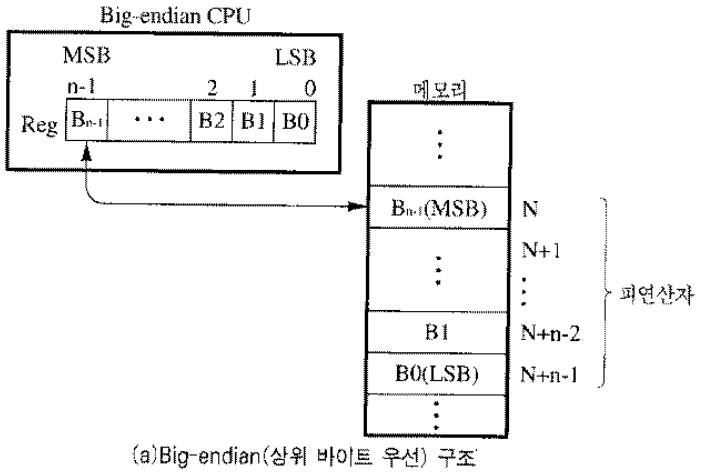
Slide 0

2장. 마이크로프로세서의 버스 전달

데이터 순서 및 정렬

Slide 1

- Big-Endian 순서와 Little-Endian 순서
  - Big-Endian 프로세서 (그림 2.1 (a))



(a)Big-endian(상위 바이트 우선) 구조

데이터 순서 및 정렬 (Cont'd)

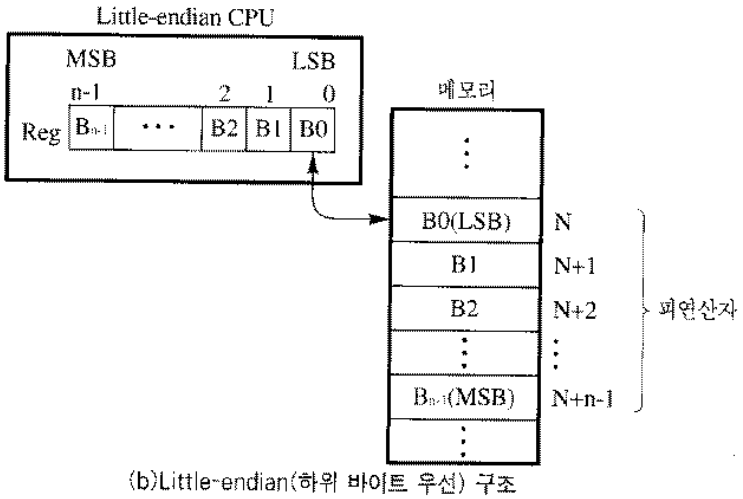
Slide 2

* high order byte first 구조
* 가장 낮은 주소에서 읽은 것이 피연산자의 최대 유효 byte 가됨
* Motorola 계열 및 SPARC RISC 형 마이크로 프로세서
* IBM 370 방식

데이터 순서 및 정렬 (Cont'd)

Slide 3

– Little-Endian 프로세서 (그림 2.1 (b))



(b)Little-endian(하위 바이트 우선) 구조

데이터 순서 및 정렬 (Cont'd)

Slide 4

  * low order byte first 구조
  * 가장 낮은 주소에서 읽은 것이 피연산자의 최소 유효 byte 가됨
  * Intel 계열
  * DEC VAX 방식
- 대부분의 RISC 프로세서
  * 초기화 시 순서 모드를 세팅할수 있음.

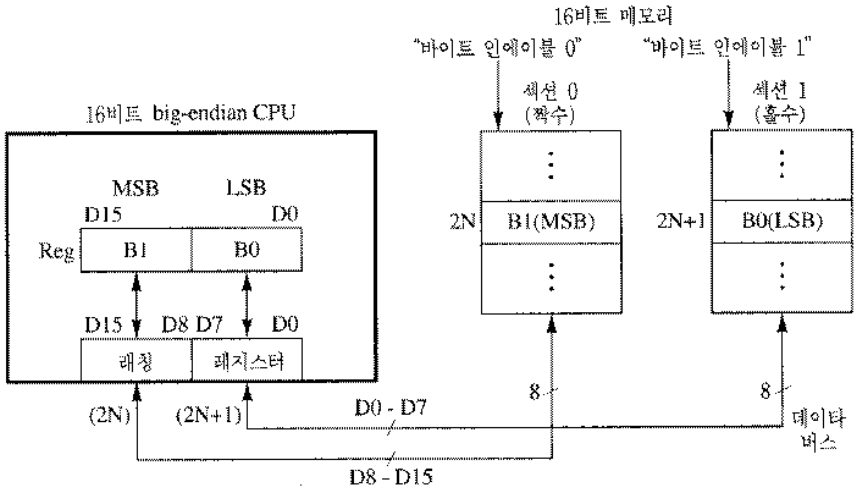
데이터 순서 및 정렬 (Cont'd)

Slide 5

- 16 비트 및 32 비트 마이크로 프로세서
  - 16 bit 제품 (그림 2.2)
    * Motorola 68000 에서 byte 지정 UDS#, LDS#
    * Intel 8086/286 에서 byte 지정 BHE#, A0
    * 읽기 : 모든 16 bit 가 읽힘 (CPU 가 알아서 처리)
    * 쓰기 : 해당 byte 만 enable 시킴 (Memory Enable 로 사용)

데이터 순서 및 정렬 (Cont'd)

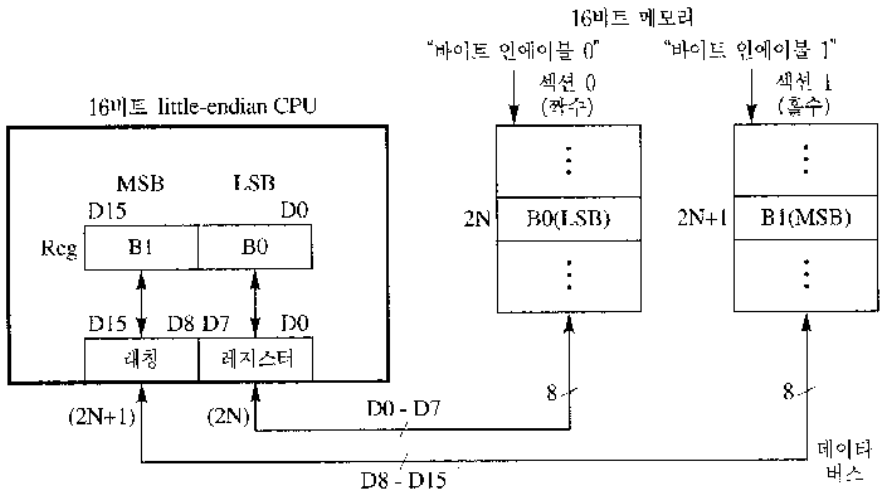
Slide 6



(a) 16비트 big-endian의 16비트 읽기

데이터 순서 및 정렬 (Cont'd)

