

최신 컴퓨터 구조(2판)

디지털 논리부터 성능 분석까지

(기출문제 풀이 과정)

제01장. 컴퓨터 시스템 개론

1.

컴퓨터가 본래의 기능을 수행하기 위해서는 전기적으로 변환된 정보들을 입력하여 처리하고 저장해야 한다. 컴퓨터에서 각종 정보를 입력하여 처리하고 저장하는 동작이 실제 일어나게 해 주는 물리적인 실체가 하드웨어다. 소프트웨어는 정보 처리의 종류를 지정하고 정보의 이동 방향을 결정하는 동작이 일어나는 시간을 지정해 주는 명령들의 집합(프로그램)이다. ㉠하드웨어의 운영을 위해서는 반드시 ㉡소프트웨어의 지원이 필요하다.

2.

컴퓨터는 중앙처리장치(CPU), 기억장치, 입출력장치로 나눈다. 논리연산장치는 중앙처리장치에 포함된다.

3.

주기억장치는 데이터나 프로그램을 일시적으로 기억시킨다.

4.

마이크로프로세서(microprocessor)는 산술논리연산장치, 제어장치, 레지스터들로 구성되어 있다.

5.

컴퓨터는 중앙처리장치(CPU), 기억장치, 입출력장치로 나눈다. 또한 중앙처리장치는 산술논리연산장치, 제어장치, 레지스터들로 구성되어 있다.

6.

제어장치는 컴퓨터에서 실행해야 할 동작을 지시하고 데이터의 흐름을 제어한다.

7.

컴퓨터에서 사용되는 버스의 종류

- 주소 버스(address bus)
- 데이터 버스(data bus)
- 제어 버스(control bus)

8.

제어 버스는 중앙처리장치(CPU)가 기억장치나 입출력장치와 데이터 전송을 할 때나, 자신의 상태를 다른 장치들에 알리기 위해 사용하는 신호를 전달한다. 이러한 신호에는 기억장치 읽기 및 쓰기 신호, 입출력장치 읽기 및 쓰기 신호, 중앙처리장치 상태 신호, 인터럽트 요청 및 허가 신호, 클록 신호 등이 있다.

9.

주소 버스나 제어 버스는 단방향 버스이지만 데이터 버스는 양방향 버스다.

10.

하드 디스크나 CD-ROM 같은 보조기억장치는 기계적인 장치가 포함되기 때문에 속도가 느려서 보조기억장치는 중앙처리장치와 직접 연결되지 않는다. 또한 입출력장치도 이와 같은 이유로 중앙처리장치와 직접 연결되지 않는다. 그러므로 보조기억장치와 입출력장치는 별도의 인터페이스(interface) 회로나 제어기(controller)를 통해 중앙처리장치와 연결되어야 한다.

11.

소프트웨어(software)는 저장장치에 저장된 특정한 목적의 하나 또는 다수의 컴퓨터 프로그램을 뜻한다. 소프트웨어는 모두 비트, 즉 0과 1의 수로 이루어져 있다.

12.

시스템 소프트웨어(system software)는 하드웨어를 관리하고 응용 소프트웨어를 실행하는 데 필요한 프로그램이다. 운영체제, 언어 번역 프로그램, 유틸리티 프로그램 등이 이에 속한다. 유틸리티 프로그램은 각종 주변 장치들을 구동하는 데 필요한 드라이버 프로그램, 백신 프로그램, TCP/IP 같이 컴퓨터를 네트워크로 연결하는 데 필요한 각종 프로그램 등을 말한다.

13.

백신 프로그램은 시스템 소프트웨어로 분류한다.

14.

시스템 소프트웨어에는 운영체제, 언어 번역 프로그램(컴파일러, 인터프리터), 유틸리티 프로그램(드라이버 프로그램, 백신 프로그램) 등이 이에 속한다.

15.

시스템 소프트웨어(system software)에는 운영체제, 언어 번역 프로그램(컴파일러, 인터프리터), 유틸리티 프로그램(드라이버 프로그램, 백신 프로그램) 등이 이에 속한다.

16.

컴퓨터가 이해할 수 있는 2진수로 표현한 언어를 기계어(machine language)라고 한다.

17.

프로그램은 보통 C, C++, Python 등과 같은 고급 언어(high-level language)로 작성한다. 이렇게 작성된 프로그램은 영문자와 숫자로 이루어져 있어 사람들은 이해하기 쉽지만, 컴퓨터는 이해할 수 없다. 따라서 고급 언어로 작성한 프로그램은 컴파일러(compiler) 또는 인터프리터(interpreter)라는 소프트웨어를 이용해 컴퓨터가 이해할 수 있는 언어(기계어)로 번역해야 한다.

18.

어셈블리어로 작성된 프로그램은 언어 번역 프로그램인 어셈블리(assembler)가 기계어로 번역해 준다. 각 어셈블리 명령어의 동작을 개략적으로 이해할 수 있도록 사용된 기호인 LOAD, ADD, STOR를 니모닉(mnemonic)이라고 한다.

19.

컴퓨터의 발전 과정을 세대별로 명확하게 설명하기는 어렵지만 컴퓨터 세대는 새로운 하드웨어 부품의 출현을 기준으로 분류되고 있다. 특히, 집적회로(Integrated Circuit, IC)가 개발된 3세대 이후에는 컴퓨터 세대에 대한 정의가 분명하지 않다.

진공관(1세대) - 트랜지스터(2세대) - 집적회로(3세대) - VLSI(4세대 이후)

20.

인텔의 공동 창업자인 고든 무어가 예언한 무어의 법칙(Moore's law)은 반도체 집적 회로의 트랜지스터 수가 24개월마다 2배로 증가한다는 법칙이다.

21.

아날로그 컴퓨터(analog computer)는 연속적인 변량을 사용하여 계산을 수행한다. 필요한 입력 데이터는 계측 기기로부터 직접 입력할 수 있으며, 신속한 입력과

그 상태에 대한 즉각적인 반응을 얻을 수 있으므로 이 유형의 컴퓨터는 프로세스 제어(process control)에 적합하다. 아날로그 컴퓨터에서는 전압, 전류, 온도, 압력 등의 데이터를 처리한다.

22.

컴퓨터에는 추론과 창조적 사고 기능은 없다.

23.

컴퓨터에는 창의성, 응용성은 없다.

24.

연산장치의 기능

- 함수 연산 기능
- 자료 전달 기능
- 제어 기능
- 입출력 기능

25.

연산 코드(op code)에 연산의 의미가 있다.

26.

하버드 구조는 폰 노이만 구조의 단점을 보완한다. 명령어 메모리 영역과 데이터 메모리 영역을 물리적으로 분리시키고, 각각을 다른 시스템 버스로 중앙처리장치에 연결함으로써 명령과 데이터를 메모리로부터 읽는 것을 동시에 처리할 수 있다.

27.

하버드 구조(Harvard)는 명령어 메모리 영역과 데이터 메모리 영역을 물리적으로 분리하고, 각각을 다른 시스템 버스로 중앙처리장치에 연결함으로써 명령과 데이터를 메모리로부터 읽는 것을 동시에 처리할 수 있다.

제02장. 데이터의 표현

1.

MSB(Most Significant Bit)는 맨 왼쪽 비트(최상위 비트)를 말하며, LSB(Least Significant Bit)는 맨 오른쪽 비트(최하위 비트)를 말한다.

2.

4개의 2진 변수로 수행할 수 있는 논리연산은 $16(=2^4)$ 가지다.

3.

- N 가지 정보를 코드로 부호화하려면 $\lceil \log_2 N \rceil$ 개가 필요하다.
- $N = 17$ 이면, $\log_2 17 = 4.xxx$ 이므로 $\lceil \log_2 17 \rceil = 5$ 비트가 필요하다.

4.

비트 n 개로는 2^n 개의 정보를 표현할 수 있다.

5.

N 가지 정보를 코드로 부호화하려면 $\lceil \log_2 N \rceil$ 개가 필요하다.

6.

- N 가지 정보를 코드로 부호화하려면 $\lceil \log_2 N \rceil$ 개가 필요하다.
- $N = 500$ 이면, $\log_2 500 = 8.xxx$ 이므로 $\lceil \log_2 500 \rceil = 9$ 비트가 필요하다.

7.

256은 $\lceil \log_2 256 \rceil = \lceil \log_2 2^8 \rceil = 8$ 이므로 R, G, B마다 8비트씩 필요하다. 따라서 화소 당 저장장소는 $24(=3 \times 8)$ 비트가 필요하며, 나타낼 수 있는 색상 수는 2^{24} 이다.

8.

Kilo = 10^3 , Mega = 10^6
Giga = 10^9 , Tera = 10^{12}

9.

- ① $101111.111_{(2)} = 2^5 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = 47.875$
- ② $10111.010_{(2)} = 2^5 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-2} = 47.25$

$$\textcircled{3} \quad 10111.001_{(2)} = 2^5 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-3} = 47.125$$

$$\textcircled{4} \quad 10111.101_{(2)} = 2^5 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = 47.625_{(10)}$$

10.

$$10110.1101_{(2)} = 2^5 + 2^3 + 2^2 + 2^1 + 2^{-1} + 2^{-2} + 2^{-4} = 46.8125_{(10)}$$

11.

10진법에서는 0부터 9까지 정의되므로 가장 큰 수인 9가 1001이므로 4비트가 필요하다.

12.

$0.1875 \times 8 = 1.5$, 앞에 정수부분을 1로 두고,
소수 부분 $0.5 \times 8 = 4 \Rightarrow 0.14_{(8)}$

13.

$$375.24_{(8)} = 3 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 + 2 \times 8^{-1} + 4 \times 8^{-2} = 192 + 56 + 5 + 0.25 + 0.0625 = 253.3125_{(10)}$$

14.

- ① $FE.D_{(16)} = 15 \times 16^1 + 14 \times 16^0 + 13 \times 16^{-1} = 254.8125_{(10)}$
- ② $FF.E_{(16)} = 15 \times 16^1 + 15 \times 16^0 + 14 \times 16^{-1} = 255.875_{(10)}$
- ③ $9F.8_{(16)} = 9 \times 16^1 + 15 \times 16^0 + 8 \times 16^{-1} = 159.5_{(10)}$
- ④ $FF.5_{(16)} = 15 \times 16^1 + 15 \times 16^0 + 5 \times 16^{-1} = 255.3125_{(10)}$

15.

$$73C.4E_{(16)} = 7 \times 16^2 + 3 \times 16^1 + 12 \times 16^0 + 4 \times 16^{-1} + 14 \times 16^{-2} = 1852.304688_{(10)}$$

16.

B의 가중치 $4096(=16^3)$, E의 가중치 $256(=16^2)$,
A의 가중치 $16(=16^1)$, D의 가중치 $1(=16^0)$

17.

2진수를 3자리씩 묶어서 8진수로 변환한다. 빈자리는 0으로 채운다.

$$100110.100101_{(2)} = 100 \ 110.100 \ 101_{(2)} = 46.45_{(8)}$$

18.

8진수 1자리를 3비트의 2진수로 변환하면 된다.

$$474_{(8)} \rightarrow 100 \ 111 \ 100_{(2)}$$

19.

2진수를 4자리씩 묶어서 16진수로 변환한다. 빈자리는 0으로 채운다.

$$110110001_{(2)} \rightarrow 0001\ 1011\ 0001 \rightarrow 1B1_{(16)}$$

20.

16진수 1자리를 4비트의 2진수로 변환하면 된다.

$$C52_{(16)} \rightarrow 1100\ 0101\ 0010_{(2)}$$

21.

8진수 1자리를 3비트의 2진수로 바꾸고, 이를 4비트씩 묶어서 16진수로 변환하면 된다. 빈자리는 0으로 채운다.

$$735.56_{(8)} = 111\ 011\ 101\ .\ 101\ 110$$

$$\rightarrow 0001\ 1101\ 1101\ .\ 1011\ 1000$$

$$= 1DD.B8_{(16)}$$

22.

16진수 1자리를 4비트의 2진수로 바꾸고, 이를 3비트씩 묶어서 8진수로 변환하면 된다. 빈자리는 0으로 채운다.

$$A4D_{(16)} = 1010\ 0100\ 1101_{(2)}$$

$$\rightarrow 101\ 001\ 001\ 101 = 5115_{(8)}$$

23.

$$111110.1000_{(2)} = 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^{-1} = 62.5_{(10)}$$

$$111110.1000_{(2)} = 0011\ 1110.1000_{(2)} = 3E8_{(16)}$$

24.

$$20_{(10)} = 010100_{(2)} \rightarrow 010\ 100 = 24_{(8)}$$

$$\rightarrow 0001\ 0100 = 14_{(16)}$$

25.

4진수 1자리는 2진수 2자리(2비트)에 대응한다. 즉, $0_{(4)} = 00_{(2)}$, $1_{(4)} = 01_{(2)}$, $2_{(4)} = 10_{(2)}$, $3_{(4)} = 11_{(2)}$ 이다. 따라서 $23.103_{(4)} = 1011.010011_{(2)}$

26.

2진수를 2자리씩 묶어서 4진수로 변환한다. 2진수를 3자리씩 묶어서 8진수로 변환한다. 2진수를 4자리씩 묶어서 16진수로 변환한다. 빈자리는 0으로 채운다.

$$10010010.011_{(2)} = 10\ 01\ 00\ 10\ .\ 01\ 10 \rightarrow 2102.12_{(4)}$$

$$10010010.011_{(2)} = 010\ 010\ 010\ .\ 011 \rightarrow 222.3_{(8)}$$

$$10010010.011_{(2)} = 1001\ 0010\ .\ 0110 \rightarrow 92.6_{(16)}$$

27.

$$\textcircled{1}\ 1101_{(2)} = 8 + 4 + 1 = 13_{(10)}$$

$$\textcircled{2}\ 23_{(5)} = 2 \times 5 + 3 = 13_{(10)}$$

$$\textcircled{3}\ 15_{(8)} = 1 \times 8 + 5 = 13_{(10)}$$

$$\textcircled{4}\ C_{(16)} = 12_{(10)}$$

28.

$$\textcircled{1}\ 101111_{(2)} = 2^5 + 2^3 + 2^2 + 2^1 + 2^0 = 47_{(10)}$$

$$\textcircled{2}\ 57_{(8)} = 5 \times 8^1 + 7 \times 8^0 = 47_{(10)}$$

$$\textcircled{3}\ 48_{(10)}$$

$$\textcircled{4}\ 2F_{(16)} = 2 \times 16^1 + 15 \times 16^0 = 47_{(10)}$$

29.

$$\textcircled{1}\ FF_{(16)} = 15 \times 16^1 + 15 \times 16^0 = 255_{(10)}$$

$$\textcircled{2}\ 257_{(10)}$$

$$\textcircled{3}\ 1111111_{(2)} = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = 255_{(10)}$$

$$\textcircled{4}\ 377_{(8)} = 3 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 = 255_{(10)}$$

30.

$$\textcircled{1}\ FF_{(16)} = 15 \times 16^1 + 15 \times 16^0 = 255_{(10)}$$

$$\textcircled{2}\ 1111111_{(2)} = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = 255_{(10)}$$

$$\textcircled{3}\ 254_{(10)}$$

$$\textcircled{4}\ 377_{(8)} = 3 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 = 255_{(10)}$$

31.

컴퓨터에서 음수를 표현하는 방법

- 부호와 절댓값 표시
- 부호화된 1의 보수 표시
- 부호화된 2의 보수 표시

32.

$$11110 \xrightarrow{\text{1의 보수}} 00001 \xrightarrow{\text{2의 보수}} 00010$$

33.

3의 1의 보수 : $0011_{(2)} \rightarrow 1100$

$$\textcircled{1}\ 1\text{의 } 2\text{의 보수} : 0001_{(2)} \rightarrow 1110 \rightarrow 1111$$

$$\textcircled{2}\ 2\text{의 } 2\text{의 보수} : 0010_{(2)} \rightarrow 1101 \rightarrow 1110$$

$$\textcircled{3}\ 4\text{의 } 2\text{의 보수} : 0100_{(2)} \rightarrow 1011 \rightarrow 1100$$

$$\textcircled{4}\ 8\text{의 } 2\text{의 보수} : 1000_{(2)} \rightarrow 0111 \rightarrow 1000$$

34.

$$\begin{array}{r} 7\ 7\ 7 \\ - \quad 5\ 6\ 3 \\ \hline 2\ 1\ 4 \end{array}$$

35.

$$284\text{의 } 9\text{의 보수} \rightarrow 999 - 284 = 715$$

$$284\text{의 } 10\text{의 보수} \rightarrow 715 + 1 = 716$$

36.

16의 보수는 “15의 보수 + 1” 이다.

$$\text{FF}_{16} \xrightarrow{15\text{의 보수}} 00\text{F}_{16} \xrightarrow{+1} 010$$

37.

8비트로 10의 절댓값을 구하면 0001010이다. 양수이면 0, 음수이면 1을 붙이면 된다.

$$+10 \rightarrow 0001010, \quad -10 \rightarrow 1001010$$

38.

$$-11_{(10)} \rightarrow -00001011_{(2)} \rightarrow 11110100 \rightarrow \text{F4}_{(16)}$$

39.

먼저 8비트로 +9를 구하면 00001001이다. 이것을 2의 보수로 바꾸면 11110111이다.

$$00001001 \xrightarrow{1\text{의 보수}} 11110110 \xrightarrow{+1} 11110111$$

40.

㉠ 부호 있는 절댓값 : 최상위 비트 1은 음수이므로 11101101을 10진수로 변환하면 -109이다.

㉡ 부호와 1의 보수 : 최상위 비트를 제외하고 반전시킨 001 0010을 10진수로 변환하면 18이므로 -18이 된다.

㉢ 부호와 2의 보수 : 부호화 1의 보수에 1을 더한 001 0011을 10진수로 변환하면 19이므로 -19가 된다.

41.

• 10의 보수는 “9의 보수 + 1” 이므로 -593의 9의 보수는 99406이고, 10의 보수는 99407이다.

• 2의 보수는 “1의 보수 + 1” 이다. 먼저 593을 2진수로 변환하면 001001010001이므로 1의 보수는 110110101110이고, 2의 보수는 110110101111이다.

42.

2진수 10비트 A의 2의 보수는 $B = 2^{10} - A$ 이므로 $A + B = 2^{10} = 1024$ 이다.

43.

 $-(2^{n-1}) \sim +(2^{n-1} - 1)$ 이므로 $n = 8$ 이면 $-(2^7) \sim +(2^7 - 1)$ 이다. 따라서 표현범위는 -128 ~ +127이다.

44.

 n 비트인 경우 2의 보수로 표현 가능한 범위는 $-(2^{n-1}) \sim +(2^{n-1} - 1)$ 이다.따라서 $n = 16$ 이면 $-(2^{15}) \sim +(2^{15} - 1)$ 이다.

45.

 n 비트인 경우 2의 보수로 표현 가능한 범위는 $-(2^{n-1}) \sim +(2^{n-1} - 1)$ 이다.

46.

8비트 데이터를 16비트로 확장하려면 먼저 부호 비트를 16비트의 최상위 비트(MSB)로 이동하고, 빈자리는 부호 비트로 채우면 된다. 따라서 1111111100101111이다.

47.

컴퓨터 내부에서는 2의 보수를 이용한 가산으로 감산 기능을 수행한다.

48.

①, ②, ③은 1의 보수에 대한 설명이다.

④ n 비트인 경우 2의 보수로 표현 가능한 범위는 $-(2^{n-1}) \sim +(2^{n-1} - 1)$ 이므로 음수의 최대 절댓값이 양수의 최대 절댓값보다 1만큼 크다.

49.

2의 보수 표현

- 연산 과정에서 자리올림수(carry)는 무시
- 연산이 간단하여 덧셈 연산이 1의 보수에 비해서 간단
- 음수표현 시 1의 보수보다 1개의 수를 더 표시 가능
- 0의 표현에 있어 1의 보수는 -0과 +0, 2의 보수는 0만 존재
- 4비트인 경우 수의 표현범위는 -8 ~ +7이다.

50.

① 2의 보수는 “1의 보수 +1” 이므로 하드웨어로 구현하기 어렵다.

② 부동소수점 표현 방식에 대한 설명이다.

④ 2의 보수 표현 방법에서 0은 한 가지가 있다.

51.

1의 보수, 부호와 절댓값 방법에서 0의 표현은 2가지가 있으나 2의 보수 방법에서는 1가지만 존재한다.

52.

부호 자리에서 캐리를 C_1 , 부호 다음 자리(MSB)에서 캐리를 C_2 라고 하면 C_1 과 C_2 가 다르면 오버플로가 발생한 것이며, 같으면 오버플로가 발생하지 않은 것이다.

53.

$$\begin{array}{r} 256 \\ + \quad 542 \\ \hline 1020 \end{array}$$

54.

$$\begin{array}{r} 1A1D \\ - \quad F9F \\ \hline A7E \end{array}$$

55.

$FFFF_{(16)} - EC00_{(16)} + 1 = 1400_{(16)} \Rightarrow$ 2진수로 변환하면,

$\Rightarrow 0001\ 0100\ 0000\ 0000 \Rightarrow 5Kbyte$

56.

$$\begin{array}{r} 011100 \\ + \quad 100011 \\ \hline 111111 \\ 111111 = 111\ 111 = 77_{(8)} \end{array}$$

57.

$$\begin{aligned} 1011_{(2)} + 34_{(8)} &= 11_{(10)} + 28_{(10)} = 39_{(10)} \\ &= 100111_{(2)} \rightarrow 0010\ 0111 = 27_{(16)} \end{aligned}$$

58.

$$\begin{array}{r} 11001 \\ - \quad 10001 \\ \hline 11001 \\ + \quad 01111 \\ \hline 101000 \end{array}$$

(가) 10001을 2의 보수로 변환하면 01111이다.

(나) 계산 결과 최상위 비트의 캐리를 무시하면 결과는 01000이다.

59.

$$\begin{array}{r} 101011 \\ - \quad 100110 \\ \hline 000101 \end{array}$$

60.

$$\begin{aligned} -15_{(10)} &\rightarrow -00001111_{(2)} \rightarrow 11110000 \rightarrow 11110001 \\ -3_{(10)} &\rightarrow -00000011_{(2)} \rightarrow 11111100 \rightarrow 11111101 \\ &\quad 11110001 \\ + \quad &\quad 11111101 \\ \hline 1 \quad &11101110 \end{aligned}$$

$\Rightarrow$ 최상위 비트의 캐리를 무시하면 결과는 11101110이다.

61.

6비트 수의 표현범위인 $-32 \sim 31$ 을 벗어나면 오버플로가 발생한다.

- ① $14 + 18 = 32$ ② $30 - 14 = 16$
③ $-20 - 4 = -24$ ④ $24 + 6 = 30$

62.

MSB에서 계산이 이루어질 때 바로 직전 캐리와 최종 캐리가 같은 값이면 정상적으로 계산된 것이고, 다른 값이면 오버플로가 발생한 것이다.

① 같다	$\begin{array}{r} 1000\ 000 \\ + \quad 0100\ 0000 \\ \hline 1100\ 0000 \\ 1\ 0000\ 0000 \end{array}$	② 다르다	$\begin{array}{r} 0000\ 000 \\ + \quad 1000\ 0000 \\ \hline 1100\ 0000 \\ 1\ 0100\ 0000 \end{array}$
③ 다르다	$\begin{array}{r} 1000\ 000 \\ + \quad 0100\ 0000 \\ \hline 0100\ 0000 \\ 0\ 1000\ 0000 \end{array}$	④ 다르다	$\begin{array}{r} 0000\ 000 \\ + \quad 1000\ 0000 \\ \hline 1100\ 0001 \\ 1\ 0100\ 0001 \end{array}$

63.

MSB에서 계산이 이루어질 때 바로 직전 캐리와 최종 캐리가 같은 값이면 정상적으로 계산된 것이고, 다른 값이면 오버플로가 발생한 것이다.

① 같다	$\begin{array}{r} 00110 \\ + \quad 010010 \\ \hline 000111 \\ 0\ 011001 \end{array}$	② 다르다	$\begin{array}{r} 11110 \\ + \quad 010010 \\ \hline 001111 \\ 0\ 100001 \end{array}$
③ 같다	$\begin{array}{r} 10000 \\ + \quad 010010 \\ \hline 111001 \\ 1\ 001011 \end{array}$	④ 같다	$\begin{array}{r} 00010 \\ + \quad 010010 \\ \hline 001011 \\ 0\ 011101 \end{array}$

64.

두 수가 모두 음수이거나 양수이면 덧셈 과정에서 오버플로가 발생할 수 있다. 즉, 수의 표현범위를 벗어날 수 있다.

65.

언팩 10진수 형식

- 1바이트를 존(zone) 부분과 디지트(digit) 부분으로 구성
- 가장 오른쪽 바이트의 존 부분에 부호를 표시

66.

팩 10진 데이터 형식으로 연산하고, 결과를 출력할 때는 언팩 10진 데이터 형식으로 변환한다.

67.

팩 10진 데이터 형식으로 연산하고, 결과를 출력할 때는 언팩 10진 데이터 형식으로 변환한다.

68.

팩(Pack) 형식에서는 1바이트에 10진수 두 자리를 표현하고, 최하위 바이트의 하위 4비트에 부호를 표시한다.

양수 = 1100 = C, 음수 = 1101 = D

6	5
---	---

7	부호
---	----

 이므로

0110	0101
------	------

0111	1100
------	------

 이다.

69.

- 언팩 10진수 형식에서 1바이트를 존(zone) 부분과 디지털(digit) 부분으로 구성
- 존 부분에는 항상 F(1111)가 들어가고 디지털 부분에는 10진수 값이 8421 BCD 코드 형식으로 들어감.
- 가장 오른쪽 바이트의 존 부분에 부호를 표시
- 양수(+)는 1100(C), 음수(-)는 1101(D), 부호가 없을 때는 1111(F)

F	4	F	2	C	6
---	---	---	---	---	---

70.

팩(Pack) 형식에서는 1바이트에 10진수 두 자리를 표현하고, 최하위 바이트의 하위 4비트에 부호를 표시한다.

양수 = 1100 = C, 음수 = 1101 = D

따라서 -456은 45 6D이다.

