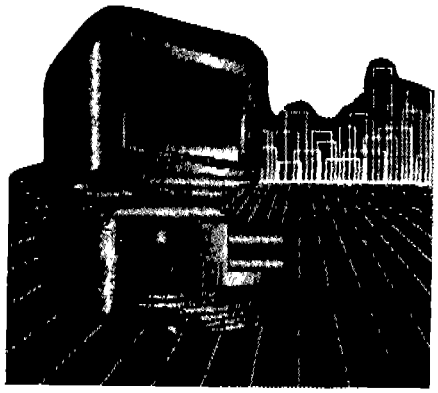
컴퓨터 메모리 구성요소의 동향



2

공학박사 이 근 철

제일전산훈련원 원장

3·1 속도 증가를 위한 실용적인 방법

캐시 시스템은 DRAM의 속도문제를 해결하기 위해 사용되는 가장 확실한 방법이긴 하지만 회로가 너무 복잡해지는 반면 그에 따른 높은 가격 때문에 쉽게 사용할 수 없으며 286을 사용하는 일반 AT에서도 사용하기 어려운 방법이다.

이런 이유로 캐시를 사용하지 않으면서 시스템의 성능을 가능한 한 높일 수 있고 비용이 저렴하게 드는 방법이 필요하다.

AT급 이상의 컴퓨터에서는 롬 바이어스에 비디오 카드나 디스크의 타입, 메모리 사이즈 등을 지정해 주는 CMOS 세트업을 하게 되는데 최근에는 CMOS 세트업 이외에도 XCMOS (Extended CMOS)세트업이라는 것을 할 수 있는 PC가 제공되고 있다.

이는 CMOS 세트업에서의 표준 세트업외에 메모리구성, 메모리 속도 등도 세트업 프로그램을 통해서 제어할 수 있게 한 것으로 메모리 인터리브(Memory Interleaving) 섀도 램(Shadow RAM), 페이지 모드 램(Page-mode RAM) 등의 방식이 있다.

3·2 메모리 인터리브

메모리 인터리브(Memory Interleaving)는 보통의 DRAM을 사용하면서도 대기상태 0를 부분적으로나마 이용할 수 있게 한 기법이다. 즉 메모리를 몇 개의 뱅크로 나누어 놓고 CPU가 한 메모리 뱅크를 액세스 할 동안 다른 뱅크는 리프레시를 하게 만드는 것이다.

실제로 대개의 프로그램은 명령어 인출(Instruction Fetch)를 할 때나 스택(Stack)과 배열(Array)를 사용할 때 메모리의 연속된 번지를 액세스하는 것이 보통이다.

여기에 두 개의 뱅크 구조를 갈라진 메모리로 사용하는데 한 뱅크에는 홀수(Odd) 번지의 데이터가, 그리고 다른 한 뱅크에는 짝수(Even) 번지의 데이터가 들어간다. CPU가 일정하게 연속된 번지를 액세스한다면 두 뱅크는 각각 교대로 사용되는 것이다.

이런 식으로 계속 사용하는 한 CPU는 메모리

를 대기상태 0에서 액세스할 수 있다. 하지만 CPU가 짝수번이나 홀수번을 연속해서 액세스할 때에는 대기상태 2가 된다.

이러한 인터리브 기법은 비용을 적게 들이면서도 느린 속도의 DRAM을 대기상태 0으로 사용할 수 있지만 캐시 시스템만큼의 성능은 기대할 수 없다.

또 프로그램이 실행되는 데 있어서 대기상태 0과 대기상태 2가 섞여서 사용된다는 것도 그다지 바람직하지 않은 일이다.

그리고 인터리브 기법을 사용하기 위해서는 기본 메모리 사이즈가 일반 시스템의 두 배가 되는 것도 단점이 된다.

하지만 그럼에도 불구하고 SRAM이 사용되는 값비싼 캐시 시스템에 비해 제어회로가 비교적 간단하다는 장점 때문에 저가의 386 시스템 등에 많이 사용되고 있다.

인터리브 기법을 사용할 때와 사용하지 않을 때의 속도를 비교해 보면 대략 15~20% 정도 차이가 난다.

4. 페이지 모드 램(Page-mode RAM)

이 기법은 앞에서의 인터리브 방식과는 달리 일반적인 DRAM을 아무 것이나 다 쓸 수 있는 것이 아니다. 반드시 페이지 모드의 액세스가 지원되는 DRAM을 사용해야 한다.

페이지 모드가 지원되는 DRAM은 지원되지 않는 것보다 가격이 조금 더 비싸지만 인터리브 방식보다는 성능면에서 낫기 때문에 충분히 사용할만한 가치가 있는 기법이라 할 수 있겠다.

가장 널리 사용되는 256K DRAM은 총 256K 비트의 용량을 가지고 있는 메모리를 말한다.