Slide 7



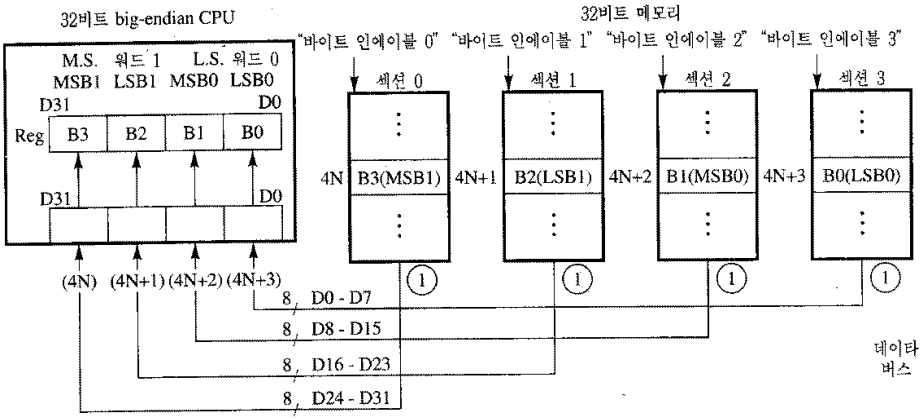
(b) 16비트 little-endian의 16비트 읽기

데이터 순서 및 정렬 (Cont'd)

Slide 8

– 32 bit 제품

∗ big-endian 32 bit 적재 (그림 2.3 a)

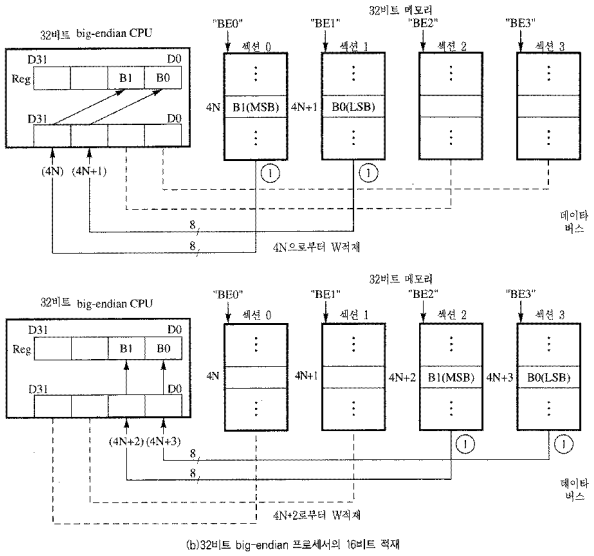


(a)32비트 big-endian 프로세서의 32비트 적재(4N으로부터 DW를 적재)

데이터 순서 및 정렬 (Cont'd)

Slide 9

∗ big-endian 16 bit 적재 (그림 2.3 b)

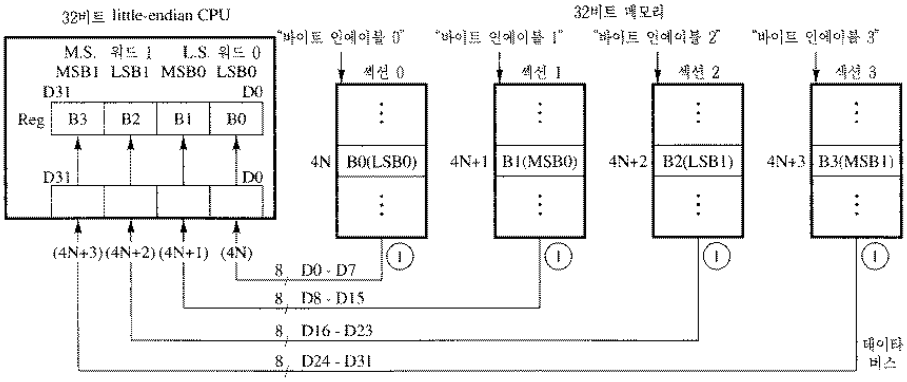


(b)32비트 big-endian 프로세서의 16비트 적재

데이터 순서 및 정렬 (Cont'd)

Slide 10

* little-endian 32 bit 적재 (그림 2.4 a)

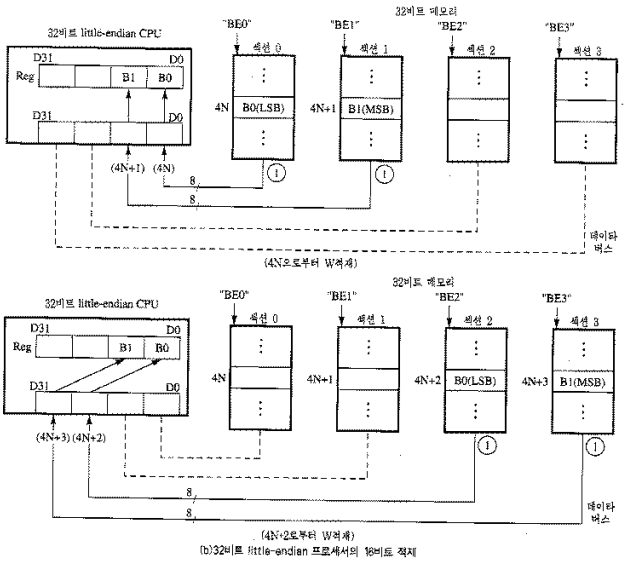


(a)32비트 little-endian 프로세서의 32비트 적재(4N으로부터 DW를 적재)

데이터 순서 및 정렬 (Cont'd)

Slide 11

* little-endian 16 bit 적재 (그림 2.4 b)

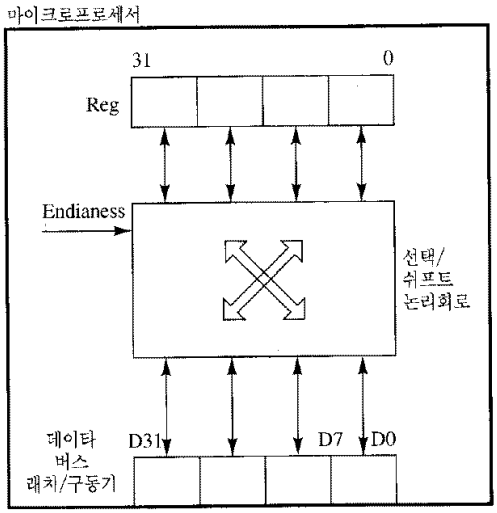


(b)32비트 little-endian 프로세서의 16비트 적재

데이터 순서 및 정렬 (Cont'd)

Slide 12

- 재구성 가능 RISC 마이크로 프로세서 (그림 2.5)



(a)내부 데이타 정렬의 단순화된 구성

데이터 순서 및 정렬 (Cont'd)

Slide 13

- 피연산자 및 명령어의 정렬
  - 최대의 성능을 얻기위해서는 메모리에 저정된 데이터는 정렬 되어있어야함
  - 정렬
    - 16비트 데이터 : A0 = 0 (2N)
    - 32비트 데이터 : A0 = A1 = 0 (4N)
    - 64비트 데이터 : A0 = A1 = A2 = 0 (8N)