71.

마지막 디지털 4비트의 C는 양수, D는 음수를 나타낸다. 따라서 $A = 4095$, $B = -3840$ 이므로 $A + B = 4095 + (-3840) = 255$ 이다.

00	00	25	5C
----	----	----	----

72.

- 언팩10진형 표현

+375

1111	0011	1111	0111	1100	0101
------	------	------	------	------	------

- 팩10진형 표현

+375

0011	0111	0101	1100
------	------	------	------

- 언팩10진형 표현은 24비트, 팩10진형 표현은 16비트이므로 8비트가 절약된다.

73.

부동소수점수를 표현하는데 필요한 정보는 부호(Sign), 지수(Exponent), 가수(Mantissa)다.

74.

부동소수점수를 표현하는데 필요한 정보는 부호(Sign), 지수(Exponent), 가수(Mantissa)다. 따라서 소수점은 필요하지 않다.

75.

- ② 아주 작은 수와 아주 큰 수의 표현에 적합하다.

76.

- 바이어스 : 127(지수 부분의 + 또는 -를 표시하기 위함)
- 지수부가 127보다 작으면 지수는 음수, 127보다 크면 양수, 127이면 지수는 0이다.

77.

정규화(normalization) : 가수(mantissa)의 가장 왼쪽 숫자(digit)가 0이 아닌 숫자가 오도록 하는 과정

78.

2진수를 정규화하면 가수는 항상 $1.xxx_{(2)}$ 로 표현된다. 실제로는 '1.'을 저장하지 않는다. 한 비트라도 더 표현함으로써 정밀도를 높일 수 있다.

79.

부동소수점 수의 국제 표준은 IEEE 754이다.

80.

- ④ 표현범위는 $\pm 1.40 \times 10^{-45} \sim \pm 3.40 \times 10^{38}$ 이다.

81.

$+14925_{(10)} \rightarrow 0011\ 1010\ 0100\ 1101_{(2)}$
 $\rightarrow 1.1101001001101_{(2)} \times 2^{13}_{(2)}$

따라서 지수부는 $127 + 13 = 140$ 이므로 2진수로 표현하면

$140_{(10)} = 1000\ 1100_{(2)} = 8C_{(16)}$

가수부는 $1101\ 0010\ 0110\ 1000_{(2)} = D268_{(16)}$

부호는 양수이므로 0

82.

- 0.01101을 정규화 : 1.101×2^{-2}

- 부호 : 0(양수)

- 지수부 : $01111111 - 10 = 01111101$

- 가수부 : 101000000000000000000000

0	01111101	101000000000000000000000
---	----------	--------------------------

83.

$-13.625_{(10)} \rightarrow -1101.\ 101_{(2)} \rightarrow -1.101101 \times 2^3$

- 부호는 음수이므로 1이다.

- 지수부는 $127 + 3 = 130$ 이므로 2진수로 표현하면 1000

0010이다.

- 가수부는 1.을 생략하면 1011 0100 0000 0000 0000 000이다.
- ∴ 1 1000 0010 1011 0100 0000 0000 0000 000

84.

$$\begin{aligned} & (1.110 \times 10^{10}) \times (9.200 \times 10^{-5}) \\ &= (1.110 \times 9.200) \times (10^{10} \times 10^{-5}) \\ &= 10.212 \times 10^5 = 1.021 \times 10^6 \end{aligned}$$

85.

0111 0100 0001
7 4 1

86.

1234₍₁₀₎를 10진수로 변환하면 668이며, 이를 8421 코드로 변환하면

0110 0110 1000
6 6 8

87.

BCD 코드에서 1010(10) ~ 1111(15)은 사용되지 않음

88.

2421 코드의 1011 = $2 \times 1 + 4 \times 0 + 2 \times 1 + 1 \times 1 = 5$

89.

3초과 코드는 8421 코드(BCD 코드)에 3을 더한 비가중치 코드이며, 자기보수 특성을 갖는다.

10진수	8421 코드	3초과 코드
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

90.

3초과 코드는 BCD 코드에 3을 더해서 만들어진 코드이다. 10진수 3에 3을 더하면 6이므로 이를 2진수로 나타내면 0110이다.

91.

3초과 코드는 BCD 코드에 3을 더해서 만들어진 코드이므로 $1011 - 0011 = 1000 \rightarrow 8$ 이다.

92.

3초과 코드는 BCD 코드에 3을 더해서 만들어진 코드이므로 0011 ~ 1100 코드 값을 갖는다. 1101은 포함되지 않는다.

93.

3초과 코드(excess-3 code)는 자기 보수성을 갖는다.

94.

그레이 코드는 현 상태에서 다음 상태로 코드의 그룹이 변화할 때 단지 하나의 비트만이 변화되는 최소 변화 코드의 일종이며, 또한 비트의 위치가 특별한 가중치를 갖지 않는 비가중치 코드로서 산술연산에는 부적합하고 입출력장치와 A/D 변환기와 같은 응용 장치에 많이 사용하고 있다.

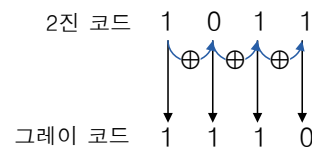
95.

그레이 코드는 비트의 위치가 특별한 가중치를 갖지 않는 비가중치 코드다.

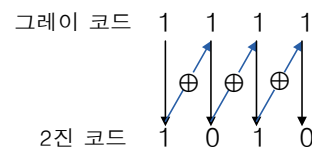
96.

그레이 코드는 AND 연산이 아닌 XOR 연산을 사용하여 만든 코드다.

97.



98.



99.

가중치 코드 : BCD 코드, 2421 코드, 51111 코드
비가중치 코드 : 3초과 코드, 그레이 코드

100.

자기보수 특성을 갖는 코드 : 3초과 코드, 2421 코드, 84-2-1 코드, 51111 코드

101.

ASCII 코드는 데이터 통신용으로 많이 쓰이고 있다.

102.

ASCII 코드의 구성

패리티	존 비트	디지트 비트
1비트	3비트	4비트

103.

ASCII 코드에서 6, 5, 4비트는 존 비트로 사용한다.

parity	zone bit			digit bit			
7	6	5	4	3	2	1	0

104.

BCD 코드, ASCII 코드, EBCDIC 코드는 문자 코드이지만 그레이 코드는 숫자 코드다.

105.

EBCDIC 코드는 8비트로 구성되므로 $256(=2^8)$ 개 문자를 표현할 수 있다.

106.

EBCDIC 코드의 비트 구성

parity	zone bit				digit bit			
8	7	6	5	4	3	2	1	0

107.

UNICODE는 컴퓨터에서 세계 각국의 언어를 통일된 방법으로 표현할 수 있게 제안된 국제적인 코드다.

108.

④ 그레이 코드는 대표적인 비가중치 코드로 인접하나 코드의 비트가 1비트만 변하여 산술연산에 부적합하다.

109.

패리티 비트(parity bit)는 정보비트에 여분으로 한자리를 추가하여 사용하는 에러 검출용 비트다.

110.

패리티 비트(parity bit)는 전송된 데이터의 오류를 검출하기 위하여 사용한다.

111.

패리티 비트는 1개의 에러만 검출할 수 있다.

112.

패리티 비트를 이용한 에러 검출 방식에서는 데이터 비트의 개수에 관계없이 1비트 에러만 검출할 수 있다.

113.

1의 개수가 짝수가 아닌 코드를 찾는다.

- ① 001101100 → 1의 개수 짝수
- ② 110100110 → 1의 개수 홀수
- ③ 110110101 → 1의 개수 짝수
- ④ 011111111 → 1의 개수 짝수

114.

XOR와 NOT 게이트를 사용하여 패리티 검사기를 구성할 수 있다.

115.

해밍코드는 데이터 수신 시 데이터 중에서 발생한 오류를 검출 및 교정이 가능한 코드다.

116.

추가되는 패리티 비트 수 산출식

$$2^p \geq d + p + 1$$

여기서 p : 패리티 비트 수, d : 데이터 비트 수32비트 데이터이므로 $d = 32$ 이다.부등식 $2^p \geq 32 + p + 1$ 을 만족하는 p 를 구하면 $p = 6$ 이다.

117.

추가되는 패리티 비트 수 산출식

$$2^p \geq d + p + 1$$

여기서 p : 패리티 비트 수, d : 데이터 비트 수

- $d = 11$ 인 경우, $2^p \geq 11 + p + 1$ 를 만족하는 p 는 $p = 4, 5, 6, \dots$ 이므로 $p = 5$ 이면 총 비트수가 16이 되므로 가능한 조합이다.
- $d = 12$ 인 경우, $2^p \geq 12 + p + 1$ 를 만족하는 p 는 $p = 5$ 이다. 총 비트수는 $12 + 5 = 17$ 비트이므로 총 비트수가 16이 되지 못한다.

118.

비트 위치	1	2	3	4	5	6	7
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7

119.

비트 위치	1	2	3	4	5	6	7
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7
P_1 영역			0		1	0	0
P_2 영역			0		1	0	0
P_4 영역			0		1	0	0

$$P_1 = D_4 \oplus D_3 \oplus D_1 = 0 \oplus 1 \oplus 0 = 1$$

$$P_2 = D_4 \oplus D_2 \oplus D_1 = 0 \oplus 0 \oplus 0 = 0$$

$$P_4 = D_3 \oplus D_2 \oplus D_1 = 1 \oplus 0 \oplus 0 = 1$$

비트 위치	1	2	3	4	5	6	7
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7
P_1 영역			0		1	0	1
P_2 영역			0		1	0	1
P_4 영역			0		1	0	1

$$P_1 = D_4 \oplus D_3 \oplus D_1 = 0 \oplus 1 \oplus 1 = 0$$

$$P_2 = D_4 \oplus D_2 \oplus D_1 = 0 \oplus 0 \oplus 1 = 1$$

$$P_4 = D_3 \oplus D_2 \oplus D_1 = 1 \oplus 0 \oplus 1 = 0$$

120.

비트 위치	1	2	3	4	5	6	7
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7
해밍코드			1		0	0	1

$$P_1 = D_3 \oplus D_5 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

$$P_2 = D_3 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

$$P_4 = D_5 \oplus D_6 \oplus D_7 = 0 \oplus 0 \oplus 1 = 1$$

⇒ 0011001

121.

비트 위치	1	2	3	4	5	6	7
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7
해밍코드			0		1	0	1

$$P_1 = D_3 \oplus D_5 \oplus D_7 = 0 \oplus 1 \oplus 1 = 0$$

$$P_2 = D_3 \oplus D_6 \oplus D_7 = 0 \oplus 0 \oplus 1 = 1$$

$$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

⇒ 0100101

122.

비트 위치	1	2	3	4	5	6	7
기호	P_1	P_2	D_3	P_4	D_5	D_6	D_7
해밍코드	0	0	1	1	0	1	1

$$P_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$

$$P_2 = P_2 \oplus D_3 \oplus D_6 \oplus D_7 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$P_4 = P_4 \oplus D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

⇒ $P_4 P_2 P_1 = 110$ 이므로 왼쪽으로부터 6번째 비트가 에러이므로 수정된 데이터는 0011001이다.

123.

- 해밍 코드는 오류 검출 및 교정이 가능한 코드다.
- 패리티 코드는 오류를 검출하는데 사용된다.
- Biquinary 코드는 7비트로 이루어져 있으며, 코드 당 1이 2개씩 포함되어 있어서 오류 검출이 가능하다.
- Excess-3 코드는 수치적 자료를 표현하는데 사용된다.

제03장. 디지털 논리회로

1.

2진수 각 비트의 값을 반전시키거나 1의 보수를 구하기 위해서 NOT 게이트를 사용한다.

2.

S는 Active-High로 동작하는 enable 단자이므로 S = 1 인 경우에만 NOT 게이트로 동작한다.

A	S	Y
0	0	Hi-Z(하이 임피던스)
1	0	Hi-Z(하이 임피던스)
0	1	1
1	1	0

3.

A	B	NOR	AND	OR	NAND
0	0	1	0	0	1
0	1	0	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

4.

A	B	NOR	NAND	XOR	OR
0	0	1	1	0	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	0	0	0	1

5.

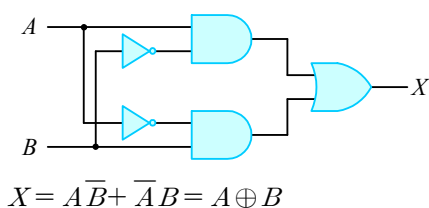
$$\begin{array}{r} 01101101 \\ ? \quad 11100110 \\ \hline 10011011 \end{array}$$

입력이 모두 1인 경우에만 출력은 0이 되고, 다른 경우에 출력은 1이므로 NAND 게이트다.

6.

입력이 모두 0인 경우에만 출력은 1이 되고, 다른 경우에 출력은 0이므로 NOR 게이트다.

7.



8.

$$\begin{array}{r} 10110 \\ \oplus \quad 01110 \\ \hline 11000 \end{array}$$

9.

A	B	NAND	XOR	XNOR	NOR
0	0	1	0	1	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	1	0

10.

- ① $A = 0, B = 0 \rightarrow F = 1$
 ② $A = 0, B = 1 \rightarrow F = 1$
 ③ $A = 1, B = 0 \rightarrow F = 1$
 ④ $A = 1, B = 1 \rightarrow F = 0$

11.

$$Y = \overline{A\bar{B}} \oplus C$$

- ① $Y = \overline{A\bar{B}} \oplus C = \overline{A0} \oplus 0 = \overline{A \cdot 1} \oplus 0 = \overline{A} \oplus 0 = \overline{A}$
 ② $Y = \overline{A\bar{B}} \oplus C = \overline{A0} \oplus 1 = \overline{A \cdot 1} \oplus 1 = \overline{A} \oplus 1 = A$
 ③ $Y = \overline{A\bar{B}} \oplus C = \overline{A1} \oplus 0 = \overline{A \cdot 0} \oplus 0 = \overline{0} \oplus 0 = 1 \oplus 0 = 1$
 ④ $Y = \overline{A\bar{B}} \oplus C = \overline{A1} \oplus 1 = \overline{A \cdot 0} \oplus 1 = \overline{0} \oplus 1 = 1 \oplus 1 = 0$

12.

제시된 회로는 XOR 게이트이다.

A	B	A + B	$\overline{AB}$	Y
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

$$\begin{array}{r} 1101 \\ \oplus \quad 0111 \\ \hline 1010 \end{array}$$

13.

$$\textcircled{4} A(\bar{A} + AB) = A\bar{A} + AAB = AB$$

14.

$$\textcircled{4} A + 1 = 1$$

15.

$$\textcircled{4} A + A = A$$

16.

$$F = \overline{AB} = \overline{A} + \overline{B}$$

17.

$$\begin{aligned} X(X + Y) &= XX + XY = X + XY \\ &= X(1 + Y) = X \end{aligned}$$

18.

$$\begin{aligned} F &= A + \overline{A}B = A(\overline{B} + B) + \overline{A}B \\ &= A\overline{B} + AB + \overline{A}B = A + B \end{aligned}$$

		B	
		0	1
A	0	0	1
	1	1	1

19.

$$\begin{aligned} Y &= \overline{AB} + \overline{A}B = \overline{AB} \cdot \overline{A}B = (A + \overline{B})(\overline{A} + B) \\ &= A\overline{A} + AB + \overline{A}B + \overline{B}B = AB + \overline{A}B \end{aligned}$$

20.

$$\begin{aligned} S &= (A + B)(\overline{AB}) = (A + B)(\overline{A} + \overline{B}) \\ &= A\overline{A} + AB + \overline{A}B + \overline{B}B = \overline{A}B + \overline{A}B \end{aligned}$$

21.

$$Y = A + AB + AC = A(1 + B + C) = A$$

22.

$$\begin{aligned} F &= (A + B)(A + C) = A + AC + AB + BC \\ &= A(1 + B + C) + BC = A + BC \end{aligned}$$

23.

입력으로부터 출력 방향으로 논리식을 쓰면
 $Y = (\overline{A} + \overline{B})B$ 가 된다.

24.

입력으로부터 출력 방향으로 논리식을 쓰면
 $F = \overline{A + B} \cdot \overline{C}$ 가 된다.

25.

입력으로부터 출력 방향으로 논리식을 쓰면
 $Y = A\overline{B} + \overline{A}B$ 가 된다.

26.

입력으로부터 출력 방향으로 논리식을 쓰고, 드모르간

의 정리를 적용하여 정리한다.

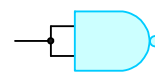
$$\begin{aligned} F &= \overline{\overline{X} \cdot \overline{Y}} + X = \overline{\overline{X} \cdot \overline{Y}} + X \\ &= \overline{\overline{X} \cdot \overline{Y}} + Y + X = X(1 + \overline{Y}) + Y = X + Y \end{aligned}$$

27.

입력으로부터 출력 방향으로 논리식을 쓰고, 드모르간의 정리를 적용하여 정리한다.

$$F = \overline{A + B} + \overline{A + B} = \overline{A + B} \cdot \overline{A + B} = A + B$$

28.

 는 NOT 게이트이므로 입력으로부터 출력 방향으로 논리식을 쓰고, 드모르간의 정리를 적용하여 정리한다.

$$\begin{aligned} F &= \overline{AB} \cdot \overline{AB} \cdot \overline{AC} = \overline{AB} + \overline{AB} + \overline{AC} \\ &= \overline{AB} + \overline{AB} + \overline{AC} \end{aligned}$$

29.

$$\begin{aligned} \overline{F} &= (\overline{A + B + C})(\overline{A + B + C}) \\ &= (\overline{A + B + C}) + (\overline{A + B + C}) \\ &= ABC + A\overline{B}C = AC(B + \overline{B}) = AC \end{aligned}$$

30.

최소항(minterm)은 입력 변수를 모두 포함하는 AND 항이며, 출력값이 1인 부분이다.

31.

$$\begin{aligned} F(A, B, C) &= A + \overline{B}C \\ &= A(\overline{B} + B)(\overline{C} + C) + (\overline{A} + A)\overline{B}C \\ &= \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + \overline{A}BC + A\overline{B}C + A\overline{B}\overline{C} + ABC \\ &= \Sigma_m(1, 4, 5, 6, 7) \end{aligned}$$

32.

		BC				
		00	01	11	10	
A	0	1	1	0	0	
	1	1	1	1	0	

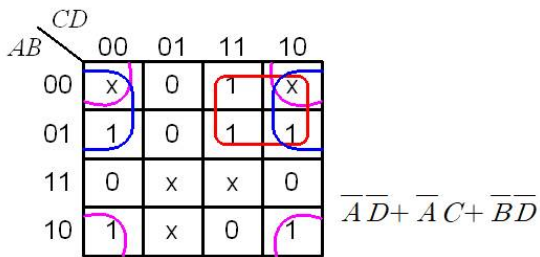
$B+AC$

33.

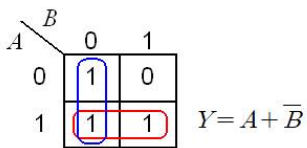
		CD				
		$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$	
AB	$\overline{A}\overline{B}$	1	0	0	1	
	$\overline{A}B$	1	1	1	1	
	$A\overline{B}$	0	0	1	0	
	AB	1	0	0	1	

$\overline{A}B + \overline{B}D + BCD$

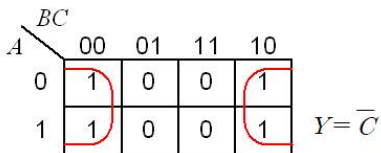
34.



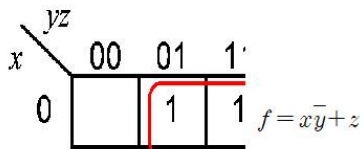
35.



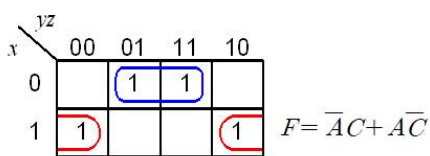
36.



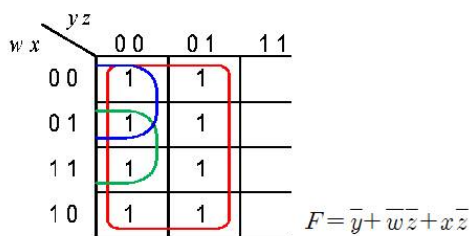
37.



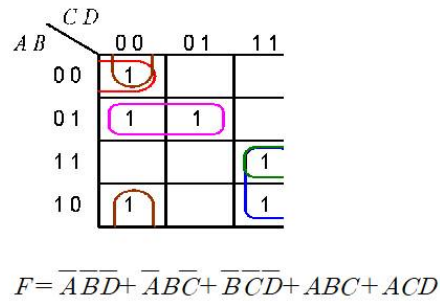
38.



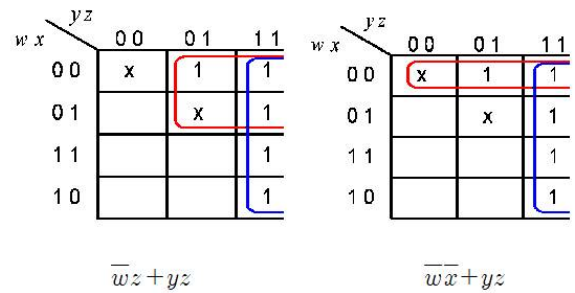
39.



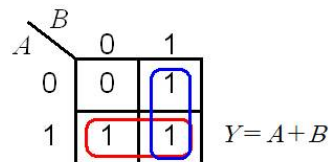
40.



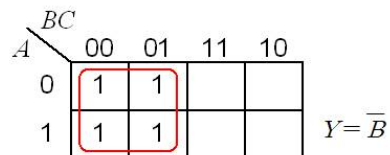
41.



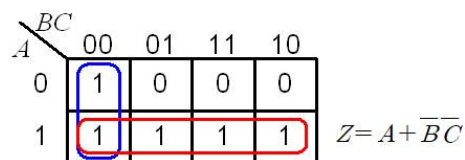
42.



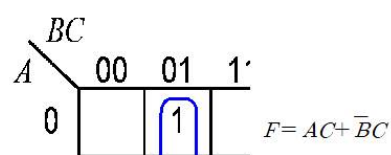
43.

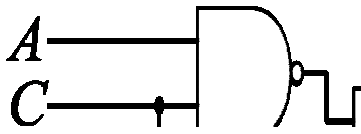


44.



45.





∴ NAND 게이트 4개가 필요하다.

46.

$$\begin{aligned} Y &= \overline{A(\overline{AB})} \cdot \overline{B(\overline{AB})} = A(\overline{AB}) + B(\overline{AB}) \\ &= A(\overline{A+B}) + B(\overline{A+B}) \\ &= A\overline{A+B} + B\overline{A+B} = A\overline{B} + \overline{A}B = A \oplus B \end{aligned}$$

47.

$$\begin{aligned} &((AB \oplus C) + \overline{CD}) + (\overline{A \oplus CD}) \\ &= ((1 \cdot 1 \oplus 0) + \overline{01}) + (\overline{1 \oplus 0 \cdot 1}) \\ &= ((1 \oplus 0) + \overline{1 \cdot 1}) + (\overline{1 \oplus 0}) \\ &= (\overline{1+1}) + (\overline{1}) = \overline{1} + \overline{1} = 0 \\ &(\overline{ABC} + B\overline{C}) \oplus (\overline{A+C})(\overline{BAD}) \\ &= (1 \cdot \overline{1} \cdot 0 + 1 \cdot \overline{0}) \oplus (\overline{1+0})(\overline{1 \cdot 1 \cdot 1}) \\ &= (0+1) \oplus (1)(0) = 1 \oplus 0 = 1 \end{aligned}$$

48.

- 조합논리회로 : 가산기, 인코더, 디코더, 멀티플렉서, 디멀티플렉서
- 순서논리회로 : 카운터, 레지스터

49.

1비트 비교기 진리표

입력		출력			
X	Y	X = Y	X ≠ Y	X < Y	X > Y
0	0	1	0	0	0
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	0	0	0
		$\overline{X \oplus Y}$	$X \oplus Y$	$\overline{X}Y$	$X\overline{Y}$

50.

제시된 조건에 대해 진리표를 작성한다.

A	B	F
0	0	0
0	1	0
1	0	1
1	1	0

$F = A\overline{B}$

51.

반가산기는 $S = \overline{A}B + A\overline{B} = A \oplus B$, $C = AB$ 로 구성된다.

52.

반가산기는 $S = \overline{A}B + A\overline{B} = A \oplus B$, $C = AB$ 로 구

성된다.

53.

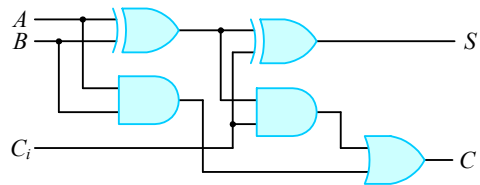
진리표로부터 논리식을 작성하면
 $S = x \oplus y$, $C = xy$ 이다.

54.

입력			출력	
x	y	C_1	C_2	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

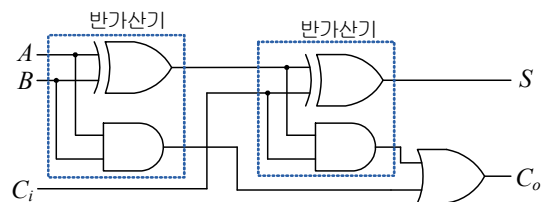
55.

전가산기는 다음과 같이 구성되므로 NAND 게이트는 포함되지 않는다.



56.

전가산기는 반가산기 2개와 OR 게이트 1개로 구성할 수 있다.



57.

제시된 회로는 전가산기다.

58.

전가산기의 진리표로부터 캐리 비트 C에 대한 논리식을 구하면 다음과 같다.

$$\begin{aligned} C &= \overline{x}yz + x\overline{y}z + xy\overline{z} + xyz \\ &= z(\overline{x}y + x\overline{y}) + xy(\overline{z} + z) \\ &= (x \oplus y)z + xy \end{aligned}$$

59.