칩의 외형을 보면 16개 핀이 있는데 데이터 입력핀 데이터 출력핀 RAS(Row Address Select) 핀 CAS(Column Address Select) 핀 전원 및 그라운드핀이 필요한데, 칩에는 그 절반인 9비트의 핀밖에 없다.

이것은 내부의 메모리 구조가 행렬구조로 되어 있어서 실제로 액세스할 때는 열(Column) 어드레스를 먼저 인가해 주고 난 다음에 행(Row)의 어드레스를 연속적으로 주기 때문이다.

이때 동일한 열 어드레스를 가진 정보들은 같은 한 페이지에 있다고 말하며 256K DRAM에서의 한 페이지는 2K 바이트의 크기를 가진다. 이러한 메모리에서 페이지 모드로 액세스가 이루어질 때는 CAS 신호가 매번 뜰 필요 없이 RAS 신호만 연속적으로 뜨며 그 각 시점마다 행 어드레스만 변경되는 것이다.

보통의 메모리 읽기 동작은 우선 8개의 행 어드레스가 인가되고 잠시 후에 8개의 열 어드레스가 인가된 다음에 데이터가 출력되는 동작이 반복되는데 비해 페이지 모드가 지원되는 DRAM에서는 동일한 행 주소(즉, 동일한 페이지)내에서 데이터가 액세스 된다고 하면 행 주소를 고정시켜 놓고 열의 주소만 변화시키면서 데이터를 액세스 하기 때문에 그 페이지 안에서의 메모리 액세스 속도는 보통의 메모리 액세스 시간보다 2배 가까이 빨라지게 되는 것이다.

이것은 물론 하나의 페이지, 즉 행 주소가 고정된 상태에서 가능하고 행 주소가 변경되는 경우에는 다시 대기상태를 사용하는 메모리 액세스 방법으로 돌아와야 한다.

이러한 페이지 모드 기법의 경우에도 앞서의 인터리브 방식처럼 그때마다 대기상태의 수가 달라지는 단점이 있다. 또한 다소 복잡한 제어회로가 필요한 점도 단점이 된다.

페이지 모드 기법을 사용했을 때는 사용하지 않은 시스템 보다 약 7%의 성능향상이 이루어진다.

5. 섀도 램(Shadow RAM)

최근에 판매되는 AT 또는 386 컴퓨터들은 처음 전원을 켜거나 리세트 시킨 직후에 우선 어느

회사의 롬 바이어스라는 메시지가 나타나고 나서 그 아래에 'Shadow RAM Enabled' 등의 메시지가 나오는 경우가 많다.

섀도 램이라는 기법은 메인 메모리와는 관련이 없이 바이어스가 저장되어 있는 롬 칩의 속도 문제를 해결하기 위해 사용된다. 일반적으로 바이어스 루틴이 기록되어 있는 EPROM(Erasable Programmerable ROM)은 속도가 상당히 느린 것이 보통이다.

보통 200ns 또는 그 이상의 액세스 시간을 가진 것을 사용하고 빨라도 150ns 이하의 액세스 타임을 가진 것은 거의 사용되지 않는다. 이것은 EPROM 자체의 특성 때문이기도 하고 한편으로 가격의 문제도 있기 때문이다.

이런 속도특성 때문에 DRAM으로 구성되어 있는 메인 메모리를 대기상태 0으로 액세스하는 시스템일지라도 롬 바이어스를 사용할 때는 두 번의 대기상태나 혹은 그보다 더 많은 대기상태가 발생하게 된다. 섀도 램 기법은 이처럼 롬의 속도가 느린 것을 해결하기 위해 사용된다.

일반적인 AT의 경우에 메모리는 1M 바이트(1024KB)를 장착하는 것이 거의 표준으로 되어 있다. 그런데 실제로 이 가운데서 도스 프로그램이 사용할 수 있는 영역은 640KB이고 나머지 384KB는 쓰지 않은 채 확장 메모리(Extended Memory) 전용으로 사용되어 왔다.

섀도 램의 아이디어는 바로 이 영역에 롬 바이어스의 내용을 복사한 다음 롬 바이어스 대신에 이 램 영역을 사용하게 만드는 것이다.

수행되는 프로그램은 자신이 롬 바이어스를 호출했을 때 롬 바이어스 대신 램이 사용되는지 전혀 모른다. 그러나 제어회로에 의해서 롬의 바이어스를 액세스하는 명령은 실제로 램에 복사된 바이어스를 읽는 것이다.

AT급 이상의 컴퓨터에는 대개 1MB의 기본 메모리 중에서 도스가 사용하는 640KB를 뺀 나머지 384KB를 확장 메모리(Extended Memory)로 사용할지 아니면 롬 바이어스 섀도 영역으로 사용할지를 사용자가 세트업을 통해 지정할 수 있게 만들어져 있다.

