Zagadnienia, które podał Patronik na wykładzie

1. Architektura von Neumanna:
   Dane przechowywane są wspólnie z instrukcjami, co sprawia, że są kodowane w ten sam sposób. Architektura ta składa się z 4 komponentów:
   a. Pamięci, która przechowuje dane programu oraz instrukcje, każda komórka ma adres,
   b. Jednostki sterującej, która przetwarza dane i instrukcje z pamięci,
   c. ALU, które wykonuje podstawowe operacje arytmetyczno-logiczne,
   d. Urządzeń I/O, które służą do komunikacji z operatorem.

   System komputerowy zbudowany na tej architekturze powinien:
   a. Mieć skończona i funkcjonalnie pełną listę rozkazów,
   b. Mieć możliwość wprowadzenia programu przez urządzenia zew.,
   c. Dane i instrukcje powinny być jednakowo dostępne dla CPU,
   d. Sekwencyjnie przetwarzać dane.
2. Architektura Harwardzka:
   Dane i instrukcje są przechowywane w różnych pamięciach, które mogą różnić się budową. Pamięć instrukcji zajmuje inną przestrzeń adresową niż pamięć danych. Przez co procesor może w tym samym czasie czytać instrukcje i pobierać dane.
3. Cechy przejrzystej architektury (AK 2 wykład 2):
   Spójność/regularność – dostępność spodziewanych rozkazów i operandów. Dostępne dodawanie z przeniesieniem i odejmowanie z pożyczką, używając takich samych operandów. Mnożenie dostępne jedynie z operandami rejestrowymi. Dzielenie również.
   Ortogonalność – niezależności funkcji i operandów. Niektóre tryby adresowania pamięci dostępne z niektórymi rozkazami. W architekturze ortogonalnej rozkaz odnoszący się do pamięci może używać dowolnego trybu adresowania. Wybór trybu adresowania jest niezależny od wyboru rozkazu. Pary rozkaz/operand są dostępne z takimi samymi trybami adresowania. Dodawanie z pewnymi trybami adresowania. Odejmowanie i inne operacje arytmetyczno-logiczne z pewnymi trybami adresowania (ortogonalność) oraz z takimi samymi trybami adresowania (regularność). Mnożenie jest wykonywane przy użyciu operandów rejestrowych, a dzielenie nie co znaczy że brak ortogonalności.
   Trafność – przeźroczystość(brak ograniczeń implementacyjnych, np. w rozmieszczeniu rozkazów w kodzie lub z zakresie danych). Oszczędność pozwala nad wykonywać operacje w zakresie jak najmniejszej ilości rejestrów np. add R1,R1.
   Ogólność – kompletność:
   Jest sprzeczna z oszczędnością(spodziewane rozkazy i operandy: dostępne i z tym samym trybem adresowania), trudna do zdefiniowania(możliwość wykonywania obliczeń w rozsądnym czasie, można wykonać inkrementacje zamiast dodawania), niedostępność rozkazów z powodu oszczędności(rozkazy potrzebne muszą być, ale mnożenie i dodawanie jest kosztowne sprzętowo).
   Ogólność – otwartość:
   Posiada dostępną przestrzeń dla przyszłych rozszerzeń. Ma regularną strukturę kodu(wolne miejsca w przestrzeni kodowej). Jednolite kodowanie argumentów(uniezależnienie działań od rozmiarów argumentów). Konsekwentny dobór funkcji(unikanie ograniczeń implementacyjnych i technologicznych).

