1. 컴퓨터 기능

가. 데이터 입출력 기능

1) 전자 계산기 외부로부터 데이터를 받거나 내보내는 기능

2) 전자 계산기와 외부 장치를 연결하는 통로 역할

나. 데이터 저장 기능

1) 외부로부터 받은 데이터나 내부에서 처리한 결과 데이터를 저장장치에 보관하는 기능

다. 데이터 처리 기능

1) 외부로부터 받은 데이터나 내부에 저장된 데이터를 내부 연산 장치를 사용하여 데이터 값을 변경하는 기능

2. 범용 컴퓨터

가. 다양한 응용 본야의 소프트웨어를 실행할 수 있도록 범용 구조를 가진 컴퓨터

1) 범용성(generality), 유연성(flexibility)을 강조

2) 특정 용도에 비해 성능이 떨어짐

3) PC, Workstation, Main-frame

3. 특정용도 컴퓨터

가. 특성 응용 분야에 맞춰 하드웨어와 소프트웨어가 최적화된 컴퓨터

1) 임베디드 시스템(Embedded System)

나. 실생활에서 사용하는 대부분의 전기전자 시스템에서 특정 용도 컴퓨터를 사용

1) 가정용 전자제품

2) 사무용 기기

3) 공장 자동화 기기

4) 항공, 우주, 군사 장비

4. 컴퓨터 하드웨어

가. 컴퓨터의 하드웨어는 중앙처리장치, 메모리, 주변장치, 버스로 구성된다

1) 중앙처리장치(CPU: Central Processing Unit): 데이터 처리

2) 메모리(Memory): 데이터 저장

3) 주변 방치(Peripherals): 데이터 입출력, CPU 보조, 시스템 동작 지원

4) 버스(Bus): 구성요소들의 상호 연결

5. 시스템 소프트웨어

가. 컴퓨터 시스템 하드웨어를 관리하거나 프로그램 개발에 필요한 프로그램

1) 초기화 프로그램: 부트로더(Bootloader)

2) 운영체제(Operating System)

가) 컴퓨터의 하드웨어 자원의 효율적 관리

나) 프로그램 실행환경 제공

3) 프로그램 개발 도구

가) 프로그램 생성 및 디버깅: 코드 편집기, 컴파일러, 인터프리터, 디버거 등

6. 펌웨어

가. 비휘발성 메모리에 내장된 소프트웨어

1) 비휘발성 메모리: MROM, EPROM, EEPROM, Flash 메모리

나. 펌웨어 용도

1) 개발이 완료되어 추가 변경 가능성이 낮은 소프트웨어의 영구 저장

가) 컴퓨터 하드웨어 제어에 필요한 필수 프로그램

(1) PC의 ROM-BIOS

나) 각종 전자기기의 내장형 소프트웨어

2) 프로그램 코드 보호 수단으로 활용 가능

가) ROM의 접근제어 기능을 활용하여 외부에서 프로그램 코드 접근 제한

다. 펌웨어 변경 방법

1) 하드웨어 Re-write

가) 소프트웨어보다 유연성이 부족

라. 최근에는 응용 프로그램을 운영체제없이 하드웨어상에서 직접 실행하도록 설계하는 방식을 말하기도 함

1) 펌웨어 개발: 응용 소프트웨어 - 하드웨어
2) 운영체제기반 개발: 응용 소프트웨어 - 운영체제 - 하드웨어

7. H/W - S/W 공조설계

가. 전자 계산기의 H/W특성과 S/W특성을 조합하여 최적의 시스템을 설계하는 방법
1) H/W특성: 고성능, 고비용
2) S/W특성: 유연성, 저비용
나. 시스템 설계의 비용 최소화 + 시스템 성능 향상
다. 전자 계산기 하드웨어와 소프트웨어를 이해하고 공조설계(Co-design)이 가능한 고수준 설계자 필요

8. 세대별 분류

가. 1세대(1946 ~ 1957): 진공관(vaccum tube)을 사용한 컴퓨터
나. 2세대(1958 ~ 1964): 트랜지스터(transistor)를 사용한 컴퓨터
다. 3세대(1965 ~ 1971): 직접회로(IC: integrated circuit)를 사용한 컴퓨터, SSI
라. 4세대(1972 ~ 1977): LSI(Large Scale Integration)
마. 5세대(1978 ~ 1991): VLSI(Very Large Scale Integration)
바. 6세대(1991 ~ 현재): ULSI(Ultra Large Scale Integration)

9. 데이터 처리 방식에 따른 분류

가. 일괄 처리(Batch Proessing) 방식
1) 여러 개의 프로그램을 사전에 정해진 순서에 따라 차례로 실행하는 방식
2) 실행중인 프로그램이 종료될 때 까지 다른 프로그램을 실행할 수 없다.
나. 시분할 처리(Time Sharing) 방식
1) 여러 개의 프로그램을 스케쥴링을 통하여 CPU를 시간적으로 분할하여 사용하는 방식
가) 가상적으로 여러개의 프로그램이 동시에 실행되는 효과
나) 컴퓨팅 자원의 이용도 향상
다. 실시간 처리(Real-time Processing) 방식
1) 프로그램 또는 작업 시작과 종료에 대한 시간 제약 조건의 만족을 보장하는 처리방식
라. 시간 제약조건의 만족요부가 시스템에 미치는 영향 정도에 따라 경성, 연성으로 구분
1) 경성(Hard) 실시간 시스템: 시스템 동작에 치명적 영향
가) 미사일 제어
2) 연성(Soft) 실시간 시스템: 시스템 성능에 영향
가) 동영상 재생

10. MPU(Micro Processor Unit)

가. 계산용 프로세서(computation-oriented)
1) 고속의 연산 및 데이터 처리: 여러 개의 고성능 ALU, H/W multiplier, FPU(Floating Point Unit), ...
2) x86(Intel), MIPS(MIPS Technologies), SH(Hitachi), ...