이것을 어떻게 사용할 것인지는 사용자가 선택할 일이지만 섀도 램을 선택하면 다음 두 가지의 사항을 고려해야 한다.

첫째로 섀도 램 사용시에는 우선 확장 메모리가 줄어든다. 1MB 메모리를 가진 시스템에서는 384KB만큼의 메모리가 줄어든 나머지가 확장 메모리로 사용되는 것이다.

둘째는 비디오 어댑터(그래픽 카드)의 롬 바이어스를 섀도하는 경우에 비디오 카드와 충돌하는 수가 있음을 알아 두어야 한다. 이것은 비디오 어댑터의 롬 바이어스 자신도 속도 증가를 위해서 확장 메모리상으로 재배치하는 경우가 있기 때문이다.

그러므로 이런 문제가 생길 경우에는 세트업을 맞춰줄 때 비디오 롬 바이어스의 섀도를 'Disable'로 해주어야 한다.

☆ ☆ ☆

다음에는 마이크로프로세서가 메모리로부터 명령어를 받는 과정과 또 다른 장치들에게 데이터를 주는 과정을 알아본다.

1. 시스템 버스

버스란 여러 장치들간에 공통적으로 접속되어 있는 전기적 통로를 말한다.

이 전기적 통로를 따라 전류가 흐르고 이것이 각 장치에 입력으로 작용되어 그 장치로 하여금 특정한 반응을 일으키게 한다.

간단히 전선으로 연결시켜 둔 것이라고 이해할 수 있다. 이 버스는 컴퓨터 시스템을 같이 묶

어 주는 접착제와 같은 기능을 하는데, 이러한 대표적인 예가 모든 마이크로 컴퓨터에 등장하는 시스템 버스이다.

여기서 마이크로 컴퓨터라고 함은 대략 PC급에서 소형 워크스테이션까지를 말한다.

그림 1에서 보면 여러 개의 버스를 사용하고 있는 시스템의 구조가 나타나 있다. CPU와 다른 주변장치들간에 버스로 연결되어 CPU가 내어 놓는 전기적 신호가 이 통로를 따라 전송될 것이다. 또한 CPU내에도 버스가 존재한다. 이를 두고 내부(Internal Bus)라고 부르기도 한다. 이는 CPU가 여러 개의 작은 CPU로 구성되어 있을 때 위에서 언급한 상황이 내부의 CPU간에 그대로 적용된다.

이러한 구조는 복잡하여 설계하기 어려운 대신 CPU의 성능을 놀라울만큼 개선시키고도 비용을 감소시키는 역할을 해왔다.

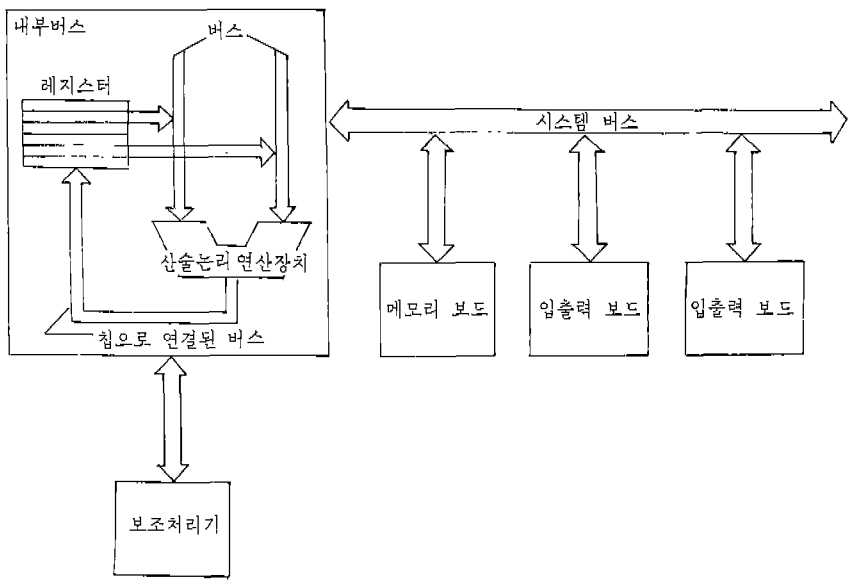
CPU는 시스템 버스를 통해 다른 장치들과 통신한다(그림에 있듯이 메모리 보드, 입출력 보드 등과 통신한다).

또 자신은 다른 프로세서와 통신하기 위해 지역 버스(Local Bus)를 두기도 한다. 여기서 장치(Device)라고 하는 것은 메모리처럼 수동적으로 단지 반응만 하는 시스템 구성요소(System Component)와 디스크 컨트롤러처럼 작지만 필요한 행동을 스스로 취할 수 있는 시스템 구성요소들을 말한다.