다수결 함수는 입력 수가 반드시 홀수이고, 반수 이상의 입력이 1일 때 출력이 1이 되는 논리함수이며, 전

가산기 회로에서 C_o 의 논리함수가 다수결 함수이다.

입력			출력	
A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

60.

- n 개의 전가산기로 구성할 수 있다.
- $(n-1)$ 개의 전가산기와 1개의 반가산기로도 구성 가능하다.

61.

레지스터의 비트 수만큼 전가산기가 필요하다.

62.

제시된 회로는 전가산기가 4개이므로 4비트 병렬 가감산기다. $S=0$ 이면 가산기, $S=1$ 이면 감산기가 된다.

63.

n 비트로 부호화된 2진 정보를 최대 2^n 개의 출력으로 변환하는 조합논리회로를 디코더(decoder)라고 한다.

64.

제시된 진리표는 2×4 디코더(decoder)다.

65.

마이크로프로세서와 RAM들을 접속하기 위하여 디코더(decoder)를 사용한다.

66.

제시된 논리회로는 입력이 2개, 출력이 4개인 2×4 디코더 회로다.

67.

2×4 디코더 : 입력이 2개, 출력은 $4(=2^2)$ 개

68.

2×4 디코더의 진리표

Y_1	Y_0	X_3	X_2	X_1	X_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

69.

$$\begin{aligned}
 F &= \overline{B} + \overline{AC} = \overline{B} + \overline{A} + \overline{C} \\
 &= (\overline{A} + A)\overline{B}(\overline{C} + C) + \overline{A}(\overline{B} + B)(\overline{C} + C) \\
 &\quad + (\overline{A} + A)(\overline{B} + B)\overline{C} \\
 &= \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC \\
 &= \Sigma m(0, 1, 2, 3, 4, 5, 6)
 \end{aligned}$$

70.

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

$$F = \overline{A}\overline{B} + AB$$

71.

X	Y	F_1	F_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

72.

제시된 회로를 분석하면 다음과 같다.

S_1	S_0	F
0	0	$A + B$
0	1	$\overline{A}B + A\overline{B}$
1	0	AB
1	1	$\overline{A}$

73.

인코더(encoder)는 키보드와 같은 입력장치에 사용한다.

74.

인코더는 입력이 2^n 개이면 출력은 n 개이므로 입력선이 $8(=2^3)$ 이면 출력선은 3개다.

75.

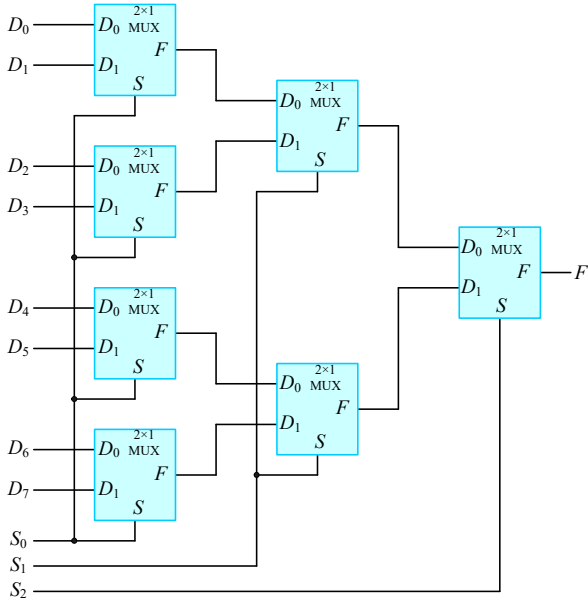
멀티플렉서는 n 개의 선택선에 의해 2^n 개의 입력 중에서 1개의 출력과 연결하는 조합논리회로이며, 데이터 선택회로(data selector)라고도 불린다.

76.

여러 개의 입력 중에서 1개를 선택하여 출력으로 연결하는 조합논리회로를 멀티플렉서라고 한다.

77.

2×1 멀티플렉서 7개를 사용하면 8×1 멀티플렉서를 구성할 수 있다.



78.

디멀티플렉서(demultiplexer)는 정보를 한 선으로 받아서 여러 개의 출력선들 중 한 개를 선택하여 정보를 전송하는 조합논리회로다. 데이터 분배기라고도 한다.

79.

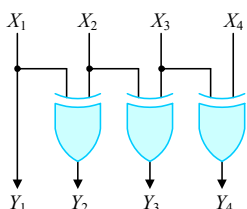
디멀티플렉서(demultiplexer)는 1개의 입력선과 n 개의 선택선에 의해 2^n 개의 출력선 중 하나를 선택한다.

80.

- $2^n \times 1$ 멀티플렉서에는 n 개의 선택 단자가 필요하므로 $4(=2^2)$ 개의 선택선이 필요하다.
- 1×2^n 디멀티플렉서에는 n 개의 선택 단자가 필요하므로 $5(=2^3)$ 개의 선택선이 필요하다.

81.

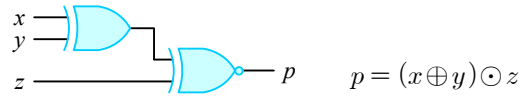
4비트의 2진수를 그레이 코드로 변환하는 논리회로



따라서 XOR 게이트는 3개가 필요하다.

82.

3비트 홀수 패리티 발생기



83.

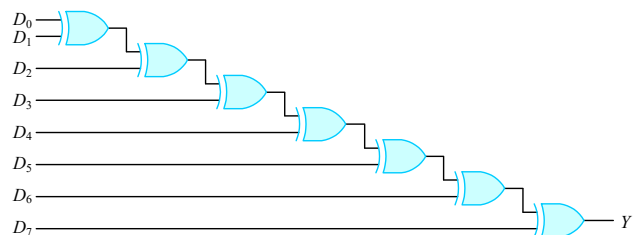
제시된 회로에서 출력 Y 에 대해 진리표를 작성하면 회로는 짝수 패리티 발생기가 된다.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

① 입력 A, B, C, D 가 모두 0이면, $Y=0$ 이다.

84.

그림은 짝수 패리티 검사기인 경우이며, 출력 $Y=0$ 이면 에러가 발생하지 않았다고 판단하며, $Y=1$ 이면 에러가 발생한 것으로 판단한다. 8비트 패리티 검사를 할 때 7개의 XOR 게이트가 필요하다.



85.

④ 감산기, 인코더, 디멀티플렉서는 조합논리회로다.

86.

④ 패리티 발생기, 멀티플렉서는 조합논리회로다.

87.

플립플롭이나 래치는 두 가지 상태 중 하나를 가지는 1비트 기억소자다.

88.

가산기는 조합논리회로다.

89.

$S = 1, R = 1$ 이면 $Q = \overline{Q} = 0$ 이 되어 부정 상태가 된다.

90.

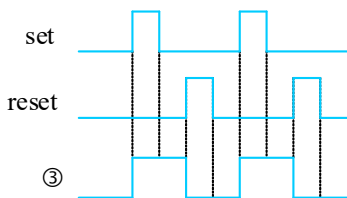
SR 플립플롭의 진리표

S	R	Q_{n+1}
0	0	Q_n (불변)
0	1	0
1	0	1
1	1	부정

91.

SR 플립플롭의 진리표를 참조하여 출력 파형을 그리면 된다.

S	R	Q_{n+1}
0	0	Q_n (불변)
0	1	0
1	0	1
1	1	부정



92.

D	$Q(t+1)$
0	0
1	1

$Q(t+1) = D$

93.

JK 플립플롭은 SR 플립플롭에서 $S = R = 1$ 일 때 발생하는 결점을 보완한 플립플롭이다.

94.

JK 플립플롭의 진리표

J	K	Q_{n+1}
0	0	Q_n (불변)
0	1	0
1	0	1
1	1	$\overline{Q_n}$ (toggle)

95.

④ $J = 1, K = 1$ 일 때 출력은 반전된다.

96.

T 플립플롭 : JK 플립플롭의 입력 J와 K를 묶어서

한 개의 입력선 T로 구성한 플립플롭으로, 1이 입력될 때마다 출력값이 반전된다.

97.

$J = K = 1$ 이므로 클록펄스(CLK)가 입력될 때마다 출력이 반전된다. 따라서 T 플립플롭으로 동작한다.

98.

$J = K = 1$ 이면 클록 펄스를 인가할 때마다 출력 Q는 상태가 반전되므로 Toggle 상태가 된다.

99.

Master-Slave 플립플롭은 어떤 형태의 플립플롭으로도 구성이 가능하며, master 플립플롭과 slave 플립플롭으로 된다. Master 플립플롭에는 클록 펄스를 그대로 인가하며, slave 플립플롭에는 반전된 클록 입력이 인가된다. M/S 플립플롭은 레이스(race) 현상을 방지하기 위해 제안된 플립플롭이지만 최근에는 레이스 현상을 피하기 위해 에지 트리거 플립플롭이 많이 사용된다.

100.

JK 플립플롭의 여기표

$Q(t)$	$Q(t+1)$	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

101.

현재 상태	입력	다음 상태	출력
A	x	A	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

102.

순서논리회로의 상태 여기표

현재 상태			다음 상태			플립플롭 입력		
A	B	C	A	B	C	T_A	T_B	T_C
0	0	0	1	0	0	1	0	0
0	1	0	0	1	1	0	0	1
0	1	1	0	0	0	0	1	1
1	0	0	1	1	1	0	1	1
1	1	1	0	1	0	1	0	1

카르노 맵을 이용하여 간소화한다.

		BC			
A		00	01	11	10
	0	1	x	0	0
	1	0	x	1	x

$T_A = \overline{A}\overline{B} + AB$

103.

JK 플립플롭의 여기표를 참조하여 문제에서 주어진 상태표를 근거로 순서논리회로의 상태 여기표를 구한다.

Q	A	Q^+	J	K
1	0	1	x	0
1	1	0	x	1
0	0	1	1	x
0	1	0	0	x

카르노 맵을 이용하여 간소화하면 다음과 같다.

		A	
Q		0	1
	0	1	0
	1	x	x

$J = \overline{A}$

		A	
Q		0	1
	0	x	x
	1	0	1

$K = A$

104.

④ JK 플립플롭의 입력 J, K 에 동시에 1이 입력되면 출력은 이전상태의 반전(toggle)이 된다.

105.

- 카운터는 T 플립플롭으로 설계한다.
- 카운터는 JK 플립플롭을 $J = K = 1$ 로 구성한 T 플립플롭으로 설계한다.

106.

비동기식 카운터를 리플 카운터(ripple counter)라고도 한다.

107.

$$0 \xrightarrow{1} 1 \xrightarrow{2} 2 \dots\dots 15 \xrightarrow{144} 16$$

$$\frac{144}{32} = 4 \dots 16(10000_{(2)})$$

108.

CP 가 모든 플립플롭에 공통으로 인가되므로 동기식 카운터이다. 또한 계수는 000에서 111까지 계수하므로 동기식 8진 상향 카운터다.

109.

- 모듈러스-14 카운터는 14가지의 상태를 갖는다.
- $\log_2 14 = 3.xxx$ 이므로 4개의 플립플롭이 필요하다.

110.

$\log_2 8 < \log_2 10 < \log_2 16$ 이므로 $\log_2 10 = 3.xxx$ 가 된다. 따라서 $\lceil \log_2 10 \rceil = \lceil 3.xxx \rceil = 4$ 개다.

111.

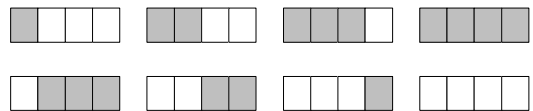
플립플롭은 1비트를 저장할 수 있으며, 플립플롭 여러 개를 일렬로 배열하여 적당히 연결함으로써 여러 비트로 구성된 2진수를 저장할 수 있게 한 것을 레지스터(register)라고 한다.

112.

시프트 레지스터는 주로 D 플립플롭으로 구성한다.

113.

예를 들어, $n = 4$ 인 경우 데이터 1011을 시프트 레지스터에 입력하는 경우 총 8가지 모드가 존재한다.



따라서 n 단으로 구성된 시프트 카운터는 $2n$ 개의 모드를 갖는다.

114.

③ 레지스터에 사용되는 플립플롭은 외부 입력을 그대로 저장하는 D 플립플롭이 적당하다.

115.

- 직렬 전송 레지스터는 동작 속도는 느리지만 회로 구성이 간단하며,
- 병렬전송 레지스터는 동작 속도는 빠르지만 회로 구성이 복잡하다.

제04장. 중앙처리장치

1.

중앙처리장치(CPU)는 산술논리연산장치, 제어장치, 레지스터로 구성되어 있다.

2.

누산기(accumulator)는 논리연산 및 수치연산을 행할 때 사용되는 레지스터이다.

3.

④ 센서 신호의 변환은 입출력장치에서 담당한다.

4.

산술논리연산장치(ALU)는 조합논리회로만으로 구성된다. 반면에 명령 레지스터, 프로그램 카운터, 어큐뮬레이터는 레지스터로 구성되며, 레지스터는 순서논리회로다.

5.

마이크로프로세서의 크기를 결정하는 가장 대표적인 요소는 레지스터의 크기다.

6.

중앙처리장치에서는 주소 버스(address bus), 데이터 버스(data bus), 제어 버스(control bus)를 사용한다.

7.

마이크로프로세서는 산술논리연산장치, 제어장치, 레지스터들로 구성되어 있으며, 내부 버스(internal bus)를 이용하여 데이터를 전달한다.

8.

연산장치(ALU)는 산술연산장치와 논리연산장치로 이루어져 있다.

9.

산술논리연산장치(ALU)는 산술연산과 논리연산을 담당한다.

10.

산술연산의 기본 연산은 덧셈이다.

11.

산술논리연산장치(ALU)는 산술연산과 논리연산을 담당한다. 어드레스 증가는 제어장치에서 수행한다.

12.

중앙처리장치는 2의 보수를 이용하여 가산함으로써 감산 기능을 처리한다. $F = A + \overline{B} + 1$ 에서 $\overline{B} + 1$ 은 2의 보수를 의미한다.

13.

제시된 구성도는 $F = A + \overline{B} + C_{in} = A + \overline{B} + 1$ 이다. 여기서 $\overline{B} + 1$ 은 B의 2의 보수를 나타낸다. 따라서 2의 보수를 이용한 덧셈으로서 최종적으로는 뺄셈 연산($F = A - B$)을 수행한다.

14.

16진수로 뺄셈을 수행하면 다음과 같다.

$$\begin{array}{r} F F F F F F 6 1 \\ - 0 0 0 0 0 4 F \\ \hline F F F F F F 1 2 \end{array}$$

15.

- 논리연산 : AND, OR, NOT
- 산술연산 : ADD

16.

논리회로에서의 인버터(inverter)와 같은 논리연산자로 이용되거나 음수의 표현에 필요한 연산자는 complement 연산(보수 연산)이다.

17.

연산 결과의 상위 바이트가 모두 0이 되므로 $D2E2_{(16)}$ 과 $00FF_{(16)}$ 를 AND 연산하면 된다.

18.

$$\begin{array}{r} 80H = 1 0 0 0 0 0 0 0 \\ 15H = \text{OR } 0 0 0 1 0 1 0 1 \\ \hline 1 0 0 1 0 1 0 1 \end{array}$$

19.

Rotate는 기존 데이터를 회전시키는 것이므로 데이터를 모두 0으로 바꿀 수는 없다.

20.

XOR 연산은 두 비트가 같으면 0이고, 서로 다르면 1이다.

$$\begin{array}{r} 1 0 0 1 0 0 1 0 \\ 0 0 0 0 1 1 1 1 \\ \hline 1 0 0 1 1 1 0 1 \end{array} \quad (\text{연산 결과})$$

$$\begin{array}{r} 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ \text{AND } \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 1} \\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0 \end{array}$$

$$\begin{array}{r} 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ \text{OR } \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 1} \\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \end{array}$$

$$\begin{array}{r} 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ \text{XNOR } \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 1} \\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0 \end{array}$$

21.

AND는 마스크(mask)를 이용하여 비수치 데이터의 불필요한 부분을 제거하는데 사용하는 연산이다.

22.

AND 연산에서 0에 대응되는 비트는 0이 된다.

23.

OR 연산은 대응되는 비트들 간에 0과 OR시켜서 데이터를 혼합하거나, 1과 OR시켜서 1을 삽입시키는 연산이다.

24.

XOR는 두 입력이 같으면 0, 다르면 1을 출력하는 논리연산이므로, XOR 연산은 두 값의 비교를 위한 연산이라고 할 수 있다.

25.

XOR 연산은 특정 비트를 반전시킬 때 사용하는 연산이다. 예를 들어, 2진수 1100 1001에서 하위 4비트만을 반전시키려는 경우 1100 1001과 0000 1111을 XOR 연산하면 된다.

$$\begin{array}{r} 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1 \\ \oplus \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 1} \\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0 \end{array}$$

26.

74를 부호와 절댓값으로 표현하면 01001010이다.
오른쪽 1비트 산술 시프트 $\Rightarrow$ 00100101 $\Rightarrow$ 37₍₁₀₎

27.

먼저 -36을 2의 보수로 표현하려면 8비트로 +36을 구하면 00100100이다. 이를 2의 보수로 변환하면 11011100이다. 부호 비트를 제외하고 왼쪽 시프트를 수행한다. 빈자리에는 0이 채워진다.

왼쪽 1비트 산술 시프트 $\Rightarrow$ 10111000

28.

먼저 -14를 2의 보수로 표현하려면 8비트로 +14를 구하면 00001110이다. 이를 2의 보수로 변환하면 11110010이다. 오른쪽 시프트를 수행하고, 빈자리에는 부호 비트인 1이 채워진다.

오른쪽 1비트 산술 시프트 $\Rightarrow$ 11111001

29.

오른쪽으로 N 번 시프트하면 기존 데이터의 $1/2^N$ 이므로 -77/4를 수행하면 -19가 된다.

30.

11011001을 2번 좌측 시프트하면(빈자리에는 0이 채워진다)

1번 시프트 $\Rightarrow$ 10110010

2번 시프트 $\Rightarrow$ 01100100

11011001을 2번 좌측 로테이트하면

1번 로테이트 $\Rightarrow$ 10110011

2번 로테이트 $\Rightarrow$ 01100111

31.

산술적 shift는 곱셈과 나눗셈을 수행하거나 혹은 곱셈과 나눗셈에 보조적으로 이용된다.

32.

산술 시프트 연산에서는 오버플로(overflow)의 발생 여부를 확인함으로써 연산 결과가 수의 범위를 넘어서는지 검사해야 한다.

33.

왼쪽으로 k 번 시프트하면 기존 데이터의 2^k 배가 되므로 여섯 번 shift-left하면 $\text{number} \times 2^6$ 가 된다.

34.

오른쪽으로 k 번 시프트하면 기존 데이터의 $1/2^k$ 배가 되므로 두 번 shift-right하면 $\text{number} \div 2^2$ 가 된다.

35.

- 논리 시프트는 왼쪽이건 오른쪽이건 빈자리에는 0을 채운다.
- 산술 시프트는 왼쪽 시프트는 빈자리를 0으로 채우지만 오른쪽 시프트인 경우는 부호비트로 채운다. 따라서 양수인 경우에는 0으로 채우지만 음수인 경우에는 1이 채워진다.

36.

홀수를 오른쪽으로 한 번 Shift하면 0.5의 오차가 발생하므로 ② 10001111₍₂₎을 우측으로 시프트하면 절단 현상이 발생한다.

37.

초기 : 11011001

- 1회 제어 신호 : 11101100
- 2회 제어 신호 : 01110110
- 3회 제어 신호 : 00111011

38.

MAR(Memory Address Register)은 CPU에서 기억장치 내의 특정번지에 있는 데이터나 명령어를 인출하기 위해 그 번지를 기억하는 역할을 한다.

39.

MAR(Memory Address Register)은 현재 수행되고 있는 연산에서 사용되는 메모리 위치 또는 I/O 장치의 주소를 저장하는 역할을 한다.

40.

MBR(Memory Buffer Register)는 기억장치를 출입하는 데이터를 일시적으로 저장하는 버퍼 레지스터다.

41.

Read하거나 Write할 때 데이터가 반드시 거쳐야 하는 레지스터는 MBR(Memory Buffer Register)이다.

42.

명령어 수행 시 주기억장치로부터 명령어를 읽어서 MBR(Memory Buffer Register)에 저장한 후, 이를 해독하기 위해 CPU 내의 명령 레지스터(instruction register, IR)로 명령어를 읽어온다.

43.

다음에 실행할 명령어의 주소를 갖고 있는 레지스터는 프로그램 카운터(program counter, PC)다.

44.

프로그램 카운터(program counter, PC)는 다음에 인출할 명령어의 주소를 가지고 있는 레지스터다.

45.

현재 수행 중인 명령어 코드를 저장하고 있는 레지스터는 IR(Instruction Register)이다.

46.

주기억장치에서 명령어를 읽어 CPU의 명령 레지스터(IR)로 가져오고, 제어장치는 명령 레지스터의 opcode를 기반으로 명령어 실행을 지시하는 세부적인 제어 신호를 발생시킨다.

47.

명령 레지스터(Instruction Register)는 프로그래머가 직접 사용할 수 없다.

48.

Program Counter(PC)에는 다음에 실행할 명령어의 주소가 들어 있으므로 데이터 레지스터라고 볼 수 없다.

49.

중앙처리장치에서 덧셈, 뺄셈, 곱셈, 나눗셈 등의 연산 결과 등을 일시적으로 저장해 두는 레지스터를 누산기(accumulator)라고 한다.

50.

연산의 중심이 되는 레지스터는 누산기다. 누산기는 연산 결과를 일시적으로 저장하는 역할을 한다.

51.

컴퓨터의 내부 상태를 나타내는 레지스터를 상태 레지스터(status register)라고 한다.

52.

CPU에서 논리연산 또는 산술연산을 수행하였을 때 그 결과의 상태를 나타내는 레지스터를 상태 레지스터(status register)라고 한다.

53.

PSW(Program Status Word)는 컴퓨터 시스템 내부에서 순간순간의 시스템 상태를 기록하고 있는 워드(word)를 의미한다.

54.

인터럽트 플래그(interrupt flag)는 인터럽트가 발생되면 세트되는 플래그이므로 연산 결과에 영향을 받지 않는다.

55.

$A - B$ 를 수행하면 다음과 같다.

$$\begin{array}{r} 11110000 \\ - 00010100 \\ \hline 11011100 \end{array}$$

11011100에서 최상위 비트는 부호를 나타내므로 $S = 1$ 이고, 연산 결과는 0이 아니므로 $Z = 0$ 이다.

56.

$$\begin{array}{r} 11111111 \\ 00100001 \\ - 11111111 \\ \hline 10010000 \end{array}$$

- 빨간색으로 표시한 부분이 같으므로 오버플로는 발생하지 않음 $\rightarrow V=0$
- 덧셈 결과 00100000은 0이 아니므로 $\rightarrow Z=0$
- 덧셈 결과 00100000에서 최상위 비트가 0이므로 양수 $\rightarrow S=0$
- 덧셈 결과 최상위 비트에서 캐리 발생 $\rightarrow C=1$

57.

스택 포인터(Stack Pointer)는 프로그래머가 직접 사용할 수 있지만, MAR, MBR, IR은 사용할 수 없다.

58.

- ④ 지정된 메모리 주소를 기억하기 위한 레지스터를 데이터 카운터(DC)라 하며, 매번 메모리 데이터를 지정할 때마다 변경되어야 한다.

59.

데이터 전송 명령에 속하는 것은 LOAD, STORE, MOVE이며, AND는 논리연산 명령이다.

60.

LOAD : 메모리의 내용을 레지스터에 전달

61.

STORE : 레지스터의 내용을 주기억장치로 전달

62.

MOVE : 하나의 레지스터에 기억된 자료를 다른 레지스터로 전달

63.

일반적으로 사용자 프로그램을 처리함에 있어서 중앙처리장치(CPU)가 가장 많이 실행하는 명령의 종류는 중앙처리장치와 주기억장치 사이의 자료 전달(LOAD, STORE)이다.

64.

명령 레지스터(Instruction Register)는 실행될 명령어를 기억하고 있다.

65.

명령어는 연산자와 오퍼랜드로 구성된다.

66.

명령어 형식에서 수행할 데이터가 저장된 곳을 나타내는 부분은 오퍼랜드(operand)다.

67.

오퍼랜드(operand)와 명령의 형식은 상관없다.

68.

오퍼랜드 개수에 따라 0-주소, 1-주소, 2-주소, 3-주소 명령어로 분류한다.

69.

오퍼랜드 개수에 따른 분류다.

70.

0-주소 명령어는 연산에 필요한 오퍼랜드와 결과의 저장장소가 모두 묵시적으로 지정된 경우로, 스택(stack)을 갖는 구조에서 사용된다.

71.

0-주소 명령어에서 모든 연산은 스택에 있는 자료를 이용하여 수행한다.

72.

0-주소 명령어에서 데이터를 기억시킬 때는 PUSH를, 꺼낼 때는 POP을 사용한다.

73.

```
PUSH A
PUSH B
PUSH C
ADD      (B + C = 5)
PUSH D
PUSH E
ADD      (D + E = 5)
MUL      (5 × 5 = 25)
POP X    (X = 25)
```

74.

1-주소 명령어는 연산 대상이 되는 2개 중 하나만 표현하고 나머지 하나는 묵시적으로 지정된 누산기(AC)를 사용한다.

75.

1-주소 명령어다. 예를 들어, ADD B는 다음과 같은 의미를 갖는다.

```
ADD B ; AC ← AC+B
```

76.

- ①

```
LOAD A ; AC ← A
ADD B ; AC ← AC+B
MUL C ; AC ← AC*C
STORE X ; X ← AC
```

77.

1-주소 명령어에서는 누산기에 기억되어 있는 자료를

사용한다. 따라서 $AC \leftarrow AC \times M[A]$ 가 된다.

78.

오퍼랜드가 2개이므로 2-주소 명령어다.

79.

최종적인 결과는 R1에 저장되어 있으므로 이를 Y로 이동하면 되므로 MOV Y, R1이다.

80.

3-주소 명령어는 연산에 필요한 오퍼랜드 2개와 결과 값의 저장장소가 모두 달라 명령어에 주소 3개를 모두 지정한다. 3-주소 명령어를 사용하면 프로그램 길이가 짧아지지만 명령어 해독 과정이 복잡해지는 단점이 있다.

