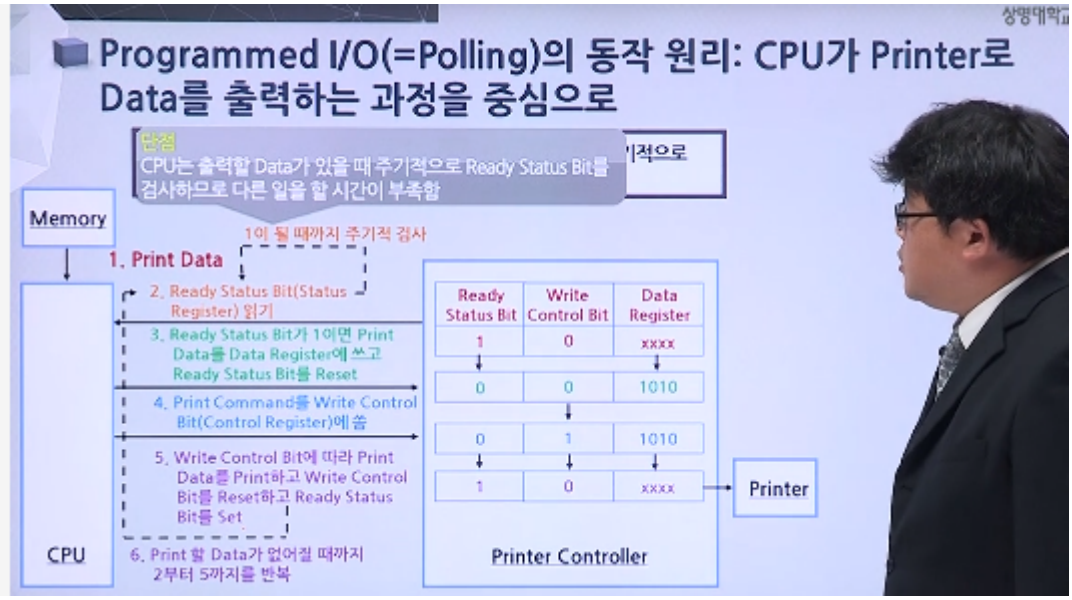
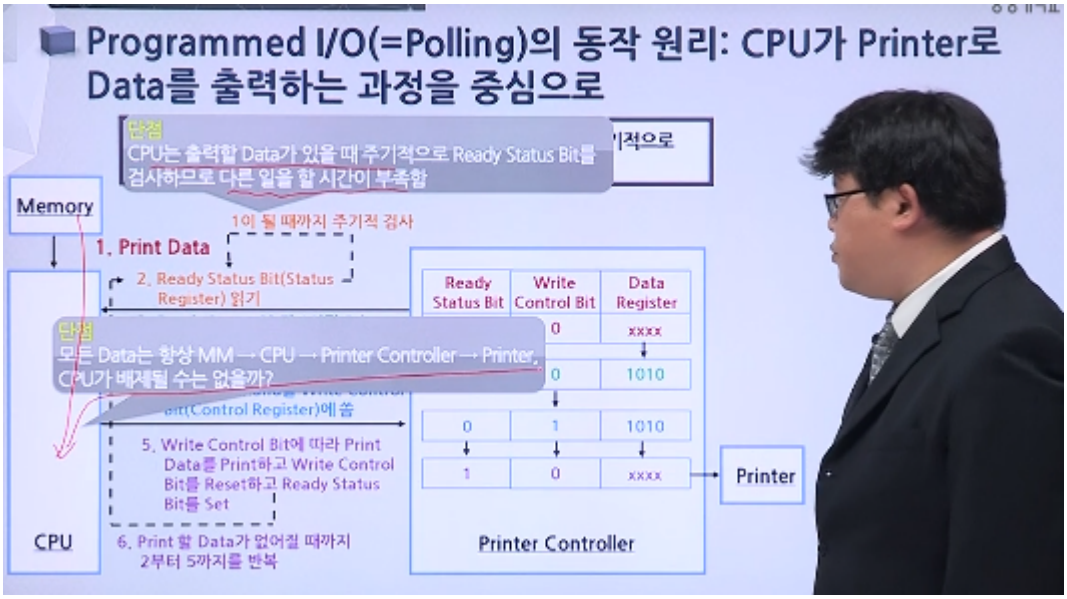
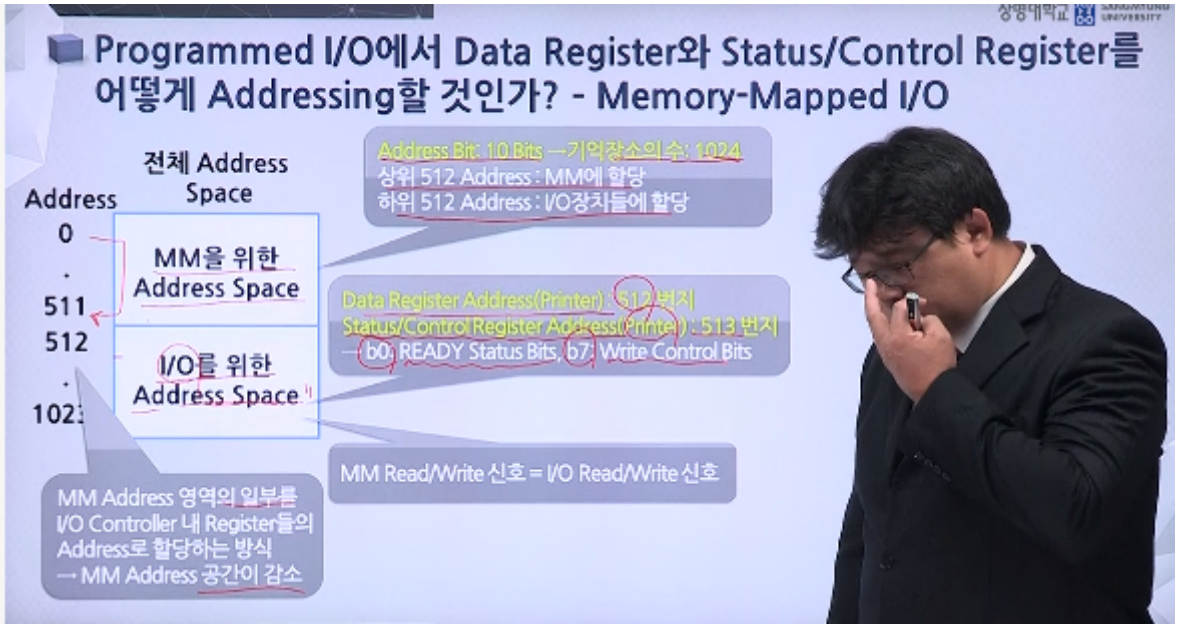
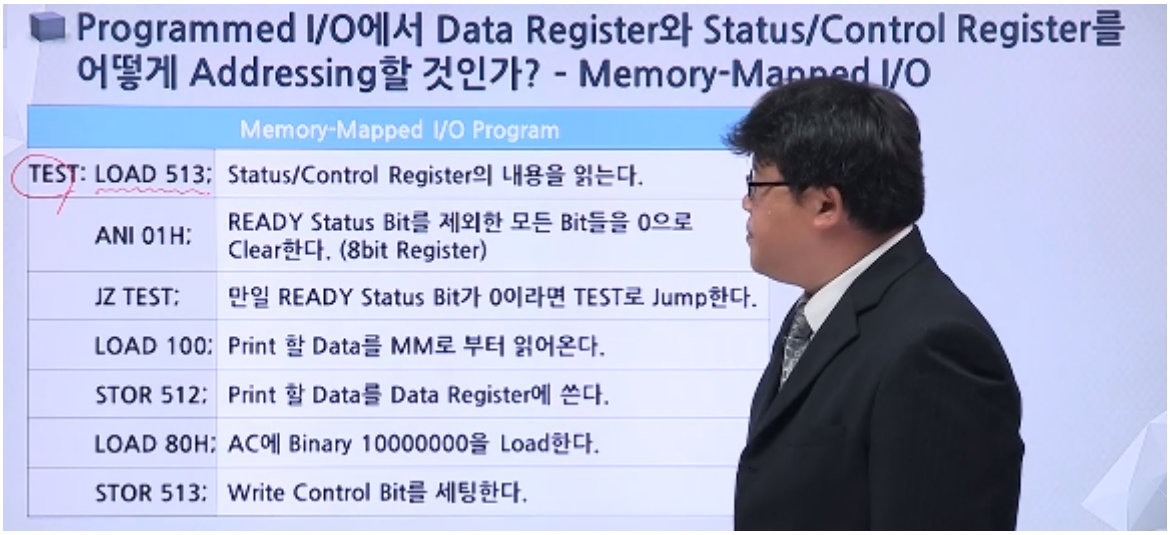
35강, 36강_I/O Device 접속 방식들(1)(2)

Created	@Aug 19, 2020 4:08 PM
Tags	RE

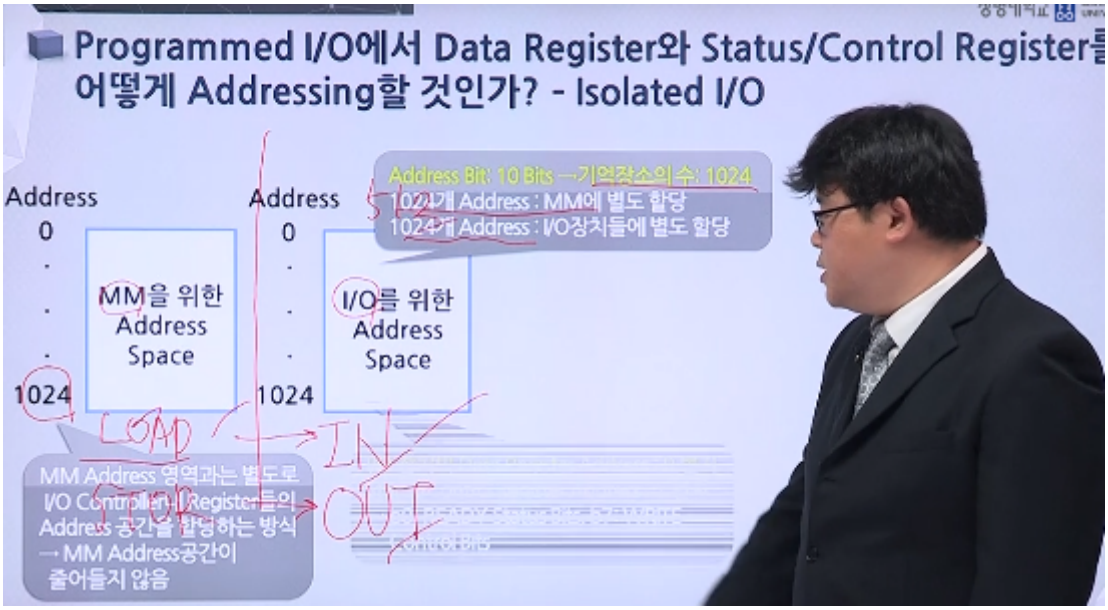