11. MCU(Micro Controller Unit)

가. 제어용 프로세서(controll-oriented)
1) 제어(control)에 특화된 마이크로 프로세서
2) 비트단위(bit-wise) 연산 지원, 빠른 인터럽트 처리, 다양한 입출력 포트 제공, ...
3) 주로 On-chip/SoC micro-processor, 8-bit 프로세서(AVR, 8051, PIC), ...

12. DSP(Digital Signal Processor)

가. 디지털 신호처리(audio, video) 전용 프로세서(DSP-specific)
1) MAC(Multiply-Accumulator), FPU, Multi-port 메모리 인터페이스, etc, ...
2) TMS320C6xxx(Texas Instruments), MSC81xx(Freescale), SHARC(Analog Devies), ...

13. ASP(Application Specific Processor)

가. 특정 용도에 최적화된 프로세서
1) Graphics Processor: GeForce(nVidia), HD(Intel), Radeon(AMD), ...
2) Java Processor: ARM926EJ (ARM), picojava(Sum Microsystems), ...
3) Network Processor: IXP(Intel), PowerQUICC(NXP), ...
4) Crypto Processor: C29x(NXP), SC300(ARM), ...

14. 컴퓨터의 구조적 특성

가. Instruction Set Architecture

1) 명령어 유형 및 표현 방법

나. 데이터 표현에 사용되는 비트 수

1) 8-, 16-, 32-, 64-bit 등 단위 데이터 표현에 사용되는 비트 수

다. Memory Addressing Method

1) 메모리 저장공간에 대한 접근 방법

라. I/O Mechanism

1) 데이터 입출력 장치에서의 데이터 처리 방법

15. 메모리 계층 구조

가. 속도, 용량 차이가 있는 메모리를 계층적으로 배치하여 메모리 접근 속도와 저장 용량을 최적화 하는 방법

나. 메모리 계층 구조

1) CPU에 가까울수록 고속, 소용량

가) CPU <-> 레지스터 <-> Cache <-> 주기억장치 <-> 보조기억장치

16. 시스템 동작에 필요한 주변장치

가. 클럭발생장치

1) 컴퓨터 구성요서에서 필요한 클럭을 생성

나. 전원제어장치

1) 소비전력 제어, 전원전압 레벨 감시

다. 리젯장치

1) 안정된 리셋 신호 생성

라. 타이머

1) 실시간 또는 시간간격 측정

2) 이벤트 발생 횟수 카운트

17. 시스템 버스

가. CPU, 메모리, 주변장치를 연결하는 주요 버스 (고속)

18. I/O 버스

가. I/O 버스

1) 주변장치, 주로 입출력 장치를 연결하는 버스 (고속/저속)

나. 버스와 버스의 연결

1) 브릿지(Bridge): 버스 사이의 속도 및 데이터 전송방식 차이 해결

19. Micro-Architecture

가. Definition of Micro-Architecture

1) 주어진 마이크로 프로세서의 명령어 세트 실행에 최적화된 하드웨어 구조

2) 동일한 명령어 세트에 대해 여러개의 마이크로 아키텍처가 가능

3) 컴퓨터 구조 = 명령어 세트 구조 (ISA) + 마이크로 아키텍처

20. Micro-Architecture 구성

가. 데이터 경로와 제어 유니트로 구성

1) Datapath: 데이터가 흘러가는 통로

2) Control Path: 데이터 경로를 제어하는 제어신호 발생회로

21. Stored-Program 방식 컴퓨터

가. 기존 컴퓨터(ENIAC) 에서의 프로그램의 입력과 변경 방법

1) 수작업으로 스위치 설정 및 케이블 연결해서 프로그래밍

나. 1945년 ENIAC의 프로그램 입력과 변경이 불편한 점을 해결하기 위해 John von Neumann이 “First Draft of a Report on the EDVAC” 논문에서 제안한 구조.

다. 주요 개념

1) 프로그램과 데이터를 메모리에 저장해서 실행

가) 프로그램 입력: 프로그램을 메모리에 저장

나) 프로그램 변경: 메모리의 내용을 변경

22. IAS 컴퓨터

가. 1946년 폰 노이만(John von Neumann)이 Pirnceton의 IAS(Institute for Advanced Studies)에서 설계한 최초의 Stored Program 방식의 전자식 컴퓨터)

나. 현대 범용 컴퓨터의 원형

다. Von Nuewmann Machine, Von Neumann 구조로 알려짐

23. IAS 컴퓨터 구조

가. 1000 x 40-bit 메모리 공간

나. 데이터

1) 1-bit sign bit, 39-bit value

다. 명령어: 20-bit

1) 8-bit op-code, 12-bit address

2) 메모리 1워드에 2개 명령어 저장

라. MAR: Memory Address Register

마. MBR: Memory Buffer Register

바. PC: Program Counter

사. IR: Instruction Register(Op-Code)

아. IBR: Instructin Register(Right Inst.)