81.

3-주소 명령어는 연산에 필요한 오퍼랜드 2개와 결과 값의 저장장소가 모두 달라 명령어에 주소 3개를 모두 지정한다. 연산 후에도 입력 데이터가 변하지 않고 보존된다.

82.

오퍼랜드가 3개이므로 3-주소 명령어다.

83.

③ R2와 R3를 더하여 R1에 값을 넣는다.

84.

3-주소 명령어를 사용하면 프로그램 길이가 짧아지지만 명령어 해독 과정이 복잡해지므로 많은 cycle time이 필요하다.

85.

내용이 연산 결과 저장으로 소멸되는 것은 2-주소 명령어다.

86.

메모리에 접근하지 않는 스택 명령의 instruction cycle time이 가장 짧다.

87.

스택 구조 CPU는 수식을 Postfix 표기법으로 변환한 후, 0-주소 명령어 형식을 사용한다.

88.

인스트럭션(명령) 설계 시 고려사항

- 데이터 구조
- 주소지정 방식

- 연산자의 수와 종류
- 사용 빈도 및 기억 공간

89.

주소지정 기능은 연산자(OP code)의 기능과 관련 없다.

90.

명령어의 구성

Operation code	Mode	Operand
----------------	------	---------

- Operation code(연산자부) : 수행해야 할 동작에 맞는 연산자를 표시하며, 흔히 OP-Code라고 한다.
- Mode : 주소부의 유효 주소가 결정되는 방법을 지정한다. 모드 비트가 0이면 직접, 1이면 간접이다.
- Operand(오퍼랜드, 주소부) : 실제 데이터에 대한 정보를 표시하는 부분이다. 기억장소의 주소, 레지스터 번호, 사용할 데이터 등을 표시한다.

따라서 버스 개수는 명령어 내 비트들의 할당에 영향을 주는 요소가 아니다.

91.

OR는 이항연산이다.

92.

complement는 단항연산이다.

93.

- 단항연산 : complement
- 이항연산 : OR, AND, XOR

94.

동일한 명령을 반복 실행하거나 명령의 실행 순서를 변경시키는 기능은 제어기능이다.

95.

$$2^8 = 256$$

96.

$$\text{명령어 수} = 2^4 = 16, \text{ 주소의 수} = 2^9 = 512$$

97.

OP code가 n 비트 일 경우 연산자의 종류 : 2^n 개
첫 번째 : $2^3 = 8$, 두 번째 : $2^6 = 64$ 이므로 $8 + 64 = 72$ 개이다.

98.

opcode가 5비트이므로 최대 $32(=2^5)$ 개 명령어를 가질 수 있다.

99.

$$\lceil \log_2 75 \rceil = \lceil 6.xxx \rceil = 7$$

100.

명령어 길이 = 40비트

오퍼랜드 길이 = $2^{16} \rightarrow 16$ 따라서 $40 - 16 \times 2 = 8$ 비트

101.

opcode 비트 수 : $2^9 = 512$ 이므로 9비트operand 종류는 8가지이므로 $2^3 = 8 \rightarrow 3$ 비트따라서 $9 + 2 \times 3 = 15$ 비트

102.

$32 - 8 - 1 - 20 = 3$ 비트이므로 레지스터는 총 $8(=2^3)$ 개다.

103.

F : function code(16개) $\rightarrow$ 4비트B : index register (3개) $\rightarrow$ 2비트I : addressing mode(2개) $\rightarrow$ 1비트M : address 공간(4,096개) $\rightarrow$ 12비트 $\Rightarrow 4 + 2 + 1 + 12 = 19$ 비트

104.

임시 어드레싱은 주소지정방식이 아니다.

105.

즉시(immediate) 주소지정방식은 명령어의 오퍼랜드에 실제 데이터를 내포하고 있는 방식이다. 별도의 기억장치를 액세스하지 않고 CPU에서 곧바로 데이터를 이용할 수 있다.

106.

즉시(immediate) 주소지정방식은 명령어의 오퍼랜드에 실제 데이터를 내포하고 있는 방식이므로 기억장치를 액세스하지 않는다.

107.

즉시(immediate) 주소지정방식은 명령어의 오퍼랜드에 실제 데이터를 내포하고 있는 방식이다. 기억장치를 액세스하지 않고 CPU에서 곧바로 데이터를 이용할 수 있어서 실행 속도가 빠르다는 장점이 있다.

108.

즉시(immediate) 주소지정방식은 명령어의 오퍼랜드에 실제 데이터를 내포하고 있는 방식이므로 메모리를 액세스하지 않는다.

109.

“ADD A, 30H” 명령어는 A의 내용과 30H를 더해서 그 결과를 A에 저장하라는 명령어이므로 즉시(immediate) 주소지정방식이다.

110.

직접 주소 지정(direct addressing)은 명령어의 주소(address)부로 유효 주소를 이용하는 방법이다.

111.

직접 주소 지정(direct addressing)은 명령어의 주소(address)부로 유효 주소를 이용하는 방법이다.

112.

직접주소지정 방식이므로 5301번지에 있는 6501이 연산될 데이터다.

113.

120번지에 저장된 200이 직접주소지정 방식의 오퍼랜드가 된다.

114.

AC에 저장된 550과 457번지에 있는 950을 더하여 그 결과를 AC에 저장한다.
 $500 + 950 = 1500$

115.

상대 주소 지정 방식에서 기억장소의 위치는 “명령어 주소부분 + PC” 이 된다.

116.

간접 주소지정방식은 기억장치를 가장 많이 액세스하므로 메모리 이용 효율이 높다.

117.

상대 주소 지정 방식은 현재 프로그램 코드가 실행되고 있는 위치에서 앞 또는 뒤로 일정한 변위만큼 떨어진 곳의 데이터를 지정하는 것이다.

118.

상대 주소 지정 방식에서 사용하는 레지스터는 프로그램 카운터(PC)다.

119.

상대 주소 지정 방식에서 기억장소의 위치는 “명령어 주소부분 + PC” 이 된다.

120.

상대주소지정방식에서 오프셋(offset)은 보통 2의 보수

로 표현한다. 오프셋이 8비트이므로 $-2^{8-1} \sim +2^{8-1}-1$ 이다.

121.

오퍼랜드(operand)가 레지스터를 지정하고, 다시 그 레지스터의 값이 유효 주소가 되는 방식을 레지스터 주소지정 방식이라고 한다.

122.

베이스(Base) 레지스터 주소지정방식
기억장소의 위치 = 명령어 주소부분 + BR(Base Register)

- 프로그램의 재배치가 용이하다.
- 다중 프로그래밍 기법에 많이 사용된다.

123.

인덱스 주소 지정방식은 배열(array)이나 표(table)의 데이터를 액세스하는 유용하다.

124.

계산에 의한 주소지정방식의 종류

- 상대 주소지정방식
- 인덱스 레지스터 주소지정방식
- 베이스 레지스터 주소지정방식

125.

상대주소 = PC + 프로그램 주소(변위)이므로 두 값을 더해주면 된다. 따라서 $2FA50_{(16)} + 0B_{(16)} = 2FA5B_{(16)}$ 이다.

126.

$256AH - 75H = 24F5H$
 $24F5H + \text{분기명령어}(3\text{byte}) = 24F8H$

127.

JUMP 명령어가 750번지에 저장되어 있으므로 JUMP 명령어가 수행될 때, PC에는 751이 저장된다.

상대주소 = PC + 프로그램 주소(변위)이므로 두 값을 더해주면 된다. 따라서

A = 56일 때 $751 + 56 = 807$ 이고,
 A = -61일 때는 $751 - 61 = 690$ 이다.

128.

간접 주소 지정 : 명령어 내의 주소부에 실제 데이터가 저장된 장소의 주소를 가진 기억장소의 번지를 표현하므로 최소한 주기억장치를 2번 이상 접근하여 데이터가 있는 기억장소에 도달한다.

129.

간접 주소지정방식은 최소한 주기억장치를 2번 이상 액세스하므로 기억장치를 가장 많이 액세스한다.

130.

③ 간접주소지정방식(indirect addressing mode)

131.

간접주소지정방식이므로 500번지에 저장된 2002가 연산장치로 전송된다.

132.

간접주소모드이므로 값을 참조하기 위해서 2번, 총 4번 메모리를 참조한다. 따라서 명령어 패치를 위한 메모리 참조 1회, 결과값을 저장하기 위한 참조 1회를 포함해서 총 6회의 메모리 참조가 이루어진다.

133.

172번지에 있는 0202가 유효 주소이며, 0202번지에 있는 3256이 데이터가 된다.

134.

간접주소지정방식이므로 2010번지의 내용과 누산기의 값을 더하고 결과를 누산기에 저장한다.

135.

SHL은 주소지정 필드가 없는 0-주소 명령으로서 묵시적 주소지정방식이다.

136.

묵시적 주소지정(implied addressing)의 예로서 누산기를 사용하는 1-주소 명령이 이에 해당한다.

137.

묵시적 주소지정(implied addressing)은 오퍼랜드의 소스나 목적지를 명시하지 않아도 암묵적으로 그 위치를 알 수 있는 주소 지정 방식이다. 누산기를 사용하는 1-주소 명령이나 스택 구조의 컴퓨터에서의 0-주소 명령어도 이에 해당한다.

138.

묵시적 주소지정(implied addressing)은 오퍼랜드의 소스나 목적지를 명시하지 않아도 암묵적으로 그 위치를 알 수 있는 주소 지정 방식이다.

139.

묵시적 주소지정(implied addressing)의 예로서 스택 구조의 컴퓨터에서의 0-주소 명령이 이에 해당한다.

140.

PUSH A 만이 묵시적 주소지정 방식이다.

141.

CPU 내의 레지스터에 오퍼랜드가 있으면 레지스터 주소 지정(register addressing) 방식이다.

142.

② 직접주소지정 방식은 명령의 주소부분이 그대로 유효 주소이다.

143.

④ 상대주소지정 방식은 프로그램 카운터와 관련이 있다.

144.

MOVE	Y,A	; Y=A
SUB	Y,B	; Y=Y-B
MOVE	T,D	; T=D
MPY	T,E	; T=T*E
ADD	T,C	; T=T+C
DIV	Y,T	; Y=Y/T

따라서 $(A-B)/(D*E+C)$

145.

프로그램 상에서 사용한 주소를 변경 없이 실제 기억 공간 내의 주소로 재배치할 수 있도록 서로 독립적이어야 한다.

146.

③ 지정할 수 있는 범위가 넓을수록 좋다.

147.

CISC(Complex Instruction Set Computer)형 프로세서는

명령어 길이와 형식이 다양하다.

148.

RISC 방식 컴퓨터는 제어장치가 단순하고 속도가 빠르다.

149.

RISC 구조 특징

- 명령어 크기가 동일하고 형식이 제한적이다.
- 주소 지정 방식이 단순하고 제한적이다.
- 모든 명령어는 한 사이클에 실행되지만 프로그램의 길이가 길다.
- 파이프라인이 쉽다.

150.

CISC 구조는 명령어 크기가 동일하고 형식이 제한적이다.

151.

- ARM 계열의 프로세서들은 주로 RISC 방식을 사용한다.
- 인텔은 CISC 방식을 많이 사용한다.

152.

RISC는 명령어가 적고, 레지스터가 많으며, 속도가 빨라서 주로 서버에서 쓰인다. CISC는 명령어가 많고, 레지스터가 적으며, 속도가 느려서 주로 PC에서 많이 쓰인다.

153.

② : RISC는 CISC에 비해 수행 속도가 빠르다.

제05장. 제어장치

1.

제어장치(Control Unit)는 명령어를 해독하고 시스템 전체에 제어 신호를 보내는 장치를 말한다.

2.

제어장치는 명령 해독기, 제어 주소 레지스터, 제어 기억장치, 제어 버퍼 레지스터, 신호 발생장치 등으로 구성되어 있다.

3.

① CPU 내의 제어신호들은 제어장치로부터 출력되는 항목이다.

4.

명령 레지스터(Instruction Register, IR)는 현재 실행 중인 명령어의 내용을 기억하는 레지스터이므로 OP code는 명령 레지스터에 들어간다.

5.

명령 레지스터(Instruction Register, IR)는 현재 실행 중인 명령어의 내용을 기억하는 레지스터다.

6.

명령 레지스터에 저장된 명령어 중에서 연산 코드(OP code)를 해독하는 장치는 명령 해독기(instruction decoder)다.

7.

마이크로 오퍼레이션(micro operation)은 명령을 수행하기 위해 CPU 내의 레지스터와 플레그의 상태 변환을 일으키는 동작으로, 마이크로 오퍼레이션이 순서적으로 일어나게 하려면 제어 신호(control signal)가 필요하다.

8.

하드와이어(hardwired) 제어방식

- 게이트, 플립플롭, 디코더 등의 디지털 회로를 이용하여 구현
- 속도가 아주 빠름
- 설계가 변경되면 제어장치를 새롭게 디자인해야함
- 주소 체계나 명령어가 복잡하면 회로도 복잡해짐

9.

하드와이어(hardwired) 제어방식은 하드웨어적으로 제어신호를 발생시키므로 속도가 빠르지만 명령어 세트의 변경이 어렵다. 플립플롭, 게이트, 디코더 등을 사

용하여 구현하므로 비용이 증가한다.

10.

하드와이어 제어 방법은 하드웨어(플립플롭, 게이트, 디코더 등)를 사용하여 구현하므로 명령어 세트의 변경이 어렵다.

11.

마이크로 명령을 수행할 수 있는 일련의 제어 워드가 ROM과 같은 기억장치에 저장된 것을 마이크로프로그램 램이라고 한다.

12.

제어기억장치는 보통 비휘발성 메모리인 ROM을 사용하여 구현한다.

13.

마이크로프로그램은 컴퓨터에 하드웨어로 내장되어 있으며, 컴퓨터의 동작을 제어하기 위한 프로그램이다.

14.

④ 마이크로 명령어(micro-instruction)의 비트 수는 프로세서가 사용하는 데이터의 비트 수와 같지 않다.

15.

③ 마이크로프로그램은 제어장치 내부의 제어 메모리에 기억된다.

16.

지정된 연산을 수행하는 동작은 마이크로 오퍼레이션의 동작을 시작하는 것과 관계없다.

17.

마이크로프로그램을 이용한 제어방식은 소프트웨어 기반 기술을 사용하므로 처리 속도가 느리다.

18.

② 주기억장치는 사용자가 그 내용을 변경시킬 수 있다.

19.

① 마이크로프로그램은 보통 ROM에 저장한다.

20.

③ 마이크로프로그램은 컴퓨터의 제작 단계에서 제어장치 내부의 제어 메모리(control memory)에 저장

한다.

21.

마이크로프로그램 제어 장치에서 마이크로프로그램은 ROM으로 구성된 제어 메모리에 저장한다. 따라서 마이크로프로그램은 읽기만 가능하다.

22.

수평적 마이크로프로그램에서는 제어 신호가 제어 신호당 1비트로 해독된 2진 형식으로 표현된다. 예를 들어 프로세서에 제어 신호가 50개 있다면 50비트의 제어 신호가 필요하다.

23.

수직 마이크로 명령(Vertical Micro Instruction)

- 제어 메모리 외부에서 디코딩 회로를 필요로 하는 마이크로 명령이다.
- 한 개의 마이크로 명령으로 한 개의 마이크로 동작만 제어할 수 있다.
- 마이크로 명령어의 비트 수가 감소된다.
- 제어 기억장치의 용량을 줄일 수 있다.
- 마이크로 명령어의 코드화된 비트들을 해독하기 위한 지연이 발생한다.

24.

③ 제어기억장치의 사용 용량이 커진다.

25.

④는 수평적 마이크로 명령어 형식에 대한 설명이다.

26.

OP code : 12bit, Flag 4개 주소 표현 : 2bit
따라서 operand 부는 $32-14=18$ bit이므로 워드 수는 2^{18} byte이다.

word 크기 : 32bit=4byte이므로

총용량 = word 크기 × 워드 수 $\Rightarrow 2^2 \times 2^{18} = 2^{20} = 1\text{MB}$

27.

마이크로 명령어는 여러 개의 필드로 구성되며, 각 필드(F1-F3)는 해독된 후 일련의 신호를 활성화한다. 모든 필드에 의해 제어 신호가 생성되면 최대 3개를 생성할 수 있다.

28.

② 저급(low-level) 언어일수록 좋다.

29.

④ 고정배선 제어방식은 마이크로프로그램 제어방식에 비해 고속의 제어가 가능하지만 융통성이 부족해 개발 수정이 용이하지 못하다.

30.

기억장치로부터 명령어를 인출하여 해독하고 해독된 명령어를 실행하기 위해 제어 신호를 발생시키는 각 단계의 세부 동작을 마이크로 오퍼레이션(micro-operation)이라고 한다.

31.

명령 처리 과정

인스트럭션 페치 → 인스트럭션 디코딩 → 오퍼랜드 페치 → 실행 → 인터럽트 조사

32.

명령어 사이클은 명령어의 인출에서 실행까지의 전체 과정을 의미한다.

33.

4가지 메이저 상태는 fetch, indirect, execute, interrupt다.

34.

프로그램 카운터로부터 다음에 실행할 명령의 주소를 읽어서 명령어를 메모리로부터 꺼내오는 명령 사이클은 인출 사이클(Fetch cycle)이다.

35.

인출 단계(fetch cycle)에서는 주기억장치에서 명령어를 가져오고, 프로그램 카운터 값을 +1 증가시킨다.

36.

인출 단계(fetch cycle)는 다음에 실행할 명령어를 기억장치로부터 CPU로 가져와 해독하는 단계로 인터럽트를 처리한 후 실행되는 단계이다.

37.

인출 단계(fetch cycle)는 명령어를 주기억장치에서 중앙처리장치의 명령 레지스터로 가져와 해독하는 단계이다.

38.

인출 단계(fetch cycle)는 명령어를 주기억장치에서 중앙처리장치의 명령 레지스터로 가져와 해독하는 단계이다.

39.

마이크로오퍼레이션에서 명령이 실행되기 위해 명령어 패치가 가장 먼저 이루어진다.

40.

인출 단계(fetch cycle)는 중앙처리장치가 모든 명령어

의 종류에 관계없이 반드시 거쳐야 한다.

41.

실행 사이클(execution cycle)에서는 실제로 명령을 수행한다.

42.

③ 인출 사이클(fetch cycle)이 실행되고 그 다음에 실행 사이클(execution cycle)이 수행된다.

43.

주기억장치에서 인출된 명령을 해독한 후 실행하는 주기를 실행 사이클(execution cycle)이라고 한다.

44.

인출(fetch) 중인 명령어를 실행(execution)한 후에 인터럽트 요청 여부를 검사하고 요청이 있으면 처리한다. 그러나 인터럽트 요청이 없다면 fetch cycle로 천이한다.

45.

$MAR \leftarrow PC$ 은 명령을 수행하는 과정에서 가장 먼저 수행되어야 하는 마이크로 오퍼레이션이다.

46.

제시된 내용은 인출 사이클(fetch cycle)이다.

47.

인출 사이클(fetch cycle)의 동작 순서

- ④ $MAR \leftarrow PC$
- ⑥ $MBR \leftarrow M(MAR)$, ③ $PC \leftarrow PC+1$
- ④ $IR \leftarrow MBR(OP)$

48.

CPU 클럭이 100MHz이므로 1주기는 $1/(100M) = 1/(100 \times 10^6) = 10ns$ 이다. 따라서 3개의 클럭으로 구성되는 인출 사이클에는 30ns가 소요된다.

49.

메모리의 내용을 AC로 가져오는 LDA(Load to AC) 마이크로 오퍼레이션이다.

50.

메모리의 내용을 AC로 가져오는 LDA(Load to AC) 마이크로 오퍼레이션이다.

51.

AC의 내용을 메모리에 저장하는 “STORE C” 명령이다.

52.

ADD 명령어의 마이크로 오퍼레이션

$MAR \leftarrow MBR(addr)$ // 실효주소를 MAR로 전송
 $MBR \leftarrow M(MAR)$ // 피연산자를 읽음
 $AC \leftarrow AC + MBR$ // 누산기와 덧셈 연산하고 AC에 저장

53.

$A - B$ 를 수행하기 위한 마이크로 동작이므로 $A \leftarrow A + \overline{B} + 1$ 이다.

54.

레지스터 R_1, R_2, R_3, R_4 를 사용하여 $A - B$ 연산을 수행하는 과정이다.

55.

ISZ(Increment and Skip if zero)

프로그램 루프의 반복실행을 위한 명령이다. 지정된 번지의 내용을 1 증가시킨 후 그 결과가 0이면 다음 명령을 skip하고, 0이 아니면 다음 명령을 수행한다.

56.

ISZ(Increment and Skip if zero)는 프로그램 루프의 반복 실행을 위한 명령이다. 지정된 번지의 내용을 1 증가시킨 후 그 결과가 0이면 다음 명령을 skip하고, 0이 아니면 다음 명령을 수행한다.

57.

BSA(Branch and Save Return Address)

서브프로그램으로 분기하고 복귀주소를 저장하는 명령이다.

58.

서브루틴의 실행이 모두 끝나고 메인으로 복귀하는 RET(return) 명령의 마이크로 연산이다.

59.

$2l_{(16)} = 0010\ 0001$ 이므로 상위 4비트는 opcode = 0010(SUB)이다. 하위 4비트 0001에서 00은 oper1(AX), oper2 = 01(BX)이다. 따라서 SUB AX, BX이다.

60.

$DX - CX = 7 - 4 = 3$

61.

BSA(Branch and Save Return Address)는 서브프로그램으로 분기하고 복귀주소를 저장하는 명령이며, 실행 시간이 가장 오래 걸린다.

62.

메모리에서 두 개의 데이터를 가져오는 데 2사이클, 연산하는 데 1사이클, 연산 결과를 메모리에 저장하는 데 1사이클이 소요되므로 전체 4사이클이 소요된다. 따라서 CPU 작업시간은 $16/10\text{MHz} = 1.6\mu\text{s}$ 이다.

63.

1클록 = 1머신 스테이트이다. 주어진 명령어가 2개의 머신 사이클이 필요하고, 각 머신 사이클은 5개의 머신 스테이트로 구성되므로 한 개의 명령은 총 10개의 머신 스테이트(총 10개 클록)가 필요하다. 1클록 당 $0.1\mu\text{s}(=1/10\text{M})$ 이므로 주어진 명령어가 수행되기 위해서는 $1\mu\text{s}$ 가 필요하다.

64.

1클록 = 1머신 스테이트이다. 주어진 명령어가 3개의 머신 사이클이 필요하고, 각 머신 사이클은 4개의 머신 스테이트로 구성되므로 한 개의 명령은 총 $10(=6+4)$ 개의 머신 스테이트(총 10개 클록)가 필요하다. 1클록 당 $0.4\mu\text{s}(=1/2.5\text{M})$ 이므로 주어진 명령어가 수행되기 위해서는 $0.4\mu\text{s} \times 10 = 4\mu\text{s}$ 가 필요하다.

65.

인스트럭션의 성능 = 수행시간 / (페치시간 + 준비시간)
 $= 10/(5+3) = 10/8 = 1.25$

66.

속도 향상지수 = 향상된 속도 / 이전 속도
 $= 10\text{초} / 5\text{초} = 2$

67.

명령 중에서 20%는 속도가 반으로 줄었고, 80%는 그대로 이므로 $0.5 \times 0.2 + 1 \times 0.8 = 0.9$ 다. 따라서 $1-1/0.9 = 0.1111$ 이므로 약 11.11%가 향상되었다.

68.

마이크로프로그램 제어기가 다음에 수행할 마이크로 인스트럭션의 주소를 결정하는데 사용하는 정보

- 인스트럭션 레지스터(IR)
- CPU의 상태 레지스터
- 마이크로 인스트럭션에 나타난 주소

69.

하나의 일을 여러 단계로 나누어 중첩되게 실행함으로써 성능을 높이는 방법은 파이프라이닝이다.

70.

파이프라인 프로세서는 2개 이상의 명령어를 동시에

수행할 수 있는 프로세서를 말한다.

71.

명령어 파이프라인(Instruction Pipelining)은 CPU의 처리 속도를 빠르게 하기 위하여 2개 이상의 명령어를 동시에 수행하는 것이다.

72.

② CPU가 필요한 데이터를 액세스할 때 읽기와 쓰기 동작을 향상시키는 기법으로는 SDRAM이나 DDR 등이 있다.

73.

파이프라인(instruction pipeline)에서 사용하는 버퍼 구조는 FIFO다.

74.

파이프라인 제어방식은 병렬처리 방법 중 명령 실행을 아주 작은 여러 개의 서브 프로세스로 분할하여 처리하는 방법을 말한다.

75.

파이프라인이 정상적인 동작에서 벗어나는 경우는 다음과 같다.

- 자원 충돌
- 데이터 의존성
- 분기 곤란

76.

파이프라인이 다음과 같은 정상적인 동작에서 벗어나는 경우에 이론적 최대 속도 증가율을 내지 못한다.

- 자원 충돌
- 데이터 의존성
- 분기 곤란

77.

파이프라인이 정상적인 동작에서 벗어나는 경우

- 자원 충돌
- 데이터 의존성
- 분기 곤란

78.

4단계 명령어 파이프라인의 수행 순서

IF(instruction Fetch) → ID(instruction Decode) → OF(Operand Fetch) → EX(Execution)

79.

파이프라이닝은 명령어 실행 단계를 여러 단계로 나누어 동시에 처리함으로써 성능을 향상시키는 기법이다. 5단계 파이프라인의 경우, 각 단계가 동시에 실행

되므로, 5개의 명령어가 동시에 실행되는 것과 같은 효과를 얻을 수 있다. 따라서, 이론적으로는 5배의 성능 향상이 가능하지만, 실제로는 파이프라인 해저드(Pipeline Hazard)와 같은 요인으로 인해 성능 향상이 제한될 수 있다.

80.