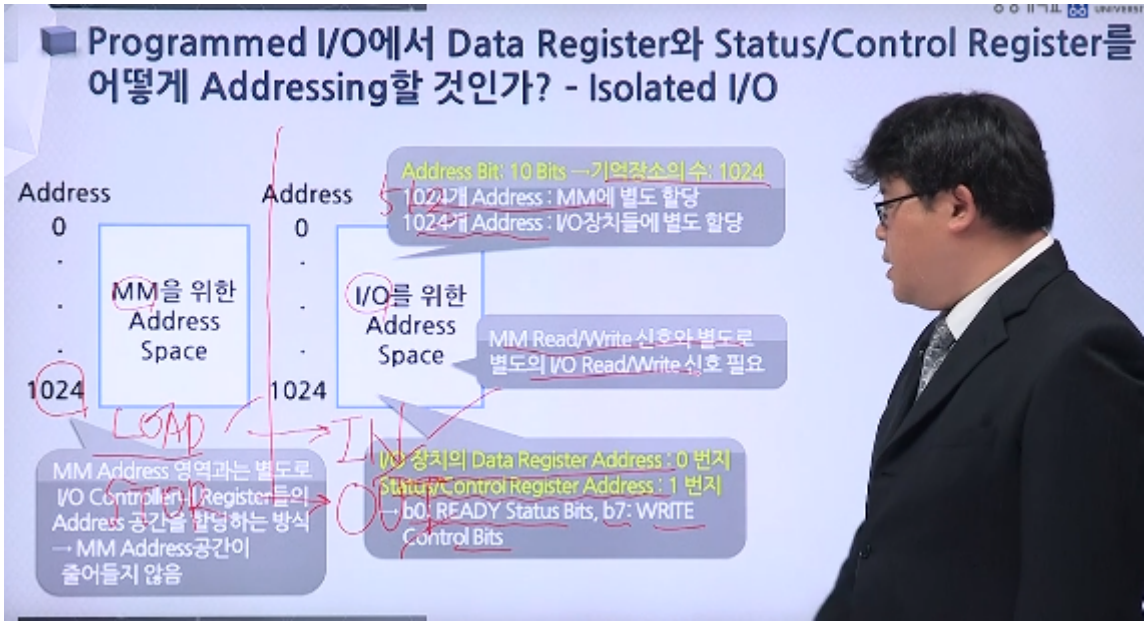

Programmed I/O(=Polling)의 동작 원리: CPU가 Printer로 Data를 출력하는 과정을 중심으로
단점
CPU는 출력할 Data가 있을 때 주기적으로 Ready Status Bit를 검사하므로 다른 일을 할 시간이 부족함
Memory
1이 될 때까지 주기적 검사
1. Print Data
2. Ready Status Bit(Status Register) 읽기
단점
모든 Data는 항상 MM → CPU → Printer Controller → Printer, CPU가 배제될 수는 없을까?