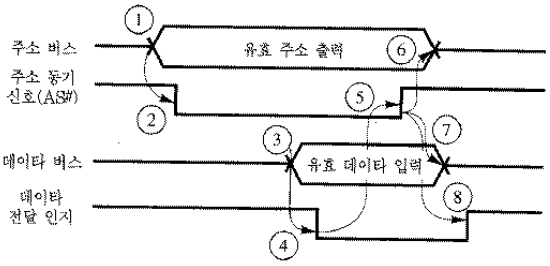
정렬된 버스 전달

Slide 14

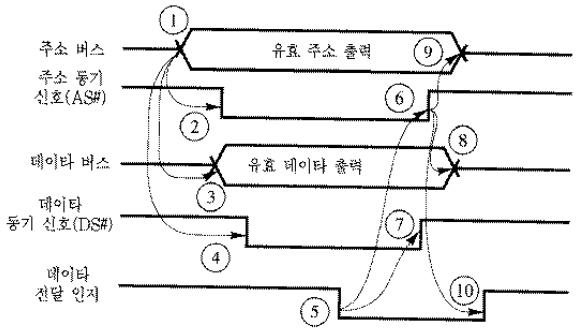
- 동기식/비동기식 버스
  - 동기식 버스 : Intel 계열
    - 버스 master 와 버스 slave 가 동일 clock 사용
    - negative 나 positive edge 에서 동작
    - 가장 느린장치 (최악의 경우) 에 의하여 버스 사이클이 결정됨
    - 보완 : 대기 프로토콜 (wait protocol) 사용 (Ready 가 안들어오면 기다림)
  - 비동기식 버스 : Motorola 계열 (68040 제외)
    - 버스 master 와 버스 slave 가 다른 clock 사용
    - 데이터와 주소에 대하여 핸드쉐이크 (hand shake) 신호 사용 : address/data strobe 이용
    - DTACK 사용
    - 비동기식 읽기 쓰기 싸이클 (그림 2.6) 타이밍 그림 (그림 2.7)

정렬된 버스 전달 (Cont'd)

Slide 15



(a)읽기 사이클의 타이밍도



(b)쓰기 사이클의 타이밍도

정렬된 버스 전달 (Cont'd)

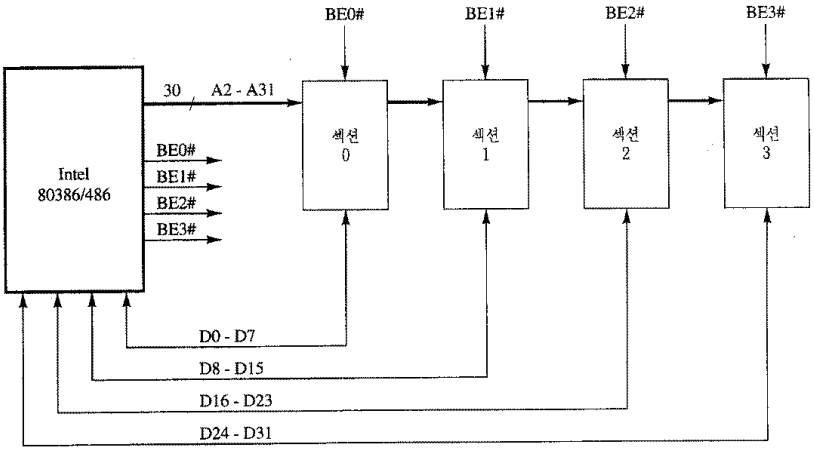
Slide 16

* 주소밖의 장치를 access 하면 감시타이머를 이용 버스 에러 (BERR#)을 발생
* 복잡한 버스 접속논리 회로, 다양한 속도의 장치 활용

정렬된 버스 전달 (Cont'd)

Slide 17

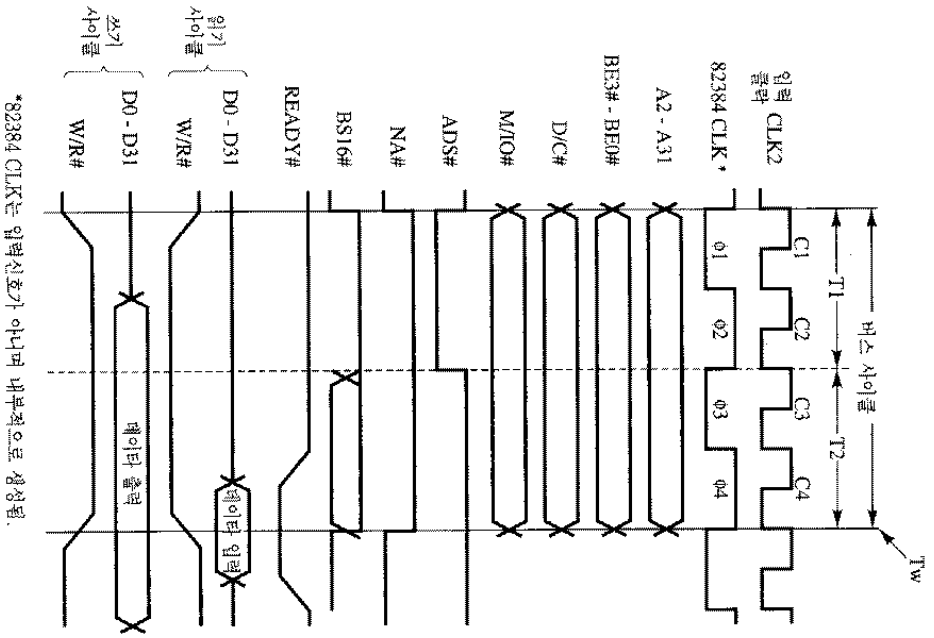
- Intel 80X86 의 버스 전달 (그림 2.8)



  - 80386 메모리 읽기/쓰기 싸이클 버스 타이밍 도 (그림 2.9), 486 (그림 2.10)

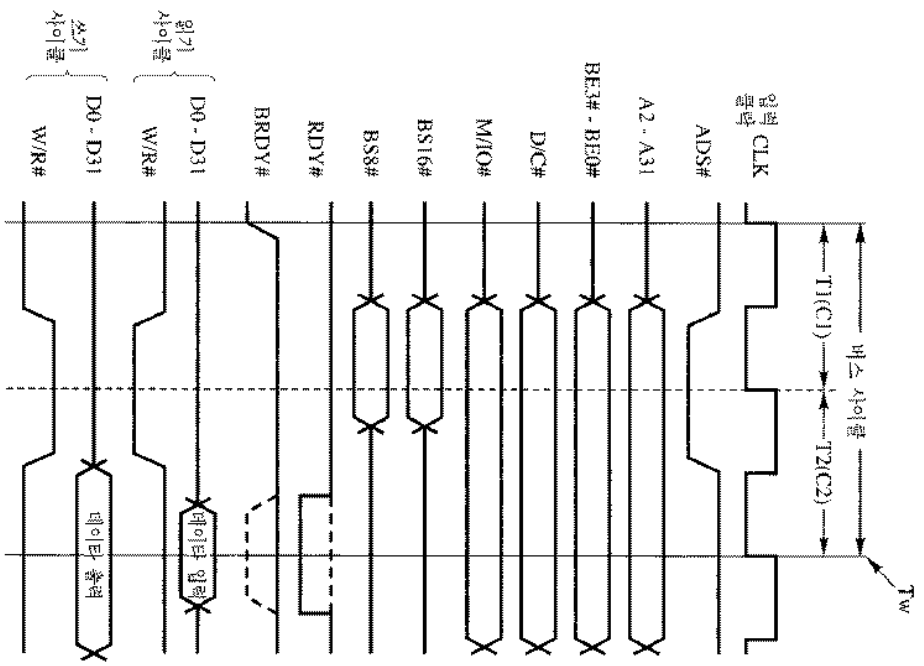
정렬된 버스 전달 (Cont'd)

Slide 18



정렬된 버스 전달 (Cont'd)

Slide 19



정렬된 버스 전달 (Cont'd)