반면에 프로세서(Processor)라는 것은 지능적인(Intelligent) 장치로서 상황에 따라 행동을 달리하는 능동적인 시스템 구성요소를 의미한다.

CPU는 물론이고 수치연산 보조 프로세서(Math Coprocessor)라든지 다중 프로세서(Multi Procesor) 환경, 즉 CPU가 여러 개 있는 환경에서 각각의 프로세서가 이러한 예에 해당된다.

이들간에도 통신이 필요하다. 예를 들어 80286(주 프로세서)이 80287(보조 프로세서)이 있음을 알기 때문에 실수 연산작업을 80287에게 넘기려 할 때나 80287 작업을 끝낸 뒤 받으려 할 때에도 서로간의 통신을 통하여 작업의 완료



<그림 1> 여러 개의 버스를 사용한 컴퓨터 시스템

를 알게 된다.

실제로 여러 프로세서가 있는 시스템에서 프로세서들이 서로 잘 활용되고 있다면 디스크 컨트롤러 같은 장치와 CPU간의 데이터 전송보다는 많을 것이다

이런 이유로 지역 버스를 각 프로세서간에 두어 데이터의 송·수신에 속도를 올리자는 것이며 궁극적으로 이는 전체 시스템의 성능향상에 도움을 준다.

CPU와 여러 장치들간에 전송이 제대로 이루어지기 위해서는 서로 협력해야 하는데 이를 명시한 규칙을 버스프로토콜(Bus Protocol)이라고 부른다.

예를 들어 CPU가 버스를 통해 전기적인 신호를 내보내고 받게 되는데, 여기서 전기적인 신호와 데이터는 같은 뜻이다.

전기적인 신호 또는 이진 스트림(Binary Stream)이라고 하는 것은 물리적인 입장에서 본 것이고 데이터라는 것은 논리적인 입장에서 본 것이다. 이 사이에 디스크 컨트롤러가 자신이 CPU에게 데이터를 보낼 것이 있다고 해서 바로 버스에 써버렸다면 버스 위에 두개의 데이터가 혼합되어 아무도 이해할 수 없는 결과를 얻게 된다.

그러나 CPU는 기계이다. CPU에 대해 이해라는 말을 쓴 것이 오히려 오해의 소지가 있다. CPU는 이해할 수 없는 데이터를 받아도 대단히 특별한 하드웨어적 검출장치가 병행적으로 돌아가며 검사해 주기 전에는 이것이 잘못된 것인지 아닌지를 전혀 알 길이 없다. 그래서 자신은 그 데이터를 늘 옳은 것으로 알고 이에 맞는 행동을 하게 되고 결과는 전혀 예측할 수 없는 쪽으로 흘러간다.

만약 CPU가 받은 데이터를 우연히 지금 작업을 그만두고 재시작(Reset) 하라는 명령으로 알았다면 전혀 엉뚱한 주소로 분기(Jump) 하라는 것으로 알고 그대로 수행한다. 따라서 우리가 그동안 작업을 한 것이 수포로 돌아가는 것은 당연하다.

그래서 서로간에 버스를 액세스함에 있어서 조금 느려질지라도 바로 쓰지 않고 버스의 상태를 보아가며 통신할 필요가 있다.

이러한 사전약속(Protocol)에 따라 하드웨어가 만들어지게 되면 위와 같은 문제는 발생하지 않는다.

버스 프로토콜의 간단한 한가지 예는 모든 장치들이 버스를 쓰기 전에 항상 CPU에게 허가를 맡는 것이다.

버스 프로토콜의 가장 간단한 한가지 예는 모든 장치들이 버스를 쓰기 전에 항상 CPU에게 허락을 받고 쓰는 방식이 있다.

여러 개의 시스템 버스가 컴퓨터 세계에 널리 알려졌는데 몇 가지를 알아 보면 Camac 버스(핵 물리학), EISA 버스(80386, 80486), Fast 버스(고 에너지 물리학), IBM-PC 및 PC/AT 버스(IBM-PC 및 PC/AT), Mass 버스(PDP-11, VAX), Mega 버스(Honeywell), Micro-channel(PS/2), Multi 버스 I (8086), Multi 버스II(80386), Nu 버스(Machintosh II), Omni 버스(PDP-11), Versa 버스(Motorola), VME 버스(680XO) 등이 있다.

1·1 버스의 작동

버스가 어떻게 작동하는지 알아 보자.

버스에 연결되어 있는 어떤 장치가 버스 전송(Bus Transfer)을 개시하고 다른 장치들은 기다리고 있다. 이 전송을 개시한 장치를 마스터(Master)라고 부르고 기다리는 장치를 슬레이브(Slave)라고 한다.