4. RISC & CISC:

Zagadnienia, które podał Patronik na wykładzie

Architektura klasyczna CISC

- CISC – Complex Instruction Set Computer
- Lista rozkazów
  - rozkazy realizujące działania proste i skomplikowane
  - rozbudowane sposoby (tryby) adresowania
  - argumenty umieszczone zwykle w pamięci
  - stałe w dodatkowych słowach kodu
  - niejednolita struktura – różna liczba słów kodu
- Organizacja – rozwiązania intuicyjne
  - akumulator lub niewiele rejestrów uniwersalnych
  - większość argumentów w pamięci, rejestry specjalizowane
  - trudne buforowanie i dekodowanie rozkazów (zmienny rozmiar)
- Skutki
  - zmienny czas wykonania tych samych etapów przetwarzania
  - bariera przepustowości pamięci

Analiza wykonania rozkazów w rozwiązaniach klasycznych (CISC)

analiza:

- rozbudowana lista rozkazów i nieregularna struktura kodu
  - zmienny czas pobrania kodu i dekodowania rozkazu
  - skomplikowany dekoder (układ sekwencyjny)
  - skomplikowane układy wykonawcze, zmienny czas wykonania działań
- większość operandów w pamięci
  - częste konflikty dostępu podczas wykonania etapów F, R i W
  - długi czas wykonania etapów F, R i W

wnioski:

- uproszczona lista rozkazów oraz stały rozmiar i struktura słowa kodu:
  - stały czas pobrania kodu i dekodowania rozkazu
  - prosty dekoder kombinacyjny
  - proste układy wykonawcze, krótki czas wykonania podstawowych działań
- większość operandów w rejestrach procesora:
  - wyeliminowanie etapu R i krótszy czas etapu W (load/store)
  - rzadkie konflikty dostępu podczas wykonania etapów F, W

Zagadnienia, które podał Patronik na wykładzie

Koncepcja RISC

- RISC – Reduced Instruction Set Computer
  →!redukcja dotyczy różnorodności, a nie liczby instrukcji (racjonalizacja)!
- *Lista rozkazów*
  - rozkazy proste
  - proste tryby adresowania
  - specjalne rozkazy komunikacji z pamięcią
  - ograniczony zakres i krótkie kody stałych
  - jednolita struktura kodu
- *Organizacja* (zasada lokalności, buforowanie informacji)
  - dużo rejestrów uniwersalnych
  - buforowanie informacji (kolejka rozkazów, cache)
- *Korzyści*
  - prawie stały czas wykonania tych samych etapów przetwarzania
  - podobny czas wykonania różnych etapów przetwarzania
  - łatwa implementacja potoku
  - możliwe włączenie niektórych rozkazów CISC (o minimalnym wpływie na μarchitekturę)

Potokowe wykonanie rozkazów w architekturze RISC

Uproszczona lista rozkazów oraz stały rozmiar i struktura słowa kodu,

- prosty dekoder kombinacyjny
- proste układy wykonawcze, krótki czas wykonania działań

Większość operandów w rejestrach procesora

- wyeliminowanie etapu R i krótszy czas etapu W (load/store)
- konflikty dostępu tylko podczas wykonania rozkazów load/store (E/W)

Niezbędne skrócenie czasu pobierania kodu z pamięci

- buforowanie pamięci

Nieuniknione przestoje (buble) wskutek konfliktów

- spekulacyjne wykonanie niektórych rozkazów
- zmiana kolejności przetwarzania rozkazów

Przypuszczenie

- podział etapów F, D, E, W na podetapy
- wzrost przepustowości
- ...

Zagadnienia, które podał Patronik na wykładzie

5. Koncepcja pamięci wirtualnej:

Pamięć wirtualna

Wirtualna przestrzeń adresowa – suma logicznych przestrzeni pamięci (widzianych w programie stanowiącym treść procesu) translacja adresu wirtualnego (logicznego) na adres rzeczywisty odwzorowanie adresu wirtualnego w pamięci fizycznej (*real memory*).

6. Stronicowanie:

Stronicowanie polega na podziale pamięci na spójne bloki ustalonej wielkości. Powoduje to znaczne ułatwienia przy obsłudze pamięci, a także brak fragmentacji zewnętrznej , chociaż występuje fragmentacja wewnętrzna. Taki pojedynczy blok, na które jest podzielona pamięć nazywa się stroną, a jego wielkość określa stała PAGE_SIZE w pliku page.h.