Slide 20

- 80386 주소 파이프 라인 사용 가능 (NA# 이용), BS16# 만 있음 (16비트 장치만 사용가능)
- 80486 버스트 전달 (burst transfer), BS16#, BS8# 이용
- 80386 의 데이터 복사 : BS16# 이 enalbe 되어 16비트 모드로 동작시 그림 2.11 과 같이 데이터 복사 →그림 2.29 참조

정렬된 버스 전달 (Cont'd)

Slide 21

	80386바이트 인에이블								자동 복사
	BE3#	BE2#	BE1#	BE0#	D24-D31	D16-D23	D8-D15	D0-D7	
바이트 쓰기	High	High	High	Low	Undef.	Undef.	Undef.	B0	No
	High	High	Low	High	Undef.	Undef.	B0	Undef.	No
	High	Low	High	High	Undef.	B0	Undef.	B0	Yes
	Low	High	High	High	B0	Undef.	B0	Undef.	Yes
2바이트 쓰기	High	High	Low	Low	Undef.	Undef.	B1	B0	No
	High	Low	Low	High	Undef.	B1	B0	Undef.	No
	Low	Low	High	High	B1	B0	B1	B0	Yes
3바이트 쓰기	High	Low	Low	Low	Undef.	B2	B1	B0	No
	Low	Low	Low	High	B2	B1	B0	Undef.	No
4바이트 쓰기	Low	Low	Low	Low	B3	B2	B1	B0	No

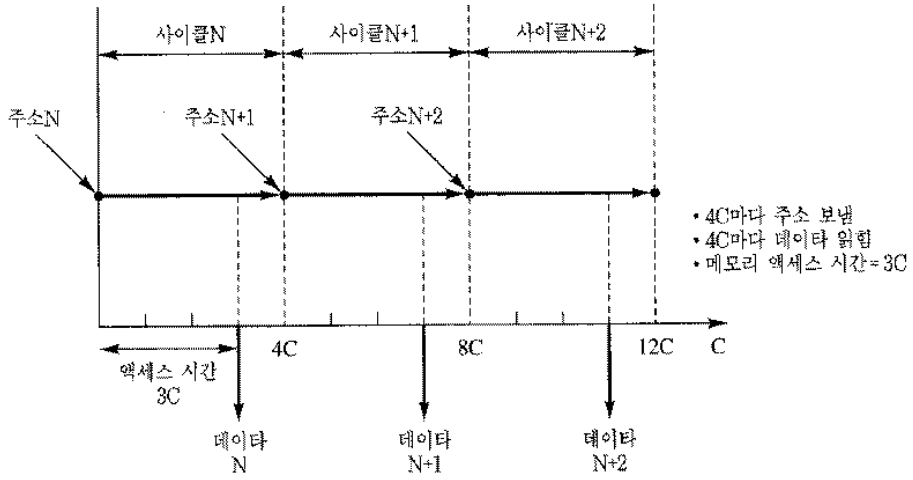
[a] 피연산자:
32비트 피연산자: B3, B2, B1, B0
16비트 피연산자: B1, B0
8비트 피연산자: B0
(B0 = 최초 유효 바이트)

- 80486 은 데이터 복사를 하지 않고 바이트 교환 (byte swap) 논리회로 사용 (2.6.2 절에서 설명) →그림 2.34 참조

다른 버스 동작 모드들

Slide 22

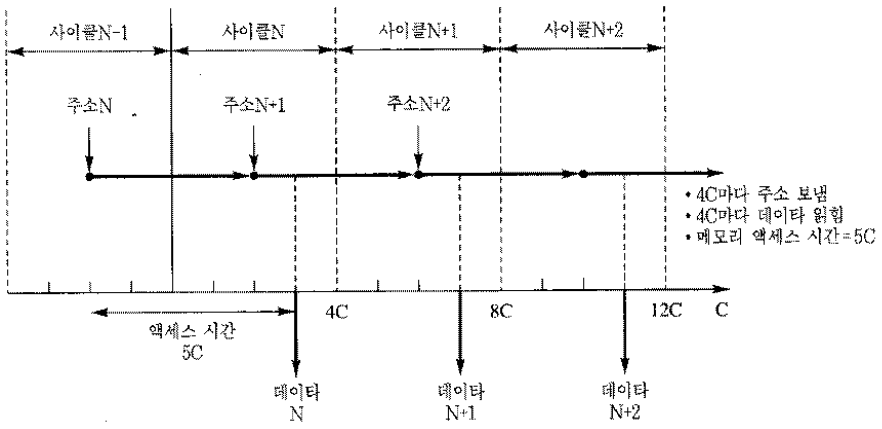
- 주소 파이프 라이닝 (그림 2.16)



(a)비파이프라인 버스동작

다른 버스 동작 모드들 (Cont'd)

Slide 23



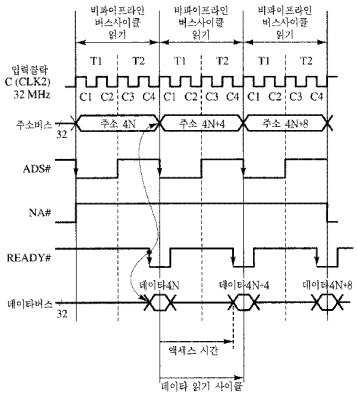
(b)파이프라인 버스동작

- 기본원리 : 현재의 버스 싸이클이 끝나지 않은 상태에서 다음 주소를 내보내어 버스싸이클을 시작한다.
- 장점 : 느린 장치에서 데이터를 빠르게 읽을수 있다.

다른 버스 동작 모드들 (Cont'd)

Slide 24

- 예 1) Intel 80386
  - NA# (Next Address) 입력핀이 active 되면 주소파이프라인 실행 (그림 2.17)

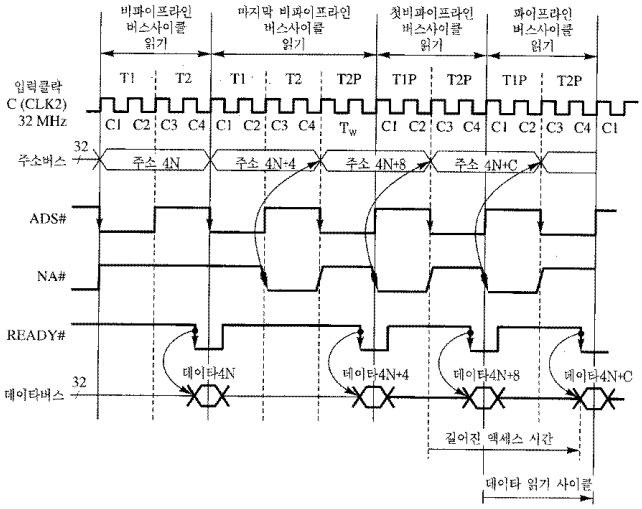


• 버스사이클 시간 = 4C = 125ns
• 데이타읽기 사이클 시간 = 4C = 125ns
• 데이타 전달속도 = 32M바이트/초
• 액세스 시간 = 3.5C = 109.375ns

(a)비파이프라인 읽기 사이클

다른 버스 동작 모드들 (Cont'd)

Slide 25



파이프라인 버스사이클 시간 = 4C = 125ns
파이프라인 데이타 읽기 사이클 시간 = 4C = 125ns
파이프라인 데이타 전달속도 = 32M바이트/초
파이프라인 액세스 시간 = 5.5C = 171.875ns

(b)파이프라인 읽기 사이클

다른 버스 동작 모드들 (Cont'd)