첫 번째 명령어 실행에는 k cycle이 걸리고, 나머지 $(N-1)$ 개 명령어들은 1cycle마다 한 개씩 실행이 종료된다. 따라서 N 개의 명령어들을 실행하는데 걸리는 시간은 $T = k + (N-1)$ 사이클이다.

81.

첫 번째 명령어의 실행에는 6cycle이 걸리고, 그 다음 명령어들은 1cycle마다 한 개씩 실행이 종료된다. 결과적으로 10개의 task를 모두 실행하는 데는 $6+9 = 15$ cycle이 필요하다.

82.

동기 고정식으로 클록 사이클 타임이 부여된다고 가정하면 마이크로 사이클 타임이 가장 긴 100ns에 지연시간 10ns를 더한 110ns로 결정되어야 한다.

83.

클록 사이클은 제일 큰 t_3 에다가 지연시간 10을 더한 110으로 결정되어야 한다. 이와 동등한 환경에서의 비파이프라인은 모두 더한($t_1 \sim t_4 + t_r$) 320이 되고 $320/110 = 2.9$ 이므로 2.9배 향상된 것이다.

84.

파이프라인 클록의 주파수가 1MHz이므로 클록 주기는 $1\mu s$ 이다. 따라서 첫 번째 명령어의 실행에는 $4\mu s$ 가 걸리고, 그 다음 명령어들은 $1\mu s$ 마다 한 개씩 실행이 종료된다. 결과적으로 10개 명령어들을 모두 실행하는 데는 $(4+(10-1)) \times 1\mu s = 13\mu s$ 가 걸린다.

85.

파이프라인 클록의 주파수가 1GHz이므로 클록 주기는 1ns이다. 따라서 첫 번째 명령어의 실행에는 4ns가 걸리고, 그다음 명령어들은 1ns마다 한 개씩 실행이 종료된다. 결과적으로 20개의 명령어들을 모두 실행하는 데는 $(4+(20-1)) \times 1ns = 23ns$ 가 걸린다.

86.

파이프라이닝에서 첫 번째 명령어 실행에는 k cycle이 걸리고, 나머지 $(N-1)$ 개 명령어들은 1cycle마다 한 개씩 실행이 종료된다. 따라서 N 개의 명령어를 실행하는데 필요한 시간은 $T = k + (N-1)$ 사이클이다.

비파이프라이닝인 경우는 N 개의 명령어를 실행하는데 $k \times N$ cycle이 필요하다. 따라서 파이프라이닝을 허용함으로써 얻어지는 속도 향상(S_p)은 다음과 같다.

$$S_p = \frac{k \times N}{k + (N-1)}$$

제06장. 기억장치

1.

기억장치의 접근 속도는 다르며, 일반적으로 빠른 순서로 나열하면 레지스터 - 캐시 메모리 - 주기억장치 - 보조기억장치 순이다.

2.

기억장치의 특성을 결정하는 요소는 기억 용량, 접근 시간, 사이클 시간, 대역폭, 데이터 전송률, 가격 등이 있다. Idle mode는 기억장치의 특성을 결정하는 요소가 아니다.

3.

액세스 시간(access time)은 주소나 제어(읽기/쓰기) 신호가 기억장치에 도착한 순간부터 데이터가 저장되거나 읽히는 순간까지의 시간을 말한다.

4.

주기억장치에서 접근 시간(access time)은 판독 신호 발생 후 자료를 메모리 버퍼 레지스터(MBR)에 옮기는 데까지의 시간이다.

5.

사이클 시간(cycle time)은 액세스 시간과 다음 액세스를 시작하기 위해, 필요한 동작에 걸리는 시간을 더한 것이다.

6.

파괴적 기억장치(destructive memory)는 읽기 동작 후에 데이터가 지워지며, 자동으로 복구되는 데 시간이 소요되어 (사이클 시간) = (액세스 시간) + (복원 시간) 관계가 성립한다. 반면 반도체나 자기디스크 같은 비파괴적 기억장치는 데이터가 파괴되지 않으므로 $M_i \geq A_i$ 관계가 성립한다.

7.

주기억장치의 대역폭(Bandwidth)은 기억장치의 자료 처리 속도를 나타내는 단위이며, 기억장치를 연속적으로 액세스할 때 초당 처리할 수 있는 비트 수이다.

8.

주기억장치의 대역폭(Bandwidth)은 기억장치의 자료 처리 속도를 나타내는 단위이며, 기억장치를 연속적으로 액세스할 때 초당 처리할 수 있는 비트 수이다.

9.

메모리의 word 개수를 늘리면 단지 메모리의 용량이

증가하는 것이다. 메모리의 대역폭 증가와는 관련이 없다.

10.

대역폭 = 데이터 버스의 길이 × 버스의 클록 주파수
 $= (32/0.5\mu s) = 64\text{Mbit/s}$

11.

대역폭 = 데이터 버스의 길이 × 버스의 클록 주파수
 $= 32\text{bit} \div 8 \times 1000\text{MHz} = 4,000\text{Mbytes/s}$

12.

대역폭 = 데이터 버스의 폭 × 버스의 클록 주파수
 $= (8/80\text{ns}) \text{ byte/s} = 108 \text{ byte/s} = 100\text{Mbyte/s}$

13.

파괴적 기억장치(destructive memory)에는 자기코어(magnetic core)나 FRAM(Ferroelectric RAM) 등이 있다.

14.

④ 보조기억장치(외부 메모리) → 디스크 캐시 → 주기억장치 → 메모리 캐시 → CPU 내부 캐시

15.

레지스터 - 캐시기억장치 - 주기억장치 - 디스크 캐시 - 보조기억장치

16.

레지스터 > 캐시 > RAM > ROM > 자기코어 > 자기디스크 > 자기테이프

17.

② CPU에 의한 액세스 빈도가 높아진다.
 ①, ③, ④는 하위 계층의 기억장치가 갖는 특징이다.

18.

④ 레지스터 → 캐시 → RAM → HDD

19.

주기억장치에는 반도체 IC 메모리가 널리 사용된다.

20.

SRAM은 컴퓨터의 주기억장치로 사용하며, 휘발성이 있어 전원이 차단되면 기억 내용이 지워지는 특성이 있다.

21.

메모리가 제대로 동작하려면 어드레스 신호, 데이터 신호 및 제어 신호가 상호 간 시간적 관계가 잘 유지되어야 한다.

22.

주기억장치에서 기억장소의 지정은 주소(Address)에 의해 이루어진다.

23.

1Mbyte = $2^{20} \times 8 \Rightarrow$ 주소선 수 = 20개

24.

256MB = 256×2^{20} Byte = $2^8 \times 2^{20}$ Byte = 2^{28} Byte
따라서 주소 버스는 최소한 28bit이어야 한다.

25.

1GB = $2^{30} \times 8$ 이므로 MAR의 길이는 30bit이다.

26.

컴퓨터의 메인 메모리는 워드 1개가 17비트이고, 256M 워드로 구성되므로 어드레스 비트 수는 28개다.
 $256M = 2^8 \times 2^{20} = 2^{28}$

27.

$$\begin{aligned} N &= \log_2 \left(\frac{M}{N} \right) = \log_2 \left(\frac{64K}{4} \right) \\ &= \log_2 \left(\frac{2^6 \times 2^{10}}{2^2} \right) = \log_2 (2^{14}) = 14 \end{aligned}$$

28.

$$4096 \times 16 = 2^{12} \times 16 \Rightarrow \text{MBR} = 16\text{비트}$$

29.

$$1024 \times 16 = 2^{10} \times 16 \Rightarrow \text{PC} = 10, \text{MAR} = 10, \text{MBR} = 16$$

30.

$$2^{15} \times 25 \Rightarrow \text{BR} = 25, \text{MAR} = 15, \text{PC} = 15$$

31.

$$2^{20} \times 32 \Rightarrow \text{PC} = 20, \text{MAR} = 20, \text{MBR} = 32$$

32.

$$\begin{aligned} 16K \times 32 &= 2^4 \times 2^{10} \times 32 = 2^{14} \times 32 \\ \Rightarrow \text{MAR} &= 14, \text{MBR} = 32 \end{aligned}$$

33.

$$2^{20} \times 8 = 1\text{Mbyte}$$

34.

$$2^{12} \times 8 = 2^2 \times 2^{10} \times 8 = 4K \times 8$$

35.

$$64K (= 2^{16})$$

36.

$$2^{17} \times 8 = 2^7 \times 2^{10} \times 8 = 128K \times 8 = 128K\text{byte}$$

37.

$$\frac{2 \times 1024 \times 8}{256 \times 8} = \frac{2^1 \times 2^{10} \times 2^3}{2^8 \times 2^3} = \frac{2^{14}}{2^{11}} = 2^3 = 8\text{개}$$

38.

$$\frac{64K \times 16}{8192 \times 8} = \frac{2^6 \times 2^{10} \times 2^4}{2^{13} \times 2^3} = \frac{2^{20}}{2^{16}} = 2^4 = 16\text{개}$$

39.

$$\frac{2^{32}}{512 \times 2^{20} \div 8} = \frac{2^{32}}{2^9 \times 2^{20} \div 2^3} = \frac{2^{32}}{2^{26}} = 2^6 = 64\text{개}$$

40.

$$\frac{64 \times 8}{64 \times 1} = 8\text{개}$$

41.

$$\frac{2^{20} \times (8+1)}{2^{16} \times 1} = 2^4 \times 9 = 144\text{개}$$

42.

- ① 1K $\times$ 4bit : MAR = 10, MBR = 4
- ② 8K $\times$ 4bit : MAR = 13, MBR = 4
- ③ 4K $\times$ 1bit : MAR = 12, MBR = 1
- ④ 4K $\times$ 8bit : MAR = 16, MBR = 8

43.

핀 배치도에서 주소선은 10개(A0~A9), 데이터 선은 8개(D1~D8)다. 따라서 메모리 용량은 $2^{10} \times 8 = 1024 \times 8$ 이다.

44.

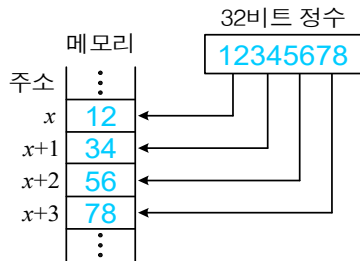
$$2^{(16-5)} = 2^{11}$$

45.

10가지 명령어를 구분하기 위해 4비트가 필요하므로
 $2^{(16-4)} = 2^{12} = 2^2 \times 2^{10} = 4K$ 워드

46.

빅 엔디언(big endian)은 상위 바이트를 낮은 주소에 저장하는 방법이다.

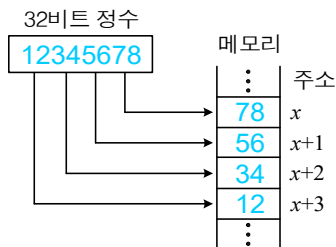


47.

빅 엔디언(big endian)은 상위 바이트를 낮은 주소에 저장하는 방법이다.

48.

리틀 엔디언(little endian)은 하위 바이트를 낮은 주소에 저장하는 방법이다.



49.

리틀 엔디언(little endian)은 리눅스(Linux), 인텔 계열의 CPU, AMD 계열의 CPU에서 사용하는 방식이다.

50.

마이크로컴퓨터에서 값이 고정되어 변하지 않는 시스템 프로그램은 ROM에 저장한다.

51.

마이크로프로그램은 지워지면 안 되므로 ROM이 사용된다.

52.

ROM은 문자 패턴을 저장하고 있으며, 변경되지 않는 데이터다.

53.

ROM에는 지워지면 안 되는 데이터를 저장한다. 소스 프로그램은 주기억장치에 저장한다.

54.

RAM은 사용자가 작성한 프로그램이나 데이터를 기억

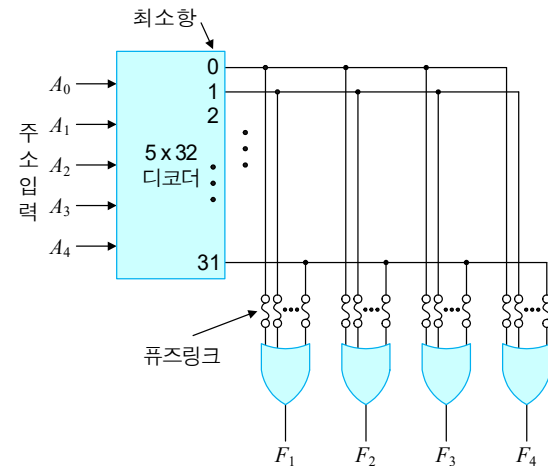
시켜 처리하기 위해 사용하는 메모리이다.

55.

ROM은 읽기만 가능한 메모리이므로 쓰기(write) 신호는 없다.

56.

ROM은 디코더와 OR 게이트로 구성된다.



57.

ROM은 디코더와 OR 게이트로 구성된다. 16×8 ROM을 구성하기 위한 디코더는 4×16 구조를 갖는다. 따라서 디코더의 출력은 AND 게이트이므로 총 16개가 필요하다. 그리고 16×8 ROM에서 8은 OR 게이트의 개수를 의미한다.

58.

ROM에는 지워지면 안 되는 데이터를 저장한다. 운영 체제는 하드 디스크(HDD)나 SSD에 저장한다.

59.

ROM에는 지워지면 안 되는 데이터를 저장한다. 소스 프로그램은 주기억장치에 저장한다.

60.

컴퓨터 시스템의 바이오스(BIOS, Basic Input/Output System)는 지워지면 안 되므로 ROM을 사용한다.

61.

Mask ROM은 제조 과정에서 데이터를 영구적으로 저장한다. 동일한 형태가 대량으로 필요할 때는 Mask ROM이 경제적이다.

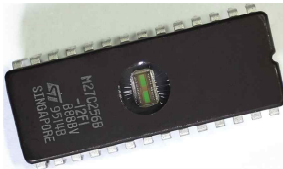
62.

PROM(Programmable ROM)은 쓰기를 한 번만 할 수 있다. 퓨즈가 손상되지 않은 상태로 출시되므로 이후

에 원하는 형태로 절단하면 된다.

63.

EPROM(Erasable PROM)은 ROM 라이터를 사용해 데이터를 써넣을 수 있고, ROM 소거기(ROM eraser)를 사용해 칩 위의 석영 유리창을 통해 자외선을 10~20분 정도 쬔어 지울 수도 있다. EPROM은 데이터를 여러 번 쓰고 지울 수 있으며, UVEEPROM(Ultra Violet EPROM)이라고도 한다.



64.

EEPROM(Electrically EPROM)은 전기적으로 기록과 삭제가 가능하다.

65.

EPROM은 자외선을 이용하여 데이터를 지우지만 EEPROM, NOR 플래시, NAND 플래시는 전기적으로 지운다.

66.

② 비휘발성(Non-volatile) 메모리는 정전이 되어도 저장된 내용이 유지된다.

67.

① 펌웨어

68.

펌웨어(firmware)에는 변경되지 않는 데이터가 저장되므로 ROM(Mask ROM, PROM, EPROM)이 적합하다. 반면에 SRAM은 전원이 제거되면 데이터가 지워진다.

69.

임베디드 보드의 ROM에 저장되어 하드웨어를 제어하기 위해 작성된 프로그램을 펌웨어(firmware)라고 한다.

70.

휘발성 메모리는 DRAM이며, FRAM, PROM, EPROM은 비휘발성 메모리다.

71.

DRAM은 Read와 Write가 가능한 주기억장치 소자로 기억 상태를 유지하기 위해 주기적으로 재생(Refresh)해야 한다.

72.

RAM은 기억된 정보를 읽어내기도 하고 다른 정보를 기억시킬 수도 있으며, 응용프로그램의 일시적 로딩, 데이터의 일시적 저장 등에 사용된다.

73.

RAM은 전원이 공급되는 동안 데이터를 기억하지만, 전원이 제거되면 데이터가 지워지는 휘발성(volatile) 메모리다.

74.

읽기/쓰기 선택은 칩(chip) 외부 신호다.

75.

SRAM은 내부 플립플롭(flip-flop)에 데이터를 기억시킨다.

76.

캐시 메모리는 고속으로 동작하는 SRAM을 사용한다.

77.

② SRAM은 DRAM에 비해서 집적도가 떨어지며, 소비 전력이 높다.

78.

DRAM은 주기적으로 refresh를 시켜주어야 하지만, SRAM은 플립플롭을 기본 소자로 하므로 refresh가 필요 없다.

79.

SRAM은 DRAM에 비해 액세스 시간이 빠르다.

80.

DRAM은 미소의 콘텐서에 전하를 충전하는 형태의 원리를 이용하는 메모리로, 재충전(refresh)이 필요하다.

81.

DRAM은 미소의 콘텐츠에 전하를 충전하는 형태의 원리를 이용하는 메모리이므로 재충전(refresh)이 필요하다.

82.

SRAM은 플립플롭의 원리를 이용한다.

83.

SRAM은 캐시 메모리에 주로 사용되며, DRAM은 주기억장치에 사용된다.

84.

DRAM은 SRAM에 비해 액세스 속도가 느리다.

85.

첫 번째 행 부터 마지막 행까지 refresh하는데 2ms가 소요되므로 행과 행사이의 시간은 $125\mu s$ 가 된다.

$$\frac{2ms}{16} = 0.125ms = 125\mu s$$

86.

- ② DRAM은 SRAM에 비해 속도가 느리다.
- ③ SRAM의 소비전력이 DRAM보다 높다.
- ④ SRAM의 memory cell은 flip-flop으로 구성되어 있다.

87.

DRAM은 커패시터를 사용하므로 재충전이 필요하고, SRAM은 플립플롭을 사용하므로 재충전이 필요 없다.

88.

- ② PROM은 한 번 쓰면 지울 수 없지만, EPROM은 데이터를 여러 번 쓰고 지울 수 있다.

89.

- address 라인 : $(1024=2^{10}) \rightarrow 10$ 개
 - data 라인 : 8개
 - chip select bit : 1개
- 따라서 총 $19(=10+8+1)$ 개 핀이 필요하다.

90.

- RAM 1:
 $A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0 = 01\ 0000\ 0000 \sim 01\ 1111\ 1111$
 $(= 100 \sim 1FF)$
- RAM 2:
 $A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0 = 10\ 0000\ 0000 \sim 10\ 1111\ 1111$
 $(= 200 \sim 2FF)$

91.

11 0000 0000 ~ 11 1111 1111 이므로 300H~3FFH다.

92.

1100 0000 ~ 1111 1111이므로 C0H ~ FFH다.

93.

중앙처리장치(CPU)와 주기억장치 사이에 고속이지만 소용량의 캐시(cache)를 두고 있다

94.

중앙처리장치(CPU)의 속도와 주기억장치의 속도 차이

가 현저하므로 중앙처리장치와 주기억장치 사이에 캐시를 두어 속도 차를 극복한다.

95.

- ① 내용에 의해서 액세스되는 메모리는 연관기억장치다.
- ② 소형 및 대형 컴퓨터 시스템에서 사용되는 개념이다.
- ④ 메모리에 접근을 각 모듈별로 액세스하도록 하는 기억장치를 복수모듈기억장치라고 한다.

96.

캐시(cache)는 중앙처리장치의 속도와 주기억장치의 속도를 가능한 같도록 하기 위한 장치이다.

97.

캐시(cache)는 주기억장치에 비해서 속도가 빠르고, 가격이 비싸다.

98.

캐시에 히트되는 비율을 나타내는 히트율(hit ratio, H)는 다음과 같이 정의된다.

$$H = \frac{\text{캐시에 히트되는 횟수}}{\text{전체 기억장치 액세스 횟수}}$$

99.

- 히트율 = 히트 횟수 / 총 접근 횟수 = $45 / 50 = 0.9$
- 미스율 = $1 - \text{히트율} = 1 - 0.9 = 0.1$

100.

$$0.9 \times 100ns + (1 - 0.9) \times 1000ns = 90ns + 100ns = 190ns$$

101.

캐시 메모리의 접근 시간을 T_C 라고 하면 다음식이 성립하므로 캐시 메모리의 접근 시간은 10ns이다.

$$0.9 T_C + 0.1 (T_C + 200) = 30$$

$$\therefore T_C = 10ns$$

102.

$$H \times 100ns + (1 - H) \times 700ns = 170ns$$

$$\leftrightarrow H = \frac{700 - 170}{600} = 0.883$$

103.

L1만 사용할 때와 L1, L2를 사용할 때의 액세스 시간을 계산하여 비교하면 된다.

- L1만 사용할 때의 액세스 시간은 찾으려는 데이터가 L1 캐시에 없을 경우 주기억장치에서 찾으므로 액세스 시간은
 메모리 액세스 시간 = L1 히트시간 + L1 미스율 $\times$ L1 미스패널티

$$=1+0.05 \times 100=6\text{사이클이다.}$$

- L1, L2 캐시를 사용할 때의 액세스 시간

L1 캐시에 데이터가 없을 경우, L2 캐시를 액세스하는데 걸리는 시간이 L1 미스페널티이고, L2 캐시에 데이터가 없을 경우 주기억장치를 액세스하는데 걸리는 시간이 L2 미스페널티이다. 즉, L2 미스페널티를 이용하여 L1 미스페널티를 구한 후 전체에 대한 액세스 시간을 계산하면 된다.

- L1 미스페널티 = L2 히트시간 + L2 미스율 $\times$ L2 미스페널티
 $= 4+0.2 \times 100 = 24\text{사이클}$
- 메모리 액세스 시간 = L1 히트시간 + L1 미스율 $\times$ L1 미스페널티
 $= 1+0.05 \times 24 = 2.2\text{사이클}$
 $\therefore 6/2.2 = 2.73$, 약 2.7배 액세스 시간이 향상된다.

104.

접근의 국부성 : 프로그램이 수행될 때 최근에 사용한 명령과 데이터를 다시 사용할 가능성이 크다는 것

105.

매핑 함수 : 주기억장치의 한 개 블록을 캐시 라인에 매핑하는 규칙

106.

캐시 메모리와 주기억장치 사이에는 블록 단위로 정보를 교환한다.

107.

캐시에 기억시키는 블록 주소의 일부를 태그 주소라고 한다.

108.

매핑 프로세스(mapping process)의 종류

- 직접 매핑(direct mapping)
- 연관 매핑(associative mapping)
- 세트-연관 매핑(set-associative mapping)

109.

직접 매핑(direct mapping)에서는 만약 같은 인덱스를 가졌으나 다른 태그를 가진 두 개 이상의 워드가 반복해서 접근된다면 적중률(히트율)이 상당히 낮아질 수 있다.

110.

직접 매핑(direct mapping)에서는 만약 같은 인덱스를 가졌으나 다른 태그를 가진 두 개 이상의 워드가 반복해서 접근된다면 적중률(히트율)이 상당히 낮아질 수 있다.

111.

기억장치에 기억된 정보를 액세스하기 위하여 주소를 사용하는 것이 아니라 기억된 정보의 일부분을 이용하여 원하는 정보를 찾는 것을 연관 기억장치(associative memory)라고 한다.

112.

캐시블록번호 = (메모리 블록번호) mod (캐시블록 수)
 메모리 블록번호 = $1200 / 16 = 75$
 $75 \text{ MOD } 64 = 11$

113.

- 주기억장치 크기가 64Mbyte이므로 주소지정에 $26(2^6 \times 2^{20})$ 비트가 필요하다. 바이트 단위로 주소가 지정되며, 워드 길이는 1바이트이다.
- 캐시 크기는 64Kbyte이므로 캐시의 주소지정에 $16(2^6 \times 2^{10})$ 비트가 필요하다.
- 블록 1개 크기가 4바이트(=4워드)이므로 주기억장치에는 $2^{24}(= 2^{26} / 2^2)$ 개의 블록이 있다.
- 블록 크기가 4바이트이므로 캐시 라인의 크기도 4바이트가 되며, 결과적으로 캐시 라인 수는 $2^{14}(= 2^{16} / 2^2)$ 개다.

완전 연관 사상에서는 블록 번호 전체를 태그로 사용하므로 캐시 1라인에 필요한 태그의 크기는 24비트가 된다.

태그	워드
24비트	2비트

114.

- 주기억장치 크기가 $32K \times 12$ 이므로 주소지정에 $15(2^5 \times 2^{10})$ 비트가 필요하다. 12비트 단위로 주소가 지정되며, 워드 길이는 12비트이다.
- 캐시 크기는 512×12 이므로 캐시의 주소지정에 $9(512=2^9)$ 비트가 필요하다.
- 블록 1개 크기가 8워드(1워드=12비트)이므로 주기억장치에는 $2^{12}(= 2^{15} / 2^3)$ 개의 블록이 있다.
- 블록 크기가 8워드이므로 캐시 라인의 크기도 8워드가 되며, 결과적으로 캐시 라인 수는 $2^6(= 2^9 / 2^3)$ 개다.

완전 연관 사상에서는 블록 번호 전체를 태그로 사용하므로 캐시 1라인에 필요한 태그의 크기는 12비트가 된다.

태그	워드
12비트	3비트

115.

가상기억장치는 사용자가 실제 기억장치보다 큰 기억장치를 사용할 수 있는 메모리 이용 기법을 말한다.

116.

캐시의 교체 알고리즘

- LRU(Least Recently Used)
- FIFO(First-In First-Out)
- LFU(Least Frequently Used)
- Random

117.

LRU(Least Recently Used)는 가장 효과적이라고 알려져 있으며, 사용되지 않은 채 가장 오래 캐시에 적체된 블록을 교체한다.

118.

LRU(Least Recently Used)는 가장 효과적인 알고리즘이라고 알려져 있다.

119.

write-through 방식은 쓰기 동작이 캐시와 주기억장치에서 동시에 발생한다.

120.

write-through 방식은 쓰기 동작이 캐시와 주기억장치에서 동시에 발생하기 때문에 쓰기 동작에 걸리는 시간이 길다는 단점이 있다.

121.

write-back 방식은 새로운 데이터가 캐시에서만 갱신되므로 주기억장치와 캐시의 데이터가 서로 일치하지 않는 경우가 발생할 수 있다.

122.

캐시메모리의 쓰기 정책

- write-through 방식 : 쓰기 동작이 캐시와 주기억장치에서 동시에 발생한다.
- write-back 방식 : 새로운 데이터가 캐시에서만 갱신된다.
- write-once 방식 : write-through와 write-back의 장점을 취한 방식으로, 처음 기록할 때는 write-through 방식이고, 이후부터는 write-back 방식을 사용한다.