CPU가 디스크 컨트롤러에게 데이터를 읽거나 쓰라고 할 때, CPU는 마스터로서 동작하고 디스크 컨트롤러는 슬레이브로 동작한다. 그러나 잠시후 디스크 컨트롤러가 마스터로서 동작하여 메모리에게 자신이 디스크로부터 읽은 데이터를

가져갈 것을 명한다.

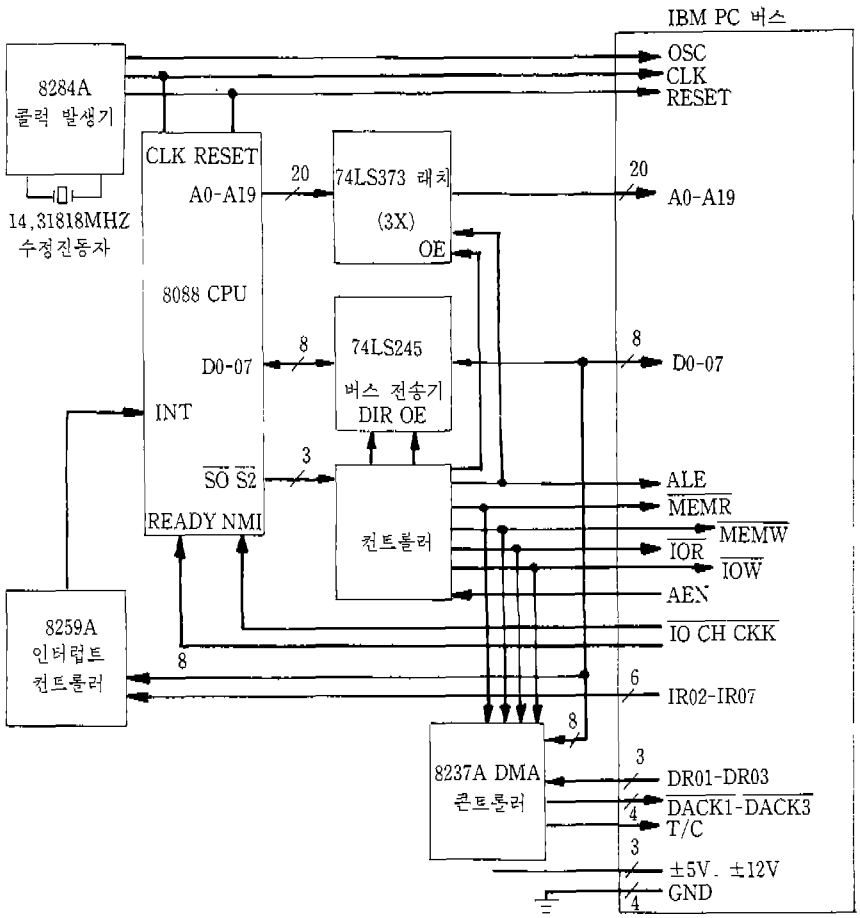
이때 버스에 지나다니는 이진 데이터의 송신 및 수신을 위해 버스 드라이버와 송신 및 수신을 위해 버스 드라이버와 버스 리시버가 존재하고 실상 이들에 여러 장치들이 연결되어 있다는 사실을 우리는 알아야 한다. 그러므로 여러 장치들은 직접 버스에 관여하는 것이 아니고 버스 드라이버나 버스 리시버를 통해 버스에 데이터를 읽고 쓰게 된다.

여러 장치들이 경우에 따라서 마스터가 되기도 하고 슬레이브가 되기도 한다.

따라서 데이터의 전송이 동시에 여러 곳에서 시작될 수 있으므로 이에 대한 대비가 필요하다.

2개 이상의 디지털 데이터가 동시에 버스에 실리게 되면 이를 충돌(Collision)이 일어났다고 하며 위에서도 말했지만 이때는 정상적인 데이터가 받아들여질 수 없다.

실제로 어느 장치든지 버스에 데이터를 전송하려고 하면 CPU와 같은 전체적인 마스터에게 버스를 액세스할 수 있는 권한을 달라고 하는 요청(Bus Request)을 하고 허락(Bus Acknowledge)을 받아야 한다.



<그림 2 > IBM-PC XT의 CPU와 버스의 기본구성도

<표 1> IBM PC 버스상의 신호(그림 2의 설명)

Signal	Lines	In	Out	Description
OSC	1		X	70 nesc Clock Signal(14.318 MHz)
CLK	1		X	210 nsec Clock Signal(4.77 MHz)
RESET	1		X	Used to reset the CPU adn I/O drvices
A0-A19	20		X	20 Address lines
D0-D7	8	X	X	8 Data lines
ALE	1		X	Address latch enable
MEMR	1		X	Memory read
MEMW	1		X	Memorywrite
IOR	1		X	I/O read
IOW	1		X	I/O Write
AEN	1	X		Address enable(asserted to have CPU iloat the bus)
IO CH CHK	1	X		I/O channel check(parity error)
IO CH RDY	1	X		I/O channel ready(insert wait states)
IRQ2-IR07	6	X		Interrupt request lines
DRQ1-DRQ3	3	X		DMA request lines
DACKD0-DACK3	4		X	DMA acknowledge lines
T/C	1		X	Terminal/count(indicates DMA completed)
Power	5			±5 Volts, ±12 volts
GND	3			Ground
Resarved	1			(Not used cn PCcard select on XT)

1·2 8088에서의 버스의 역할

그림 2 을 보자.