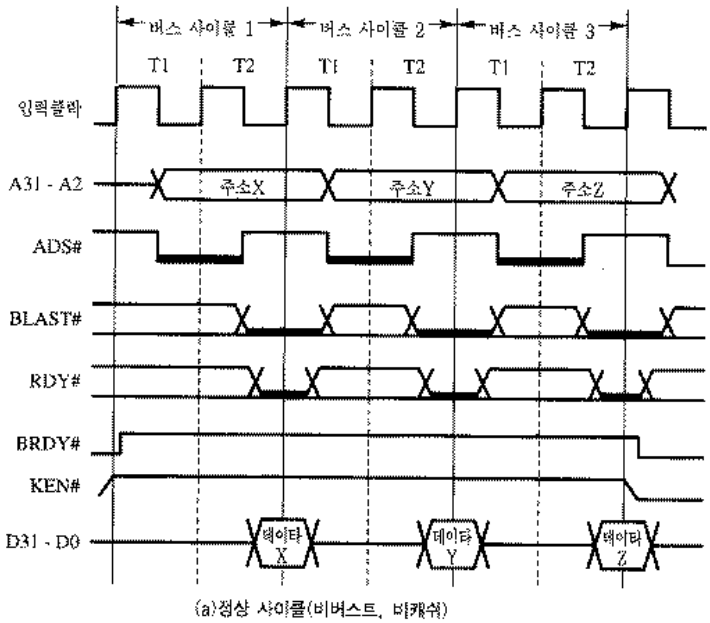
Slide 26

- 1단계 주소 파이프라인 (하나의 버스 싸이클에서 한 개의 다음 주소만 가능)
- I860 : 2단계 주소 파이프라인 가능

- 버스트 전달
  - 하나의 주소를 내보내고 연속되는 다수의 이중워드 (32 bit) 를 인출
  - 보통 캐쉬를 채울 때 사용
  - 80486 의 버스트 전달 (주소파이프라인 (80386) 은 사용하지 않음)(그림 2.18)

다른 버스 동작 모드들 (Cont'd)

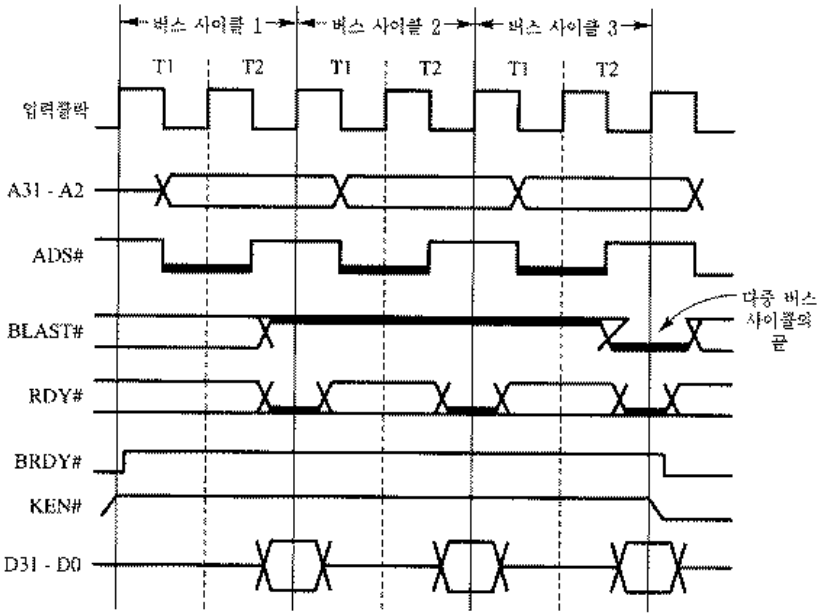
Slide 27



(a)정상 사이클(비버스트, 비캐쉬)

다른 버스 동작 모드들 (Cont'd)

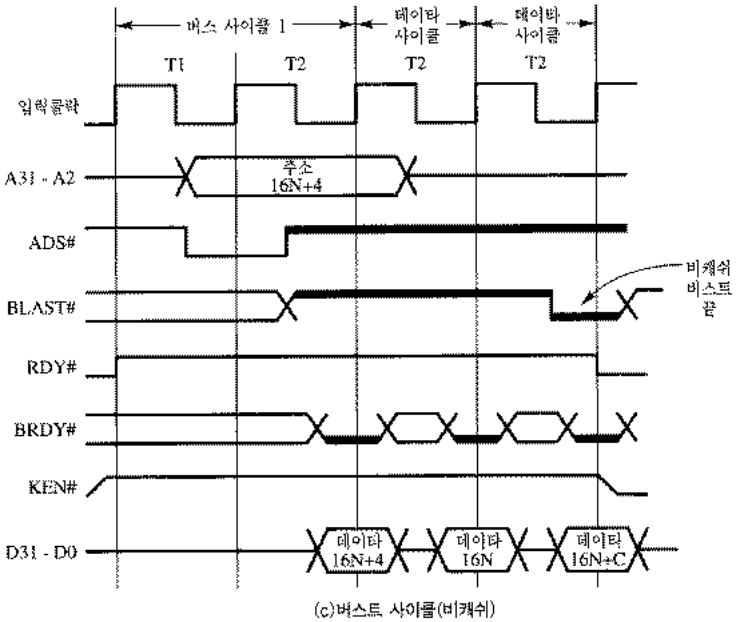
Slide 28



(b)다중 버스 사이클(비버스트, 비캐쉬)

다른 버스 동작 모드들 (Cont'd)

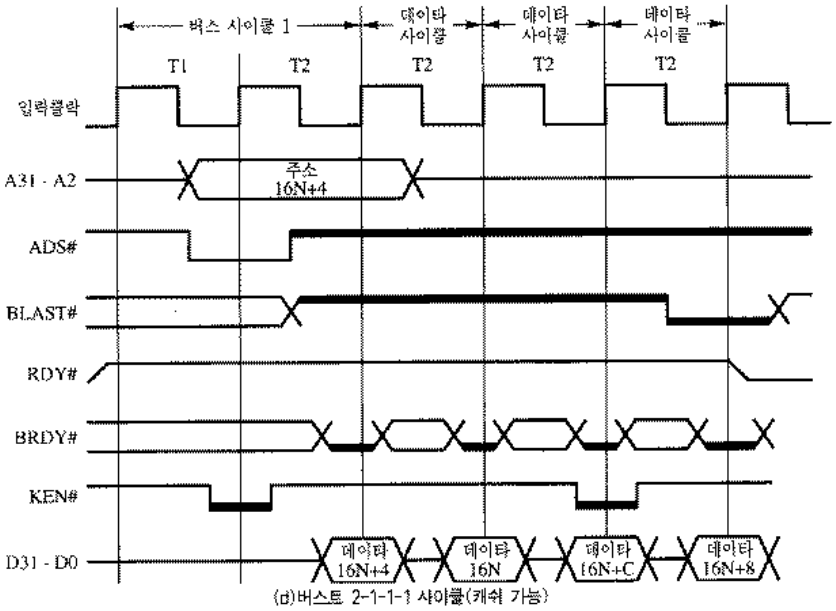
Slide 29



(c)버스트 사이클(비캐쉬)

다른 버스 동작 모드들 (Cont'd)

Slide 30



(d)버스트 2-1-1-1 사이클(캐쉬 가능)

다른 버스 동작 모드들 (Cont'd)