5. Write Control Bit에 따라 Print Data를 Print하고 Write Control Bit를 Reset하고 Ready Status Bit를 Set
6. Print 할 Data가 없어질 때까지 2부터 5까지를 반복
CPU
Ready Status Bit
Write Control Bit
Data Register
0
xxxx
0
1010
0
1
1010
1
0
xxxx
Printer
Printer Controller

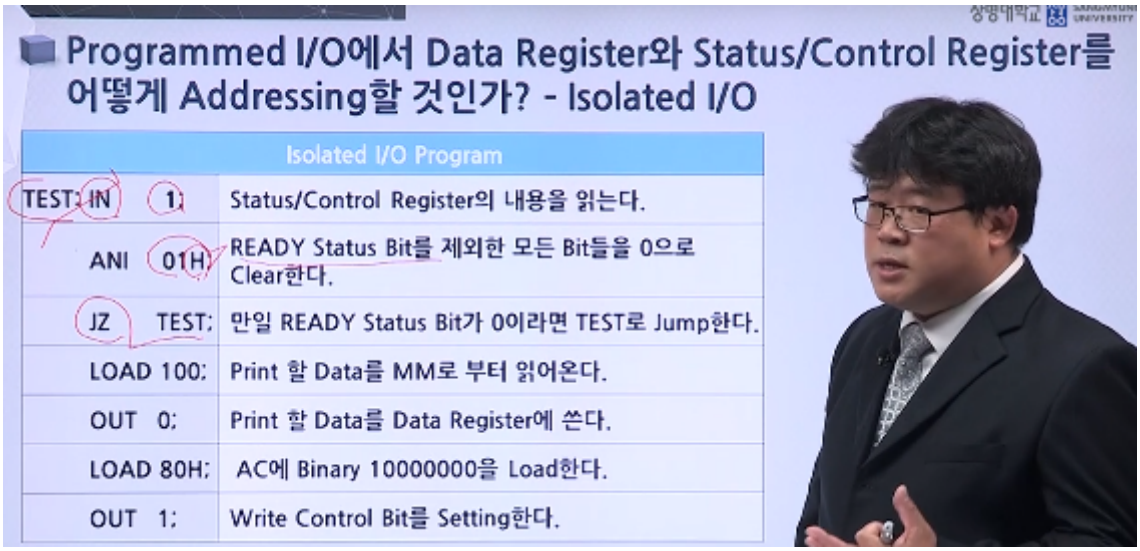

