują się na dole pętli.
- Z drugiej strony, gałęzie skierowane do przodu są tylko w około 50% przypadków.

RISC - potokowanie (pipelininig)

- Dlatego logiczne byłoby, aby procesory zawsze podążały za gałęzią, gdy wskazuje ona wstecz, ale nie, gdy wskazuje do przodu.
- Inne metody przewidywania gałęzi są mniej statyczne: procesory korzystające z przewidywania dynamicznego przechowują historię dla każdej gałęzi i używają jej do przewidywania przyszłych gałęzi.
- Te procesory mają rację w swoich przewidywaniach w 90% przypadków.

- Jeszcze inne procesory rezygnują z całej próby przewidywania gałęzi.
- System RISC/6000 pobiera i rozpoczyna dekodowanie instrukcji z obu stron gałęzi.
- Kiedy określa, którą gałąź należy zastosować, wysyła odpowiednie instrukcje w dół potoku do wykonania.

RISC - potokowanie (pipelininig)

- Aby procesory były jeszcze szybsze, opracowano różne metody optymalizacji potoków.
- Superpipelining odnosi się do podziału potoku na więcej etapów.
- Im więcej jest etapów potoku, tym szybszy jest potok, ponieważ każdy etap jest wtedy krótszy.
- W idealnym przypadku potok z pięcioma etapami powinien być pięć razy szybszy niż procesor bez potoku (lub raczej potok z jednym etapem).
- Instrukcje są wykonywane z szybkością, z jaką każdy etap jest zakończony, a każdy etap zajmuje jedną piątą czasu, jaki zajmuje instrukcja niepotokowa.

RISC - potokowanie (pipelininig)

- W ten sposób procesor z 8-stopniowym potokiem (MIPS R4000) będzie nawet szybszy niż jego 5-stopniowy odpowiednik.
- MIPS R4000 dzieli swój potok na więcej części, dzieląc kilka kroków na dwa.
- Na przykład pobieranie instrukcji odbywa się teraz w dwóch etapach, a nie w jednym.
- Etapy są przedstawione niżej:
  1. **Instruction Fetch (First Half)**
  2. **Instruction Fetch (Second Half)**
  3. **Register Fetch**
  4. **Instruction Execute**
  5. **Data Cache Access (First Half)**
  6. **Data Cache Access (Second Half)**
  7. **Tag Check**
  8. **Write Back**

RISC - potokowanie (pipelininig)

- Pobieranie instrukcji (pierwsza połowa)
- Pobieranie instrukcji (druga połowa)
- Zarejestruj się Pobierz
- Wykonaj instrukcję
- Dostęp do pamięci podręcznej danych (pierwsza połowa)
- Dostęp do pamięci podręcznej danych (druga połowa)
- Sprawdzanie tagów
- Zapisz do pamięci podręcznej danych

Superskalarność

tps://pl.wikipedia.org/wiki/Superskalarno%C5%9B%C4%87

- **Superskalarność** (ang. *superscalar*) – cecha mikroprocesorów oznaczająca możliwość jednoczesnego wykonywania kilku rozkazów maszynowych, realizowana poprzez zwielokrotnienie skalarnych jednostek wykonawczych[1].
- Pierwszym procesorem Intela z rodziny x86 wykorzystującym fragmentaryczną superskalarność był procesor Pentium, który miał dwie jednostki wykonawcze, z czego jedną zubożoną, mogącą wykonywać tylko proste instrukcje; Pentium Pro posiadał już 3 jednostki wykonawcze.
- Większość procesorów superskalarnych nie ma w pełni zduplikowanych jednostek wykonawczych kodu, jednak mogą mieć one wiele ALU, jednostek zmiennopozycyjnych itp., wobec czego pewne instrukcje będą wykonywane znacznie szybciej, a inne znów wolniej.

Superskalarność

- Pełne wykorzystanie wszystkich jednostek wykonawczych zależy od tego, czy w programie nie występują zależności między kolejnymi instrukcjami – tj. czy kolejna instrukcja jako argumentu nie potrzebuje wyników poprzedniej. Np. instrukcje:

```
a = b + 5
c = a + 10
```

nie będą mogły zostać wykonane równolegle, ponieważ wartość c zależy od wyliczanej wcześniej a. Jeśliby jednak usunąć zależność i napisać równoważnie:

```
a = b + 5
c = b + 15
```

realizacja superskalarna tych instrukcji będzie możliwa.

Superskalarność

- Minimalizacja zależności jest kluczowa, aby możliwe było pełne użycie dostępnych zasobów mikroprocesora – o właściwe rozmieszczenie instrukcji dba programista lub kompilator.
- Ponadto współczesne procesory, np. Pentium Pro i nowsze, mogą zmieniać kolejność wykonywanych instrukcji zachowując jednak zależności między nimi
- – aby w pełni wykorzystać jednostki wykonawcze odpowiednie mechanizmy wyszukują instrukcje niezależne od siebie i wykonują je równolegle.

RISC - potokowanie (pipelininig)

- Potoki superskalarne obejmują wiele równoległych potoków.
- Wewnętrzne elementy procesora są replikowane, dzięki czemu może on uruchamiać wiele instrukcji w niektórych lub wszystkich etapach potoku.
- RISC System/6000 ma rozgałęziony potok z różnymi ścieżkami dla instrukcji zmiennoprzecinkowych i całkowitych.
- Jeśli w programie występuje mieszanka obu typów, procesor może utrzymywać oba rozgałęzienia działające jednocześnie.