자. AC: Accumulator

차. MQ: Multiplier Quotient

24. Von Neumann Structure

가. 프로그램과 데이터가 동일 메모리에 저장된 형태의 컴퓨터 구조

나. Characteristics

1) 데이터와 명령어가 하나의 Read/Write 가능한 메모리에 저장된다

2) 메모리에 저장된 내용은 저장된 데이터 유형과 상관없이 위치정보만을 사용하여 접근한다

3) 프로그램은 Sequential으로 실행된다.

25. Von Neumann Structure의 단점

가. 프로그램 코드와 데이터를 동시에 접근할 수 없다.

1) 병렬화를 통한 성능향상에 치명적 약점

나. 해결방법: 하바드(Harvard) 구조

26. Harvard Structure

가. 프로그램 코드와 데이터를 별도 메모리에 저장하는 구조

1) 특징: 프로그램 코드와 데이터에 대한 동시 접근이 가능

2) 장점: 명령어의 병렬 실행이 가능하여 성능 향상에 유리

3) 단점: 메모리 구조가 복잡. 비용 증가

27. 명령어 사이클

가. 각 명령어 실행은 동일한 절차로 이루어짐

나. 하나의 명령어 실행 절차를 명령어 사이클(Instruction Cycle)로 정의

1) 1개의 명령어 사이클은 여러 개의 동작 사이클(Operation Cycle)로 구성

다. 명령어 사이클에서 수행하는 작업들은 CPU에 따라 다름

28. 명령어 사이클 구성

가. 명령어 사이클이 5개의 동작 사이클로 이루어진 경우

1) 명령어 인출(IF: Instruction Fetch)

가) 프로그램 메모리에서 명령어 읽어오기

2) 명령어 해석(ID: Instruction Decode)

가) CPU내 제어 장치에서 명령어 해석하기

3) 데이터 인출(OF: Operand Fetch)

가) 명령어 실행에 필요한 데이터를 데이터 메모리에서 읽어오기

4) 실행(EX: Execute)

가) CPU의 ALU에서 데이터 처리 (산술 및 논리 연산)

5) 결과 저장(WB: Write-Back)

가) 데이터 처리 결과를 레지스터 또는 데이터 메모리에 저장하기

29. 클럭 주파수

가. 클럭 주파수(Clock Frequency)

1) 초당 클럭 신호 변화율: 초당 클럭 펄스 수(Pulses per second)

가) clock speed, clock rate

2) 헤르츠(Hertz) 단위로 표시: 예) 500MHz, 4GHz

나. 클럭 주파수가 높을수록 CPU처리능력이 향상

30. CPU클럭 속도

가. Clock cycle

1) 클럭 펄스 주기(clock pulse period)

2) CPU Clock Cycle

나. Clock cycle time(CCT)

1) 클럭 사이클 사이의 시간 간격

31. CPI

가. CPI(Cycles Per Instruction)

1) 명령어 하나를 실행하는데 소모되는 클럭 사이클 수

나. 명령어 유형에 따라 명령어 실행에 필요한 클럭 사이클 수가 다르기 때문에 주어진 프로세서 또는 프로그램의 실행성능은 형균 CPI(Average CPI)를 사용하여 표시

다. 프로그램의 평균 CPI계산

1) i: 명령어 유형, 모두 n개의 명령어 유형이 있다고 가정

2) li: i-번째 유형의 명령어 수

3) CPIi: i-번째 유형 명령어의 CPI

4) Ic: 프로그램이 포함하고 있는 전체 명령어 수

5) 수식: $CPI = \dfrac{\sum_{i=1}^{n}(CPI_i \times I_i)}{I_c}$

라. 주어진 프로그램의 실행에 필요한 CPU Time

1) τ: clock cycle time, 1/f

2) $T = I_c \times CPI \times \tau$

마. CPU성능 = 1/CPU Time으로 표현 가능

32. MIPS

가. MIPS(Millions of Instruction Per Second)

가) 가장 많이 사용되는 프로세서 성능 표시방법

나) 초당 몇 백만개의 명령어를 실행할 수 있는가? 를 표시

다) MIPS = (프로그램에 포함된 전체 명령어 수) / (프로그램 실행에 소요된 CPU Time * 10^6)

라) 평균 CPI를 사용한 계산: $MIPS = \dfrac{I_c}{T \times 10^6} = \dfrac{f}{CPI \times 10^6}$

