POSTECH 기출문제 정리

2023-24학년도 일반1차 대학원 입시 구술 시험 문제

💻 컴퓨터구조

(10 pts) ISA의 설계 방식 중 CISC와 비교할 때 Reduced Instruction Set Computer (RISC)의 장단점을 각 두가지씩 설명하시오.

장점

- 필수적이고 중요한 Instruction만 가지고 있기에 사용 빈도에 대한 효율이 좋음
- Instruction이 규격화되어 있기에 Pipelining을 적용할 수 있음
- Instruction이 간단하기에 1-clock 내외로 Instruction 실행 가능

단점

- 비교적 적은 수의 Instruction을 가지고 있기에 하나의 프로그램을 실행하기 위해 Instruction을 많이 사용해야 함
- 하나의 Process를 실행할 때 필요한 Instruction 수가 많기에 메모리 공간을 절약하기 어려움
- 적은 수의 Addressing mode를 지원함 (메모리 접근 최소화, 단순화 → Register 적극 활용)

[CISC]

장점: 적은 수의 Instruction으로 프로그램을 실행할 수 있음 / 프로그램 실행 시 메모리 공간 절약 / 다양한 Addressing mode 지원

단점: Instruction의 사용 빈도 효율이 좋지 않음 (20%가 80% 작업) / Instruction 길이가 가변적이기에 Pipelining에 적합하지 않음 / Instruction이 복잡하기에 여러 Clock에 걸쳐 Instruction을 실행

(10 pts) CPU를 5-stage pipeline으로 구현할 때 각 stage의 역할은 무엇인가?

1. IF stage: PC (Program Counter) register의 정보를 사용하여 Memory로부터 Instruction을 Fetch하고, PC register의 값을 업데이트함
2. ID stage: Instruction을 Decoding 하고, Register의 값을 읽음
3. EX stage: Instruction의 종류에 따라 Operation을 실행하거나 Address를 계산함

4. MEM stage: 필요하다면 Memory access 후 값을 읽거나 씀
5. WB stage: 계산 결과 또는 Memory로부터 읽어 온 데이터를 Register에 씀

(15 pts) Pipelined CPU에서 발생하는 세가지 hazard는 무엇인가? 각각에 대해 설명하시오.

- Structural hazards

  하드웨어 자체가 Pipelining에 적합하지 않아서 생기는 Hazard이다. 하나의 Resource를 여러 Instruction이 동시에 사용하려고 할 때 발생한다. 예를 들어, Instruction Fetch와 데이터 저장을 위한 Memory access를 동시에 시도하는 경우가 있다. 이는 Resource duplication을 통해 해결 가능하다. Instruction Memory와 Data Memory를 분리하는 방법(Harvard Architecture)이 대표적이다.

- Data hazards

  실행하고자 하는 Instruction이 이전 Instruction 결과의 영향을 받아 생기는 Hazard이다. 연산 결과가 Register에 저장되기 전에 해당 Register 값을 사용하려고 할 때 발생한다. Stall (대기), Forwarding 등의 방법을 통해 해결 가능하다.

- Control hazards

  jump, beq 등 Branch instruction으로 인해 Instruction 실행 순서가 바뀌어 발생하는 Hazard이다. Pipeline 속 Instruction을 실행하지 않을 경우 발생한다. Branch prediction, 최적화된 Branch processing 등의 방법을 통해 해결 가능하다.

(15 pts) 기본적인 two-bit saturation counter를 이용한 branch predictor의 동작 방식에 대해 설명하시오. (branch history table에 대한 indexing 포함)

2bits로 4개의 State를 정의한다. 이때, Prediction은 “Prediction taken”과 “Prediction not taken”으로 나뉜다. 00과 01은 “Prediction”, 10과 11은 “Prediction not taken”으로 정의한다. Branch 여부에 따라 State 값은 달라지지만, Prediction이 연속 두 번 틀려야 Prediction 값이 바뀐다. Branch 여부와 Prediction의 변화를 Indexing 하여 Branch History Table에 기록한다. Branch History Table 정보를 다음 Branch Prediction에 활용한다. Two-bit로 Branch Prediction을 할 경우, Prediction의 적중률을 높일 수 있다.

(10 pts) 현대의 컴퓨터의 memory hierarchy가 어떻게 구성되었고, 각 level은 어떤 메모리 디바이스를 이용하여 구현되는지 설명하라.

현대 컴퓨터는 CPU에 가까운 순으로 Register, Cache, Main memory, Secondary storage의 계층 구조를 가지고 있다. Register는 CPU에 내장되어 있는 데이터 저장소로서 작업에 필요한 데이터와 Instruction을 일시적으로 저장한다. Cache는 SRAM (Static Random Access Memory)로 구현되어 있다. SRAM은 속도가 빠르고 비용이 비싸다는 특징이 있다. Main memory는 DRAM (Dynamic Random Access Memory)로 구현된다. DRAM은 속도가 비교적 느리고 저렴하다. Secondary storage는 Disk로 구현되어 있으며, Disk는 속도가 가장 느리고 비용도 가장 저렴하다.