123.

캐시 액세스 충돌 제거를 위해 분리 캐시(split cache)를 사용한다.

124.

캐시 설계 시 고려사항으로는 캐시 용량, 사상방식, 교체 알고리즘, 쓰기 정책, 라인 크기, 캐시 수 등이다. 따라서 하드 디스크 용량은 캐시 설계 시 고려 대상이 아니다.

125.

캐시와 주기억장치에 저장된 데이터가 항상 일치하도록 관리하면서, 이 과정에서 발생하는 추가적인 작업(예: 데이터 동기화, 무효화 등)을 최소화해야 한다.

126.

프로그래머에 의하여 보여지는 주소를 가상 주소라고 할 때, 이들 주소의 집합을 주소 공간이라고 한다.

127.

가상기억장치는 사용자가 실제 기억장치보다 큰 기억장치를 사용할 수 있는 메모리 이용 기법이다.

128.

설치된 물리적인 메모리보다 더 큰 메모리를 요구하는 프로그램을 로드(load)할 수 있도록 보조기억장치인 HDD 또는 SSD에서 해당하는 용량만큼 확장하여 사용하는 것을 가상메모리라고 한다.

129.

가상메모리(virtual memory) : 보조기억장치에 저장된 프로그램과 데이터 중에서 프로그램 수행에 필요한 부분을 주기억장치로 옮길 때 부족한 주기억장치의 용량을 확장하기 위해 보조기억장치의 일부를 마치 주기억장치의 일부로 사용하는 것을 말한다.

130.

가상기억장치는 사용자가 실제 기억장치보다 큰 기억장치를 사용할 수 있는 메모리 이용 기법이며, 속도를 향상시키기 위한 방법은 아니다.

131.

④는 캐시 메모리에 대한 설명이다.

132.

가상기억장치(Virtual Memory)는 보조기억장치의 일부 용량을 주기억장치처럼 가상하여 사용할 수 있도록 하는 기법으로, 처리 속도가 CPU 속도에 비해 매우 느리다.

133.

가상기억장치는 사용자가 실제 기억장치보다 큰 기억장치를 사용할 수 있는 메모리 이용 기법이다.

134.

가상기억장치를 도입하면 실질적인 기억 공간보다 훨씬 큰 논리적 공간을 주소화할 수 있지만, 운영체제의 설계가 복잡해지고, 가상기억장치를 채택하지 않는 시스템에서의 실행 속도보다 느리다는 단점이 있다.

135.

가상기억장치(Virtual Memory)는 프로그램의 크기가 주기억장치의 용량보다 클 때 사용한다.

136.

매핑(Mapping)은 가상기억장치에서 주기억장치로 페이지를 옮겨 넣을 때 주소를 조정하는 것을 의미한다.

137.

사상 함수(매핑 함수) : 인스트럭션이 수행될 때 주기억장치에 접근하려면 인스트럭션에서 사용한 주소는 주기억장치에 직접 적용될 수 있는 기억장소의 주소로 변환되어야 한다. 이때 주소로부터 기억장소의 변환에 사용되는 것을 말한다.

138.

페이징 기법은 가상기억장치에 보관된 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 다음 나뉜진 프로그램을 동일하게 나뉜진 주기억장치의 영역에 적재시켜 실행하는 기법이다.

139.

페이징 기법은 가상기억장치에 보관된 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 다음 나뉜진 프로그램을 동일하게 나뉜진 주기억장치의 영역에 적재시켜 실행하는 기법이다.

140.

페이지 크기가 커질수록 페이지 테이블 크기가 작아져서 주기억장치 공간을 절약할 수 있지만, 참조되지 않는 정보까지 함께 로드될 수 있어서 기억 공간 낭비가 발생할 수 있다. 또한, 페이지 크기가 클수록 디스크 접근 시간이 늘어날 수 있다.

141.

세그먼트(segmentation) 기법은 가상기억장치에서 블록의 크기가 가변인 방식을 말한다.

142.

FIFO(First-In First Out)는 가장 간단한 알고리즘으로 주기억장치에 가장 오래 있었던 페이지를 교체한다. 구현하기가 쉽다는 장점이 있으나, 어떤 상황에서는 페이지가 너무 자주 교체되는 단점이 있다.

143.

LFU(Least Frequently Used) 알고리즘은 페이지를 교체해야 할 때 페이지 중에서 사용 빈도가 가장 낮은 페이지를 선택해서 제거하는 방법이다.

144.

페이지 부재(page fault)는 CPU가 액세스한 가상 페이지가 주기억장치에 없는 경우를 말한다. 페이지 부재가 발생하면 해당 페이지를 디스크에서 주기억장치로 가져와야 한다.

145.

$$\frac{\text{DAFFH} - \text{B000H}}{100\text{H}} = 2\text{BH} = 43_{(10)}$$

146.

기억 공간의 크기가 $64\text{K}(=2^{16})$ 이므로 주소 레지스터는 16비트다.

147.

- 페이지 : $2^{16} / 2^9 = 2^7 = 128$
- 블록 : $2^{12} / 2^9 = 2^3 = 8$

148.

- 실제 페이지 주소 = $2^{19} / 2^{12} = 2^7 \rightarrow 7$
- 가상 페이지 주소 = $2^{32} / 2^{12} = 2^{20} \rightarrow 20$

149.

$$2^4 \times 2^8 \times 2^8 = 2^{20}$$

150.

- 페이지 개수 : $2^{32} / 2^{12} = 2^{20}$
- 페이지 당 4byte이므로 $4 \times 2^{20} \text{ byte} = 4\text{Mbyte}$

151.

4비트	8비트	8비트
세그먼트 번호	페이지 번호	워드필드

$$\begin{aligned} \text{세그먼트의 최대 크기} &= \text{페이지 번호} + \text{워드 필드} \\ &= 8\text{비트} + 8\text{비트} = 16\text{비트} \end{aligned}$$

$$\text{따라서 } 2^{16} \text{ word} = 2^6 \times 2^{10} \text{ word} = 64\text{K word}$$

152.

주기억장치 관리 기법 중 연속할당 기법은 고정 분할 기법과 가변분할 기법이 있다.

153.

연관기억장치(Associative Memory)는 기억장치에 기억된 정보를 액세스하기 위하여 주소를 사용하는 것이 아니라 기억된 정보 일부분을 이용하여 원하는 정보를 찾는 것을 말한다. CAM(content addressable memory)이라고도 한다.

154.

연관기억장치(Associative Memory)는 주소를 사용하는 것이 아니라 기억된 주소 일부분을 이용하여 원하는 정보를 찾는 기법이다. 저장 공간의 확대가 목적인 기법은 가상기억장치다.

155.

④는 캐시 메모리(Cache Memory)에 대한 설명이다.

156.

연관기억장치(Associative Memory)는 기억장치에 기억된 정보를 액세스하기 위하여 주소를 사용하는 것이 아니라 기억된 정보의 일부분을 이용하여 원하는 정보를 찾는 기법이다. CAM(content addressable memory)이라고도 한다.

157.

연관기억장치는 기억장치에서 데이터를 찾을 때 주소에 의한 접근이 아닌, 내용의 일부를 이용해 접근하는 방식이며, 병렬 검색이므로 빠르지만 하드웨어 비용이 증가한다.

158.

연관기억장치(Associative Memory)는 별도의 판독회로 구성이 필요하므로 하드웨어 비용이 증가한다.

159.

Thrashing : 가상기억장치 시스템에서 프로그램이 접근한 페이지나 세그먼트를 디스크에서 주기억장치로 올려놓기 위한 페이지 폴트가 너무 자주 일어나 프로그램의 처리 속도가 급격히 떨어지는 상태를 말한다.

160.

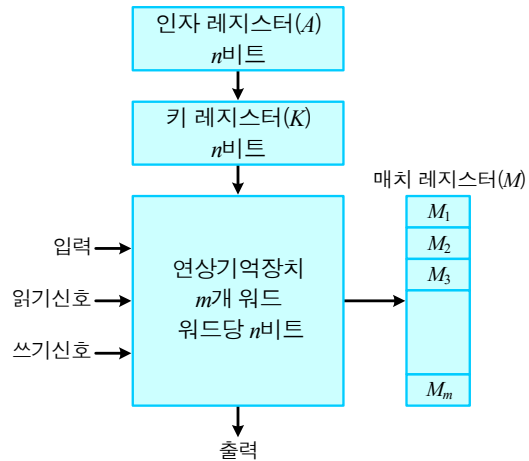
연관 기억장치(Associative Memory)는 기억장치에서 자료를 찾을 때 주소에 의해 접근하지 않고, 기억된 정보 일부분을 이용하여 원하는 정보가 기억된 위치를 알아낸 후 그 위치에서 나머지 정보에 접근하는 기억장치로, CAM(Content Addressable Memory)라고도 한다.

161.

연관 기억장치(Associative Memory)는 기억장치에서 자료를 찾을 때 주소에 의해 접근하지 않고, 기억된 정보 일부분을 이용하여 원하는 정보가 기억된 위치를 알아낸 후 그 위치에서 나머지 정보에 접근하는 기억장치로, CAM(Content Addressable Memory)라고도 한다.

162.

연관기억장치 구조



163.

② 연관기억장치(associative memory) 또는 CAM은 DRAM보다 가격이 비싸다.

164.

② 액세스가 진행되는 동안 CPU는 결과를 기다리지 않고 내부적으로 다른 연산을 수행한다.

165.

인터리빙(interleaving)은 모듈화된 기억장치의 주소를 한 개의 기억장치에만 집중시키지 않고, 여러 기억장치의 모듈에 분산시켜서 처리 능력을 높이는 방법이다.

166.

인터리빙(interleaving)은 중앙처리장치(CPU)가 기억 모듈에 중복적인 데이터 접근을 방지하기 위해서 연속된 데이터 또는 명령어들을 기억장치모듈에 순차적으로 번갈아 가면서 처리하는 방식이다.

167.

메모리 인터리빙(memory interleaving)은 중앙처리장치(CPU)와 기억장치 사이에 실질적인 대역폭(bandwidth)을 늘리기 위한 방법이다.

168.

디스크 인터리빙(disk interleaving)은 데이터를 디스크에 분산 저장하는 기술로 데이터가 다수의 블록들로 이루어져 있을 때 블록들을 라운드 로빈(round-robin) 방식으로 디스크에 균등하게 분산 저장한다.

169.

디스크 인터리빙(disk interleaving)은 데이터를 디스크에 분산 저장하는 기술을 말하며, Low-order와 High-order 방식이 있다.

170.

메모리 액세스의 효율 증대시키기 위해 메모리 인터리빙을 사용한다.

171.

기억장치의 액세스 속도를 향상시키기 위해 캐시 메모리, 메모리 뱅킹, 메모리 인터리빙 등의 방법이 사용된다. 가상메모리는 사용자가 실제 기억장치보다 큰 기억장치를 사용할 수 있는 메모리 이용 기법이다.

172.

메모리 인터리빙은 명령들의 Memory Access 충돌을 막기 위해서 사용한다.

173.

인터리빙(interleaving)은 모듈화된 기억장치의 주소를 한 개의 기억장치에만 집중시키지 않고, 여러 기억장치의 모듈에 분산시켜서 처리 능력을 높이는 방법이다.

174.

④는 가상기억장치에 대한 설명이다.

175.

④ 인터리빙 기법을 이용하여 모듈의 수에 해당하는 데이터 워드 수를 동시에 읽어 패치한다.

176.

③은 연관기억장치(associative memory)의 특징이다.

177.

복수 모듈 기억장치 처리 시 주소가 완전히 인터리브될 때 처리 속도가 증가한다.

178.

복수 모듈 기억장치는 데이터를 전달할 때 시분할(Time sharing) 개념을 사용한다.

179.

메모리 인터리빙은 메모리 공간 확대의 목적이 아니라 접근 속도의 향상이 목적이다.

제07장. 보조기억장치

1.

자기디스크(magnetic disk), 자기테이프(magnetic tape), DVD 등은 보조기억장치로 사용되며, SDRAM은 주기억장치로 사용된다.

2.

RAM이나 ROM은 주기억장치로 사용되며, 자기디스크는 보조기억장치로 사용된다.

3.

자기디스크(magnetic disk), 자기 드럼(magnetic drum), USB 메모리 등은 임의 접근 방식이며, 자기테이프(magnetic tape)는 순차 접근 방식이다.

4.

자기 드럼은 임의 접근(random access) 방식을 사용한다. 내용에 의한 접근 방식은 연관 기억장치(Associative Memory)에 사용된다.

5.

자기코어, 자기디스크, 자기드럼 방식은 임의 접근 방식이며, 자기테이프(magnetic tape)는 순차 접근 방식이다.

6.

CD-ROM은 보조기억장치로서 읽기만 가능한 기억장치다. 반면에 하드 디스크, 플로피디스크, USB 저장장치는 읽기/쓰기가 가능한 저장장치다.

7.

보조기억장치는 CPU에 의한 기억장치의 접근 빈도가 낮다.

8.

보조기억장치는 주기억장치보다 정보를 읽는 속도가 느리다.

9.

보조기억장치는 주기억장치보다 액세스 속도가 느리다.

10.

가상메모리로 사용할 수 있는 보조기억장치로 자기디스크(magnetic disk)가 적당하다.

11.

실린더는 디스크 판의 같은 위치에 놓여있는 트랙들의 집합을 의미한다. 따라서 실린더는 자기디스크의 구성 요소는 아니다.

12.

섹터(sector)는 자기디스크에서 읽기/쓰기 작업이 이루어지는 최소 단위다.

13.

액세스 시간 = 탐색시간 + 회전지연시간 + 데이터전송시간

14.

액세스 시간 = 탐색시간 + 회전지연시간 + 데이터전송시간

15.

탐색시간(seek time)은 헤드가 원하는 실린더를 찾을 때까지의 시간이다.

16.

탐색시간(seek time)은 헤드가 원하는 트랙(또는 실린더)를 찾을 때까지의 시간이다.

17.

① 탐색시간(seek time)

18.

회전속도가 7200rpm인 디스크는 초당 120회(=7200/60) 회전하므로 1회전하는 시간은 약 8.33ms(=1/120)이다.

19.

윈체스터 디스크(Winchester disk)는 오류 발생률을 낮추면서 저장 용량을 늘리기 위해 개발된 디스크 드라이브다. 오염 물질이 들어갈 수 없도록 헤드가 드라이브 패키지에 들어 있다.

20.

평균 액세스 시간 = 탐색시간 + 회전지연시간 + 데이터 전송 시간
= 20ms + 8.3ms + 0.5ms = 28.8ms

21.

- 탐색시간 : 12ms
- 회전지연시간 : 회전속도가 7200rpm이면 초당

$120(=7200/60)$ 회전하므로 1회전하는 시간은 약 $8.33\text{ms}(=1/120)$ 이다. 평균 회전지연시간은 $4.17\text{ms}(=8.33/2)$ 다.

- 데이터 전송 시간 : $512/(10 \times 106) = 0.0512\text{ms}$
- 제어기의 지연시간 : 1ms
- ∴ 평균 액세스 시간 = $12\text{ms} + 4.17\text{ms} + 0.0512\text{ms} + 1\text{ms}$
= 17.22ms

22.

디스크가 양면이라고 가정하면
 $2 \times 200 \times 4 \times 320 = 512,000$ 워드다.

23.

영문자 1개는 8비트(=1byte)이므로
 $1.28\text{Mbyte} / 1\text{byte} = 1.2 \times 2^{20} = 1,258,291$ 이다.

24.

디스크 저장장치 용량 = 디스크 개수 × 디스크 당 면 수 × 디스크 당 트랙 수 × 트랙 당 섹터 수 × 섹터 당 용량
 $= 2 \times 2 \times 8 \times 16 \times 512 \text{ byte} = 262,116\text{byte}$
 $= 262,116 / 1024 \text{ byte} = 256\text{Kbyte}$

25.

최대 데이터 전송 속도 = 초당 회전수 × 트랙당 섹터 수 × bit
 $60\text{rpm} = \text{분당 } 60\text{회전} = \text{초당 } 1\text{회전}$
 $512\text{byte} = 512 \times 8(\text{bit})$
 최대 데이터 전송 속도 = $1 \times 16 \times 512 \times 8$
 $= 65536\text{bps} = 65536 / 1024 = 64\text{Kbps}$

26.

레코드 수 = $8 \times 2 \times 16 \times 8 = 2^3 \times 2^1 \times 2^4 \times 2^3 = 2^{11}$
 따라서 주소지정에는 11비트가 필요하다.

27.

실린더는 자기디스크(magnetic disk) 표면에서 같은 위치에 놓여있는 트랙들의 집합을 의미한다.

28.

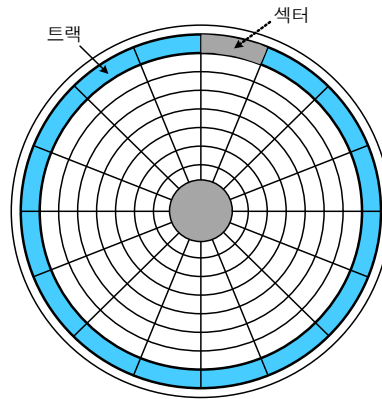
실린더는 자기디스크(magnetic disk) 표면의 같은 위치에 놓여있는 트랙들의 집합을 의미한다.

29.

(실린더의 개수 = 한 면의 트랙의 개수)이므로 실린더는 200개다.

30.

① 트랙(track)



31.

등각속도(CAV) 방식에서 트랙의 위치에 따라 데이터 저장밀도가 달라진다. 반면에 등선속도(CLV) 방식에서는 모든 트랙의 저장밀도가 같다.

32.

등각속도(CAV) 방식에서 트랙의 위치에 따라 데이터 저장밀도가 달라지므로 저장 공간이 낭비된다.

33.

③ 자기디스크 : CAV, 콤팩트디스크 : CLV

34.

자기디스크는 자기 드럼보다 액세스 시간이 느리다. 자기 드럼은 데이터를 저장하는 회전하는 실린더로, 헤드가 데이터 위치로 이동할 필요 없이 데이터를 빠르게 읽고 쓸 수 있다. 반면, 자기디스크는 데이터가 저장된 트랙과 섹터로 이동해야 하므로 액세스 시간이 더 오래 걸린다.

35.

디스크를 사용하려면 처음에 파티션을 만들고 포맷한다.

36.

모든 하드 디스크 제조사들은 IDEMA(International Disk Drive Equipment and Materials Association, 국제 디스크 드라이브 장비 및 재료 협회)를 통해 2011년부터 기존 512바이트 섹터 표준을 4K 섹터 표준으로 변경하기로 하였다.

37.

- ① sequential access가 가능하다.
- ② 입력 및 출력 장치로 사용된다.
- ③ 각 블록 사이에는 IBG(Inter Block Gap)라는 간격이 있다.

38.

① sequential access가 가능하다.

39.

자기테이프(magnetic tape)는 순차 접근 기억장치(Sequential Access Storage Device, SASD)다.

40.

자기테이프의 기록 밀도는 1인치당 저장될 수 있는 문자나 바이트 수인 BPI(Byte Per Inch)로 나타낸다.

 $600 \times 5 = 3,000\text{byte}$

41.

블록 하나에 들어가는 레코드 수를 blocking factor라고 한다. 따라서 $4 \times 3 \times (4 + 250) = 766\text{byte}$ 다.

42.

IRG가 많으면 기록 용량이 적어지므로 레코드 여러 개를 묶어 하나의 블록으로 만들면(blocking) 저장 효율이 증가하고 한 번에 여러 개의 레코드를 읽을 수 있어 전송 속도가 빨라진다.

43.

블록과 블록 사이의 간격을 IBG(Inter Block Gap)라고 한다.

44.

스테이징(staging)은 데이터가 테이프로 백업되는 과정에서 디스크를 캐시처럼 활용하는 기술이다.

45.

라이브러리 프로그램은 자기디스크에 저장한다.

46.

자기테이프(magnetic tape)는 순차 접근 기억장치(Sequential Access Storage Device, SASD)다.

47.

자기테이프는 순차 처리만 할 수 있는 대용량 저장 매체다.

48.

CD-ROM은 한 번 기록된 후 읽기만 가능하고, 쓰거나 재기록은 불가능하다. CD-ROM은 공장에서 제작될 때 정보가 기록되고, 사용자는 그 내용을 읽을 수만 있다.

49.

CD-ROM과 DVD에서 사용하는 레이저의 파장은 같지

않다. CD-ROM은 주로 780nm 파장의 적외선 레이저를 사용하고, DVD는 650nm 또는 405nm 파장의 적색 또는 청자색 레이저를 사용한다. DVD가 CD보다 파장이 짧기 때문에 더 많은 정보를 저장할 수 있다.

50.

SSD(Solid State Drive)는 하드 디스크 드라이브(HDD)와 비슷하게 동작하면서 기계적 장치인 HDD와는 달리 반도체 IC(플래시 메모리)를 이용하여 정보를 저장한다. SSD는 광을 이용한 저장장치가 아니다.

51.

블루레이 디스크(Blu-ray Disc) 고선명(HD) 비디오 데이터를 저장하기 위해 짧은 파장(405nm)을 갖는 레이저를 사용하는 광기록 방식 저장 매체다.

52.

블루레이 디스크(Blu-ray Disc)는 파장이 405nm인 청색 레이저를 사용한다.

53.

RAID(redundant array of inexpensive disks)는 디스크 시스템의 성능과 신뢰성을 향상시키기 위해서 디스크 드라이브의 배열을 구성하여 하나의 유니트로 패키지 함으로써 액세스 속도를 크게 향상시키고 신뢰도를 높였다.

54.

RAID는 여러 개의 작은 디스크들을 배열(array) 구조로 연결하고, 하나의 유니트(unit)로 패키지 함으로써 액세스 속도를 크게 향상시키고 신뢰도를 높였다.

55.

단순히 디스크 배열을 구성한다고 해서 신뢰도가 높아지지는 않는다.

56.

디스크 오류 발생 시, 디스크를 재구성하지 않고 복사본으로 대체하여 데이터를 복구할 수 있는 RAID 레벨은 RAID-1이다. RAID-1은 미러링이라고도 하며, 데이터를 두 개 이상의 디스크에 동시에 기록하여 하나의 디스크가 고장나더라도 다른 디스크에 있는 복사본으로 즉시 대체하여 데이터를 복구할 수 있다.

57.

RAID-2는 검사 디스크를 추가하고 해밍 코드(Hamming code)를 사용해 오류를 검사하고 정정할 수 있다.

58.

RAID-3는 대형 레코드가 많이 사용되는 업무에서 단일 사용자 시스템에 적합하다.

59.

RAID-4에서 패리티 디스크에 액세스가 집중되어 병목 현상이 발생하는 문제점을 해결하기 위해 RAID-5를 개발하였다. RAID-5에서는 패리티 블록도 모든 데이터 디스크에 분산하여 저장한다.

60.

플래시 메모리(flash memory)는 전원이 끊겨도 저장된 데이터를 보존하는 ROM의 장점과 정보 입출력이 자유로운 RAM의 장점을 모두 지녔다. 디지털 캠코더/카메라, 스마트폰, PDA, 게임기, MP3 플레이어 등에서 사용되었다.

61.

플래시 메모리(flash memory)는 블록 단위로 전기적으로 데이터를 지우고 다시 기록할 수 있는 비휘발성 컴퓨터 기억장치다.

62.

SSD(Solid State Drive)는 하드 디스크 드라이브(HDD)와 비슷하게 동작하면서 기계적 장치인 HDD와는 달리 반도체 IC(플래시 메모리)를 이용하여 정보를 저장한다.

63.

메모리 셀에 데이터를 저장하는 방식에 따른 분류

- SLC(Single Level Cell) : 1셀에 1비트 저장
- MLC(Multi Level Cell) : 1셀에 2비트 저장
- TLC(Triple Level Cell) : 1셀에 3비트 저장
- QLC(Quadruple Level Cell) : 1셀에 4비트 저장

64.

메모리 셀에 데이터를 저장하는 방식에 따른 분류

- SLC(Single Level Cell) : 1셀에 1비트 저장
- MLC(Multi Level Cell) : 1셀에 2비트 저장
- TLC(Triple Level Cell) : 1셀에 3비트 저장

제08장. 버스와 입출력

1.

입출력 제어기(I/O 제어기)는 입출력장치 인터페이스와 컴퓨터 시스템 사이에 데이터의 이동을 제어하는 장치다.

2.

CPU가 기억장치 및 입출력장치와의 통로인 시스템 버스는 다음과 같다.

- 주소 버스(address bus)
- 데이터 버스(data bus)
- 제어 버스(control bus)

3.

데이터 버스(data bus)는 양방향성이다. 반면에 주소 버스나 제어 버스는 단방향성이다.

4.

입출력장치가 주기억장치에 비해 저속이므로 이를 해결하기 위해 입출력 제어기가 필요하다.

5.

기억장치는 처리 속도가 nano(10^{-9}) 단위인 전자적인 장치이고, 입출력장치는 milli(10^{-3}) 단위인 기계적인 장치이므로 동작 방식에는 많은 차이가 있다.

비교 항목	입출력장치	기억장치
동작의 속도	느리다	빠르다
동작의 자율성	타율/자율	타율
정보의 단위	byte(문자)	word(워드)
착오 발생률	많다	적다

6.

입출력장치는 데이터 입출력에만 관여하므로 연산기능은 필요하지 않다.

7.

입출력 인터페이스는 동작 방식이나 데이터 형식이 서로 다른 컴퓨터 내부의 주기억장치나 CPU의 레지스터와 외부 입출력장치 간의 데이터를 원활하게 전송하기 위한 방법을 제공한다. 따라서 기기의 크기는 상관없다.

8.

기억장치는 처리 속도가 nano(10^{-9}) 단위인 전자적인 장치이고, 입출력장치는 milli(10^{-3}) 단위인 기계적인 장치이므로 동작 방식에는 많은 차이가 있다.

비교 항목	입출력장치	기억장치
동작의 속도	느리다	빠르다
동작의 자율성	타율/자율	타율
정보의 단위	byte(문자)	word(워드)
착오 발생률	많다	적다

9.