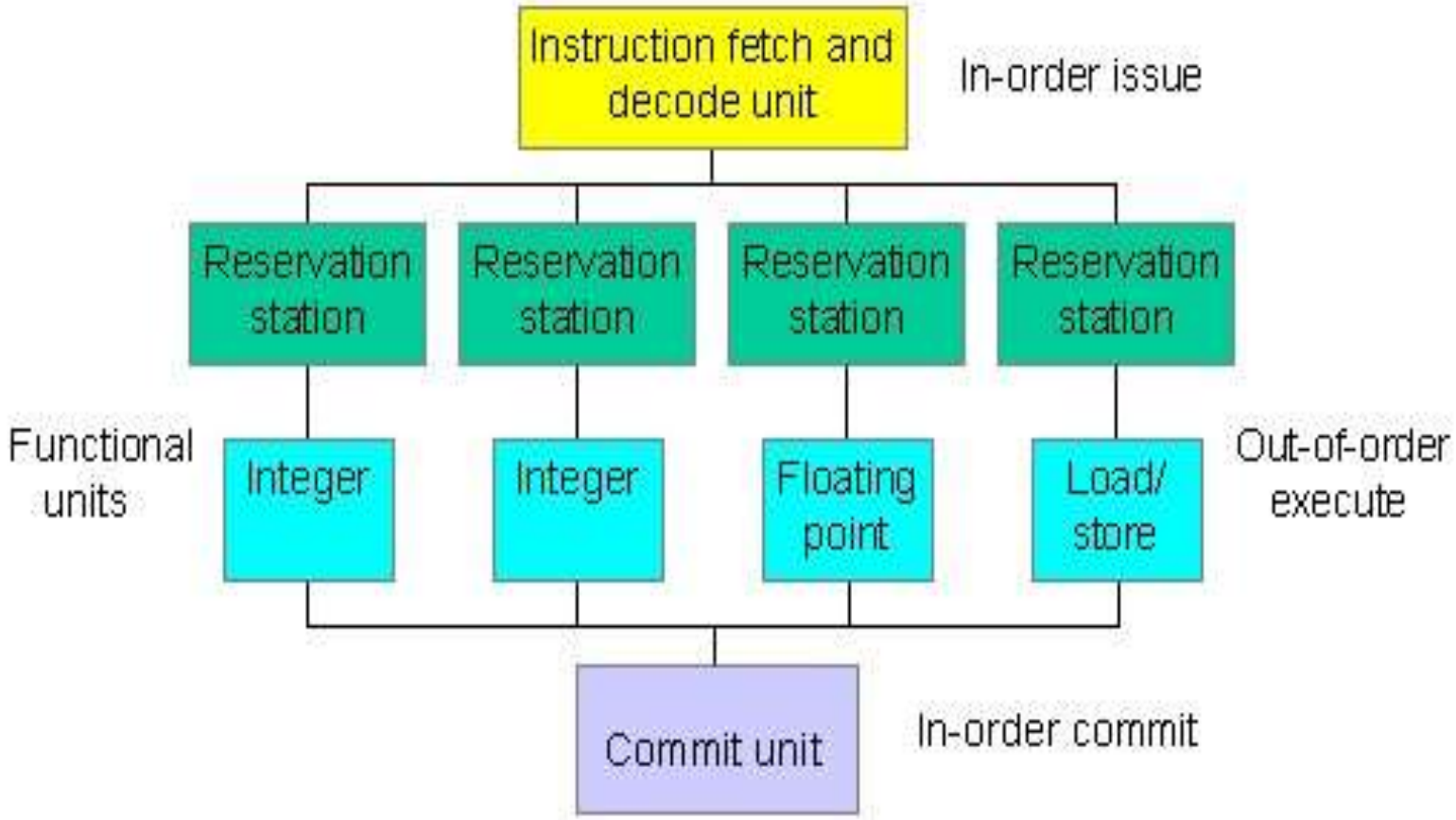
RISC - potokowanie (pipelininig)

- Oba typy instrukcji dzielą dwa początkowe etapy (pobieranie instrukcji i wysyłanie instrukcji) przed rozgałęzieniem.
- Często jednak superskalarne tworzenie potoków odnosi się do wielu kopii wszystkich etapów potoku (w przypadku prania oznaczałoby to cztery pralki, cztery suszarki i cztery osoby składające ubrania).
- Wiele współczesnych procesorów próbuje znaleźć od dwóch do sześciu instrukcji, które może wykonać na każdym etapie potoku.
- Jeżeli jednak niektóre z instrukcji są zależne, wykonywane są tylko pierwsze instrukcje.

RISC - potokowanie (pipelininig)

- Potoki dynamiczne mają możliwość planowania w czasie przestojów.
- Taki dynamiczny potok jest podzielony na trzy jednostki: jednostkę pobierania i dekodowania instrukcji, od pięciu do dziesięciu jednostek wykonawczych lub funkcjonalnych, oraz jednostkę zatwierdzającą.
- Każda jednostka wykonawcza ma stacje rezerwacji, które działają jako bufory i przechowują operandy i operacje.

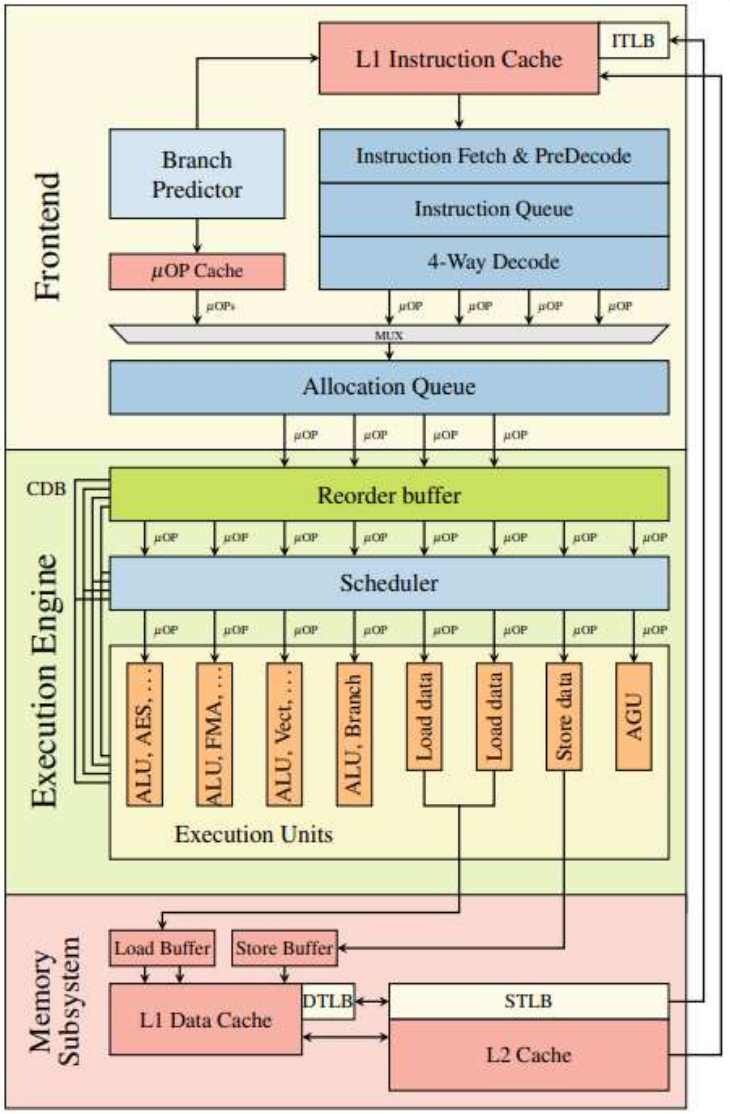
RISC - potokowanie (pipelininig)