나. 예제

1) CPU Clock Frequency = 500MHz로 가정했을 경우,

명령어 유형	평균 CPI	명령어 비율(%)
연산	2	50
Load/Store	2	20
실행흐름제어	4	10
기타	1	20

2) CPI = 2*0.5 + 2*0.2 + 4*0.1 + 1*0.2 = 2.0

3) MIPS = 500 * 10^6 / (2.0 * 10^6) = 250

33. MFLOPS

가. MFLOPS(Millions of Floating-Point Operations Per Second)
1) 부동 소수점 연산 성능 표시 방법
2) 초당 몇 백만개의 부동 소수점 연산 명령어를 실행할 수 있는가? 를 표시
3) MFLOPS = (프로그램에 포함된 전체 부동 소수점 연산 명령어 수) / (프로그램 실행에 소요된 CPU Time * 10^6)
4) 부동 소수점 연산을 포함한 프로그램에만 적용 가능
5) 명령어 대신 동작(Operation)을 카운트

34. Synthetic 벤치마크

가. Synthetic 벤치마크
1) 많은 응용 프로그램들에서 통계 분석하여, 그 결과를 바탕으로 인위적으로 작성된 프로그램을 사용하여 성능을 평가하는 방법
2) 대표적 Synthetic 벤치마크
가) 웻스톤(Whetstone): 1972년 Algol언어로 작성. 주로 실수 연산을 포함
나) 드라이스톤(Dhrystone): 1984년 Ada언어로 작성. 주로 정수 연산을 포함

35. CPU 처리속도 향상

가. 분기 예측
1) 분기 예측(Branch-prediction)
가) 분기 명령어 다음에 실행해야할 명령어(Branch Target)를 예측하여, 다음에 실행할 명령어의 사전 실행(Pre-Execution)이 가능
나) 예측 방법
(1) 프로그램 실행 기록을 보관하여 다음 분기 명령의 Target 어드레스 예측에 활용
다) 분기예측 이점
(1) 프로그램의 실행 흐름을 유지함으로써 실행 흐름 변경에 따른 손실을 제거
라) 분기 예측 실패시 부담
(1) 명령어들을 미리 가져와서 병렬 실행할 경우, 이전상태로 복구(Flushing) 작업 필요

나. 데이터 흐름 분석
1) 데이터 흐름 분석(Data-flow analysis)
가) 명령어들 사이의 데이터 의존성(Data Dependency)을 분석해서 최적의 실행순서를 도출
(1) 데이터 의존성이 없는 명령어들의 실행순서 변경

36. CPU와 메모리 사이의 성능 불균형 해소

가. CPU와 메모리 사이의 성능 불균형 해결방법
1) CPU와 메모리 사이의 데이터 버스 폭(Data Bus Width) 확장
2) 연결 대역폭(Bandwidth) 확장
가) 고속 버스나 계층적 버스를 사용하여 대역폭 확장
3) 캐시(Cache)를 사용하여 메모리 접근 방법 개선
가) 계층적 캐시를 사용하여 메모리 접근 빈도를 감소: on-chip 캐시 사용, 캐시 복잡도 증가하는 문제

37. 컴퓨터의 동작속도 향상

가. 컴퓨터의 동작속도 향상
1) 시스템 클럭 속도를 증가
2) 클럭 속도 향상으로 발생하는 문제
가) 소모 전력 밀도 증가: 냉각 또는 방열 장치 필요
나) RC Delay 증가: 동작 주파수가 올라감에 따라 RC에 의한 전압강하 심화
다) Clock Skew 발생: 클럭 전달경로 길이 차이로 인한 왜곡 현상 증가

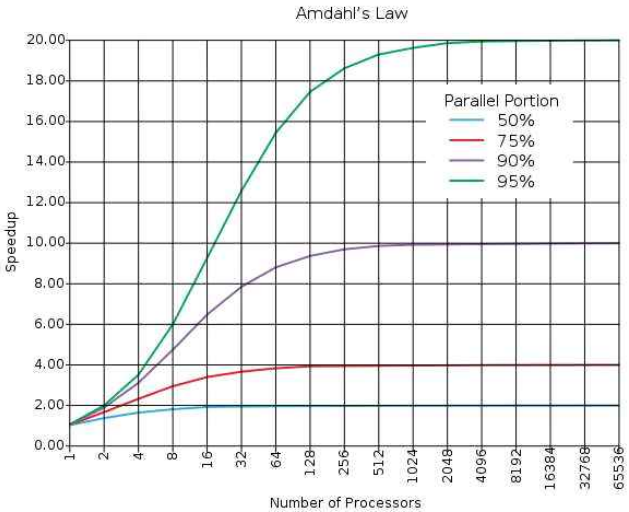
38. Clock Skew

가. Clock Skew 발생
1) 클럭 전달경로 길이 차이로 인한 클럭 신호의 전달시간 차이 발생
가) 동작 타이밍 편차로 인한 회로의 오동작 가능성

39. Amdahl의 법칙

가. 1967년 Gene Amdahl이 제안한 법칙

1) 컴퓨팅 자원의 개선으로 얻을 수 있는 컴퓨터의 성능 향상에 대한 법칙
가) 예를 들면, 단일 프로세서와 여러 개의 프로세서를 사용했을 경우, 컴퓨터 성능향상
2) 프로세서 수가 증가하면 할수록, 병렬 처리를 위한 부담이 커져서 더 이상의 성능향상을 기대하기 힘들어진다
나. 주로 병렬처리 컴퓨터에서 멀티프로세서를 사용해서 얻을 수 있는 최대 성능향상을 예상하는데 사용
다. 프로세서 수와 성능향상 관계