IBM-PC/XT의 CPU와 버스의 기본 구성도가 나타나 있다. 앞에서 예시했던 8088이 좌측에 있다. 다른 라인은 생략하고 몇 가지 중요한 라인들만 그려져 있다.

좌측 맨 위를 보면 8284A라는 클럭 발생기(Clock Generator)가 있다. 이것이 클럭을 발생시키게 된다.

클럭이 왜 필요한가? 어쩌면 가장 먼저 다루어야 할 부분인지도 모르는 이것을 쉽게 지나칠 수가 없다. 필자는 컴퓨터에 대해 일자무식인 채로 학부과정에 들어왔다. 그때에는 단지 수업에 대한 준비로서의 공부 외에는 일체 손대지 않은 관계로 디지털과 아날로그의 차이를 2학년이 될 때까지도 구별하지 못하고 있었다.

그러다가 2학년 2학기에 와서 배운 M.Mano 선생의 Digital Logic 이라는 책이 그 모든 것의 기초를 깨닫게 해 주었고 필자는 비로서 컴퓨터의 원리와 그 서광을 만나게 되었다.

여러분들은 IBM-PC 호환기종에 반드시 따라다니는 몇 MHz라는 표시를 보았을 것이다.

이것이 컴퓨터 성능의 척도가 되어 왔다.

어느새 4.77MHz에서 시작한 인텔사의 마이크로 프로세서가 랜드마크(Landmark) 118MHz에 이르게 되었다. 이는 두말할 것도 없이 클럭 진동수의 증가가 가져온 결과이다. 물론 그러한 증거가 견딜만한 CPU가 탄생했다고 말해야 더 옳겠지만, 클럭이라는 것은 이렇게 이해하자. 1클럭에 한가지 일이 일어난다. 이 클럭의

주기를 두고 머신 사이클(Machine Cycle)이라고 부른다.

CPU 내부를 보면 수만, 수십만개의 트랜지스터들이 이 사이클 내에 분주히 움직이며 작업을 한다. 이 사이클이 짧아지면, 즉 클럭 진동수가 증가하면 각 트랜지스터들이 작업을 하고 다음 트랜지스터에 넘겨서 어떠한 일의 결말을 보기도 전에 진행중인 작업이 중지된다.

반대로 길어지면 일은 리드미컬하게 잘 진행되지만 주어진 시간에 얻어지는 결과의 수가 적어지며 이는 곧 성능의 감소를 의미한다. 따라서 CPU가 따라갈 수 있는 한 클럭 진동수를 늘리는 것이 최선의 방법이다.

'85년도에 처음 나온 80386은 클럭 진동수를 늘리는 것이 최선의 방법이다. '85년도에 처음 나온 80386은 클럭 진동수가 12.5MHz로 지금의 80286과 같았다. 현재는 33MHz까지 증가했다. 이는 산술적으로 2.6배 이상 증가한 것을 의미한다. 물론 클럭 진동수가 같다고 해서 똑같은 성능을 보이는 것은 아니다. 과거의 80386이 12.5MHz였지만 지금의 80286보다는 훨씬 빠르다.

8284A는 초당 14,318,180번 진동하는 진동자(Oscillator)를 갖고 있고 이를 3배로 늘려 CPU와 버스에 제공하고 있다. 그림 3에 나타난 CLK가 4,770,000번 진동하여 초기 XT의 클럭 진동수인 4.77MHz를 얻게 된다.

또 이것이 버스에 가해져 버스의 전송 용량과도 깊이 관계하게 된다.