일반적인 컴퓨터 시스템은 CPU, 메모리, 외부 장치(하드 디스크, 키보드, 모니터, 마우스 등)로 구성된다. CPU와 장치 제어기들은 메모리 사이클을 두고 경쟁하며 동시에 실행된다. 경쟁 상황에서 공유하는 메모리에 순차적으로 접근할 수 있도록 메모리 제어기가 메모리에 대한 접근을 제어한다.

10.

입출력장치의 속도가 CPU의 속도보다 느려서 발생하는 CPU의 idle time(시간 낭비)을 줄이기 위해 입출력장치용 버퍼(buffer)를 사용한다.

11.

버스 마스터(bus master) : 시스템 버스에 접속되는 요소 중에서 버스 사용의 주체가 되는 요소들. 예를 들면 : CPU, 기억장치 모듈, I/O 제어기 등을 말한다.

12.

인터럽트 우선순위(interrupt priority)는 여러 개의 인터럽트 요구들이 동시에 들어올 때 그들 중 하나를 선택하기 위해서 사용한다.

13.

회전 우선순위(rotating priority) 방식 : 중재 동작이 끝날 때마다 모든 마스터들의 우선순위가 한 단계씩 낮아지고 가장 우선순위가 낮았던 마스터가 최상위 우선순위를 가지도록 하는 가변 우선순위 방식이다.

14.

회전 우선순위 방식은 모든 버스 마스터들이 공정하게 버스를 사용할 수 있게 해준다.

15.

가변 우선순위 방식

- 회전 우선순위(rotating priority) 방식
- 임의 우선순위(random priority) 방식
- 동등 우선순위(equal priority) 방식
- 최소-최근 사용(Least-Recently Used: LRU) 방식

16.

핸드셰이크(handshake)는 버스 사용을 중재하는 방법이 아니다. 핸드셰이크는 통신에서 데이터 전송 전에 두 장치 간에 연결을 설정하는 과정을 의미한다.

17.

버스 중재를 위한 신호

- 버스 요구(bus request) 신호 : 버스 마스터가 버스 사용을 요구했음을 알리는 신호
- 버스 승인(bus grant) 신호 : 버스 사용을 요구한 마스터에게 사용을 허가하는 신호
- 버스 사용중(bus busy) 신호 : 현재 버스가 사용되고 있는 중임을 나타내는 신호

따라서 버스 에러 신호는 버스 중재 신호가 아니다.

18.

하드웨어적으로 우선순위를 결정하는 방식을 데이저-체인(daisy-chain)이라고 한다.

19.

소프트웨어 폴링 방식은 소프트웨어로 우선순위를 결정하기 때문에 우선순위 변경이 쉽다.

20.

슈퍼스칼라(super scalar)는 CPU 내에 파이프라인을 여러 개 두어 여러 명령어를 동시에 실행하는 기술이다. 따라서 버스 경합을 줄이기 위한 방법이 아니다.

21.

입출력 인터페이스는 동작 방식이나 데이터 형식이 서로 다른 컴퓨터 내부의 주기억장치나 CPU의 레지스터와 외부 입출력장치 간의 2진 정보를 원활하게 전송하기 위한 방법을 제공한다.

22.

I/O 인터페이스는 I/O 장치의 데이터와 CPU나 메모리에서 다루는 데이터와의 속도 차이, 전압 레벨 차이, 전송 사이클 길이 차이 등의 차이점을 해결한다.

23.

입출력 인터페이스는 ② 전기적인 신호(signal), ③ 정보교환 코드(code), ④ 전송제어 방식(protocol)이 서로 다른 컴퓨터 내부의 주기억장치나 CPU의 레지스터와 외부 입출력장치 간의 2진 정보를 원활하게 전송하기 위한 방법을 제공한다.

24.

입출력 인터페이스는 주소 인코딩(address encoding) 회로가 필요하지 않다.

25.

Memory mapped I/O는 중앙처리장치(CPU)로부터 발생되는 메모리 읽기 신호와 쓰기 신호를 이용하여 입출력장치에 대한 읽기와 쓰기를 수행할 수 있는 방식이다.

26.

메모리 맵 입출력방식은 메모리와 입출력 주소 사이의 구분이 필요 없으므로 입출력을 위한 제어 신호도 필요 없다.

27.

Memory-mapped I/O는 컴퓨터에서 CPU가 입출력장치에 접근할 때, 입출력과 메모리의 주소 공간을 분리하지 않고 하나의 메모리 공간에 취급하는 방식이다.

28.

Memory mapped I/O 방식은 메모리 공간의 일부를 입출력장치에 할당하므로 주소 공간이 줄어든다.

29.

②, ③, ④는 Isolated I/O 방식에 대한 설명이다.

30.

Memory mapped I/O에서는 메모리 주소 공간의 감소를 초래하므로 기억장치의 이용 효율이 낮다.

31.

④ 기억장치의 명령을 입출력 명령으로 사용하는 것이 가능하다.

32.

Isolated I/O는 입출력 레지스터들을 액세스할 때는 반드시 별도의 명령어들을 사용해야 하며, 입출력 주소 공간이 기억장치 주소 공간과는 별도로 지정될 수 있는 장점을 갖는다.

33.

③ 주소 공간이 나뉘어 있으므로 CPU로부터 2개의 I/O 읽기와 I/O 쓰기 신호가 필요하다.

34.

② 메모리 맵 입출력 방식은 메모리에 대한 제어 신호만 필요하고, 메모리와 입출력 주소 사이의 구분은 필요하지 않다.

35.

I/O-mapped 입출력에서는 메모리 주소 공간과 I/O 주소 공간이 분리되어 있다. I/O-mapped 입출력은 메모리

리 주소 공간과 I/O 주소 공간을 별도로 관리하는 방식다.

36.

버스 중재 기능은 I/O 제어기의 주요 기능이 아니다. I/O 제어기의 주요 기능은 다양한 입출력장치들을 관리하고 제어하는 것이다. 버스 중재는 여러 장치들이 동시에 버스를 사용하려고 할 때, 누가 먼저 버스를 사용할지 결정하는 과정이다. 이 과정은 CPU나 다른 시스템 구성 요소에서 처리하며, I/O 제어기의 역할은 버스 중재 이후에 데이터 전송을 관리하는 것이다.

37.

③은 동기식 버스에 대한 설명이다.

38.

①은 동기식 인터페이스(synchronous interface)에 대한 설명이다.

39.

Programmed I/O

- 원하는 I/O가 완료되었는지의 여부를 검사하기 위해서 CPU가 상태 레지스터를 계속 조사하여 I/O가 완료되었으면 MDR(MBR)과 AC 사이의 자료 전송을 CPU가 직접 처리하는 I/O 방식(=폴링)이다.
- 입출력에 필요한 대부분의 일을 CPU가 해주므로 인터페이스는 MDR, flag, 장치번호 디코더로만 구성하면 된다.

40.

중앙처리장치의 처리를 가장 많이 필요로 하는 것은 Programmed I/O 또는 폴링(polling) 방식이다.

41.

Programmed I/O는 입출력에 필요한 대부분의 일을 CPU가 해주므로 인터페이스는 데이터 레지스터(MDR), flag, 장치번호 디코더로만 구성하면 된다. 따라서 단어 계수기는 포함되지 않는다.

42.

Programmed I/O는 입출력에 필요한 대부분의 일을 CPU가 해주므로 비효율적이다.

43.

Programmed I/O에서 입출력 수행은 주기억장치에 적재된 입출력 프로그램에 의해 수행된다.

44.

제시된 내용은 프로그램에 의한 I/O(programmed I/O)에 관한 설명이다.

45.

중앙처리장치의 입출력 명령을 직접 수행해서 주기억장치와 입출력장치 사이에 데이터를 전달하도록 하는 입출력 제어기는 DMA를 말한다. ①에서 하나의 제어기로 여러 종류의 I/O 장치들을 공통적으로 제어하는 기능은 채널의 기능이다.

46.

④ I/O 제어기는 데이터의 흐름을 동기화하고 주변장치와 컴퓨터 사이의 전달 속도를 관리한다.

47.

인터럽트(interrupt)에 의한 입출력방식은 입출력하기 위해 CPU가 계속 flag를 검사하지 않고, 데이터를 전송할 준비가 되면 입출력 인터페이스가 CPU에게 알려 입출력이 이루어지는 방식을 말한다.

48.

인터럽트(interrupt)에 의한 입출력방식은 입출력을 위해 CPU가 계속 flag를 검사하지 않고, 입출력장치의 요구가 있을 경우에만 데이터를 전송하는 방식이다.

49.

인터럽트(interrupt) 제어방식은 CPU가 현재 수행 중인 프로그램의 실행을 중단하고, 입출력 기기와 데이터 전송을 수행한 다음, 다시 원래의 상태로 복귀하는 입출력 제어방식이다.

50.

인터럽트는 CPU가 순차적으로 실행되는 과정에서 예기치 않은 사건이 발생했을 때, 현재 실행 중인 작업을 중단하고 해당 사건을 우선적으로 처리하는 메커니즘이다.

51.

I/O 장치와 메모리 간에 CPU를 통하지 않고 고속으로 직접 데이터를 주고받는 입출력 제어 방법을 DMA(Direct Memory Access)라고 한다.

52.

입출력장치와 메모리 간에 CPU를 통하지 않고 고속으로 직접 데이터를 주고받는 입출력 제어 방법을 DMA (Direct Memory Access)라고 한다.

53.

DMA(Direct Memory Access)에서는 입출력 자료 전송 시 CPU를 거치지 않기 때문에 CPU의 부담 없이 보다 빠른 데이터의 전송이 가능하다.

54.

DMA(Direct Memory Access)에서는 입출력 자료 전송 시 CPU를 거치지 않기 때문에 CPU의 부담 없이 보다 빠른 데이터의 전송이 가능하므로 매우 효율적이다.

55.

DMA(Direct Memory Access)는 입출력 자료 전송 시 CPU를 거치지 않기 때문에 매우 효율적인 입출력방식이다.

56.

DMA 내부 레지스터

- Data Register : 임시 데이터 보관
- Address Register : 소스와 목적지 주소
- Data Counter Register : 전송 데이터 길이
- Control Register : 제어비트 설정으로 I/O 수행

57.

CPU에서 DMA 제어기로 보내는 자료

- I/O 장치의 주소
- 데이터가 있는 주기억장치의 시작 주소
- DMA를 시작시키는 명령
- 입출력하려는 자료의 양
- 입력 또는 출력을 결정하는 명령

58.

DMA의 구성 요소

- 인터페이스 회로 : CPU와 입출력장치와의 통신 담당
- 주소 레지스터(Address Register) 및 주소 라인 : 기억장치의 위치 지정을 위한 번지 기억 및 전송
- 워드 카운트 레지스터(Word Count Register) : 전송되어야 할 워드의 수 기억
- 제어 레지스터(Control Register) : 전송 방식 결정
- 데이터 레지스터(Data Register) : 전송에 사용할 자료나 주소를 임시로 기억하는 버퍼 역할을 함

59.

CPU에서 DMA 제어기로 보내는 자료

- I/O 장치의 주소
- 데이터가 있는 주기억장치의 시작 주소
- DMA를 시작시키는 명령
- 입출력하려는 자료의 양
- 입력 또는 출력을 결정하는 명령

60.

DMA는 CPU의 개입 없이 메모리와 주변장치 사이에서 데이터 전송을 수행한다.

61.

인터럽트에 의한 입출력 제어에서는 CPU의 상태 보존은 반드시 필요하지만 DMA에서는 아니다.

62.

DMA는 자료 전송에 CPU의 레지스터를 직접 사용하지 않는다.

63.

DMA 방식은 Programmed I/O 방식에 비해 데이터의 전송 속도가 매우 빠르다.

64.

③ 인터럽트가 발생하면 수행하고 있던 프로그램은 정지되며 인터럽트 처리 루틴의 수행을 위하여 CPU는 인스트럭션을 수행한다.

65.

DMA 방식은 입출력 자료 전송 시 CPU를 거치지 않기 때문에 CPU의 부하가 증가되지 않는다.

66.

④ DMA 제어방식은 프로그램 I/O 제어방식 또는 인터럽트 I/O 제어방식보다 매우 빠르다는 장점이 있다.

67.

버스 요청은 200회(1워드가 16비트)가 필요하며, 데이터 전송 후 인터럽트 신호를 발생시켜 CPU에게 입출력 종료를 알린다.

68.

사이클 스틸링(Cycle Stealing)은 DMA 제어기가 한 번에 한 데이터 워드를 전송하고 버스의 제어를 CPU에게 돌려주는 방법을 말한다.

69.

사이클 스틸링은 DMA 전송 중에 CPU가 메모리 버스를 사용하지 않는 시간을 활용하여 DMA 전송을 가능하게 하는 방식이다.

70.

사이클 스틸은 CPU가 내부적으로 명령어를 해독하거나 연산을 수행하는 등 시스템 버스를 사용하지 않는 시간 동안에만 시스템 버스를 사용하기 때문에 CPU는 메모리 참조가 필요 없는 오퍼레이션을 계속 수행할 수 있지만 주기억장치에 접근할 수는 없다.

71.

DMA 방식에 의한 사이클 스틸은 주기억장치 사이클

의 한 주기만 정지된다.

72.

사이클 스틸링(cycle stealing)이 발생하면 CPU는 해당 사이클 동안 다른 작업을 수행하지 못하고 정지된다. 사이클 스틸링은 DMA(Direct Memory Access)와 같은 하드웨어 장치가 메모리에 직접 접근할 때 CPU의 자원을 잠시 빌려 사용하는 현상이다. 이때 CPU는 메모리 접근 권한을 DMA에게 넘겨주고 해당 사이클 동안 아무런 작업을 할 수 없게 된다.

73.

- 사이클 스틸은 CPU의 상태 보존이 필요 없다.
- 인터럽트는 CPU의 상태 보존이 필요하다.
- 정전도 인터럽트의 일종이다.

74.

DMA의 전송 절차

㉞ 버스 사용 요구 → ㉡ 버스 사용 허가 → ㉠ 데이터 전송 → ㉢ 인터럽트

75.

$4M/8K = 500$, 따라서, $1500 \times 500 = 750000$ 이다.

76.

CPU와 입출력장치와의 처리 속도의 차이를 해결하기 위하여 입출력만을 위한 전용 처리장치로서 CPU처럼 독자적으로 주기억장치에 저장된 명령어를 해독하여 처리할 수 있는 I/O 프로세서를 채널이라고 한다.

77.

- 채널(I/O 프로세서)은 DMA 방법으로 입출력을 수행하므로 DMA의 확장된 개념으로 볼 수 있다.
- DMA 제어기의 한계를 극복하기 위하여 고안된 방식이다.

78.

입출력장치와 주기억장치 사이의 데이터 전송을 담당하는 입출력 전달 장치를 채널 장치라고 한다.

79.

채널(channel)은 주기억장치와 입출력장치 사이에 있다.

80.

채널 명령어(CCW, Channel Command Word)는 CPU 명령어와 구별하기 위해 Command라고 부른다.

- 명령코드(Operation Code) : 입출력 명령을 지정함
- 데이터 주소(Data Address) : 블록의 시작 주소를 표시함

- 플래그(flag) : CCW의 링크 주소 등 통신 목적이나 상태 기록을 위한 부분
- 워드 카운터(Word Counter) : 전송될 블록의 단어 수를 표시함

81.

채널 명령어의 구성 요소와 I/O device 처리 속도는 관계가 없다.

82.

채널 명령어 구조

OP code	flag bit	operand부(address)
---------	----------	-------------------

83.

채널의 종류

- Selector Channel : 고속 입출력장치(자기디스크, 자기테이프, 자기 드럼) 1개와 입출력하기 위해 사용한다.
- Multiplexer Channel : 저속 입출력장치(카드 리더, 프린터) 여러 개를 동시에 제어하는 채널이며, 바이트 멀티플렉서 채널이라고도 한다.
- Block Multiplexer Channel : 동시에 여러 개의 고속 입출력장치를 제어한다(Selector와 Multiplexer 방식을 결합).
- 속도 비교 : Selector > Block > Multiplexer

84.

입출력 동작이 시작되어 끝날 때까지 하나의 입출력 장치가 전용으로 쓸 수 있는 채널로서 고속 장치에 주로 쓰이는 채널은 셀렉터 채널(Selector Channel)이다.

85.

Selector Channel은 고속 입출력장치 1개와 입출력하기 위해 사용한다.

86.

Selector Channel은 비교적 고속 입출력장치(자기디스크) 1개와 입출력하기 위해 사용한다.

87.

Multiplexer Channel은 저속 입출력장치(프린터) 여러 개를 동시에 제어하는 채널이며, 바이트 멀티플렉서 채널이라고도 한다.

88.

채널에서 버스트 방식(burst mode)은 전송을 요구한 장치에서 한꺼번에 여러 바이트를 전송하는 방식으로 출력 형태는 AAABBBCCCC가 된다.

89.

멀티플렉서 채널과 셀렉터 채널의 구분은 입출력장치의 속도 차이에 있다.

90.

채널 제어기는 하나의 입출력 명령에 의해 여러 블록의 자료를 입출력할 수 있다.

91.

모뎀(MODEM)은 통신제어장치다.

92.

② multiplexer 채널은 **저속** 입출력 장치용이고, select 채널은 **고속** 입출력 장치용이다.

93.

④ 멀티플렉서 채널(multiplexer channel)은 복수 개의 입출력장치를 동시에 제어할 수 있다.

94.

- ① 멀티플렉서 채널은 저속 입출력장치 여러 개를 동시에 제어하는 채널 형태이다.
- ② 셀렉터 채널은 고속 입출력장치 여러 개를 동시에 제어하는 채널 채널 형태이다.
- ④ 채널은 입출력장치가 작동 중일 때마다 중앙처리장치의 지시 없이 동작한다.

95.

입출력(Input-Output) 제어 방식으로는 Programmed I/O, Interrupt I/O, DMA에 의한 I/O, Channel에 의한 I/O가 있다.

96.

① DMA는 하나의 인스트럭션으로 하나의 데이터 워드를 입출력할 수 있다.

97.

입출력(Input-Output) 제어 장치로는 DMA, 채널, 입출력 프로세서(IOP)가 있다.

98.

채널 제어기에 의한 입출력 방식이 가장 성능이 우수하다.

99.

Cycle Steal을 이용하는 DMA는 한 번에 한 데이터 워드를 전송하고 버스의 제어를 CPU에게 돌려준다.

100.

- I/O의 비동기 데이터 전송방식 : 스트로브 펄스와 핸드셰이킹
- I/O 제어 방식 : 프로그램, 인터럽트, DMA, 채널

101.

입출력 방식으로는 프로그램에 의한 방식(중앙처리장치에 의한 입출력), DMA(Direct Memory Access) 방식, 채널 제어기에 의한 입출력이 있다.

102.

I/O 제어방식의 종류

- Programmed I/O
- Interrupt I/O
- DMA(Direct Memory Access)에 의한 I/O
- Channel에 의한 I/O

103.

DMA에 의한 입출력과 채널 제어기에 의한 입출력은 CPU의 개입 없이 입출력 동작이 이루어진다.

104.

입출력(I/O) 방식은 입출력할 때, CPU를 통과하는 방법과 CPU를 거치지 않는 방법의 2가지로 크게 나눈다. 후자의 예는 **DMA**를 이용한 입출력이나 **IOP**를 이용한 입출력을 의미한다. 한편, 전자는 프로그램 제어 입출력이라 하며, 이 방식은 CPU와 입출력장치의 속도 차이 때문에 비효율적이다.

제09장. 인터럽트

1.

인터럽트는 컴퓨터에서 어떤 특수한 상태가 발생하면 현재 실행하고 있는 프로그램이 일시 중단되고, 그 특수한 상태를 처리하는 프로그램으로 옮겨져 처리한 후 복귀하여 다시 원래의 프로그램을 처리하는 것을 말한다.

2.

인터럽트(interrupt)는 컴퓨터 시스템에 예기치 않은 일이 발생했을 때 제어 프로그램에게 알려주는 것을 말한다.

3.

컴퓨터는 **인터럽트** 요청 신호가 입력되면 프로그램 실행 중에 있는 CPU가 정상적인 처리를 멈추고 **인터럽트**에 대한 처리를 마친 후 정상적인 처리를 다시 수행하게 된다.

4.

인터럽트란 컴퓨터가 정상적인 작업을 수행하는 도중에 발생하는 예기치 않은 일들에 대한 서비스를 수행하는 기능이므로 모두 에러(error)에 대한 복구 기능만을 가지고 있다고 볼 수 없다.

5.

프로그램 수행 중에 인터럽트가 발생하였을 경우 수행 중인 명령(instruction)을 끝내고 인터럽트를 처리한다.

6.

④ 인터럽트가 발생하면 현재 수행 중인 프로그램을 중단하고 인터럽트 처리를 수행한 후 다시 복귀하여 중단되었던 프로그램을 계속한다.

7.

임의의 부프로그램에 대한 호출은 인터럽트 발생과는 무관하다.

8.

프로그램 내 A 루틴에서 B 루틴으로의 연결은 인터럽트 발생과는 무관하다.

9.

처리할 데이터양이 많은 경우는 인터럽트 발생과는 무관하다.

10.

분기 명령의 실행은 정상적인 프로그램 처리 과정이며, 인터럽트 발생과는 무관하다.

11.

CPU와 I/O 장치 간의 정보전달에 인터럽트를 사용한다. 입출력하기 위해 CPU가 계속 flag를 검사하지 않고, 데이터를 전송할 준비가 되면 입출력 인터페이스가 컴퓨터에게 알려 입출력이 이루어지는 방식이다.

12.

정전이나 전원 이상은 외부 인터럽트의 원인이다.

13.

외부 인터럽트(external interrupt) 중에서 입출력 인터럽트(Input-Output Interrupt)는 입출력장치가 데이터의 전송을 요구하거나 전송이 끝났음을 알릴 경우에 발생한다.

14.

타이밍 장치에 의해 발생하는 인터럽트는 외부 인터럽트(external interrupt)에 해당한다.

15.

프로그램 검사 인터럽트(Program Check Interrupt)는 내부 인터럽트(internal interrupt)에 해당한다.

16.

소프트웨어 인터럽트(software interrupt)는 프로그램 처리 중 명령의 요청에 의해 발생하는 것으로, 가장 대표적인 형태는 감시 프로그램을 호출하는 SVC(SuperVisor Call) 인터럽트가 있다. SVC 인터럽트는 사용자가 SVC 명령을 써서 의도적으로 호출한 경우이다.

17.

오퍼레이터나 타이머에 의해 의도적으로 발생하는 인터럽트는 외부 인터럽트(external interrupt)에 해당한다.

18.

잘못된 명령이나 데이터를 사용할 때 발생하며, 트랩(trap)이라고도 부른다.

- 0으로 나누기(divide by zero)가 발생한 경우
- overflow 또는 underflow가 발생한 경우
- 프로그램에서 명령어를 잘못 사용한 경우

- 부당한 기억장소의 참조와 같은 프로그램의 오류
- 스택의 overflow, 불법적인 명령의 실행, 무한 루프

19.

내부 인터럽트는 잘못된 명령어나 데이터를 사용할 때 발생한다. 이외에도 캐시를 액세스할 때 발생하는 캐시 미스(cache miss)와 가상기억장치에서 발생하는 페이지 폴트(page fault)가 있다.

20.

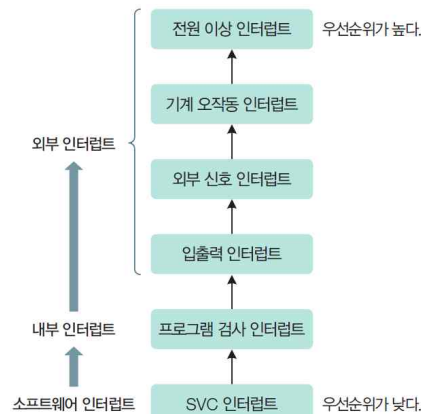
SVC 인터럽트(supervisor call interrupt)는 컴퓨터 제어를 감독자에게 넘겨주기 위해 현재 수행 중인 프로그램에 소프트웨어적으로 발생시키는 인터럽트다.

21.

정전(power fail)은 인터럽트 우선순위가 가장 높다.

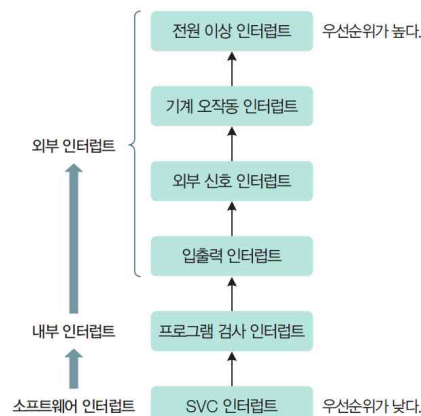
22.

㉠ 정전이나 전원의 끊어짐 → ㉡ 기계적 고장 → ㉢ 프로그램의 에러 → ㉣ 입력과 출력



23.

㉠ 전원 이상 → 기계 착오 → 외부 신호 → 입출력 → 명령의 잘못 사용 → 슈퍼바이저 호출



24.

CPU가 어떤 프로그램을 순차적으로 수행하는 도중에 외부로부터 인터럽트 요구가 들어오면 원래의 프로그램을 중단하고 인터럽트 서비스 루틴(interrupt service routine)을 먼저 수행한다.

25.

④는 인터럽트 우선순위에 관련된 내용이다.

26.

인터럽트의 처리 루틴의 순서

- 인터럽트 인식
- 현재 상태 보존
- 요청 인터럽트 서비스로 분기 및 서비스
- 사용자 상태 복구 및 재개

27.

인터럽트 동작 순서

- 인터럽트 요청 신호가 발생한다.
- 현재 수행 중인 명령을 완료하고 복귀주소를 저장한다.
- 어느 장치가 인터럽트를 요청했는지 찾는다.
- 인터럽트 서비스 루틴을 수행한다.
- 보존한 프로그램 상태로 복귀한다.

28.