라. 멀티 프로세서를 사용함으로써 얻는 성능향상(speed-up)
1) T: 단일 프로세서를 사용할 때의 프로그램 실행시간
2) f: 프로그램중 병렬처리 가능한 부분
3) 1-f: 프로그램중 시리얼 처리해야하는 부분
4) N: 프로세서 수

$$Speedup = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}}$$

$$= \frac{T(1-f)+Tf}{T(1-f)+\frac{Tf}{N}} = \frac{1}{(1-f)+\frac{f}{N}}$$

5) f가 작으면 병렬처리 효과가 약하다
6) N이 아주 커지면 성능 향상은 1/(1 - f)로 수렴한다. 결국, 프로세서가 어느 정도를 넘어서면 프로세서를 추가함으로써 얻어지는 성능 향상이 작아진다

40. Moore의 법칙

가. 하나의 칩에 집적될 수 있는 트랜지스터의 수가 매년 두 배로 증가하고 있다
1) 반도체 기술 발전에 따른 전자 계산기의 성능 향상 예측
2) 인텔 공동 창업자인 Fordon Moore가 1965년에 언급
3) 1970년대에 들어와서 18개월마다 두 배 증가하는것으로 떨어짐

41. 연관 접근

가. 연관 접근(associative access)
1) 기억장소는 저장된 데이터 내용과 비교함으로써 식별된다
2) 접근 시간은 이전 접근 장소와 무관하며, 항상 일정: 비교 부담(overhead)
가) cache

42. 메모리 접근 시간

가. 메모리 접근 시간(Memory Access Time)
1) 메모리에서 read/write 동작을 수행하는데 걸리는 시간
가) Read access time: 어드레스와 제어 신호가 메모리에 도착하는 순간부터 데이터가 읽혀지는 순간까지 걸리는 시간
나) Write access time: 어드레스와 제어 신호가 메모리에 적용되고, 데이터 쓰기가 완료될 때 까지 걸리는 시간

43. 메모리 확장

가. 메모리 확장(Memory Expansion)
1) 여러 개의 메모리 칩을 연결하여 대용량의 메모리를 구성하는 것
가) 용량 확장(Size/Capacity Expansion): 메모리의 용량을 확장
나) 워드 확장(Word-length Expansion): 메모리 입출력 데이터 폭을 확장

44. 메모리 인터리빙

가. Interleaved 메모리
1) 메모리를 뱅크(bank)로 구성
2) 메모리 액세스를 병렬화

45. Big-endian

가. Left-to-right(western culture language style)
나. 가장 낮은 어드레스에 상위 바이트(most significant byte) 부터 차례로 저장
다. IBM, Motorola, SUN, most RISC's, Internet

46. Little-endian

가. Right-tp-left(arithmetic operation style)
나. 가장 낮은 어드레스에 하위 바이트(least significant byte) 부터 차례로 저장
다. Intel, VAX

47. PROM

가. Programmable ROM (PROM)
1) 현장에서 한 번만 쓰여질 수 있다. (OTP: one-time programmable)
2) 제조가 완료된 후에 사용자가 필요에 따라 임의로 쓸 수 있다.
가) 특별한 장치(ROM writer)가 필요
3) 융통성과 편의성 제공: 대량 생산에 유용

48. EPROM

가. Erasable PROM (EPROM)
1) 자외선 (UV)로 데이터 삭제
2) EPROM writer/programmer 를 사용하여 데이터 저장
3) 데이터 저장 후 차광 씰 부착 보관

49. EEPROM

가. Electrically Erasable PROM (EEPROM)
1) 전기적으로 바이트 단위로 데이터 삭제
2) 읽기보다 쓰기 동작에 많은 시간 소요

50. Flash 메모리

가. EEPROM의 특수 형태
1) 비휘발성 메모리
2) EPROM과 EEPROM의 중간 정도의 가격과 성능
3) EPROM보다 빠른 데이터 삭제
4) 높은 집적도
5) 블록 단위 삭제: 바이트 단위 삭제 불가
나. 단점
1) 블록 단위 삭제
2) 메모리 마모(wear): 유한한 삭제-쓰기 동작
가) Wear leveling: 메모리 셀을 균등하게 사용함으로써 메모리 전체 수명을 연장하는 작업
나) Over provisioning: 플래시 메모리에 필요한 작업(wear leveling, garbage collection, bad block manage 등)에 필요한 예비 공간을 준비하는 작업

51. 명령어 설계시 고려사항

가. 연산 종류(Operation Repertoire)
1) 연산 종류 수, 연산 작업 유형, 연산의 복잡성
나. 데이터 유형(Data Types)
다. 명령어 형식(Instruction Formats)

1) 명령어 길이(비트 수), 오퍼랜드 개수, 명령어 필드의 크기

라. 레지스터(Registers)

1) 명령어들에 의해 참조될 레지스터들의 수

2) 레지스터들의 용도

마. 주소지정 (Addressing) 방식

52. 명령어의 오퍼랜드

가. 명령어의 오퍼랜드는 명령어 실행에 필요한 데이터를 말한다

1) 명령어 유형에 따라 구성방법과 해석방법이 달라진다

나. 오퍼랜드 유형

1) 주소(Address)