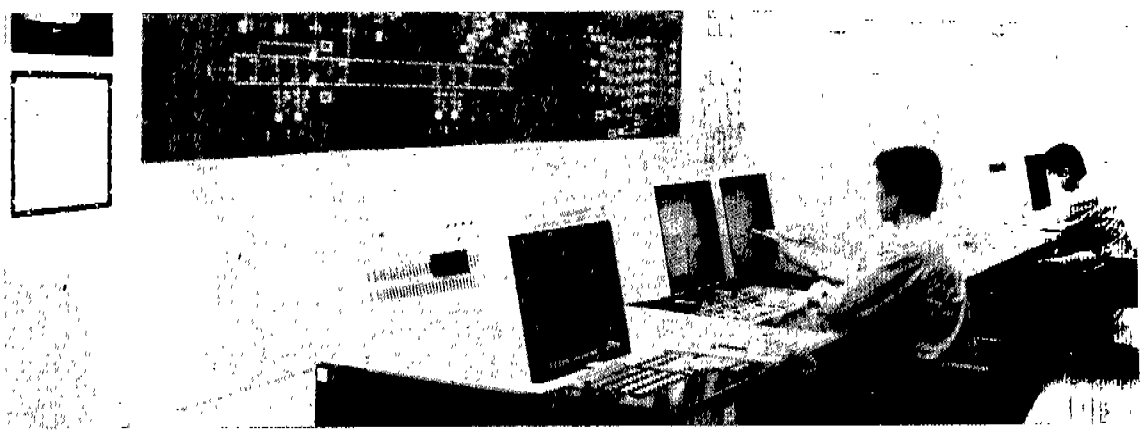
8288 버스 컨트롤러가 CPU에서 발생된 버스 제어를 적절하게 분배한다. 이 신호는 버스에 연결되어 있는 다른 장치들에게 전송되어 그들과의 협력을 용이하게 해준다.

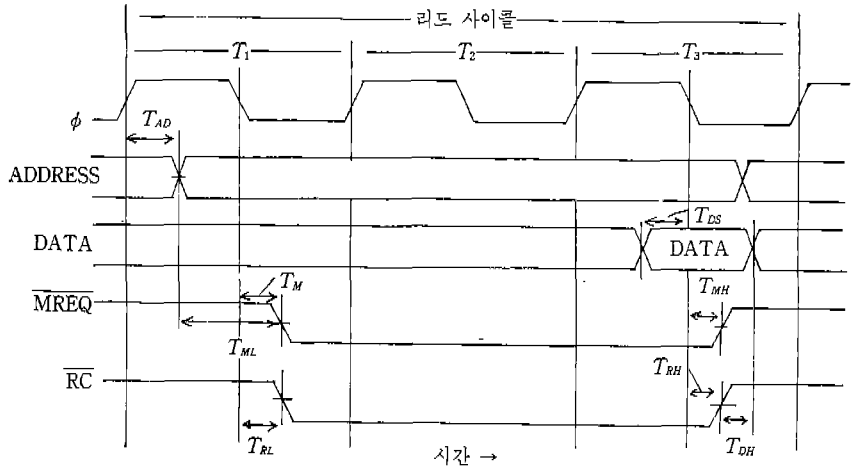
74LS245는 버스와 CPU 양쪽으로 데이터를 주고 받는데 방향을 정하고 데이터를 잠시 보관하는 등의 기능을 해준다. 여기에서 보듯이 버스는 각 장치들의 신호(중소, 데이터, 제어신호 등)를 전송해 주는 책임을 지고 있다.

한마디 더 언급한다면 왜 8284A의 진동수를 위와 같이 정했느냐 하는 것에 대해 이야기하겠다.

처음 XT가 탄생될 때를 생각해 보자. 개인이 컴퓨터를 소유한다는 것은 도저히 꿈도 꿀 수 없는 일이었다. 성능에 비해 지금과 비교하면 10배 정도의 비용이 들었다고 할 수 있다. 따라서 그 당시 가격을 낮추기 위해 하다못해 모니터의 가격이라도 줄이려고 했던 IBM의 심정을 읽을 수 있다.

위의 진동수는 북미식을 채택한 TV 주파수의 정수배로서 모니터 대신 TV를 사용할 수 있게 하는 전략이 담겨져 있는 것이다. 앞으로 연재를 진행하면서 DMA 등에 대해서도 자세히 언급할





<그림 3 > 동기 버스의 작동과정

<표 2 > 그림 3의 신호 설명

기 호	의 미	최소 필요값	최대 허용치	단 위
T_{AD}	출력지연		110	nesc
T_{ML}	MREQ 하기 전의 시간	60		nsec
T_M	MSEQ가 시작하기 전의 지연시간		85	nsec
T_{RL}	RD가 시작하기 전의 지연시간		85	nsec
T_{DS}	Data setup 시간	50		nsec
T_{MH}	MREQ가 끝나기 전의 지연시간		85	nsec
T_{RH}	RD가 끝나기 전의 지연시간		85	nsec
T_{DH}	RD를 없애기 위한 Data hold time	0		nsec

기회가 있을 것이다.

2. 버스의 분류와 종재

버스는 그들의 클럭 운용과 관계해서 두가지 분류로 나눌 수 있다.