Slide 31

* 다중버스싸이클은 외부 시스템이 버스트 방식을 지원하면 버스트 싸이클로 전환될수 있음
* 버스트 싸이클은 : BRDY# 에의하여 ready 를 보냄, RDY# 이 먼저 enable 되면 버스트 모드 동작 안함
* BLAST# 은 버스트 싸이클의 종료를 나타냄
* 버스트 모드에서는 외부회로나 메모리자체에서 연속되는 주소 생성 (memory 의 nibble 모드)
* 첫 번째 버스트 access 2 clock 필요, 연속되는 access 1 clock 필요
* 버스트 모드는 보통 읽기 동작을 의미
* 80486 은 32 비트 버스 쓰기는 지원안함, 16 비트 포트나 8비트 포트에서 32 비트 버스트 쓰기 지원
* 버스트 모드에서 캐쉬채우기 순서 (80486 예)
  →메모리 인터리빙에의하여 결정됨 (그림 2.19)
  →인터리브 되지 않은 메모리에서도 동일 순서로 인출 해야함
  →버스트/비버스트, 캐쉬/비캐쉬 사이클에 대한 규칙 (그림 2.20)

다른 버스 동작 모드들 (Cont'd)

Slide 32

	비버스트	버스트
비캐쉬 방식	정상적 버스 사이클	· 주소 순서가 적용됨(외부 하드웨어가 관리함) · 80486은 전달을 완료시키는 데에 필요한 수의 바이트만을 버스트 시킴.
캐쉬 방식 (캐쉬 라인 채우기)	· 첫 사이클에 BE3#-BE0#은 캐쉬 가능 메모리에 외부 하드웨어에 의하여 해석되지 않음 (모두 활성으로 가정됨) · 항상 16 바이트 캐쉬 라인을 인출함 · 주소 순서가 적용됨 (프로세서가 관리함)	· BE3#-BE0#은 첫 사이클 동안 캐쉬 가능 메모리에 해석되지 않음 · 항상 16 바이트 캐쉬 라인을 인출함 · 주소 순서가 적용됨(메모리 접속 하드웨어가 관리함

다른 버스 동작 모드들 (Cont'd)

Slide 33

→동일칩 캐쉬 (on-chip cache) 를 채우기위한 기능 (그림 2.21)

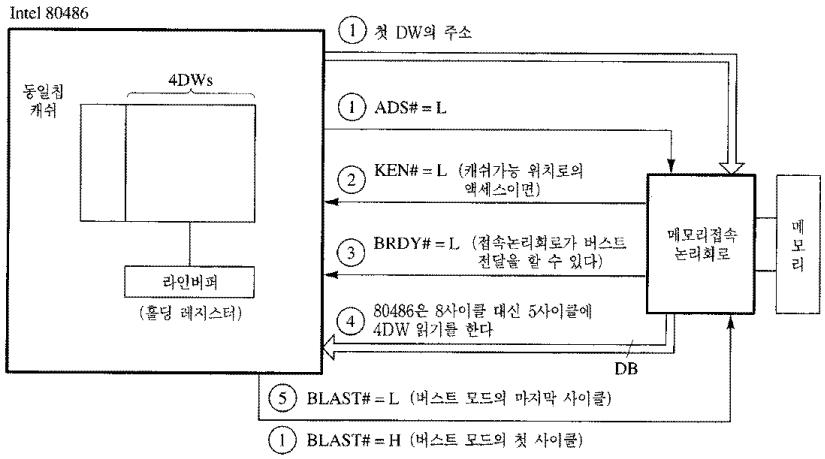


그림 2.21 Intel 80486의 동일칩 캐쉬의 라인을 채우기 위한 버스트 전달 모드의 기능 설명.

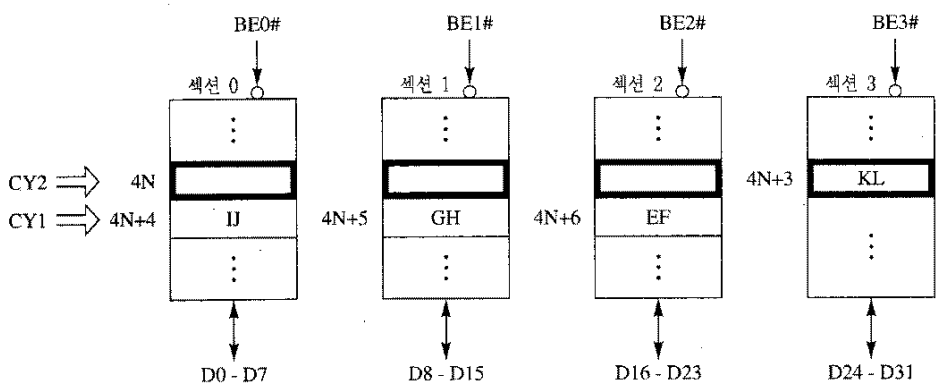
오정렬된 데이터의 전달

Slide 34

- 정렬 적용 프로세서
  - 모든 명령어나 데이터는 정렬되어야한다.
  - 정렬되지 않으면 트랩이 일어남 →소프트웨어로 처리 →비효율적
  - 예) 68000, MIPS R-series, Intel I860
- 비정렬 허용 프로세서
  - 마이크로 프로세서가 자동으로 알아서 처리
  - 대부분의 프로세서
  - 80486 CR0 (Control Register 0)에 정렬 제한 적용 여부 결정 비트가 있음, 0 이면 제한 적용안함 (386과 호환), 1이면 인터럽트 발생
- 오정렬된 데이터를 전달할 다중버스 싸이클은 endian 에 상관없이 항상 최대 유효 바이트를 먼저인출
- Intel 의 오정렬 데이터 전달 (그림 2.24, 2.25)

오정렬된 데이터의 전달 (Cont'd)

Slide 35



(a)4N+3에 오정렬된 이중워드

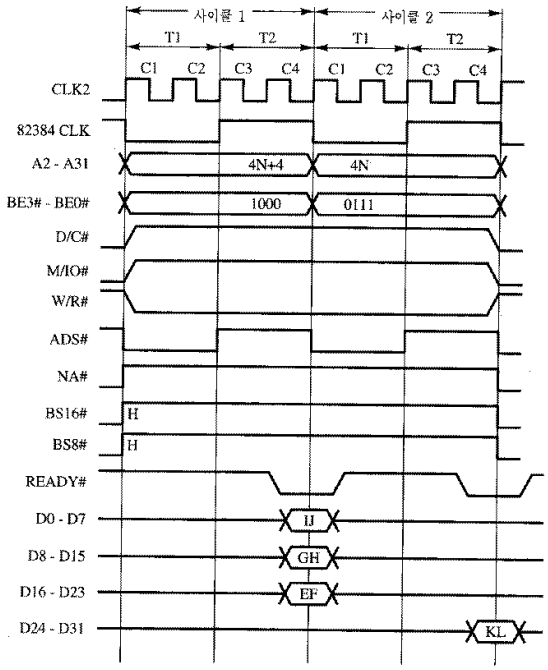
CY1 = 사이클1:
- $A_2 - A_{31} = 4N + 4$
- BE0# - BE3# = 0001
- D0 - D7 ← IJ, D8 - D15 ← GH, D16 - D24 ← EF

CY2 = 사이클2:
- $A_2 - A_{31} = 4N$
- BE0# - BE3# = 1110
- D24 - D31 ← KL

(b)각 버스 사이클 동안 버스선의 값들

오정렬된 데이터의 전달 (Cont'd)