(15 pts) Cache miss의 세 가지 핵심적인 원인 (3C's model)를 설명하라.

Compulsory miss, Conflict miss, Capacity miss가 있다.

Compulsory miss란 Cache에 처음 접근할 때 데이터가 존재하지 않아 생기는 Miss이다. 프로그램을 처음 시작할 때 발생할 수 있다.

Conflict miss는 여러 데이터 Block이 하나의 Cache location에 할당되어 생기는 Miss이다. 이는 Cache size와 무관하기에 Cache 공간이 남아도 발생할 수 있다.

Capacity miss란 Cache 내 저장 공간이 부족하여 생기는 Miss이다. Cache size를 늘림으로써 해결 가능하다.

(15 pts) Cache의 성능을 향상시키기 위하여 적용할 수 있는 최적화 방법을 3가지 제시하라.

Mapping 방식, Replacement 방식, Multi-level Cache 방법이 있다.

Mapping 방식이란 하나의 Cache line을 여러 데이터 Block에 할당하여 Cache 공간을 효율적으로 관리하는 것이다. Mapping 방식은 Cache line 할당 방법에 따라 Direct mapping, Fully-associate mapping, Set-associate mapping으로 나눌 수 있다.

Replacement 방식이란 새로운 데이터 Block을 Cache에 저장해야 할 때 교체할 Block을 정하는 방법이다. Random, FIFO (First In First Out), LRU (Least Recently Used), LFU (Least Frequently Used)가 있다.

Multi-level Cache 방식은 CPU와 Main memory 사이에 Cache를 1개 더 추가하여 Cache를 2개 사용하는 것이다. 이때 1st-level Cache는 CPU 내부에 존재하며, 더 작고 속도가 빠르다는 특징이 있다. 2nd-level Cache는 CPU와 Main memory 사이에 존재하며, 비교적 큰 Block size를 다룬다. 2-level Cache를 적용하면 Miss panalty를 줄일 수 있게 된다.

🚥 운영체제

1. Operating Systems Concepts

(1) **[10 points]** Explain key difference(s) between a process and a thread.

Process는 Memory에 적재되어 실행 중인 프로그램을 의미한다. Thread는 Process를 구성하는 실행 흐름의 단위이다. 즉, 하나의 Process는 여러 개의 Thread를 가질 수 있다. 이때 Thread는 Process의 자원을 공유하며 실행된다.

(2) **[15 points]** Explain the four conditions for a deadlock to occur.

Deadlock을 야기하는 4가지 상황에는 상호 배제, 점유와 대기, 비선점, 원형 대기가 있다. 상호 배제란 특정 자원을 한번에 하나의 Process만 사용할 수 있는 것이다. 점유와 대기는 어떤 Process가 자원을 할당 받은 상태에서 다른 Process가 할당 받은 자원을 기다리는 상황이다. 비선점은 다른 Process가 사용 중인 자원을 강제로 빼앗을 수 없어 기다려야 하는 상황이다. 원형 대기는 여러 Process가 원의 형태로 서로의 자원을 기다리는 것이다. 이렇게 4가지 상황에서 여러 Process가 서로를 기다려 실행이 멈추는 Deadlock이 발생할 수 있다.

2. Memory Management

(1) **[10 points]** What is demand paging?

Process를 Memory에 적재할 때 해당 Process를 구성하는 Page들 전체를 저장하는 것이 아니라, 실행에 필요한 Page들만 Memory에 적재될 수 있게 하는 Paging 기법이다.

(2) **[20 points]** How can a multi-level page table mitigate the storage overhead?

Multi-level page table이란 실제 page table을 여러 조각으로 나눈 뒤 Outer page table을 정의하여 Outer page table이 실제 page table 조각들을 가리키게 하는 방법이다. Multi-level page table을 사용할 경우, page table 전체가 아닌 Outer page table만 Memory에 저장하면 되기에 저장 공간을 절약할 수 있다.

(3) **[10 points]** What is the key drawback of using a multi-level page table from the perspective of address translation performance?

Page table 계층이 늘어날수록 page fault가 발생했을 경우 더 많은 Memory 참조가 필요해진다. 이는 시스템의 성능 저하를 야기하기에 단점이 될 수 있다.

3. Storage Systems

(1) **[15 points]** Redundant Array of Inexpensive Disks (RAID)

(a) **[5 points]** Explains the goals (or benefits) of RAID

RAID란 여러 하드 디스크나 SSD를 마치 하나의 장치처럼 사용하는 기술이다. RAID 기술을 통해 민감한 데이터의 안전성을 보장할 수 있으며, 시스템의 성능을 높일 수 있다.

(b) **[10 points]** Compares RAID-0/RAID-4/RAID-5

RAID-0은 여러 개의 보조기억장치에 데이터를 단순히 나누어 저장하는 구성 방식이다. RAID-0을 사용하면, 데이터가 보조기억장치의 개수만큼 나뉘어 저장된다. 하지만 이는 하나의 장치가 망가질 경우 데이터를 복구할 수 없다는 치명적 단점이 존재한다.