RISC - potokowanie (pipelininig)

- Podczas gdy jednostki funkcjonalne mają swobodę wykonywania poza kolejnością, jednostki pobierania/dekodowania i zatwierdzania instrukcji muszą działać w celu utrzymania prostego zachowania potoku.
- Kiedy instrukcja jest wykonywana i wynik jest obliczany, jednostka zatwierdzająca decyduje, kiedy można bezpiecznie przechowywać wynik.
- W przypadku utknięciam, procesor może zaplanować wykonanie innych instrukcji, dopóki problem nie zostanie rozwiązany.
- To, w połączeniu z wydajnością wielu jednostek wykonujących instrukcje jednocześnie, sprawia, że dynamiczny potok jest atrakcyjną alternatywą

- Uproszczona ilustracja pojedynczego rdzenia mikroarchitektury Intel Skylake. Instrukcje są dekodowane na µOP-y i wykonywane poza kolejnością w silniku wykonawczym przez poszczególne jednostki wykonawcze.



Wykonywanie poza kolejnością - podsumowanie

- **Wykonywanie poza kolejnością** (ang. Out-of-order execution) to technika optymalizacji, która pozwala maksymalnie wykorzystać wszystkie jednostki wykonawcze rdzenia procesora.
- Zamiast przetwarzać instrukcje ściśle w sekwencyjnej kolejności programu, procesor wykonuje je, gdy tylko dostępne są wszystkie wymagane zasoby.
- Gdy jednostka wykonawcza bieżącej operacji jest zajęta, inne jednostki wykonawcze mogą wykonywać inne operacje. Stąd instrukcje mogą być wykonywane równolegle, o ile nie korzystają ze wspólnych zasobów (są niezależne).
- W praktyce wykonywanie poza kolejnością może być nawet spekulatywnie w pewnym zakresie.
- Potem jest albo zatwierdzane bądź odwołane, np. następne instrukcje w warunkowych skokach, zanim zakończy sie ewaluacja warunku, wykonywana jest najbardziej prawdopodobna (w tej sytuacji) instrukcja.

Wykonywanie spekulatywne

- https://pl.wikipedia.org/wiki/Wykonywanie_spekulatywne

- **Wykonywanie spekulatywne** (ang. *speculative execution*) – zdolność mikroprocesorów,przetwarzających potokowo instrukcje maszynowe programu, do wykonywania instrukcji znajdujących się już po skoku warunkowym, co do którego jeszcze nie wiadomo, czy nastąpi, a więc czy kolejne instrukcje zostaną kiedykolwiek wykonane.
- Ten mechanizm jest zwykle stosowany wraz z mechanizmem prognozowania skoków/odgałęzień (ang. *branch prediction*), który bazuje na historii realizacji skoków w programie, do optymalizacji wydajności i wykorzystania zasobów systemu.
- Ostatecznie wyniki uzyskane z wyprzedzeniem zostaną albo uwzględnione, albo odrzucone – w zależności od tego czy skok zostanie wykonany zgodnie z wynikiem mechanizmu predykcji skoków, czy też nie.

Wykonywanie spekulatywne

- Przykład w pseudokodzie, gdzie użycie określenia `adres xxx:` pozwala na uproszczenie opisu adresacji instrukcji programu znajdujących się w pamięci operacyjnej, a `...` to ciąg dowolnych instrukcji tego programu:

```
    a := 0
adres 100:
    a := a + 1
    c := c - 1
    ...
    jeśli a < 10 wtedy skok_do_adresu_100
    jeśli c = 20 wtedy skok_do_adresu_200
adres 120:
    b := 0
    ...
adres 200:
    c := 0
```

Wykonywanie spekulatywne

- W powyższym przykładzie, gdy mechanizm predykcji skoków określi, że jest duże prawdopodobieństwo wykonania skoku do adresu 200, to do potoku wykonawczego procesora zostaną wstawione instrukcje od adresu 200, a nie od adresu 120.

- Jeśli jednak w rzeczywistej realizacji programu okaże się, że skok do adresu 200 nie nastąpi (wartość c jest różna od 20), to potok zostaje wyczyszczony, a tym samym wyniki tego fragmentu kodu anulowane, i ładowane są do niego instrukcje od adresu 120.

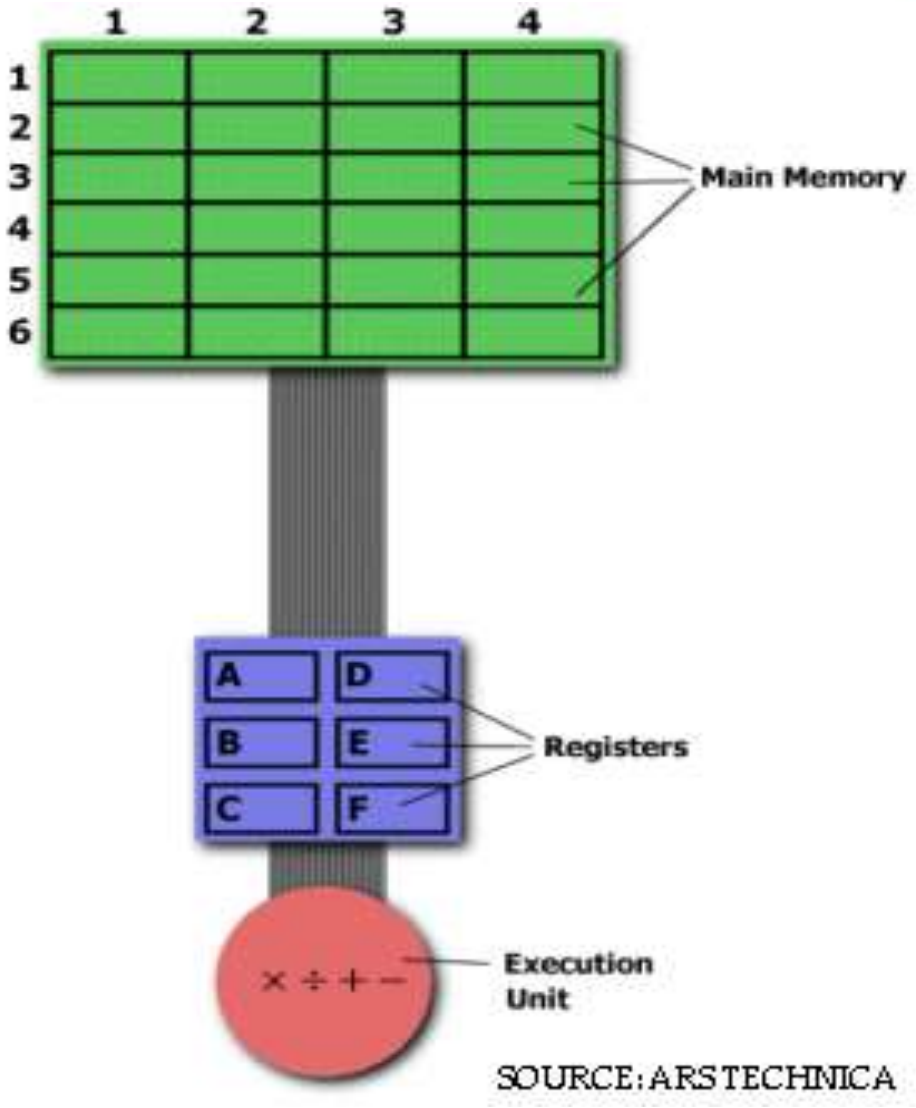
- Tego typu przetwarzanie jest stosowane w nowoczesnych mikroprocesorach, które mają zdolność wykonywania instrukcji poza kolejnością, do optymalnego wykorzystywania jednostek wykonawczych i uzyskiwania dzięki temu maksymalnej wydajności.

