Architektura komputerów

Wykład 5

Jan Kazimirski

Podstawowe elementy komputera. Procesor (CPU) c.d.

Architektura CPU

- Jednostka arytmetyczno-logiczna (ALU)
- Rejestry
- Układ sterujący przebiegiem programu
- Szyna wewnętrzna procesora
- Inne bloki (np. FPU, rozszerzenia multimedialne itp.)

Rejestry

- Rejestry dostępne dla użytkownika – dostępne dla programisty do tymczasowego przechowywania danych
- Rejestry statusu i kontrolne – wykorzystywane przez CPU do sterowania wykonywaniem instrukcji, niektóre używane przez system operacyjny (rozkazy uprzywilejowane)

Rejestry dostępne dla użytkownika

- Rejestry ogólnego przeznaczenia – bez określonej funkcji, wykorzystanie zależy od rozkazu.
- Rejestry danych – mogą przechowywać wyłącznie dane
- Rejestry adresów – mogą przechowywać wyłącznie adresy (np. rejestry segmentowe)
- Rejestry znaczników operacji – stosowane w skokach warunkowych

Wybór typów rejestrów

- Architektura oparta na rejestrach ogólnego przeznaczenia
  - Elastyczność
  - Dłuższe rozkazy (adresowanie rejestrów)
- Architektura oparta na rejestrach specjalizowanych
  - Mniejszy wybór rejestrów dla programisty
  - Krótkie rozkazy (domyślne rejestry)

Liczba rejestrów

- Duża liczba rejestrów zwiększa koszt CPU
- Duża liczba rejestrów zmniejsza liczbę odwołań do pamięci i daje większą elastyczność programiście
- Duża liczba rejestrów zwiększa długość rozkazu (wskazanie rejestru)
- Typowe liczby rejestrów 8-32

Rejestr znaczników

- Przechowuje znaczniki stanu procesora i status ostatnio wykonanej operacji arytmetycznej
- Przykładowe znaczniki: znaku, zera, parzystości, przeniesienia, kierunku zmiany rejestru indeksowego, pracy krokowej
- Wykorzystywany w instrukcjach warunkowych

Rejestry kontrolne

- Licznik programu (PC lub IP) – wskazuje następną instrukcję do wykonania
- Rejestr instrukcji (IR) – przechowuje aktualnie wykonywaną instrukcję
- Rejestr adresu pamięci (MAR) – przechowuje adres komórki pamięci
- Rejestr bufora pamięci (MBR) – przechowuje daną odczytaną lub zapisywaną do pamięci

Rejestry kontrolne c.d.

- Inne rejestry kontrolne – zależnie od implementacji, np.:
  - Rejestry kontroli dostępu do pamięci, sterowania pamięcią podręczną, ochrony pamięci
  - Rejestry trybu pracy procesora
  - Rejestry służące do testowania i optymalizacji kodu
  - Inne ...

Przepływ danych w CPU
(1 argument, adresowanie bezpośrednie)