2) 수(Numeric Data)

가) 정수(integer), 부동 소수점(floating point), 10진수(decimal)

3) 문자(Character)

가) ASCII

4) 논리적 데이터(Logical data)

가) 비트 단위 처리

(1) 기억장치의 효율적 활용

(2) 비트별 조작처리 가능

53. 오퍼랜드 저장 위치

가. 명령어 내부

1) 실행할 명령어 내부에 포함

2) 즉치(Immediate)값, 상수(Constant)값

나. CPU 내부 레지스터

1) 레지스터 이름(R0, R1, R2, ... Rn)으로 표기

다. 메모리

1) 프로그램 메모리 또는 데이터 메모리

라. I/O 장치의 내부 레지스터 또는 메모리

54. 스택

가. 스택(Stack)

1) 프로시저 사이의 공용 데이터 저장공간

가) 데이터 메모리 공간의 일부를 LIFO(Last-In-First-Out) 형태로 사용

2) 전용 명령어 사용: PUSH(데이터 저장) /POP(데이터 인출)

3) 스택의 현 위치

가) Top of Stack: 스택의 저장위치

나) 스택 포인터(Stack Pointer)를 사용하여 위치정보 저장

나. 스택 유형

1) Ascending / Descending: 스택 어드레스 증가 방향에 따라 구분

2) Full / Empty: Top of Stack 위치에 따라 구분

다. 스택에 저장되는 데이터

1) 복귀 주소(Return Address)

2) 함수 전달인자(Function Parameters)

3) CPU 상태정보(Status)

4) 사용중인 레지스터 내용

5) 함수에서 사용할 지역변수(Local Variables)

55. 스택 프레임

가. 스택 프레임(Stack Frame)

1) 하나의 프로시저를 호출하기 위해 저장되는 복귀 주소를 포함한 데이터들의 전체 집합

56. RAS

가. RAS(Row Address Strobe): 주소 버스에 Row 주소가 출력될 때 '0'로 설정

57. CAS

가. CAS(Column Address Strobe): 주소 버스에 Column 주소가 출력될 때 '0'로 설정

58. Memory Refresh

가. 메모리 재충전(Memory Refresh)

1) DRAM의 경우, 데이터를 저장하고 있는 메모리 셀은 시간이 지남에 따라 저장된 전하가 누설
   가) 주기적 재충전이 필요
   나) 모든 메모리 셀은 정해진 주기내에 반드시 재충전되어야 한다
2) 재충전 빈도와 방법은 반도체 제조기술과 메모리 셀 설계기술에 따라 다름
3) 재충전 작업으로 인한 대역폭 손실 발생

59. 명령어 사이클 RTL 동작

가. 명령어 사이클

1) Instruction Fetch(IF)
   가) 프로그램 메모리에서 명령어 가져오기
2) Instruction Decode(ID)
   가) 명령어 해석하기
3) Data Fetch(DF)
   가) 명령어 실행에 필요한 데이터 가져오기
4) Execute(EX)
   가) 명령어 실행
5) Write Back(WB)
   가) 실행 결과 저장

나. 컴퓨터의 명령어 사이클 설명을 위한 컴퓨터의 RTL 구성요서

1) PC(Program Counter): 실행할 명령어의 저장위치 정보(어드레스) 저장
2) IR(Instruction Register): 실행할 명령어 저장
3) MAR(Memory Address Register): 메모리 접근에 사용되는 어드레스 저장
4) MBR(Memory Buffer Register): 메모리에 읽거나 쓸 데이터 값
5) R(Register): 데이터 임시 저장용 레지스터
6) ALU(Arithmetic & Logic Unit): 산술/논리 연산회로
7) 메모리: 명령어와 명령어 실행에 필요한 데이터 저장

다. IF동작 사이클

1) 프로그램 메모리에서 명령어를 읽어오는 동작 사이클
   가) 명령어의 저장위치에 대한 정보: PC(Program Counter)에 저장
   나) 읽어온 명령어의 저장 위치: IR(Instruction Register)
      (1) MAR <- PC
      (2) MBR < M[MAR]
      (3) IR <- MBR
2) 다음 명령어를 가져오기 위해 PC값 업데이트
   가) PC++: 어드레스 증가량은 명령어 길이에 따라 다름
   나) 실행위치는 t2또는 t3
      (1) MAR <- PC
      (2) MBR <- M[MAR]
      (3) IR <- MBR, PC++

라. ID동작 사이클

1) IF동작 사이클에서 읽어온 명령어를 해석해서 각 컴퓨터 구성요소에서 필요한 제어신호 발생
2) 제어 유니트 구성방법에 따라 ID 동작 사이클에 소요되는 클럭 수가 달라짐
   가) 제어신호 직접전달
      (1) ID 동작 사이클 생략 가능
      (2) ID 동작 사이클을 IF 동작 사이클에 포함
   나) 제어신호

(1) 생성된 제어신호를 래치에 저장
(2) C <- IR Decoding

마. OF동작 사이클

1) ID동작 사이클에서 해석 결과 연산에 필요한 데이터(오퍼랜드)가 필요한 경우, 데이터 메모리에서 데이터를 읽어온다
(1) MAR <- IR.addr
(2) MBR <- M[MAR]
(3) R <- MBR