7. Tryby adresowania: Opracowanie

Nazwa	Ile elementów	Przykład	Działanie	Schemat	ADDR	Uwagi
Zwarte (compact)	Zero-elem.					
Błyskawiczne (quick)	Zero-elem.					Ponoć używane w RISC
Natychmiastowe (immediate)	Zero-elem.	Movl $4, %eax	Eax = 4	CONST_REG	%eax	
Bezwzględne (absolute)	Jedno-elem.	Movl $4, (%eax)	Mem(eax)=4	CONST_ADDR	%eax	
Bezwzględne pośrednie (absolute indirect)	Jedno-elem.	Jmp (SF000)	Skok	JMP		Używane tylko przez JMP
Rejestrowe bezpośrednie (register direct)	Jedno-elem.	Mov %eax, %ebx	Ebx=eax	REG_REG	%ebx	
Rejestrowe pośrednie (register indirect)	Jedno-elem.	Mov %eax, (%ebx)	Mem(ebx)=eax	REG_ADDR	%ebx	Używane tylko przez bx,si,di
Rejestrowe pośrednie z modyfikacją (register indirect modified)	Jedno-elem.					
Bazowe z przemieszczeniem (based)	Wielo-elem.	Movl $0, 2(%eax)	Mem(eax+2)=0	CONST_ADDR	%eax+2	
Indeksowe z przemieszczeniem (indexed)	Wielo-elem.	Movl $30, a(,%edi,4)	a+(edi*4)=30	DATA_ARRAY (,REG,CONST)	a+(%edi*4)	
Bazowo-indeksowe (base-indexed)	Wielo-elem.	Movl $60, (%ebx,%edi,4)	Ebx+(edi*4)=60	DATA_POINT(BASE,INDEX, SIZE)	%ebx+(%edi*4)	
Względne z przemieszczeniem (PC-relative)	Wielo-elem.					
Względne indeksowe (PC based indexed)	Wielo-elem.					

8. Arytmetyka stałoprzecinkowa: AK1
9. Efekt szamotania: Opracowanie + tutaj

Zagadnienia, które podał Patronik na wykładzie

Szamotanie - Proces się szamoce jesli spędza więcej czasu na stronicowaniu niż na wykonaniu.

Biernat:
Szamotanie następuje gdy:
Suma zbiorów roboczych procesów aktywnych > rozmiar dostępnej pamięci
Heurystyka:
Nie wymieniaj bloku, który jest częścią zbioru roboczego aktywnego procesu i nie uaktywniaj procesu, którego zbiór roboczy nie może zostać w całości odwzorowany w pamięci głównej

Warunki wstępne -
System dąży do utrzymania wykorzystania procesora ´ – Jesli użycie procesora spada – zwiększany jest stopień wieloprogramowości. Uzywany jest algorytm globalnego zastępowania stron. Jesli proces potrzebuje więcej ramek pamięci – może je odebrać innym procesom ´ Mechanizm powstawania szamotania • Jeden z procesów zwiąksza zapotrzebowanie na ramki pamięci: – Ramki są zabierane innym procesom – Zblizając się do minimalnej ilości ramek, procesy częściej wykonują stronicowanie ´ • Wzrost zapotrzebowania na urządzenie stronicujące: – Procesy oczekują w kolejce do pamięci pomocniczej – Spadek wykorzystania procesora Mechanizm powstawania szamotania • System obserwuje spadek wykorzystania procesora: – Następuje zwiększenie stopnia wieloprogramowosci. . . ´ • Nowe procesy potrzebują ramek pamięci: – Dalsze ograniczanie ilosci dostępnych ramek ´ – Dalsze nasilenie stronicowania • System utyka – procesy cały czas sp˛edzaj ˛a na stronicowaniu Mechanizm powstawania szamotania 12

10. Zasada Lokalności:

Zasada lokalności

W *komputerze z programem zintegrowanym* programy i dane mają tendencję do skupiania w wymiarze przestrzennym i czasowym