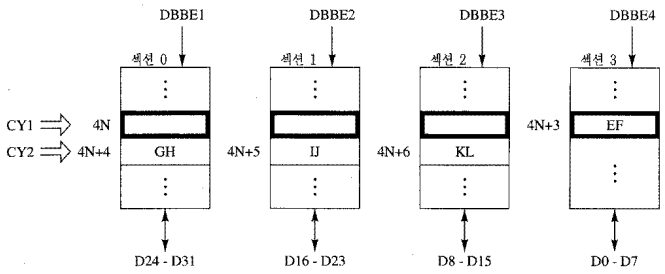
Slide 36



오정렬된 데이터의 전달 (Cont'd)

Slide 37

- Motorola 의 오정렬 데이터 전달 (그림 2.26, 2.27)



(a)4N+3에 오정렬된 이중워드

CY1 = 사이클1:
- $A_2 - A_{31} = 4N$
- DBBE4 = H
- 크기 = DW
- $D_0 - D_7 \leftarrow EF$

CY2 = 사이클2:
- $A_2 - A_{31} = 4N + 4$
- DBBE1 = DBBE2 = DBBE3 = H
- 크기 = 3B
- $D_{24} - D_{31} \leftarrow GH$
- $D_{16} - D_{23} \leftarrow IJ$
- $D_8 - D_{15} \leftarrow KL$

(b)각 버스 사이클 동안 버스선의 값들

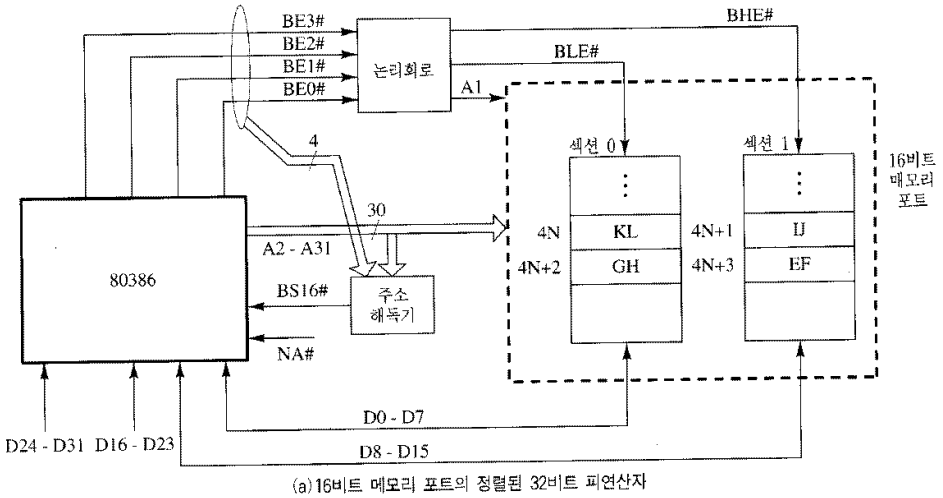
동적 버스 크기 조정

Slide 38

- 버스 슬레이브 device 의 버스 크기에 동적으로 맞춤
- 마이크로 프로세서 (피연산자의 크기 표시자 출력) slave device (포트크기 표시자 로 응답)
  - Motorola 계열 : SIZ1, SIZ0 보냄 DSACK1#, DSACK0# 이 입력됨
  - Intel 계열 : 피연산자크기 표현 없음 BS16#, BS8# 이 입력
- 좁은 포트의 연결
  - Motorola 계열 : 상위데이타 버스선에 연결
  - Intel 계열 : 하위데이타 버스선에 연결
- 80386의 예 (그림 2.29, 2.30, 2.31) 16비트 device 만연결 가능

동적 버스 크기 조정 (Cont'd)

Slide 39



(a) 16비트 메모리 포트의 정렬된 32비트 피연산자

동적 버스 크기 조정 (Cont'd)

Slide 40

i386 과 i486 CPU 신호				8-, 16-비트 버스 신호[a]			비고
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE#(A0)	
H*	H*	H*	H*	X	X	X	X — 활성화된 바이트 없음
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	X	X	X	X — 인접한 바이트 아님
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	X	X	X	X — 인접한 바이트 아님
L*	H*	L*	H*	X	X	X	X — 인접한 바이트 아님
L*	H*	L*	L*	X	X	X	X — 인접한 바이트 아님
L	L	H	H	H	L	L	
L*	L*	H*	L*	X	X	X	X — 인접한 바이트 아님
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

동적 버스 크기 조정 (Cont'd)

Slide 41

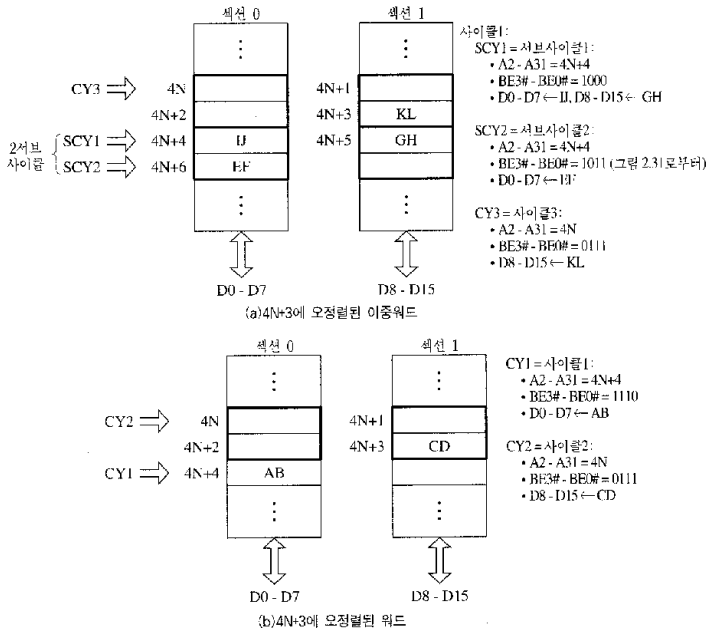
"현재의 서브 사이클"				BS8#인 "다음 서브 사이클"				BS16#인 "다음 서브 사이클"			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

주 : "n"은 요청을 만족시키기 위한 다른 버스사이클이 필요하지 않음을 뜻한다.