RISC versus CISC

Najprostszym sposobem zbadania zalet i wad architektury RISC jest porównanie jej z jej poprzedniczką: architekturą CISC (Complex Instruction Set Computers).Mnożenie dwóch liczb w pamięci RAM

RISC versus CISC

- Obok - diagram przedstawiający schemat przechowywania dla zwykłego komputera.
- Pamięć główna podzielona jest na lokalizacje ponumerowane od (wiersz) 1: (kolumna) 1 do (wiersz) 6: (kolumna) 4.
- Jednostka wykonawcza odpowiada za wykonanie wszystkich obliczeń.
- Jednak jednostka wykonawcza może działać tylko na danych, które zostały załadowane do jednego z sześciu rejestrów (A, B, C, D, E lub F).
- Załóżmy, że chcemy znaleźć iloczyn dwóch liczb — jeden przechowywany w lokalizacji 2:3, a drugi w lokalizacji 5:2 — a następnie przechowywać go z powrotem w lokalizacji 2:3.



RISC versus CISC

- Podejście CISC
- Podstawowym celem architektury CISC jest wykonanie zadania w jak najmniejszej liczbie linii kodu asemblera.
- Osiąga się to poprzez konstrukcję sprzętu procesora, który jest w stanie zrozumieć i wykonać szereg operacji.
- Do tego konkretnego zadania procesor CISC byłby przygotowany z określoną instrukcją (nazwiemy ją "MULT").
- Przy wykonaniu, instrukcja ta ładuje dwie wartości do oddzielnych rejestrów, mnoży operandy w jednostce wykonawczej, a następnie przechowuje produkt w odpowiednim rejestrze.
- W ten sposób całe zadanie mnożenia dwóch liczb można wykonać za pomocą jednej instrukcji:

```
MULT 2:3, 5:2
```

RISC versus CISC

- MULT to „złożona instrukcja”.
- Działa bezpośrednio na komórkach pamięci komputera i nie wymaga od programisty jawnego wywoływania funkcji ładowania lub przechowywania.
- Bardzo przypomina polecenie w języku wyższego poziomu.
- Na przykład, jeśli „a” reprezentuje wartość 2:3, a „b” reprezentuje wartość 5:2, to polecenie to jest identyczne z poleceniem w języku C

„a = a * b”.

RISC versus CISC

- Jedną z głównych zalet tego systemu jest to, że kompilator musi wykonać bardzo mało pracy, aby przetłumaczyć instrukcję języka wysokiego poziomu na asembler.
- Ponieważ długość kodu jest stosunkowo krótka, do przechowywania instrukcji wymagana jest bardzo mała pamięć RAM.
- Nacisk kładziony jest na budowanie skomplikowanych instrukcji bezpośrednio w sprzęcie.

RISC versus CISC

- **Podejście RISC**
- Procesory RISC używają tylko prostych instrukcji, które można wykonać w jednym cyklu zegara.
- Tak więc opisane powyżej polecenie „MULT” można podzielić na trzy oddzielne polecenia:
  - „LOAD”, które przenosi dane z banku pamięci do rejestru,
  - „PROD”, które znajduje iloczyn dwóch argumentów znajdujących się w rejestrach, oraz
  - „ STORE”, który przenosi dane z rejestru do banków pamięci.
- Aby wykonać dokładnie serię kroków opisanych w podejściu CISC, programista musiałby zakodować cztery linie asemblera:

```
LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A
```

RISC versus CISC

- Na pierwszy rzut oka może się to wydawać znacznie mniej wydajnym sposobem wykonania operacji.
- Ponieważ jest więcej linii kodu, więcej pamięci RAM jest potrzebne do przechowywania instrukcji poziomu asemblera.
- Kompilator musi również wykonać więcej pracy, aby przekonwertować instrukcję języka wysokiego poziomu na kod tej postaci.

RISC versus CISC

CISC

Nacisk na sprzęt
Zawiera multi-zegar
złożone instrukcje
Pamięć do pamięci:
"LOAD" oraz "STORE"
włączone do instrukcji
Małe rozmiary kodów,
wiele cykli zegara
Tranzystory (microcode)
używane
do przechowywania
złożonych instrukcji

RISC

Nacisk na oprogramowanie
Pojedynczy zegar,
tylko skrócona instrukcja
rejestr do rejestru:
"LOAD" i "STORE"
są niezależnymi
instrukcjami
Instrukcja na cykl,
duże rozmiary kodu
Więcej tranzystorów
w rejestrach pamięci

RISC versus CISC

CISC

- wiele operacji wielocyklowych
- mikro-kodowane wielocyklowe operacje
- register-mem i mem-mem
- wiele trybów
- wiele formatów i długości
- ręczne składanie kodu, aby uzyskać dobre wykonanie
- kilka rejestrów

RISC

- Wykonanie jednocyklowe
- Sterowanie hardwarowe
- Architektura Load/Store
- Tylko kilka trybów adresowania pamięci
- Format instrukcji/operacji o stałej długości
- Poleganie na kompilatorze w celu optymalizacji wykonania
- Wiele rejestrów (dobre dla kompilatorów)