- **lokalność przestrzenna** (*spatial locality*)
  Jest prawdopodobne użycie informacji z sąsiednich lokacji pamięci
  - kody rozkazów – zależność *lokacyjna sekwencyjna* (licznik rozkazów)
  - struktury danych – skupione (zmienne robocze) lub regularne (tablice)
- **lokalność czasowa** (temporal locality)
  Kod użyty będzie zapewne ponownie użyty w nieodległym czasie
  - kody rozkazów – pętle programowe
  - struktury danych
    – zmienne robocze: ciągłe używanie
    – struktury regularne: wielokrotne użycie elementów

WNIOSEK: Utworzyć w pobliżu *procesora* **bufor** zawierający **kopie danych** z pamięci (głównej) aktualnie używanych i korzystać z kopii zamiast z oryginału

Zagadnienia, które podał Patronik na wykładzie

11. Aspekty czasowe wykonywania kodu (np. czas wykonania wszystkich iteracji pętli): Opracowanie
12. Model programowy x64 i model abstrakcyjny:

Model programowy procesora

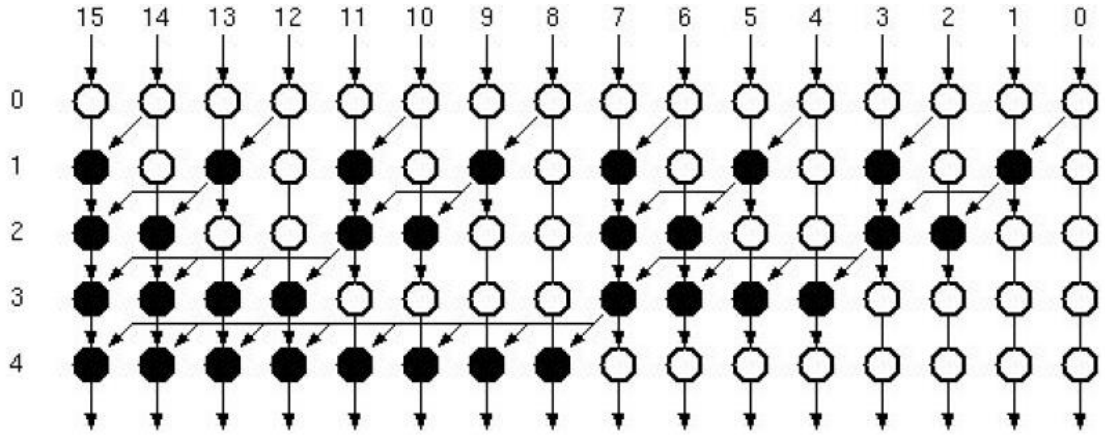
wykaz rejestrów i ich cechy

- rejestry ogólnego przeznaczenia *GPR* (*general purpose registers*)
- rejestry specyficzne (niespójność architektury)