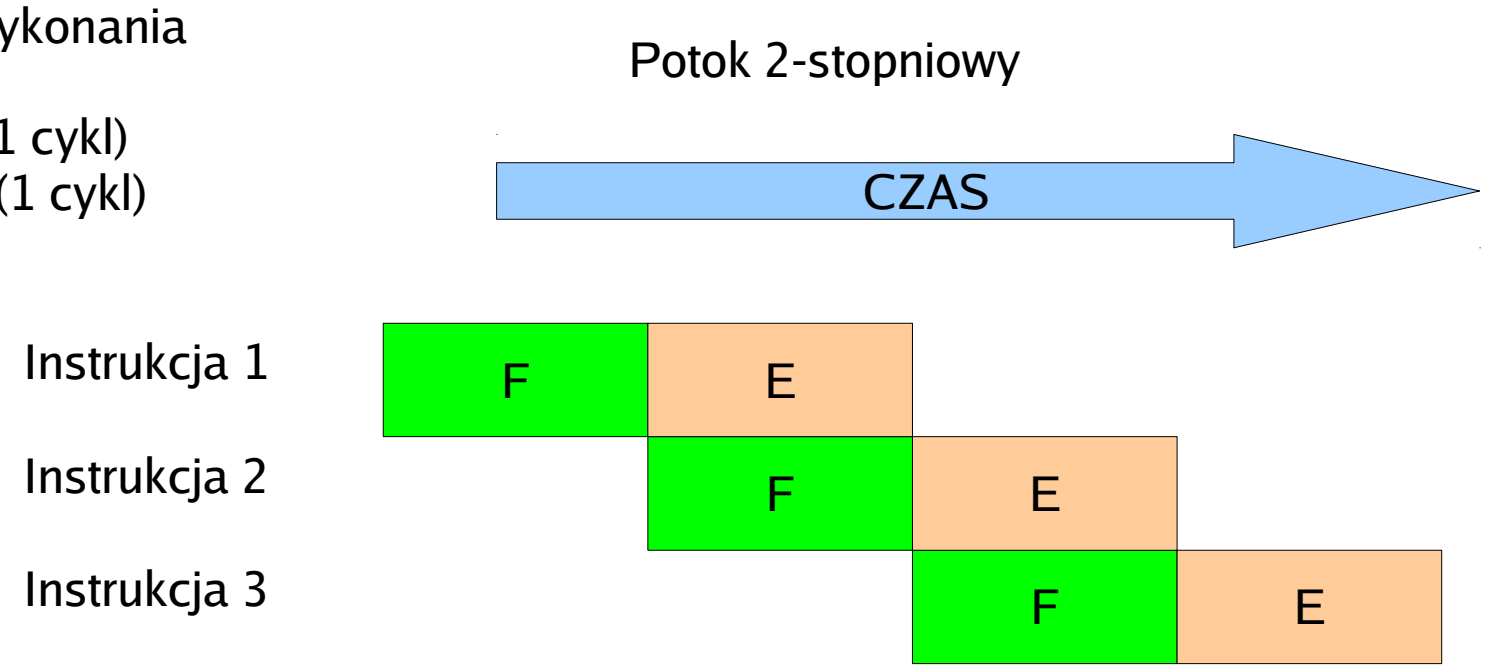
- Adres z rejestru PC umieszczany jest w MAR i wystawiany na szynę adresową
- Odczytane dane (rozkaz) umieszczane są w MBR a później kopiowane do IR. PC jest zwiększany o 1
- Adres argumentu umieszczany jest w MAR. Żądanie odczytu z pamięci
- Dane do operacji pobierane są z MBR
- Po wykonaniu operacji przejście do kolejnego rozkazu

Potoki

- Wykonanie instrukcji przez CPU składa się z kilku kolejnych etapów
- Etapy wykonywane są kolejno przez różne komponenty CPU
- Zwiększenie wydajności: rozpoczęcie wykonywania kolejnej instrukcji zanim zakończy się wykonywanie poprzedniej

Potok - przykład

Dwa etapy wykonania rozkazu:
F – pobierz (1 cykl)
E – wykonaj (1 cykl)



Sumaryczny czas wykonania kodu - 3*2 = 6 cykli
Z zastosowaniem potoku – 4 cykle

Potoki c.d.

- Bardziej złożone instrukcje można rozbić na wiele faz
- Przy większej liczbie faz wykonywania instrukcji potoki mogą być dłuższe
- Długie potoki = dalszy wzrost wydajności (**teoretyczny!**)

Przykładowe rozmiary potoków

- 5 – Pentium
- 8 – R4000, UltraSPARC T2
- 9 – UltraSPARC
- 10 – Pentium III
- 14 – Intel Core, Pentium PRO
- 20-31 – Pentium 4, Pentium D

Potoki - problemy

- Kolizje przy wykorzystaniu modułów CPU (np. ALU)
- Problem spójności danych – poprzednia instrukcja zmienia dane instrukcji następnej
- Problem skoków warunkowych – skoki całkowicie unieważniają potok – musi być opróżniony i wypełniany od nowa.