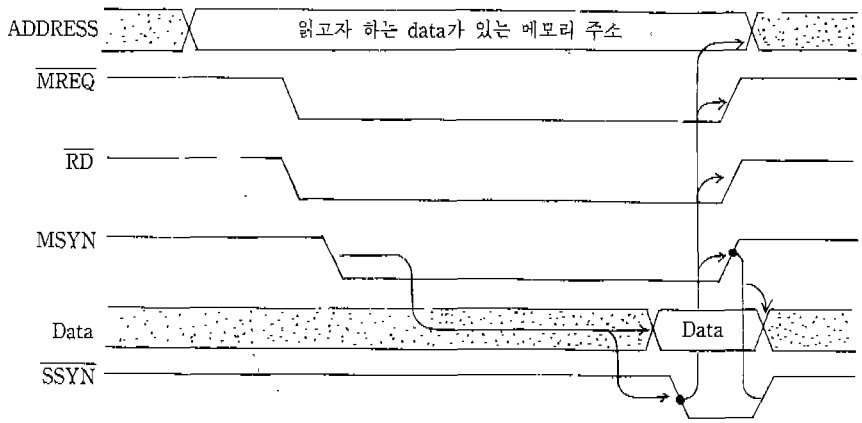
클럭이 버스에 전달되어 버스를 엑세스하려는 장치에게 영향을 미치는 동기, 형태와 시스템 버스에 클럭이 특별히 존재하지 않기 때문에 각 장치들이 전체적인 클럭에 무관하게 동작하며 그때 그때 버스의 클럭을 조정하는 비동기 형태로 나뉜다.

2·1 동기 버스

동기 버스라는 것은 수정 진동자(Crystal Oscillator)에 의해서 구동되는 라인을 갖고 있다.

버스 상의 데이터는 직각 파형(Square Wave)이 이진 스트림으로 전송되는 식으로 이동되며 그 주파수는 대략 5MHz에서 50MHz 사이이다. 모든 버스에 관계된 동작은 이 버스 사이클의 정수배로 이루어진다.

그림 4를 보자. CPU가 메모리로부터 데이터를 읽어 들이는 것을 클럭과 연관해서 분석한 그림이다. 검은 점선 부분은 무의미한 데이터이고



<그림 4 > 비동기 버스의 작동과정

된 부분이 유효한 데이터이다.

메모리 읽기 사이클은 세 클럭 사이클을 필요로 한다. 그 이유는 메모리의 반응속도와 버스상에 전송속도 명령어 처리를 위한 단계적 진행의 필요성 때문이다.

첫번째 사이클 T1에서는 메모리에 읽을 데이터가 있는 주소를 버스에 띄우고 메모리를 액세스하기 원한다는 신호(MREQ)를 낸다. 그러면 그것이 읽는 작업(RD)임을 알리는 신호가 형성된다.

사이클 T2에서는 메모리가 자신의 데이터를 꺼내고 사이클 T3에서는 원하는 데이터의 출력을 버스로 내어 보내게 된다.

이 방식은 클럭 진동수에 밀접한 관계를 맺고 있기 때문에 전반적인 속도가 진동수에 의해 고정되어 정해진다.

한 바이트의 데이터가 매 750nsec(0.00000075)마다 전송되면 전송하는 최대용량은 1.33Mbyte/sec(1byte/0.00000075)이고 다른 진동수를 써서 이 사이클을 반으로 줄이면, 2.67Mbyte/sec(1byte/0.000000375)의 전송을 할 수 있다.

그러나 무작정 버스 사이클을 줄임으로써 속도를 증사시키는 것은 여러 라인 상의 신호가 서로 다른 속도로 전송되는 버스 휨(Bus Skew)이 발생하게 되어 더 심각한 결과를 초래한다.

2·2 비동기 버스

위의 동기방식에 대해 비동기식 방식을 갖는 버스(Asynchronous Bus)도 있다. 이는 시스템 클럭이 없고 각 장치들이 클럭을 조정하는 방식이다. 그림 4를 보자. 위의 동기 방식과 비교하여 보면 MSYN(Master SYN chronization), SSYN(Slave SYN chronization) 두 신호가 더 있다. 마스터는 주소와 MREQ, RD를 버스에 띄우고 MSYN을 띄운다. 그러면 슬레이브는 그 신호를 받자마자 가능한 빨리 필요한 작업을 하고 SSYN을 띄워 작업이 완료되었음을 알린다.

마스터는 SSYN을 보고 자신이 만든 모든 신호를 끊고 SSYN을 끊어 준다. 이는 개념적으로 완벽한 시나리오이다.

이 버스 시스템의 특징은 고정된 클럭이 없기 때문에 속도의 가변화가 가능해 각 장치들이 반응만 빨리 하면 전체 속도의 증가가 크게 일어난다는 것이다.

그것은 버스에서의 전송은 빛의 전파속도이므로 버스 자체에서의 제한점은 없다고 볼 수 있기 때문이다.

☞ 다음 호에 계속