RISC versus CISC

- Podejście RISC niesie ze sobą również kilka bardzo ważnych korzyści.
- Ponieważ każda instrukcja wymaga tylko jednego cyklu zegara do wykonania, cały program zostanie wykonany w przybliżeniu w tym samym czasie, co wielocyklowe polecenie „MULT”.
- Te "instrukcje zredukowane" RISC wymagają mniej tranzystorów w chipie procesora niż instrukcje złożone, pozostawiając więcej miejsca na rejestry ogólnego przeznaczenia.
- Ponieważ wszystkie instrukcje są wykonywane w jednakowym czasie (tj. jeden zegar), możliwe jest potokowanie

RISC versus CISC

- Oddzielenie instrukcji „LOAD” i „STORE” w rzeczywistości zmniejsza ilość pracy, którą musi wykonać komputer.
- Po wykonaniu polecenia "MULT" w stylu CISC, procesor automatycznie kasuje rejestry.
- Jeśli jeden z operandów ma być użyty do innego obliczenia, procesor musi ponownie załadować dane z banku pamięci do rejestru.
- W RISC operand pozostanie w rejestrze, dopóki w jego miejsce nie zostanie załadowana inna wartość.

RISC versus CISC

- **Równanie wydajności**
- Następujące równanie jest powszechnie używane do wyrażania wydajności **komputera:**

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

- Podejście CISC próbuje zminimalizować liczbę instrukcji na program, poświęcając liczbę cykli na instrukcję.
- RISC działa odwrotnie, zmniejszając liczbę cykli na instrukcję kosztem liczby instrukcji na program.

RISC versus CISC

- **Podsumowując**

  - CISC (complex instruction set computer)
    - VAX, Intel X86, IBM 360/370, etc.
  - RISC (reduced instruction set computer)
    - MIPS, DEC Alpha, SUN Sparc, IBM 801

RISC versus CISC

- **10 najlepszych (najczęściej używanych) instrukcji dla 8086:**

° Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

RISC versus CISC

Porównanie

Reduced Instruction Set Computer

RISC-V = Reduced Instruction Set Computer (RISC)

- ≈ 200 instrukcji po 32 bity,
- 4 formaty
- wszystkie operandy w rejestrach
  - prawie wszystkie mają po 32 bity
- ≈ 1 tryb adresowania: Mem[reg + wartość natychmiastowa]

RISC versus CISC

Porównanie cd

x86 = Complex Instruction Set Computer (ClSC)

- > 1000 instrukcji,
  - każda od 1 do 15 bajtów
- operandy w dedykowanych rejestrach ogólnego przeznaczenia
- rejestry, pamięć, na stosie, …może mieć 1, 2, 4, 8 bajtów, ze znakiem lub bez
- 10s trybów adresowania
  - np. Mem[segment + reg + reg * skala + przesunięcie]

RISC versus CISC

- **Przeszkody na drodze RISC**
- Pomimo zalet przetwarzania opartego na RISC,
- chipy RISC czekały ponad dekadę, aby zdobyć przyczółek w świecie komercyjnym.
- Wynikało to w dużej mierze z braku wsparcia oprogramowania.

RISC versus CISC

- **Przeszkody na drodze RISC**
- Chociaż linia Power Macintosh firmy Apple zawierała chipy oparte na RISC,
- a Windows NT był kompatybilny z RISC,
- Windows 3.1 i Windows 95 zostały zaprojektowane z myślą o procesorach CISC.
- Wiele firm nie chciało zaryzykować nowej technologii RISC.
- Bez zainteresowania komercyjnego, twórcy procesorów nie byli w stanie wyprodukować chipów RISC w wystarczająco dużych ilościach, aby ich cena była konkurencyjna.

RISC versus CISC

- **Przeszkody na drodze RISC**
- Kolejną poważną przeszkodą była obecność Intela.
- Chociaż ich chipy CISC stawały się coraz bardziej nieporęczne i trudne do opracowania,
- Intel miał zasoby, aby przebić się przez rozwój i produkować potężne procesory.
- Chociaż chipy RISC mogły przewyższyć wysiłki Intela w określonych obszarach, różnice nie były wystarczająco duże, aby przekonać kupujących do zmiany technologii.

RISC versus CISC

- **Ogólna zaleta RISC**
- Obecnie Intel x86 jest prawdopodobnie jedynym układem, który zachowuje architekturę CISC.
- Wynika to przede wszystkim z postępu w innych obszarach technologii komputerowej.
- Cena pamięci RAM drastycznie spadła.
- W 1977 roku 1 MB pamięci DRAM kosztował około 5000 USD.
- Do 1994 r. ta sama ilość pamięci kosztowała tylko 6 USD (po uwzględnieniu inflacji).
- Technologia kompilatora stała się również bardziej wyrafinowana, dzięki czemu wykorzystanie pamięci RAM w architekturze RISC i nacisk na oprogramowanie stały się idealne.

RISC vs. CISC w erze PC i post-PC

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Firmy AMD i Intel wykorzystały 500-osobowe zespoły projektowe i doskonałą technologię półprzewodnikową, aby zniwelować różnicę w wydajności między procesorami x86 i RISC.
- Zainspirowani korzyściami wynikającymi z potokowania prostych i złożonych instrukcji, zaprojektowali dekoder instrukcji tłumaczący na bieżąco złożone instrukcje x86 na wewnętrzne mikroinstrukcje typu RISC.
- następnie AMD i Intel zrealizowały w swoich procesorach potokowe wykonanie mikroinstrukcji RISC.

RISC vs. CISC w erze PC i post-PC

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Wszelkie pomysły, które projektanci RISC wykorzystywali dla wydajności, oddzielne pamięci podręcznej (cache) instrukcji i danych, pamięci podręcznej drugiego poziomu na chipie, głębokie potoki oraz pobieranie i wykonywanie kilku instrukcji jednocześnie, mogły zostać włączone do x86.
- AMD i Intel dostarczały około 350 milionów mikroprocesorów x86 rocznie w szczytowym momencie ery PC w 2011 roku.
- Duże wolumeny i niskie marże przemysłu PC oznaczały również niższe ceny niż komputery oparte na RISC.

RISC vs. CISC w erze PC i post-PC

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext
- Biorąc pod uwagę setki milionów komputerów PC sprzedawanych każdego roku na całym świecie, oprogramowanie komputerowe stało się gigantycznym rynkiem.
- Podczas gdy dostawcy oprogramowania dla rynku Unix oferowaliby różne wersje oprogramowania dla różnych komercyjnych RISC ISAsAlpha, HP-PA, MIPS, Power i SPARC, rynek komputerów PC cieszył się jednym ISA, więc twórcy oprogramowania dostarczali oprogramowanie „shrink wrap”, które było kompatybilne binarnie tylko z ISA x86.
- Znacznie większa baza oprogramowania, podobna wydajność i niższe ceny sprawiły, że do 2000 r. x86 zdominował zarówno rynek komputerów stacjonarnych, jak i małych serwerów.