- 80386의 16비트 포트 오정렬 데이터 읽기 규칙 (그림 2.32, 2.33)
  - * 이중워드의 경계를 넘으면 →최대 유효 부분을 먼저
  - * 이중워드 내에서는 →최소 유효부분 먼저

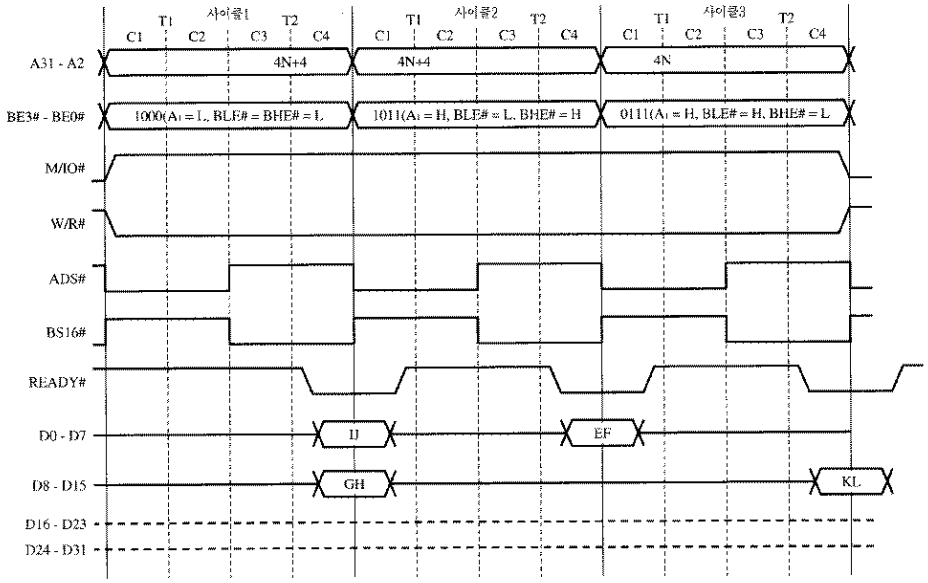
동적 버스 크기 조정 (Cont'd)

Slide 42



동적 버스 크기 조정 (Cont'd)

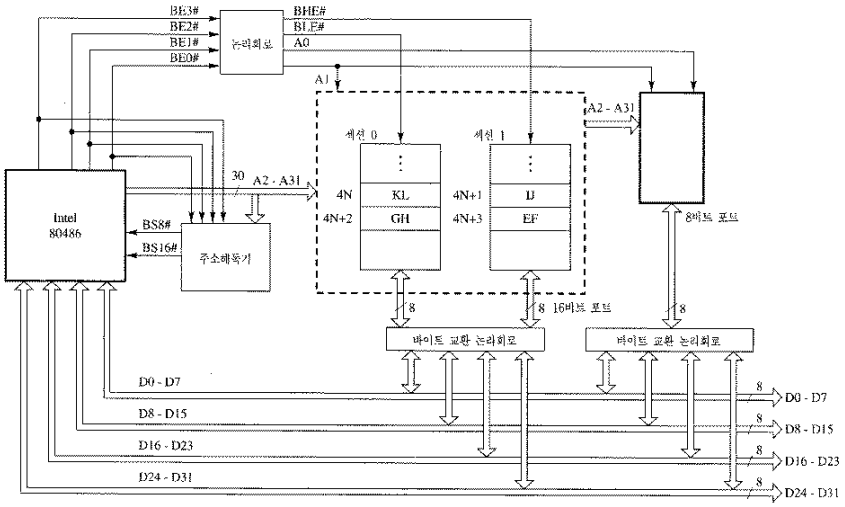
Slide 43



동적 버스 크기 조정 (Cont'd)

Slide 44

- 80486의 예 (그림 2.34, 2.35) 8비트 16비트 가능



(a)80486을 16비트 및 8비트 포트에 접속하기

동적 버스 크기 조정 (Cont'd)

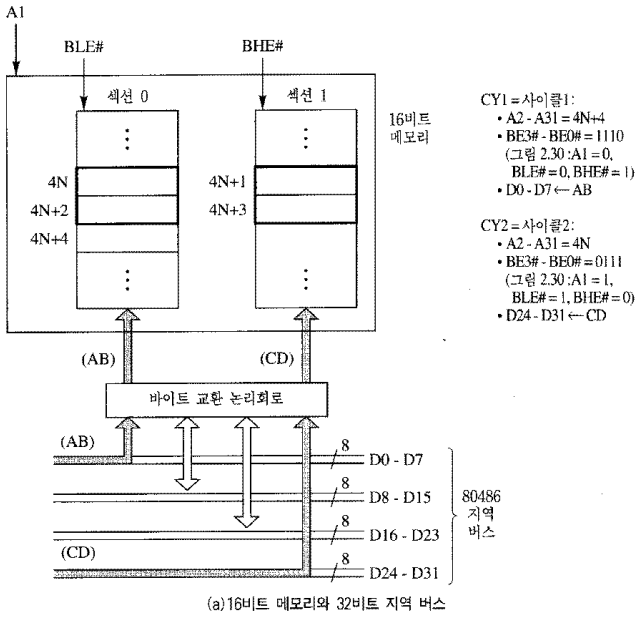
Slide 45

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	w BS16#
1	1	1	0	D7 - D0	D7 - D0	D7 - D0
1	1	0	0	D15 - D0	D7 - D0	D15 - D0
1	0	0	0	D23 - D0	D7 - D0	D15 - D0
0	0	0	0	D31 - D0	D7 - D0	D15 - D0
1	1	0	1	D15 - D8	D15 - D8	D15 - D8
1	0	0	1	D23 - D8	D15 - D8	D15 - D8
0	0	0	1	D31 - D8	D15 - D8	D15 - D8
1	0	1	1	D23 - D16	D23 - D16	D23 - D16
0	0	1	1	D31 - D16	D23 - D16	D31 - D16
0	1	1	1	D31 - D24	D31 - D24	D31 - D24

(b)여러 버스크기로 읽히는 데이타 핀들(8)

동적 버스 크기 조정 (Cont'd)

Slide 46



(a) 16비트 메모리와 32비트 지역 버스

지역버스 중재

Slide 47

- 버스 중재핀 (그림 2.44)

Intel 8086/286/386	Motorola 680 × 0	Intel 80486	Intel i860
HOLD(입력)	$\overline{BR}$(입력)	HOLD(입력)	HOLD(입력)
HLDA(출력)	$\overline{BG}$(출력)	HLDA(출력)	HOLDA(출력)
최대모드(8086)	$\overline{BGACK}$(입력)	BOFF#(입력)	BREQ(출력)
$\overline{RQ}/\overline{GT_0}$(양방향)		BREQ(출력)	
$\overline{RQ}/\overline{GT_1}$(양방향)			

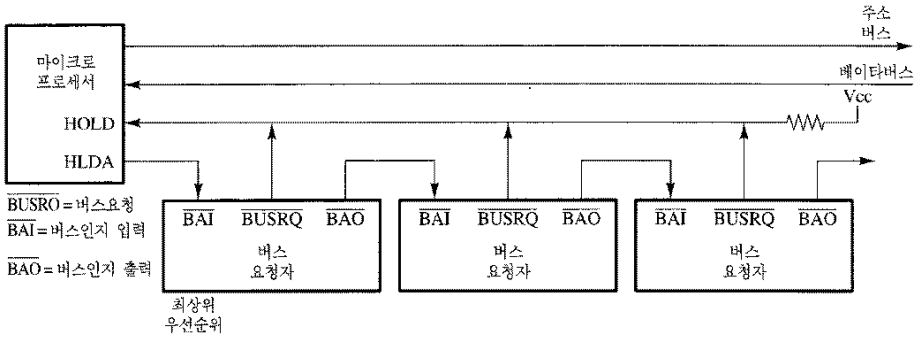
  - HOLD
    * 입력선 다른 버스 마스터가 이 선을 통하여 버스 요구
    * 소프트웨어적으로 마스킹 가능
    * 버스제어권을 넘기면 모든 버스 출력핀을 고임피던스 상태로 만듬

지역버스 중재 (Cont'd)

Slide 48

- HLDA
  - 버스 제어권을 넘김을 다른 버스 마스터에게 알림
  - 인터럽트 보다 높은 우선순위

- 직렬 중재 기법 (그림 2.45)



- 여러개의 버스 매스터가 지역버스에 연결되어 있을 때 중재 기법