tryby adresowania

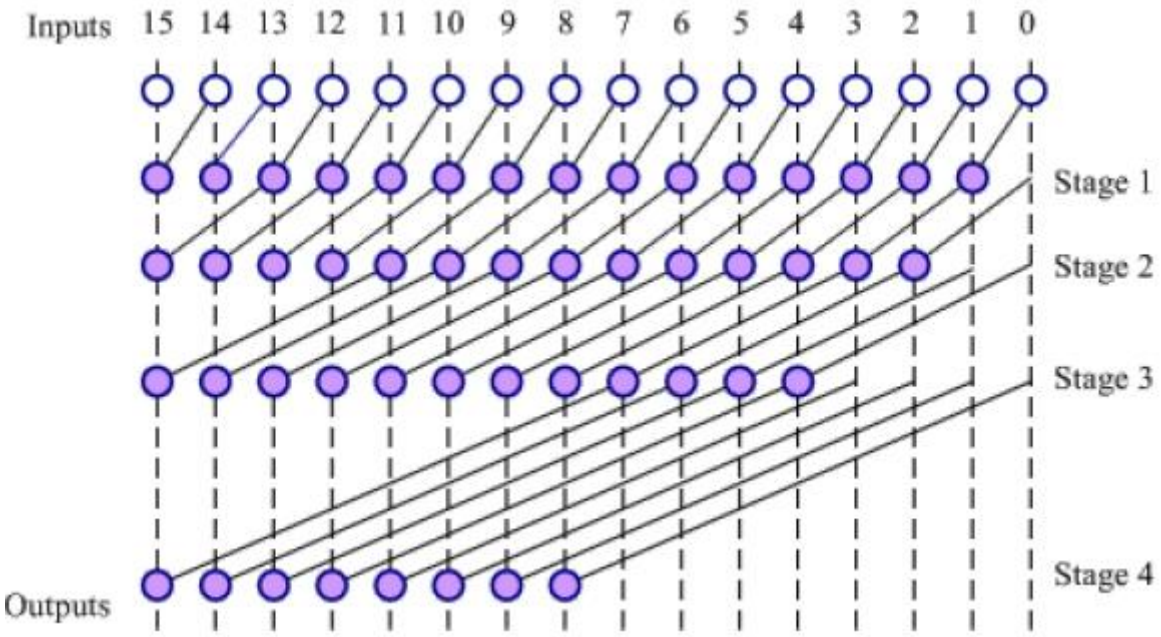
- sposoby tworzenia adresu w pamięci (głównej)
- ograniczenia (niespójność architektury) specyfikacja działań
- sposób tworzenia wyniku i jego syndromów
  - zasady
  - odstępstwa od zasad (niespójności architektury)
- ograniczenia użycia argumentów
  - nakaz użycia
  - zakaz użycia
- interpretacja argumentów
  - interpretacja kodów liczb
  - sposób tworzenia i przekształcania stałych (rozszerzenia kodu)

13. Sumatory prefiksowe: AK1
    Sklansky:

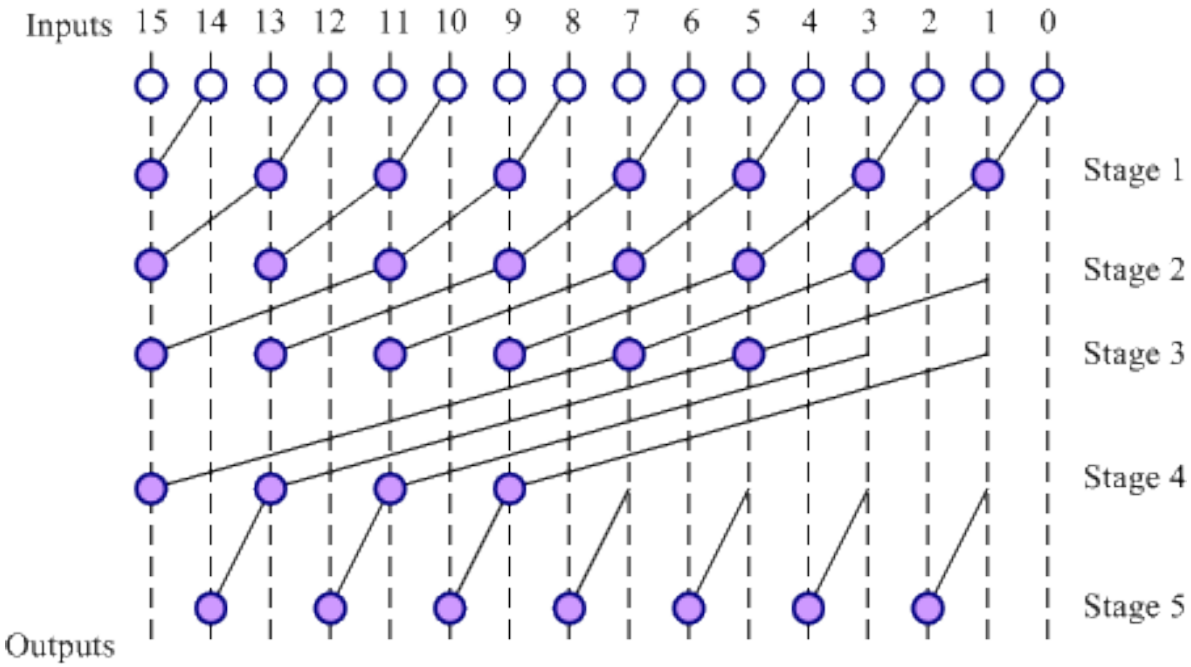


Zagadnienia, które podał Patronik na wykładzie

Kogge'a-Stone'a:



Carlsona:



14. IEEE 754: AK1
15. Kodowanie liczb: AK1
16. Specjalne wartości zmiennoprzecinkowe: AK1

Wartośc	Znak	Wykładnik	Mantysa
NaN	+/-	1……..1	Niezerowa
+/- INF	+/-	1……..1	Zerowa
+/- ZERO	+/-	0……..0	Zerowa
Zdenormalizowana	+/-	0……..0	Niezerowa (ukryte 0)

Zagadnienia, które podał Patronik na wykładzie

17. Koncepcja pętli i rozgałęzień: Wykład 8

Rozgałęzienia

decyzje → zmiana porządku instrukcji

- wytworzenie warunku – porównanie, wykonanie działania
- wybór warunku – jawne lub implikowane wskazanie przesłanki
- użycie warunku – rozkazy warunkowe
  - jawne wykonanie rozgałęzienia (instrukcja), ominięcie
    - rozgałęzienie zwykłe (branch), skok warunkowy
      if *warunek* **then goto** *adres*
    - rozgałęzienie ze śladem (branch & link)
      if *warunek* **then goto** *adres* and **link**
  - warunkowe wykonanie instrukcji
    if *warunek* **then** *polecenie*
    - pułapka (trap) – warunkowe wykonanie funkcji
      if *warunek* **then call** *exception*
  - przechowanie stanu logicznego warunku (spełniony – niespełniony) **if** *warunek* **then** zmienna:= *TRUE* **else** zmienna:= *FALSE*

18. Pamięć i ochrona pamięci:

Ochrona procesu

Spójność systemu (*system integrity*)

- bezpieczeństwo systemu (*security*)
- prywatność procesów (*privacy*)

Ochrona zasobów:

- zapobieganie (*prevention*) naruszeniu spójności systemu
- reagowanie na ingerencję w mechanizm ochrony
  - wykrywanie (*detection*) błędów i ataków
  - rozpoznawanie i neutralizacja skutków ingerencji
  - unieważnianie (*nullify*) działań ingerujących w mechanizm ochrony

Ochrona zasobów na poziomie architektury maszyny rzeczywistej:

- w przestrzeni kodów – uniemożliwienie wykonania instrukcji uprzywilejowanych (*privileged*) w procesie użytkownika
- w przestrzeni operandów – wykluczenie wykonania w trybie użytkownika operacji używających zastrzeżonych operandów
- w przestrzeni danych – przypisanie każdej danej znacznika (*tag*), sprzeczne z klasyczną koncepcją pamięci

19. Przełączanie procesów:

Zagadnienia, które podał Patronik na wykładzie

Jeden procesor, wiele zadań (procesów)

Procesy współbieżne – jeden rozpoczyna się przed zakończeniem drugiego

Poprawność procesu (programu współbieżnego)

- własność żywotności
  - każde oczekiwane zdarzenie (działanie) nastąpi
  - każdy proces ma realną szansę wykonania (sprawiedliwość, uczciwość)
  - własność bezpieczeństwa
  - program jest zawsze w stanie pożądanym

Wymagania – warunki konieczne poprawności programu sekwencyjnego

- żywotność – każdy proces zostanie wykonany → przydział procesora
  - na czas wykonania procesu – możliwe zagłodzenie innych procesów
  - okresowo na ustalony czas – podział czasu (time-sharing)
- bezpieczeństwo – ochrona procesu
  - prywatność
    – osobne przestrzenie adresowe → przestrzeń wirtualna
    – ograniczenie dostępu: tryb dostępu: (r-w-x)