2) 실행할 명령어 유형에 따라 데이터 저장위치 정보(어드레스)의 저장장소와 어드레스 해석방법이 다르다

가) 어드레스 저장위치
(1) 명령어 레지스터(IR)
(2) 또 다른 레지스터: 인덱스(Index) 레지스터 등
(3) 메모리: 어드레스가 메모리에 저장되어 있는 경우. 다양한 어드레싱 방법 적용
(가) MAR <- Index
(나) MBR <- M[MAR]
(다) R <- MBR

나) 어드레스 해석방법
(1) 어드레싱 모드에 따라 다름: 직접 어드레싱, 간접 어드레싱
(가) MAR <- IR/IDX
(나) MBR <- M[MAR]
(다) R <- MBR
(라) MAR <- R
(마) MBR <- M[MAR]
(바) R* <- MBR

3) 오퍼랜드 접근을 위한 어드레스 계산이 필요한 경우, 어드레스 계산 사이클이 추가될 수 있다

바. EX동작 사이클

1) ID동작 사이클에서 해석한 결과에 따라 필요한 연산을 수행한다

가) 연산 유형: 산술/논리 연산, 데이터 전송

2) 실행할 연산 유형에 따라 실행시간 차이 발생
(1) R3 <- R1 + R2

사. WB동작 사이클

1) EX실행 결과를 레지스터 또는 메모리에 저장한다

가) 레지스터에 저장하는 경우, 레지스터 구조에 따라 WB사이클 불필요할 수 있다.
(1) R3 <- R1 + R2

나) 메모리에 저장하는 경우, OF동작사이클 처럼 다양한 어드레싱 방법을 사용할 수 있다.
(1) MAR <- IR
(2) MBR <- R
(3) M[MAR] <- MBR

60. 명령어 파이프라인

가. 명령어 실행과정에 파이프라인 적용

1) 작업: 명령어 사이클
2) 단위 작업: 동작 사이클

나. 명령어 파이프라인

1) 하나의 명령어 사이클을 구성하는 여러 개의 동작 사이클들을 파이프라인 형태로 실행
2) 명령어 선인출(Prefetch) 가능
3) 여러 개의 명령어들을 중첩 실행하는것이 가능

다. 선인출(Prefetch)

1) 실행 사이클은 메인 메모리를 액세스 하지 않음

가) 실행 사이클 동안 다음 명령어를 미리 인출할 수 있음

(1) 명령어 선인출(prefetch)

2) 명령어 인출시 고려사항

가) 실행 사이클이 인출 사이클 보다 길다

(1) 여러개의 명령어를 선인출 할 수 있을까?

나) 조건 분기 명령어 실행시 다음에 실행할 명령어의 주소를 알 수 ㅇ벗다

(1) 추정(Guess)

61. 파이프라인 해저드

가. 파이프라인 실행조건이 만족되지 않을 경우, 파이프라인 실행을 계속할 수 없어, 파이프라인 일부분이 멈추어야 (stall)한다. 이러한 파이프 라인 멈춤을 파이프라인 해저드(pipline hazard)라고 하며, 파이프 라인 버블 (pipeline bubble)이라고도 한다.

나. 자원 해저드(resource hazard)

1) 파이프라인에 들어와있는 명령어들이 동일한 자원을 사용할 때 발생

2) 구조적 해저드(structural hazard)

다. 데이터 해저드(data hazard)

1) 파이프라인에 들어와 있는 명령어들 사이에 데이터 의존성이 존재할 때 발생

가) 쓰기후 읽기(RAW: read after write): true dependency

나) 읽기후 쓰기(WAR: write after read): anti-dependency

다) 쓰기후 쓰기(WAW: write after write): output dependency

라. 제어 해저드(control hazard)

1) 파이프라인에 들어와있는 명령어들의 실행 흐름이 바뀔 때 발생

가) 분기 목적지 예측이 잘못되어 이미 파이프 라인으로 들어온 불필요한 명령어들을 버려야 (flush)할 때 발생

2) 제어 해저드를 최소화 하는 방법

가) 분기 목적지 예측의 정확도 향상

(1) 다중 스트림

(2) 분기 목적지의 선인출

(3) 루프 버퍼

(4) 분기 예측

(5) 지연 분기

62. CISC

가. 등장 배경

1) 컴퓨터 기술 발전으로 하드웨어 비용은 절감되는 반면, 소프트웨어 비용이 상대적으로 증가

2) HPL(high-level programming language) 등장: 알고리즘 표현력 향상, 프로그램과 연관성이 떨어지는 하드웨어에 대한 이해를 컴파일러가 전달. 구조화 프로그램/객체지향 개념 프로그래밍 기술 도입

3) Semantic gap(HPL이 제공하는 기능과 컴퓨터 구조사이의 갭) 발생 비효율적인 프로그램 실행, 과도한 프로그램 크기 증가, 컴파일러 복잡도 증가를 유발