Problem z zasobami CPU

- Problem: Te same zasoby CPU używane w różnych fazach wykonywania instrukcji (np. ALU do wyliczania adresu i wykonywania obliczeń)
- Rozwiązanie: Umieszczenie dodatkowych elementów w CPU – np. sumator do wyznaczania adresu (nie korzysta z ALU).

Problem spójności danych

- Problem: Wykonanie kolejnych instrukcji zależy od danych wyliczonych (lub zmienionych) w instrukcji poprzedniej.
- Rozwiązanie: Mechanizmy kopiowania danych „w przód” w potoku, wprowadzanie „pustych przebiegów” w potokach lub reorganizacja kolejności wykonywania instrukcji

Skoki warunkowe

- Problem: Skok do innego niż kolejny adresu unieważnia potok
- Rozwiązanie:
  - Zwielokrotnienie liczby potoków
  - Wstępne pobieranie instrukcji docelowej skoku
  - Bufor pętli
  - Przewidywanie skoków
  - Opóźnione skoki

Potoki i instrukcje skoku c.d.

- Zwielokrotnienie liczby potoków
  - W przypadku skoku warunkowego w potokach umieszczane są obie instrukcje (dla obu wartości warunku)
  - Umożliwia pracę bez przestoju niezależnie od warunku
  - Problem z dostępem do rejestrów i kilkoma kolejnymi skokami warunkowymi
  - Implementowane m.in. IBM 370/168, IBM 3033

Potoki i instrukcje skoku c.d.

- Wstępne pobieranie instrukcji docelowej
  - Po detekcji instrukcji skoku warunkowego pobierana jest również instrukcja docelowa (razem z następną)
  - Jeżeli skok nastąpi to nie trzeba już pobierać instrukcji docelowej
  - Implementowane m.in. w IBM 360/91

Potoki i instrukcje skoku c.d.

- Bufor pętli
  - Niewielka pamięć podręczna dołączona do bloku pobierania instrukcji
  - W buforze przechowywane są ostatnio pobrane instrukcje (lub następne – przy wstępnym pobieraniu)
  - Skoki o niewielkim zasięgu będą obsłużone za pomocą bufora
  - Implementowane m.in. w CRAY-1

Potoki i instrukcje skoku c.d.

- Przewidywanie skoków
  - Statyczne
    - Never taken – założenie, że skoku nie będzie
    - Always taken – założenie, że skok zawsze będzie
    - By opcode – decyzja zależy od rodzaju skoku
  - Dynamiczne – na podstawie historii poprzednich skoków.
    - Bit skoku – poprzednie wykonanie
    - Tablica historii skoku – bardziej zaawansowana – historia wykonań instrukcji, adres docelowy

Potoki i instrukcje skoku c.d.

- Opóźnione skoki
  - Cel: maksymalne wykorzystanie potoku
  - Skok wykonywany jest później niż wynika to z kodu
  - Za instrukcją skoku warunkowego (w tzw. „branch delay slot”) umieszczane są instrukcje wykonywane niezależnie od warunku

CISC vs. RISC

CISC

- Rozwój architektury CISC (Complex Instruction Set Computer)
  - Wzrost kosztów oprogramowania względem sprzętu
  - Rozwój języków wysokiego poziomu
  - Próba dopasowania architektury sprzętu
    - Duża liczba instrukcji
    - Duża liczba trybów adresowania
    - Wsparcie dla instrukcji języków wysokiego poziomu

Charakterystyka działania programu

- Badania wydajności i charakterystyki wykonania programów pisanych w językach wysokiego poziomu
- Aspekty:
  - Liczba i rodzaj wykonywanych instrukcji
  - Liczba i rodzaj operandów
  - Charakterystyka przepływu sterowania (pętle itp.)

Charakterystyka ... c.d.