인터럽트 요청 신호가 발생한 순간부터 인터럽트 서비스 루틴이 시작될 때까지 시간을 인터럽트 반응 시간(interrupt response time)이라 하고, 이 시간은 짧을수록 좋다.

29.

함수 호출이나 인터럽트 발생 시에 복귀주소를 저장하기 위해서 또는 수식을 계산할 때 스택을 사용한다.

30.

인터럽트 시스템의 기본 요소

- 인터럽트 요청(interrupt request) 신호
- 인터럽트 처리 루틴(interrupt processing routine)
- 인터럽트 서비스 루틴(interrupt service routine)

31.

인터럽트 서비스 루틴을 수행하기 전에 수행 중인 프로그램 상태, 즉 프로그램이 사용한 모든 레지스터 내용과 다음에 수행할 명령어 주소가 저장된 프로그램 카운터(PC)를 안전한 장소에 보관한다.

32.

인터럽트 처리 루틴에서는 수행 중인 프로그램 상태,

즉 프로그램이 사용한 모든 레지스터 내용과 다음에 수행할 명령어 주소가 저장된 프로그램 카운터(PC)를 안전한 장소에 보관한다.

33.

인터럽트가 발생하면 프로그램의 현재 상태(프로그램 카운터 내용)를 반드시 스택에 보관해야 한다.

34.

인터럽트 발생 시 복귀주소를 스택에 저장한다.

35.

인터럽트 발생 시 복귀주소를 스택(stack)에 저장한다.

36.

누산기(Accumulator), 상태 레지스터(State Register), 프로그램 카운터(Program Counter) 등은 스택에 저장하며, 명령 레지스터(instruction register)는 스택에 저장할 필요가 없다.

37.

프로그램 카운터의 값, 프로세서의 상태 조건 값, 프로세서 내의 레지스터 내용은 스택에 저장하며, 프로그램의 크기는 스택에 저장할 필요가 없다.

38.

인터럽트 발생 시 스택의 내용은 확인할 필요가 없다.

39.

인터럽트가 발생하면 프로세서의 상태 보존이 필요한데, 그 이유는 인터럽트를 요청한 해당 장치에 대한 인터럽트 서비스를 완료하고 원래 수행 중이던 프로그램으로 복귀하기 위해서다.

40.

③ 수행 중인 프로그램을 보조기억장치에 보관할 필요는 없다.

41.

인터럽트 서비스 루틴을 수행한 후에는 인터럽트 처리 시 보존시켰던 프로그램 카운터(PC) 및 누산기(AC) 등의 내용을 복구한다.

42.

인터럽트 서비스 프로그램은 실행 중이더라도 우선순위가 더 높은 다른 인터럽트를 처리할 수 있다.

43.

① 프로그램 사태에 관련된 일부 레지스터의 내용은

스택에 보관한다.

44.

인터럽트를 요청한 장치를 소프트웨어로 판별하는 방법을 폴링(Polling)이라고 한다.

45.

폴링 방식은 인터럽트 발생 시 가장 높은 우선순위의 인터럽트 자원(Source)부터 인터럽트 요청 플래그를 차례로 검사해서, 우선순위가 가장 높은 Interrupt 자원(Source)을 찾아내어 이에 해당하는 인터럽트 서비스 루틴을 수행하는 방식이다.

46.

소프트웨어적인 방법은 우선순위 변경이 쉬우며, 별도의 하드웨어가 필요 없으므로 경제적이다. 그러나 많은 인터럽트가 있을 경우 그들을 모두 조사하는 데 많은 시간이 걸려 반응 시간이 느리다는 단점이 있다.

47.

- 벡터 인터럽트(vectored interrupt) : 하드웨어에 의해 인터럽트를 벡터 테이블을 이용하여 수행
- 폴링 방식 인터럽트 : 소프트웨어 인터럽트

48.

벡터 인터럽트 방식에서는 인터럽트를 발생한 장치가 CPU에게 분기할 곳에 대한 정보를 제공하는데, 이 정보를 인터럽트 벡터라고 한다.

49.

Vectored Interrupt 방식은 CPU와 인터럽트를 요청할 수 있는 장치 사이에 장치번호에 해당하는 버스를 병렬이나 직렬로 연결하여 요청 장치의 번호를 CPU에 알리는 방식이다.

50.

벡터 인터럽트 방식에서는 인터럽트를 발생한 장치가 CPU에게 분기할 곳에 대한 정보를 제공하는데, 이 정보를 인터럽트 벡터라고 한다.

51.

인터럽트 벡터는 인터럽트가 발생했을 때 항상 일정한 주소로 분기하여 처리하기 때문에 해당 분기의 주소를 필수적으로 알고 있어야 한다.

52.

- ① 인터럽트가 발생했을 때는 반드시 CPU의 현재 상태를 보존해야 한다.
- ② 인터럽트 발생 시 CPU는 인터럽트를 요구한 장치

의 요구를 처리한다.

- ④ 인터럽트 서비스 루틴 처리를 수행한 후 이전에 수행 중이던 프로그램의 처음 상태가 아닌 인터럽트 직전에 처리하던 위치로 복귀한다.

53.

- ④ 하드웨어에 의한 판별 방법은 소프트웨어에 의한 판별 방법보다 속도가 빠르다.

54.

- ② 단일 인터럽트 요청 신호 회선 체계에서는 소프트웨어에 의해 인터럽트를 확인하는 폴링 방법과 하드웨어 의한 데이지 체인 방법이 가능하다. 이 경우 복귀주소인 PC의 값을 메모리 0번지, 스택, 인터럽트 벡터 등 다양하게 저장한다.

55.

- ① 단일 인터럽트 요청 신호 회선 체계의 경우 장치 식별이 필요하다.
 ② 폴링 방식에서는 소프트웨어적으로 인터럽트를 요청한 장치를 식별한다.
 ③ 벡터 인터럽트 방식은 하드웨어에 의한 장치 식별 방식이다.

56.

소프트웨어적으로 인터럽트의 우선순위를 결정하는 인터럽트 방식을 폴링 방식이라고 한다.

57.

소프트웨어적인 방법은 우선순위 변경이 쉬우며, 별도의 하드웨어가 필요 없으므로 경제적이다. 그러나 많은 인터럽트가 있을 경우 그들을 모두 조사하는 데 많은 시간이 걸려 반응 시간이 느리다는 단점이 있다.

58.

인터럽트 마스크는 인터럽트가 발생하였을 때 요구를 받아들일지 받아들이지 않을지를 결정하는 것이다. 일반적으로 인터럽트 마스크 레지스터를 사용하여 각 비트의 값에 따라 해당 인터럽트의 허용 여부를 결정할 수 있다.

59.

제시된 지문은 소프트웨어적으로 인터럽트를 처리하는 폴링에 대한 설명이다.

60.

- ①은 하드웨어적인 방법(데이지 체인 등)에 대한 설명이다.

61.

데이지 체인(daisy-chain) 방식은 인터럽트가 발생하는 모든 장치를 한 개의 회선에 직렬로 연결한다. 우선순위가 높은 장치를 선두에 위치시키고 나머지를 우선순위에 따라 차례로 연결한다.

62.

데이지 체인(daisy-chain) 방식은 인터럽트가 발생하는 모든 장치를 한 개의 회선에 직렬로 연결한다. 우선순위가 높은 장치를 선두에 위치시키고 나머지를 우선순위에 따라 차례로 연결한다.

63.

데이지 체인(daisy-chain) 방식은 하드웨어를 이용하여 우선순위를 결정한다.

64.

데이지 체인(daisy-chain) 방식은 하드웨어를 이용하여 우선순위를 결정한다.

65.

데이지 체인(daisy-chain) 방식은 인터럽트가 발생하는 모든 장치를 한 개의 회선(인터럽트 신호선)에 직렬로 연결한다. 우선순위가 높은 장치를 선두에 위치시키고 나머지를 우선순위에 따라 차례로 연결한다.

66.

소프트웨어로 우선순위를 결정하는 방식은 응답 속도가 느리다.

67.

하드웨어로 우선순위를 결정하는 방식은 유연성이 떨어진다.

68.

데이지 체인(daisy-chain) 방식은 인터럽트가 발생하는 모든 장치를 한 개의 회선에 직렬로 연결한다.

69.

데이지 체인(daisy-chain) 방식은 인터럽트가 발생하는 모든 장치를 한 개의 회선에 직렬로 연결한다. 우선순위가 높은 장치를 선두에 위치시키고 나머지를 우선순위에 따라 차례로 연결한다. 마스크 레지스터는 우선순위와는 상관이 없으며, 인터럽트 허용 여부를 결정하는데 사용한다.

70.

- ① 하드웨어적으로 가장 높은 순위의 인터럽트의 소스부터 차례로 검사하여 그중 가장 우선순위가 높

은 소스를 찾아낸다.

- ③은 인터럽트 마스크 레지스터에 대한 설명이다.
- ④ CPU에서 멀수록 우선순위가 낮다.

71.

- ④ 폴링 방식이 데이지 체인 방식보다 속도가 느리다.

72.

- ③ 하드웨어에 의한 우선순위 부여는 유연성이 떨어지지만, 인터럽트 반응 시간은 빠르다.

73.

- ② 소프트웨어에 의한 방법에는 폴링 방법을 이용하며 인터럽트 반응속도가 느리다.

74.

- ④ 인터럽트가 발생되면 해결 후에 작업을 계속하기 위해서 어떠한 경우라도 반드시 수행 중인 프로그램의 상태 정보를 스택 등에 보관하여야 한다. 수행 중인 프로그램을 보관하지 않는다.

75.

인터럽트의 병렬 우선순위는 인터럽트가 발생하는 각 장치를 개별적인 회선으로 연결한다. 따라서 폴링과는 상관 없다.

76.

DMA(Direct Memory Access)는 입출력장치가 직접 주기억장치에 접근(Access)하여 데이터를 입출력하는 방식으로 입출력 전송이 CPU의 레지스터를 경유하지 않고 수행되는 입출력 제어 방식이다.

77.

- 하드웨어적 인터럽트 판별 방식 : 데이지 체인 방식
- 소프트웨어적 인터럽트 판별 방식 : 폴링 방식
- 벡터 인터럽트(vectored interrupt) : 하드웨어 신호로 특정 번지의 서브루틴을 수행하는 것

78.

인터럽트 서비스 루틴(Interrupt Service Routine)은 우선순위와는 상관없다.

79.

인터럽트를 동시에 처리할 수 없다. 다만 동시에 인터럽트가 발생하면 우선순위에 따라 순차적으로 처리한다.

80.

우선순위를 변경할 수는 있지만 해제하는 기능은 없다.

제10장. 병렬 컴퓨터 구조

1.

슈퍼 스칼라(super-scalar)는 파이프라이닝을 여러 개 두고 병행하여 실행하는 기술을 말한다.

2.

SISD(Single Instruction stream Single Data stream)

- 명령 하나가 데이터 하나를 처리하는 구조
- 제어장치가 한 개의 명령을 번역한 후 프로세서를 작동시켜 명령을 처리할 때 기억장치에서 한 개의 데이터를 꺼내서 처리

3.

SIMD(Single Instruction stream Multi Data stream)

- 한 개의 명령으로 여러 데이터를 동시에 처리하는 구조
- 다수의 프로세서가 한 개의 제어장치에 의해 제어
- 기억장치모듈은 모든 프로세서에 공유

4.

SIMD(Single Instruction stream Multi Data stream)

- 한 개의 명령으로 여러 데이터를 동시에 처리하는 구조
- 다수의 프로세서가 한 개의 제어장치에 의해 제어됨
- 배열처리기(Array Processor)에 의한 동기적 병렬처리가 가능함

5.

다수의 프로세서(PU)가 한 개의 제어장치(CU)에 의해 제어되며, 기억장치모듈은 모든 프로세서에 공유되는 구조이므로 SIMD(Single Instruction stream Multi Data stream)다.

6.

SIMD는 다수의 프로세서(PU)가 한 개의 제어장치(CU)에 의해 제어되며, 배열처리기 구조를 갖는다.

7.

SIMD(Single Instruction stream Multi Data stream)는 다수의 프로세서(PU)가 한 개의 제어장치(CU)에 의해 제어되는 구조를 갖는다.

8.

SIMD는 다수의 프로세서(PU)가 하나의 제어 프로세서(CU)에 의해 제어되며 주로 배열이나 벡터 처리에 적합한 구조를 갖는다. 예를 들어, 멀티미디어 자료를 처리하는 SIMD 명령어가 있다. 커다란 이미지를 작은

조각으로 나누어 여러 프로세서에서 처리한 후 다시 합치는 응용에 사용할 수 있다. 또는 3차원 물체를 만들고 그 재료의 특징이나 광원 위치에 따라 자연스러운 이미지를 만드는 멀티미디어 렌더링 처리용으로 제작될 수 있다.

9.

- 단일 명령어 스트림 - 단일 데이터 스트림 (SISD)
- 단일 명령어 스트림 - 복수 데이터 스트림 (SIMD)
- 복수 명령어 스트림 - 단일 데이터 스트림 (MISD)
- 복수 명령어 스트림 - 복수 데이터 스트림 (MIMD)

10.

MISD는 데이터 스트림 하나에 다중의 명령 스트림이 작동한다. 실제로 존재하는지는 명확하지 않지만 일부 사람들은 파이프라인 컴퓨터를 MISD로 간주하기도 한다. 그러나 그렇게 활용도가 높지 않고 응용 분야도 거의 없다고 볼 수 있다

11.

MISD는 데이터 스트림 하나에 다중의 명령 스트림이 작동한다. 실제로 존재하는지는 명확하지 않지만 일부 사람들은 파이프라인 컴퓨터를 MISD로 간주하기도 한다. 그러나 그렇게 활용도가 높지 않고 응용 분야도 거의 없다고 볼 수 있다

12.

- 단일 명령어 스트림 - 단일 데이터 스트림 (SISD)
- 단일 명령어 스트림 - 복수 데이터 스트림 (SIMD)
- 복수 명령어 스트림 - 단일 데이터 스트림 (MISD)
- 복수 명령어 스트림 - 복수 데이터 스트림 (MIMD)

13.

- 단일 명령어 스트림 - 단일 데이터 스트림 (SISD)
- 단일 명령어 스트림 - 복수 데이터 스트림 (SIMD)
- 복수 명령어 스트림 - 단일 데이터 스트림 (MISD)
- 복수 명령어 스트림 - 복수 데이터 스트림 (MIMD)

14.

- 단일 명령어 스트림 - 단일 데이터 스트림 (SISD)
- 단일 명령어 스트림 - 복수 데이터 스트림 (SIMD)
- 복수 명령어 스트림 - 단일 데이터 스트림 (MISD)
- 복수 명령어 스트림 - 복수 데이터 스트림 (MIMD)

15.

- 단일 명령어 스트림 - 단일 데이터 스트림 (SISD)

- 단일 명령어 스트림 - 복수 데이터 스트림 (SIMD)
- 복수 명령어 스트림 - 단일 데이터 스트림 (MISD)
- 복수 명령어 스트림 - 복수 데이터 스트림 (MIMD)

16.

병렬처리 시스템은 시스템 내에 여러 프로세서를 통해 처리작업을 분담하여 동시에 처리할 수 있다. 따라서 많은 양의 데이터를 처리하고 빠르게 작업을 완료할 수 있으며 많은 입출력장치의 요구를 수용할 수 있다.

17.

병렬처리 방식은 여러 개의 마이크로프로세서가 하나의 프로그램 상의 서로 다른 태스크(task)를 동시에 처리함으로써 부하를 분담하여 처리 속도를 향상시킨 방식이다.

18.

파이프라인(pipeline)은 하나의 프로세서를 여러 개의 서브프로세스(subprocess)로 나누어 각 서브프로세스가 동시에 서로 다른 데이터를 처리하도록 하는 기법이다.

19.

명령 파이프라인은 명령어를 읽어 순차적으로 실행하는 프로세서에게 적용되는 기술로, 한 번에 하나의 명령어만 실행하는 것이 아니라 하나의 명령어가 실행되는 도중에 다른 명령어 실행을 시작하는 식으로 동시에 여러 개의 명령을 실행하는 기법이다.

20.

파이프라인(pipeline) 기법은 한 명령어의 수행이 끝나기 전에 다른 명령어의 수행을 시작하는 연산 방법을 말한다.

21.

파이프라인 프로세서(pipeline processor)는 하나의 프로세서로 여러 명령어를 처리할 수 있다.

22.

파이프라인(pipeline)은 하나의 프로세서를 여러 개의 서브프로세스(subprocess)로 나누어 각 서브프로세스가 동시에 서로 다른 데이터를 처리하도록 하는 기법이다.

23.

- 파이프라인(Pipeline)은 명령어의 연산 과정을 몇 개의 단계로 구분하여, 한 명령어의 수행이 끝나기 전에 다른 명령어의 수행을 시작하는 기법이다.

- 4단계 명령어 파이프라인의 수행 순서는

IF(Instruction Fetch) → ID(Instruction Decode) → OF(Operand Fetch) → EX(Execution) 이다.

24.

명령 파이프라인(instruction pipeline)은 FIFO 구조를 갖는다.

25.

병렬처리의 목적은 처리 속도의 향상에 있다.

26.

병렬처리 방법

- 파이프라인 프로세서(pipeline processor)
- 벡터 프로세서(vector processor)
- 배열 처리기(array processor)
- 데이터 흐름 컴퓨터(data flow computer)

27.

벡터 형태의 데이터를 처리하는데 가장 효율적인 병렬 처리기는 배열 처리기(array processor)다.

28.

배열 프로세서(array processor)

- PE(Processing Element)라고 불리는 다수의 연산기를 갖는 동기적 병렬 프로세서다.
- 명령 해독 및 제어는 제어장치가 하고 PE들은 명령 해독 능력이 결여된 수동적 장치로서 명령 처리만 한다.
- 각 PE들은 데이터 운행 연결망에 의해 상호 연결되어 PE(ALU)들을 중복 이용함으로써 공간적 병렬성을 얻을 수 있다.

29.

모든 처리장치(Processing Element: PE)들이 하나의 제어 유니트(Control Unit: CU)의 통제 하에 동기적으로 동작하는 시스템을 배열 프로세서(array processor)라고 한다.

30.

큐브(Cube)는 약결합(Loosely-Coupled) 시스템에서 사용되는 상호연결 구조이다.

31.

상호연결 구조에는 3가지 방식이 사용되고 있다.

- 공유 버스(Bus)
- 크로스바 스위치(Crossbar Switch)
- 다단계 상호연결망(Multistage Interconnection Network)

32.

다중처리기 상호 연결 방법 중 공유 버스(시분할 공유 버스)는 각종 장치들(프로세서, 기억장치, 주변장치 등)을 단일 버스로 연결한 구조로 한 시점에 단지 하나의 전송만이 가능하다.

33.

다중 포트 메모리(Multi-port memory)

크로스바 교환행렬과 시분할 공유 버스 시스템을 혼용한 방법이다. 하나의 프로세서에 하나의 버스가 할당되어 버스를 이용하려는 프로세서 간 경쟁이 적다.

34.

밀 결합 시스템(tightly-coupled system)은 기억장치가 모든 프로세서들에게 공유되는 공유기억장치 구조를 갖는다(다중프로세서 시스템).

35.

- 다중 처리기(Multi-Processor)란 하나의 시스템에 여러 개의 처리기와 하나의 기억장치를 이용하여 하나의 작업을 각 처리기에서 분산하여 신속하게 처리하는 장치이다.
- 각 처리기들은 하나의 기억장치를 공동으로 이용한다.

36.

- ③ 하나의 복합적인 운영체제에 의해 전체 시스템이 제어된다.

37.

다중 처리기(Multi-Processor)란 하나의 시스템에 여러 개의 처리기와 하나의 기억장치를 이용하여 하나의 작업을 각 처리기에서 분산하여 신속하게 처리하는 장치이다.

38.

- ③ CPU를 한 개만 두고 동시에 여러 프로그램을 수행할 수 있다.

39.

데이터 흐름 컴퓨터(data flow machine)는 어떤 명령에서 필요한 피연산자가 모두 준비되었을 때 비로소 그 명령을 수행한다.

40.

벡터 처리기에서 사용할 수 있는 알고리즘으로 가장 적합한 알고리즘은 Systolic 알고리즘이다.

41.

비트 슬라이스(bit slice)는 2비트, 4비트 등의 소단위 칩을 여러 개 접속하여 16비트, 32비트, 64비트 등의 CPU를 만드는 기법이다. 비트 슬라이스 마이크로프로세서는 구성면에서 단일 칩으로 된 마이크로프로세서보다 매우 다양하며, 특히 데이터의 정확성을 어느 수준까지 끌어 올릴 수가 있고 명령문의 구성을 자유롭게 할 수 있다.

42.

비트 슬라이스(bit slice)는 2비트, 4비트 등의 소단위 칩을 여러 개 접속하여 16비트, 32비트, 64비트 등의 CPU를 만드는 기법이므로 용도에 맞게 CPU를 구성할 수 있다.

43.

- 양극성 소자(bipolar)로 구성하면 속도가 빠르다.
- 2비트, 4비트 등의 소단위 칩을 여러 개 접속하여 16비트, 32비트, 64비트 등의 CPU를 만드는 기법이므로 단일 칩으로 제작이 안 된다.

44.

비트 슬라이스 마이크로프로세서(bit-slice microprocessor)는 2비트, 4비트 등의 소단위 칩을 여러 개 접속하여 16비트, 32비트, 64비트 등의 CPU를 만드는 기법으로 용도에 맞게 CPU를 구성할 수 있다.

45.

다중 프로그래밍(multi programming)은 CPU를 한 개만 두고 여러 개의 프로그램을 주기억장치 내에 기억시켜 놓고 동시에 실행하도록 한다.

46.

메모리 용량 문제는 다중처리기에 의한 시스템을 구성할 때 고려사항이 아니다.

47.

프로세서들이 공통으로 사용하는 데이터들이 공유 기억장치에 저장되므로 별도의 프로세서 간 통신 메커니즘이 필요하지 않다.

48.

다중 포트 메모리(Multi-port memory)에서는 하나의 프로세서에 하나의 버스가 할당되어 버스를 이용하려는 프로세서 간 경쟁이 적다.

49.

프로세서 간의 통신은 공유 기억장치를 통하여 이루어진다.

50.

매크로 프로세서는 원시 프로그램에 존재하는 매크로 호출 부분에 매크로 프로그램을 삽입하여 확장된 원시 프로그램을 생성하는 시스템 소프트웨어다. 따라서 병렬처리와는 상관없다.

51.

① 파이프 라인 방식, ② 배열 방식, ③ VLSI 처리기 방식, ④ 벡터 방식은 병렬 처리기 방식이다. 따라서 정답 없음

52.

병렬처리를 위한 해결 사항

- 분할의 문제 : 하나의 작업을 최대의 병렬성을 갖도록 분리하기 위한 문제
- 스케줄링의 문제 : 분할된 작업들을 처리기에 할당하기 위한 문제
- 동기화의 문제 : 여러 프로세스에 의해 공유되는 자원이 있을 경우 동시에 여러 프로세스가 공유자원에 접근하여 작업을 수행할 경우 작업의 일관성이 무너질 수 있으므로 각 프로세스가 순서적으로 공유자원에 접근하도록 제어하기 위한 문제

53.

병렬처리 컴퓨터에서 처리기의 개수가 늘어나면 처리 속도가 빨라지지만, 처리기를 N 개 사용한다고 해서 처리 속도가 정확히 N 배 빨라지지는 않는다.

54.

선형 처리 방식을 고속화하기 위해 병렬처리 방식이 제안되었다. 따라서 선형 처리 방식은 고속 연산의 실현 방법이라고 볼 수 없다.

55.

다중 프로그래밍(multi programming)은 CPU를 한 개만 두고 여러 개의 프로그램을 주기억장치 내에 기억시켜 놓고 동시에 실행하도록 한다. 따라서 병렬성을 얻기 위한 방법으로 볼 수 없다.

56.

분산 및 병렬 처리시스템은 CPU를 여러 개 사용하는

시스템이다. 이러한 시스템은 처리시간이 빨라지고 처리량 및 사용 가능도가 향상되며 많은 자원을 동시에 공유하기에 적합한 시스템이다. 그러나 여러 개의 CPU를 관리하고 제어하는 소프트웨어 개발이 어렵고 보안성 및 적응성이 하나의 CPU를 사용할 때보다는 떨어진다.

57.

멀티프로그래밍(multi programming)은 CPU를 한 개만 두고 여러 개의 프로그램을 주기억장치 내에 기억시켜 놓고 동시에 실행하도록 한다.

58.

멀티프로그래밍(multi programming)은 CPU를 한 개만 두고 여러 개의 프로그램을 주기억장치 내에 기억시켜 놓고 동시에 실행함으로써 컴퓨터 시스템 자원 활용률을 극대화한다.

59.

컴퓨터 클러스터(computer cluster) : 여러 대의 컴퓨터들이 연결되어 하나의 시스템처럼 동작하는 컴퓨터들의 집합

60.

제시된 지문은 결함허용 시스템(fault-tolerant system)을 말한다.

61.

클라우드 컴퓨팅 서비스

- 소프트웨어 서비스(SaaS, software as a service) : 네트워크를 통해 소프트웨어를 온라인으로 이용하는 방식
- 플랫폼 서비스(PaaS, platform as a service) : 운영체제를 빌려 쓰는 방식
- 인프라 서비스(IaaS, infrastructure as a service) : 서버나 스토리지, 데이터베이스, 네트워크를 필요에 따라 이용할 수 있게 서비스를 제공하는 형태

제11장. 성능 분석과 측정

1.

메가플롭스(MFLOPS)

- Million of Floating Point Operations Per Second
- 1초 동안에 실행되는 부동소수점 연산의 수를 100만을 단위로 하여 나타낸 수를 말한다.

2.

메가플롭스(MFLOPS)는 1초 동안에 실행되는 부동소수점 연산의 수를 100만을 단위로 하여 나타낸 수를 말한다.

$$\text{MFLOPS} = \frac{\text{프로그램내의 부동소수점 연산 개수}}{\text{수행시간} \times 10^6}$$

3.

성능을 측정하는 데 사용되는 단위는 MIPS이다.