4) 해결방법: CISC화

가) 명령어 세트 크기 증가, 복잡한 어드레싱 방식 도입, HPL기능을 하드웨어화

(1) 컴파일러 작성 용이

(2) 실행 효율 향상: 복잡한 작업을 micro-code로 구현

(3) 복잡한 HPL 기능을 명령어 수준에서 지원

나. 특성

1) CISC프로세서 특성

가) 하나의 명령어에서 수행하는 작업이 많다

나) 다양한 어드레싱 모드를 사용

다) 명령어의 길이가 가변적이다

(1) 명령어 해석이 복잡

라) 풍부한 명령어 세트

(1) 많은 수의 명령어

(2) 복잡한 명령어

마) 프로그램 길이가 짧다
2) CISC로 가는 이유
가) 컴파일러를 간단하게 구현할 수 있다.
나) 프로그램 크기가 줄고, 실행속도가 빨라지게 된다
3) CISC에 대한 반론
가) 간단한 컴파일러
(1) 복잡한 명령어는 사용하기 힘들다
(2) 최적화가 오히려 어렵다
나) 프로그램 크기절약 + 실행속도 향상 ???
(1) CISC프로그램 크기가 RISC프로그램보다 작다고 보장하기 힘들다
(2) 긴 명령어가 여러 개의 짧은 명령어 실행보다 빨리 실행된다고 보장하기 어렵다
4) 대표적 CISC 프로세서
가) x86프로세서
(1) 어드레싱 모드: 12가지
(2) 명령어 길이 1 ~ 12바이트
나) Motorla 680x0
(1) 어드레싱 모드: 18가지
(2) 명령어 길이: 2 ~ 10 바이트

63. RISC

가. Reduced Instruction Set Computer
1) 간단한 명령어 세트 구조를 가진 컴퓨터
나. RISC 프로세서의 주요 특징
1) 간단한 명령어 구조
가) 대부분 고정길이 명령어 사용
나) 간단한 어드레싱 모드 및 명령어 포멧 사용
다) 동작 클럭당 1개 명령어 실행 가능
2) 명령어 세트에 속한 명령어 수가 상대적으로 적음
3) 비교적 많은 레지스터 (GPR) 사용
가) Load-Store 구조: 모든 연산은 레지스터를 사용
나) 효율적인 레지스터 사용을 위해 컴파일러 역할 강조
4) 명령어 실행 파이프라인 최적화 강화
5) Harvard 메모리 구조: 명령어와 데이터 전송경로 분리
6) 간단한 명령어 decoder 회로
가) Micro-code 대신 hardwired회로 사용
7) 컴파일러 최적화 용이
가) Primitive 명령어에 대한 다양한 최적화 기술 적용 가능
다. 대표적 RISC 프로세서
1) ARM
가) ARM Holdings
나) ARM2(The first ARM)@1985, ARMv8(64-bit)@2011
2) MIPS
가) MIPS Technologies
나) R2000(The first MIPS)@1985, R4000(64-bit)@1991
3) SPARC
가) Oracle(Sun Microsystems)
나) SPARC V7(The first SPARC)@1986, SPARC V9(64-bit)@1993

64. RISC와 CISC비교 방법

가. RICS와 CISC의 우열 판단 어려움
1) RISC와 CISC특성을 절충한 프로세서 설계

나. 비교 방법

1) 정량적 비교

가) 프로그램 크기와 실행속도 비교

2) 정성적 비교

가) 고급 언어 지원 능력 및 칩 면적 비교

다. 비교의 어려움

1) 직접 비교 어려움

2) 성능 측정을 위한 테스트 프로그램 부재

3) 하드웨어와 컴파일러 영향을 분리해서 비교하기가 어려움

CISC	RISC
명령어 길이 가변	명령어 길이 고정
복잡한 어드레싱 모드	간단한 어드레싱 모드
적은 레지스터 사용	많은 레지스터 사용
메모리 오퍼랜드 허용	레지스터 오퍼랜드만 허용
하드웨어 강조	소프트웨어 강조
다중 클럭을 사용하는 복잡한 명령어	단일 클럭을 사용하는 단순 명령어
대부분 명령어가 메모리 접근 가능(Memory-to-Memory)	Load/Store 명령어만 메모리 접근 가능(Register-to-Register)
복잡한 명령어 디코더 필요(Micro-programmed)	간단한 명령어 디코더 필요(Hardwired)
프로그램당 명령어 수가 적음	프로그램당 명령어 수가 많음

라. 구조적 차이

1) CISC: Micro-program을 사용한 제어

2) RISC: Hardwired Logic을 사용한 제어

마. 명령어 포맷 비교

1) CISC: 복잡한 구조, 가변 길이

2) RISC: 간단한 구조, 고정길이

65. Superscalar

가. 여러 개의 명령어를 동시에 인출하여 실행

나. 명령어들의 병렬 실행 여부는 하드웨어적으로 판단

다. CISC, RISC 명령어들을 병렬처리 할 수 있는 하드웨어 구조(Superscalar CISC, Superscalar RISC)

라. Superscalar 프로세서 사례

1) PowerPC

가) 6개의 독립적 실행 유니트

(1) Branch Execution Unit

(2) Load/Store Unit

(3) 3개의 Integer Unit

나) In-order issue

다) Register renaming

2) Pentium

가) 3개의 독립적 실행 유니트 사용

(1) 2개의 Integer Unit

(2) Floating point Unit

나) In-order Issue