- Liczba i rodzaj wykonywanych instrukcji. Rezultaty badań:
  - Duży udział prostych instrukcji przypisania (proste przesłania danych)
  - Duży udział instrukcji warunkowych (skoki warunkowe)
  - Mniejszy udział skoków i wywołań podprogramów – ale duży koszt

Charakterystyka ... c.d.

- Liczba i rodzaj operandów. Rezultaty badań:
  - Większość odwołań do zmiennych skalarnych (80% w Pascalu i C)
  - Większość odwołań do zmiennych lokalnych w ramach procedury
  - Średni stosunek dostępów do operandów w pamięci - 0.5 i w rejestrach – 1.4 na instrukcję

Charakterystyka ... c.d.

- Przepływ sterowania. Rezultaty badań:
  - Wywołania procedur – bardzo kosztowne czasowo
  - Zwykle niewielka liczba parametrów wywołania
  - Parametry wywołania to zwykle wartości skalarne
  - Zwykle niewielka „głębokość” wywołań

Charakterystyka ... c.d.

- Wnioski:
  - Optymalizacja powinna dotyczyć najbardziej istotnych i najbardziej kosztownych instrukcji
  - Większa liczba rejestrów – lokalne odwołania, mniej dostępów do pamięci
  - Wydajna implementacja potoków – duża liczba instrukcji warunkowych
  - Mniejsza liczba instrukcji i trybów adresowania

Duża liczba rejestrów

- Duży udział prostych przypisań w programie – wykorzystanie rejestrów
- Podejście programowe – kompilator optymalizuje wykorzystanie rejestrów (najczęściej używane zmienne)
- Podejście sprzętowe – dostępna większa liczba rejestrów

Okna rejestrów

- Kilka zestawów rejestrów dla zmiennych lokalnych
- Wywołanie podprogramu zmienia zestaw rejestrów. Powrót przywraca poprzedni zestaw
- Nakładające się okna rejestrów – przekazywanie danych bez fizycznego transferu
- Bufor kołowy rejestrów
- Rejestry na zmienne globalne

CISC c.d.

- Motywy dla rozwoju CISC:
  - Uproszczenie kompilatorów ?
    - Problem: Złożone instrukcje maszynowe – mniejsza elastyczność
    - Problem: Trudniejsza optymalizacja
  - Mniejsze programy ?
    - Mniej instrukcji asemblera <> mniejszy kod
    - Skomplikowane, wielo-bajtowe instrukcje
    - Zmniejszenie kosztów pamięci – jej oszczędność nie jest priorytetem

CISC c.d.

- Motywy dla rozwoju CISC:
  - Szybsze programy ?
    - Bardziej złożona logika CPU
    - Bardziej złożone – wydajniejsze instrukcje
    - Zastosowanie mikrokodu
    - Efekt: Złożona budowa CPU – program wykorzystujący proste instrukcje będzie wykonywał się wolniej!

RISC - charakterystyka

- Jedna instrukcja na cykl
- Operacje rejestr-rejestr
- Mniej trybów adresowania
- Mniej różnych formatów instrukcji
- Brak mikrokodu (sprzętowa realizacja instrukcji)
- Stały rozmiar instrukcji

RISC c.d.

- Jedna instrukcja na cykl
  - Prosta sekwencja wykonania instrukcji – pobranie operandów z rejestrów, operacja, umieszczenie wyniku w rejestrze
  - Instrukcja RISC ~ mikroinstrukcja CISC
  - Sprzętowa – wydajna realizacja (brak mikrokodu)

RISC c.d.

- Operacje rejestr-rejestr
  - Operacje wykonywane na rejestrach – uproszczone instrukcje (vs. rejestr-rejestr, rejestr-pamięć, pamięć-pamięć w CISC)
  - Prosta implementacja
  - Tylko proste instrukcje LOAD i SAVE to transferów pamięć-rejestr.

RISC c.d.

- Mniej trybów adresowania
  - Bardziej złożone tryby adresowania (np. Indeksowe) realizowane programowo – kompilator
  - Więcej pracy dla kompilatora – mniej skomplikowane obwody CPU

RISC c.d.