RISC vs. CISC w erze PC i post-PC

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Apple pomógł zapoczątkować erę post-PC z iPhonem w 2007 roku.
- Zamiast kupować mikroprocesory, producenci smartfonów zbudowali własne systemy na chipie (SoC) przy użyciu projektów innych firm, w tym procesorów RISC firmy ARM.
- Projektanci urządzeń mobilnych cenili powierzchnię matrycy i efektywność energetyczną tak samo jak wydajność, co niekorzystnie wpływało na CISC ISA.
- Co więcej, pojawienie się Internetu Rzeczy znacznie zwiększyło zarówno liczbę procesorów, jak i wymagane kompromisy w zakresie rozmiaru matrycy, mocy, kosztów i wydajności.

RISC vs. CISC w erze PC i post-PC

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Tendencja ta zwiększyła znaczenie czasu i kosztów projektowania, co jeszcze bardziej niekorzystnie wpływało na procesory CISC.
- W dzisiejszej erze post-PC, dostawy x86 spadały o prawie 10% rocznie od szczytu w 2011 roku, podczas gdy chipy z procesorami RISC wzrosły do 20 miliardów.
- Obecnie 99% 32-bitowych i 64-bitowych procesorów to procesory RISC.

RISC vs. CISC w erze PC i post-PC

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Podsumowując ten historyczny przegląd, możemy powiedzieć, że rynek rozstrzygnął debatę RISC-CISC;
- CISC wygrał późne etapy ery PC,
- ale RISC wygrywa erę post-PC.
- Od dziesięcioleci nie było nowych certyfikatów CISC ISA.

- Konsensus co do najlepszych zasad ISA dla procesorów ogólnego przeznaczenia jest dziś nadal RISC, **35 lat po ich wprowadzeniu**.

Konwergencja CISC i RISC

- Najnowocześniejsza technologia procesorowa znacznie się zmieniła od czasu wprowadzenia chipów RISC na początku lat 80-tych.
- Ponieważ wiele ulepszeń jest używanych zarówno przez procesory RISC, jak i CISC, granice między tymi dwiema architekturami zaczęły się zacierać.
- Wydaje się, że obecnie obie architektury niemalże przyjęły strategie drugiej strony.

Konwergencja CISC i RISC

- Ponieważ szybkość procesora wzrosła, układy CISC są teraz w stanie wykonać więcej niż jedną instrukcję w ramach jednego zegara.
- Pozwala to również chipom CISC na korzystanie z potoków.
- Dzięki innym ulepszeniom technologicznym, możliwe jest teraz umieszczenie o wiele więcej tranzystorów na jednym chipie.
- Daje to procesorom RISC wystarczająco dużo miejsca na wprowadzanie bardziej skomplikowanych, podobnych do CISC poleceń.

Konwergencja CISC i RISC

- Chipy RISC wykorzystują również bardziej skomplikowany sprzęt, wykorzystując dodatkowe jednostki funkcyjne do wykonywania superskalarnego.
- Wszystkie te czynniki skłoniły niektóre grupy do twierdzenia, że jesteśmy teraz w erze „post-RISC”, w której oba style stały się tak podobne, że rozróżnienie między nimi nie ma już znaczenia.

Konwergencja CISC i RISC

- Należy jednak zauważyć, że chipy RISC nadal zachowują kilka ważnych cech.
- Chipy RISC ściśle wykorzystują jednolite instrukcje jednocyklowe.
- Zachowują również architekturę typu rejestr-rejestr, ładowanie/przechowywanie (Load/Store). I pomimo rozszerzonych zestawów instrukcji, chipy RISC nadal mają dużą liczbę rejestrów ogólnego przeznaczenia.

Konwergencja CISC i RISC

Jednoczesna wielowątkowość

- Jednoczesna wielowątkowość (Simultaneous Multi-Threading SMT) umożliwia jednoczesne wykonywanie wielu wątków.
- Wątki to szereg zadań wykonywanych naprzemiennie przez procesor.
- Normalne wykonywanie wątków wymaga włączania i wyłączania wątków na procesorze, ponieważ pojedynczy proces (wątku) dominuje nad procesorem przez chwilę.

Konwergencja CISC i RISC

Jednoczesna wielowątkowość

- Pozwala to na bardziej wydajne wykonywanie niektórych zadań, które wymagają oczekiwania (na dostęp do dysku lub użycie sieci).
- SMT umożliwia jednoczesne wykonywanie wątków, przeciągając instrukcje do potoku z różnych wątków.
- W ten sposób wiele wątków postępuje w swoich procesach i żaden wątek nie dominuje nad procesorem w danym momencie.

Konwergencja CISC i RISC

Przewidywanie wartości

- Przewidywanie wartości jaką wygeneruje dana instrukcja ładowania.
- Wartości ładowania na ogół nie są losowe, a około połowa instrukcji ładowania w programie pobiera tę samą wartość, co w poprzednim wykonaniu.

Konwergencja CISC i RISC

Przewidywanie wartości

- Zatem przewidywanie, że wartość obciążenia będzie taka sama, jak ostatnio, przyspiesza procesor, ponieważ umożliwia komputerowi kontynuowanie pracy bez konieczności oczekiwania na dostęp do pamięci ładowania.
- Ponieważ ładowanie jest zwykle jedną z najwolniejszych i najczęściej wykonywanych instrukcji, to ulepszenie powoduje znaczną różnicę w szybkości procesora.

Konwergencja CISC i RISC

- Współpraca Intela i RISC-V zmienia zasady gry.

- Dzisiaj RISC-V z interesującego podejścia do architektury procesorów staje się dominująca architekturą.

Konwergencja CISC i RISC

- Architektura została x86 zaimplementowana w procesorach Intel, Cyrix, AMD, VIA Technologies i wielu innych firmach; istnieją również implementacje otwarte, takie jak platforma Zet SoC (obecnie nieaktywna).

- Niemniej jednak spośród nich tylko Intel, AMD, VIA Technologies i DM&P Electronics posiadają licencje na architekturę x86,
  - ale tylko dwie pierwsze z nich nadal produkują nowoczesne procesory 64-bitowe.