RAID-4는 오류를 검출하고 복구하기 위한 패리티 (Parity) 정보를 위한 장치를 두는 방식이다. 패리티를 저장한 장치를 사용하여 다른 장치들의 오류를 검출하고 복구할 수 있다. 이때 패리티 저장 장치에 병목 현상이 발생할 수 있다는 위험성이 존재한다.

RAID-5는 오류 검출 및 복구를 위한 패리티 정보를 분산하여 저장하는 방식이다. RAID-5를 사용하면, RAID-4의 패리티 장치 병목 현상을 해결할 수 있다.

RAID-1은 단순히 보조기억장치의 데이터를 복제하여 저장하는 방식이다. 어떠한 데이터를 저장할 때 원본 장치와 복사본 장치에 모두 써야 하는 것이 특징이다. 복구가 매우 간단하다는 장점이 있지만, 저장 공간의 효율이 떨어진다.

RAID-6은 기본적으로 RAID-5와 비슷하지만 서로 다른 패리티 정보를 2개 두는 방식이다. 즉, 오류를 검출하고 복구하는 수단을 2개 두는 것이다. 데이터의 안전성은 다른 방식에 비해 높아지지만, 쓰기 속도가 느리다는 단점이 있다.

🎡 알고리듬 분석

1. **Given the following list of integers: [5, 3, 8, 4, 2, 7, 1], use the bubble sort algorithm to sort the list in ascending order. (2pt) Show each step of the sorting process, swapping adjacent elements if they are in the wrong order.**

Bubble Sort는 인접한 두 Record의 대소 관계를 비교하여 Record의 순서를 바꾸는 방식이다. 오름차순을 적용하여 Bubble Sort를 진행하면 아래와 같은 단계로 Sorting이 완성된다.

첫 번째 순회 후 배열은 [3, 5, 4, 2, 7, 1, 8]

두 번째 순회 후 배열은 [3, 4, 2, 5, 1, 7, 8]

세 번째 순회 후 배열은 [3, 2, 4, 1, 5, 7, 8]

네 번째 순회 후 배열은 [2, 3, 1, 4, 5, 7, 8]

다섯 번째 순회 후 배열은 [2, 1, 3, 4, 5, 7, 8]

여섯 번째 순회 후 배열은 [1, 2, 3, 4, 5, 7, 8]로 최종 Sorting 결과가 된다.

2. **(2pt) Explain the basic idea behind Dijkstra's algorithm for finding the shortest path between two nodes in a weighted graph with non-negative edges. (1pt) Write a simple pseudocode for the algorithm and (1pt) analyze its time complexity.**

Dijkstra 알고리즘은 기본적으로 Negative-weight edge가 없다고 가정한다. 따라서 Negative-weight edge를 허용하는 Bellman-ford 알고리즘 보다 빠르게 실행할 수 있다. Dijkstra 알고리즘은 Priority Queue를 사용한다. 이때 Source vertex와의 Distance가 우선순위의 기준이 된다. 또 하나의 특징은 Vertex를 크게 두 집합으로 나눈다는 점이다. 최종 Shortest path가 결정된 Vertex들의 집합인 S와 그렇지 않은 Vertex들을 나누어 문제를 해결한다. Dijkstra 알고리즘은 아래의 단계로 진행된다.

모든 Vertex에 대해 Initialization (Distance를 무한, Predecessor를 null, Source의 Distance를 0)

Set S 선언 (처음에는 Empty)

Priority Queue Q 선언 후 모든 Vertex 삽입

Q 속 Vertex가 모두 없어질 때까지 아래 과정 반복

Q에서 Extract-min 하여 Vertex u 선언

Vertex u를 Set s에 추가

Vertex u와 인접한 모든 Vertex v에 대해 Relaxation

Dijkstra 알고리즘의 Time complexity는 Θ(E*logV)이다. Priority Queue가 binary heap으로 구현되었다는 가정하에 Q 사용이 Θ(logV)이며, 모든 Edge에 대해 반복되기에 Θ(E*logV)가 된다.

3. **(1pt) Explain the concept of dynamic programming and (1pt) provide an example of a problem that can be solved using this technique. (1pt) Design a solution and (1pt) analyze its time and space complexity**

Dynamic Programming은 “풀고자 하는 문제를 여러 Subproblem으로 나누고, Subproblem의 optimal solution을 구하여 문제를 해결하는 방법”이다.

n번째 피보나치 수를 구하는 문제에 Dynamic Programming을 적용할 수 있다.

Fib[n]을 n번째 피보나치 수라고 하면,

Fib[0] = 0, Fib[1] = 1, Fib[i] = Fib[i-1] + Fib[i-2] 로 정의할 수 있다.

이때 n번째의 피보나치 수를 구하기 위해 i가 2일 때부터 i가 n이 될 때까지 반복문을 실행해야 한다.

따라서 Time complexity는 세타(n)이 될 것이다.

또한 n번째 피보나치 수를 구하기 위해 0부터 n-1번째까지의 피보나치 수가 저장되어야 하기에 Space complexity는 O(n)이다.