- Prosty format instrukcji
  - Instrukcje o stałej długości – łatwe dekodowanie
  - Wskazanie rejestrów operandów w instrukcji – szybki dostęp do operandów
  - Optymalizacja pobierań instrukcji oparta na ich długości.
  - Optymalizacja stronicowania (instrukcja nie trafia na granicę strony).

RISC i potoki

- Prosty format instrukcji RISC pozwala na lepszą optymalizację potoków
- Techniki optymalizacji:
  - Reorganizacja kolejności instrukcji
  - Rozwijanie pętli
  - Opóźniony skok warunkowy (delayed branch) i transfer danych z pamięci (delayed load)

RISC vs. CISC

- Porównanie RISC i CISC jest trudne.
- Brak porównywalnych rozwiązań sprzętowych
- Duży wpływ jakości kompilatorów
- Współczesne rozwiązania nie są „czyste”
  - Architektury RISC implementują bardziej złożone instrukcje znane z CISC
  - Architektury CISC stosują rozwiązania RISC – jednostki kontrolne oparte o mikrokod

Architektura superskalarna

- Proste instrukcje (działające na danych skalarnych) mogą być wykonywane niezależnie w kilku potokach.
- Zwielokrotnienie jednostek wykonawczych – jednoczesne wykonywanie operacji
- Zmiana kolejności wykonywanych instrukcji

Superskalarność - ograniczenia

- Zależność danych (True data dependency) – rezultat wcześniejszej instrukcji jest operandem następnej
- Problem skoków warunkowych
- Problem z jednoczesnym dostępem do zasobów (szyny systemowe, cache, jednostki obliczeniowe itd.)

Parallelizm programowy i sprzętowy

- Równoległość na poziomie programu
  - Poszczególne instrukcje są od siebie niezależne
  - Kolejne instrukcje mogą być wykonywane jednocześnie
- Równoległość na poziomie procesora
  - Zdolność do efektywnej realizacji równoległości programowej
  - Liczba potoków i jednostek wykonawczych

Strategie pobierania instrukcji

- Strategia pobierania i wykonywania instrukcji – aspekty:
  - Kolejność pobierania instrukcji
  - Kolejność wykonywania instrukcji
  - Kolejność w której instrukcje modyfikują zawartość rejestrów i pamięci

Strategie pobierania instrukcji c.d.

- Pobieranie i wykonywanie instrukcji w kolejności
  - Najmniej wydajne rozwiązanie
  - Możliwe pobranie więcej niż jednej instrukcji
  - Konieczność wstrzymywania wykonywania instrukcji w przypadku zależności
- Pobieranie instrukcji w kolejności, wykonywanie poza kolejnością
  - Problem z zależnościami typu read-write

Strategie pobierania instrukcji c.d.

- Pobieranie i wykonywanie poza kolejnością
  - Odseparowanie potoku pobierania instrukcji od potoku wykonywania
  - Instrukcje mogą być pobierane z wyprzedzeniem
  - Po zwolnieniu jednostki wykonawczej wybierana jest jedna ze zdekodowanych instrukcji
  - Kolejność wykonywania instrukcji nie jest związana z ich ułożeniem w programie

Zmiana nazw rejestrów

- Wykonywanie instrukcji poza kolejnością (Out-of-Order execution) może prowadzić do problemów z zawartością rejestrów
- Wykonywane poza kolejnością instrukcje modyfikują rejestry w niewłaściwy sposób
- Rozwiązanie – dynamiczny przydział rejestrów, te same logiczne rejestry przypisywane są innym rejestrom fizycznym.

Superskalarność - podsumowanie

- Jednoczesne pobieranie wielu instrukcji
  - Zwielokrotnione potoki pobierania instrukcji
  - Pobieranie z wyprzedzeniem
  - Przewidywanie skoków warunkowych
- Zwielokrotnione potoki wykonawcze i współpracujące z nimi moduły
- Mechanizmy zapewniające spójność danych i poprawność operacji wykonywanych równolegle