Konwergencja CISC i RISC

- Od 2021 r. większość sprzedawanych komputerów stacjonarnych, laptopów i konsol do gier (z wyjątkiem Nintendo Switch) opiera się na architekturze x86 podczas gdy kategorie mobilne, takie jak smartfony lub tablety, są zdominowane przez ARM;
- x86 nadal dominuje w segmentach stacji roboczych intensywnie korzystających z mocy obliczeniowej i w chmurze obliczeniowej,
- podczas gdy najszybszy superkomputer jest oparty na architekturze ARM; trzy następne nie są już oparte na x86

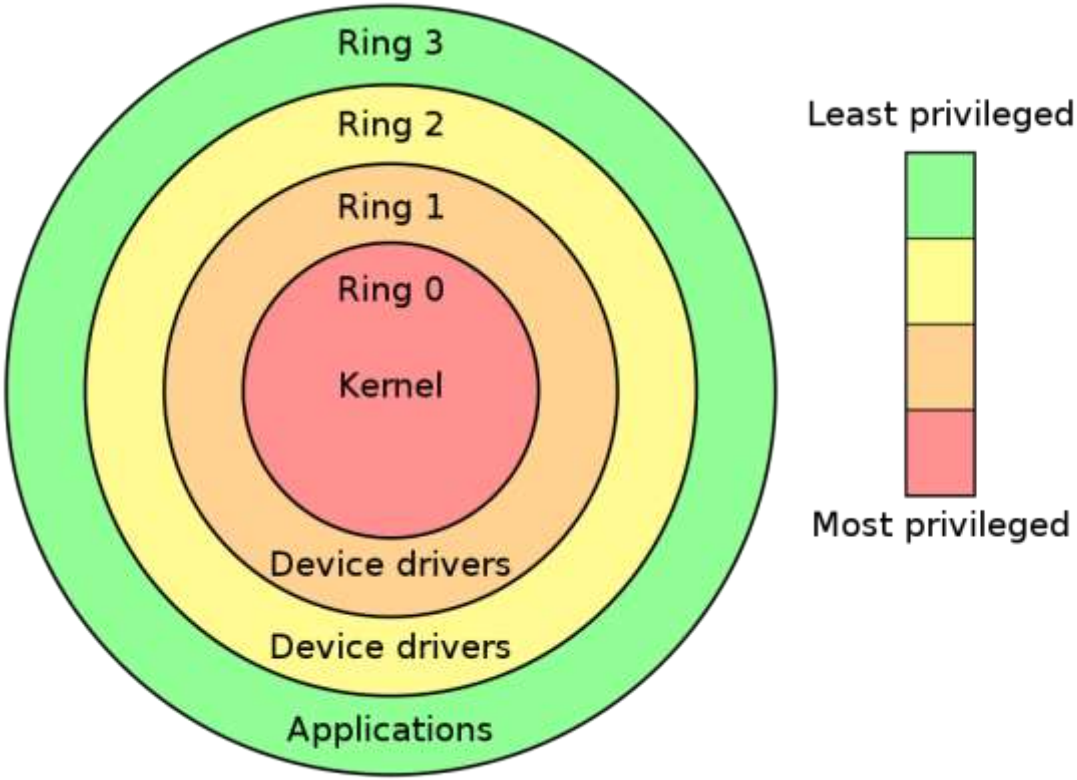
Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- W latach siedemdziesiątych architekci procesorów skupili się na zwiększeniu bezpieczeństwa komputera za pomocą pierścieni ochronnych.
- Ci architekci dobrze zrozumieli, że większość błędów będzie znajdować się w oprogramowaniu, ale wierzyli, że wsparcie architektoniczne może pomóc.
- Funkcje te były w dużej mierze nieużywane przez systemy operacyjne, które celowo koncentrowały się na rzekomo łagodnych środowiskach (takich jak komputery osobiste), a funkcje te wiązały się wówczas ze znacznym obciążeniem, więc zostały wyeliminowane.

Przeoczone bezpieczeństwo

- https://en.wikipedia.org/wiki/Protection_ring#/media/File:Priv_rings.svg
- Privilege rings for the x86 available in protected mode
- generally hardware-enforced by some CPU architectures that provide different CPU modes at the hardware or microcode level.



Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- W społeczności programistów wiele osób uważało, że weryfikacja formalna i techniki, takie jak mikrojądra, zapewnią skuteczne mechanizmy tworzenia wysoce bezpiecznego oprogramowania.
- Niestety, skala współczesnych systemów informacyjnych i dążenie do wydajności oznaczało, że takie techniki nie nadążały za wydajnością procesora.
- W rezultacie duże systemy oprogramowania nadal mają wiele luk w zabezpieczeniach,
- a efekt jest spotęgowany przez ogromną i rosnącą ilość danych osobowych w Internecie oraz korzystanie z przetwarzania w chmurze, które udostępnia fizyczny sprzęt potencjalnym przeciwnikom.

Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext
- Chociaż architekci komputerowi i inni być może powoli zdawali sobie sprawę z rosnącego znaczenia bezpieczeństwa, zaczęli uwzględniać obsługę sprzętową maszyn wirtualnych i szyfrowania.
- Niestety spekulacje wprowadziły nieznaną, ale istotną lukę w zabezpieczeniach wielu procesorów.
- W szczególności luki w zabezpieczeniach Meltdown i Spectre doprowadziły do powstania nowych luk, które wykorzystują luki w mikroarchitekturze, umożliwiając szybki wyciek chronionych informacji. niewidocznych na poziomie ISA
- W 2018 roku badacze pokazali, jak wykorzystać jeden z wariantów Spectre do wycieku informacji przez sieć bez wczytywania kodu przez atakującego na docelowy procesor.

Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Chociaż ten atak, zwany NetSpectre, ujawnia informacje powoli, fakt, że pozwala on dowolnej maszynie na ta sama sieć lokalna (lub w tym samym klastrze w chmurze), która ma zostać zaatakowana, tworzy wiele nowych luk w zabezpieczeniach.
- Następnie zgłoszono dwie kolejne luki w architekturze maszyn wirtualnych.
- Jedna z nich, o nazwie Foreshadow, umożliwia penetrację mechanizmów bezpieczeństwa Intel SGX zaprojektowanych w celu ochrony danych o największym ryzyku (takich jak klucze szyfrowania).
- Co miesiąc odkrywane są nowe luki w zabezpieczeniach.

Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Ataki boczne nie są nowe, ale w większości wcześniejszych przypadków luka w oprogramowaniu pozwalała na powodzenie ataku.
- W atakach Meltdown, Spectre i innych jest to wada w implementacji sprzętu, która ujawnia chronione informacje.
- Istnieje zasadnicza trudność w sposobie, w jaki architekci procesorów określają, jaka jest prawidłowa implementacja ISA,
- ponieważ standardowa definicja nie mówi nic o skutkach wydajności wykonania sekwencji instrukcji, a jedynie o widocznym przez ISA stanie architektury wykonania.

Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Architekci muszą przemyśleć swoją definicję prawidłowej implementacji ISA, aby zapobiec takim błędom w zabezpieczeniach.
- Jednocześnie powinni przemyśleć ilr uwagi poświęcają na bezpieczeństwo komputerów, i jak architekci mogą współpracować z projektantami oprogramowania w celu wdrożenia bezpieczniejszych systemów.
- Architekci (i wszyscy inni) zbytnio polegają na większej liczbie systemów informatycznych, aby dobrowolnie pozwolić, aby bezpieczeństwo było traktowane jako coś mniej niż pierwszorzędny problem projektowy.

Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

Wniosek końcowy

- „Najciemniejsza godzina jest tuż przed świtem”. Tomasz Fuller, 1650

- Aby czerpać korzyści z lekcji historii, architekci muszą docenić, że innowacje w zakresie oprogramowania mogą również inspirować architektów,
- że podniesienie poziomu abstrakcji interfejsu sprzęt/oprogramowanie daje możliwości dla innowacji,
- oraz że rynek ostatecznie rozstrzyga debaty na temat architektury komputerowej.
- iAPX-432Itanium ilustrują, w jaki sposób inwestycje w architekturę mogą przekroczyć ewentualne zyski,
- podczas gdy S/360, 8086 i ARM zapewniają wysokie roczne zwroty przez dziesięciolecia bez końca.

Przeoczone bezpieczeństwo

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Koniec skalowania Dennarda i prawa Moore'a oraz spowolnienie wzrostu wydajności standardowych mikroprocesorów to nie są problemy, które trzeba rozwiązać, lecz nowe architektury komputerów
  - skalowanie Dennarda: jeśli gęstość tranzystorów podwoi się, zużycie energii (przy dwukrotnej liczbie tranzystorów) pozostaje takie samo.
  - Prawo Moore'a mówi, że liczba tranzystorów podwaja się mniej więcej co dwa lata.

- Języki i architektury wysokiego poziomu, specyficzne dla domeny, uwalniające architektów z zastrzeżonych zestawów instrukcji, wraz z zapotrzebowaniem na poprawę bezpieczeństwa, zapoczątkują nowy złoty wiek dla architektów komputerowych.
- Wspomagane przez ekosystemy open source, sprawnie opracowane chipy w przekonujący sposób zademonstrują postęp, a tym samym przyspieszą komercyjne zastosowania.

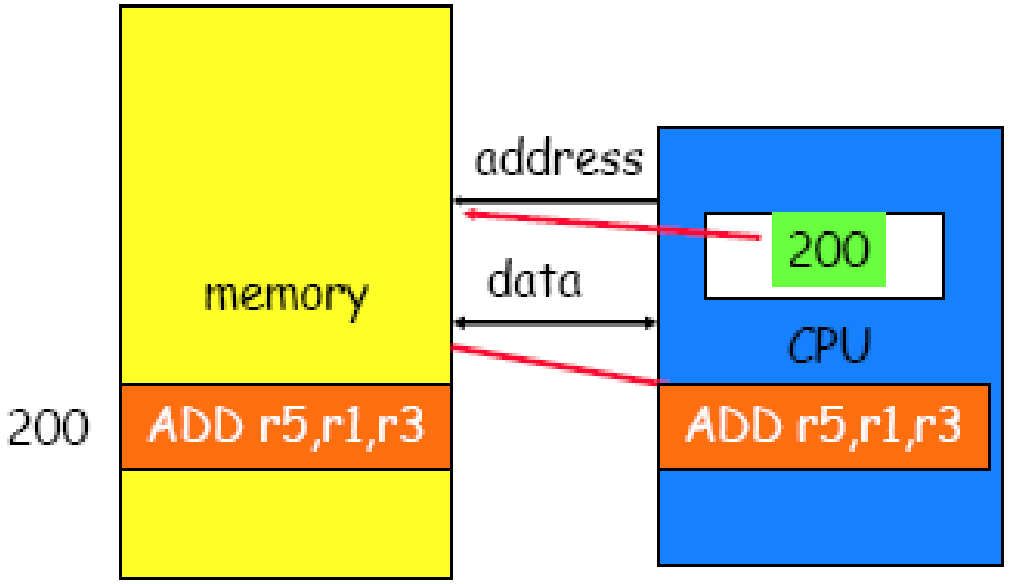
Przeoczone bezpieczeństwo

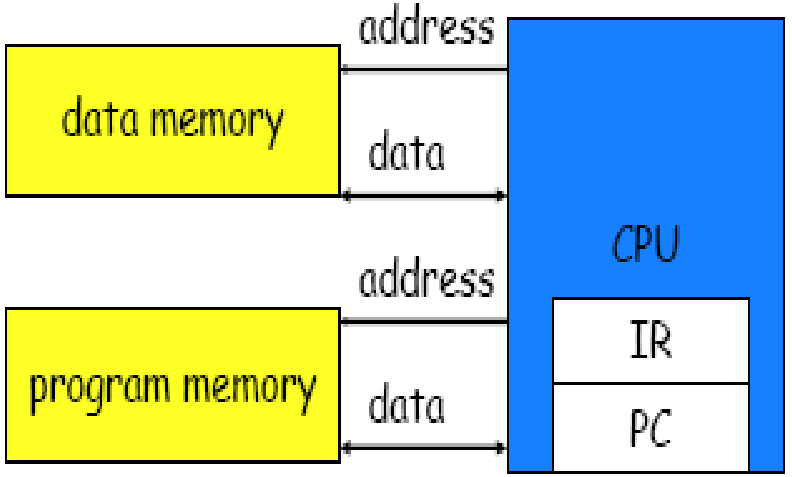
- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Nowym podejściem do projektowania ISA dla procesorów ogólnego przeznaczenia będzie prawdopodobnie RISC, który przetrwał próbę czasu.
- Oczekuj takiej samej szybkiej poprawy, jak w ostatnim złotym wieku, ale tym razem pod względem kosztów, energii i bezpieczeństwa, a także wydajności.

- **W następnej dekadzie nastąpi kambryjska eksplozja nowatorskich architektur komputerowych,**
- **co oznacza ekscytujące czasy dla architektów komputerowych w środowisku akademickim i przemysłowym.**

von Neumann versus Hardward





Dziękuję za uwagę!

Slajdy na podstawie wykładów prof. Stanisława Ambroszkewicza