Programmed I/O에서 Data Register와 Status/Control Register를 어떻게 Addressing할 것인가? - Memory-Mapped I/O
전체 Address Space
Address
0
511
512
1023
MM을 위한 Address Space
I/O를 위한 Address Space
Address Bit: 10 Bits → 기억장소의 수: 1024
상위 512 Address : MM에 할당
하위 512 Address : I/O장치들에 할당
Data Register Address(Printer) : 512 번지
Status/Control Register Address(Printer) : 513 번지
b0: READY Status Bits, b7: Write Control Bits
MM Read/Write 신호 = I/O Read/Write 신호
MM Address 영역의 일부를 I/O Controller 내 Register들의 Address로 할당하는 방식
→ MM Address 공간이 감소

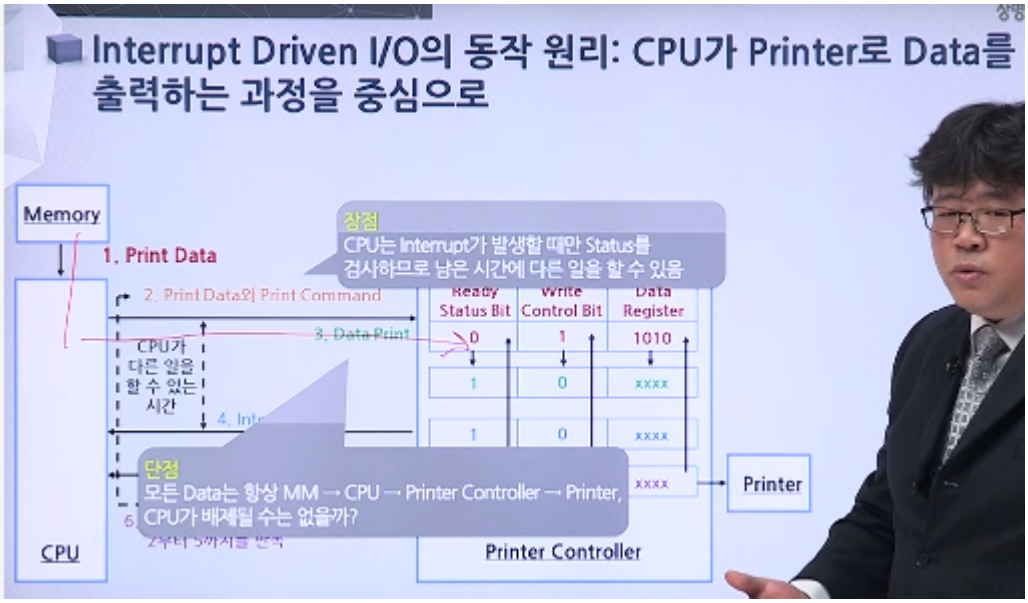


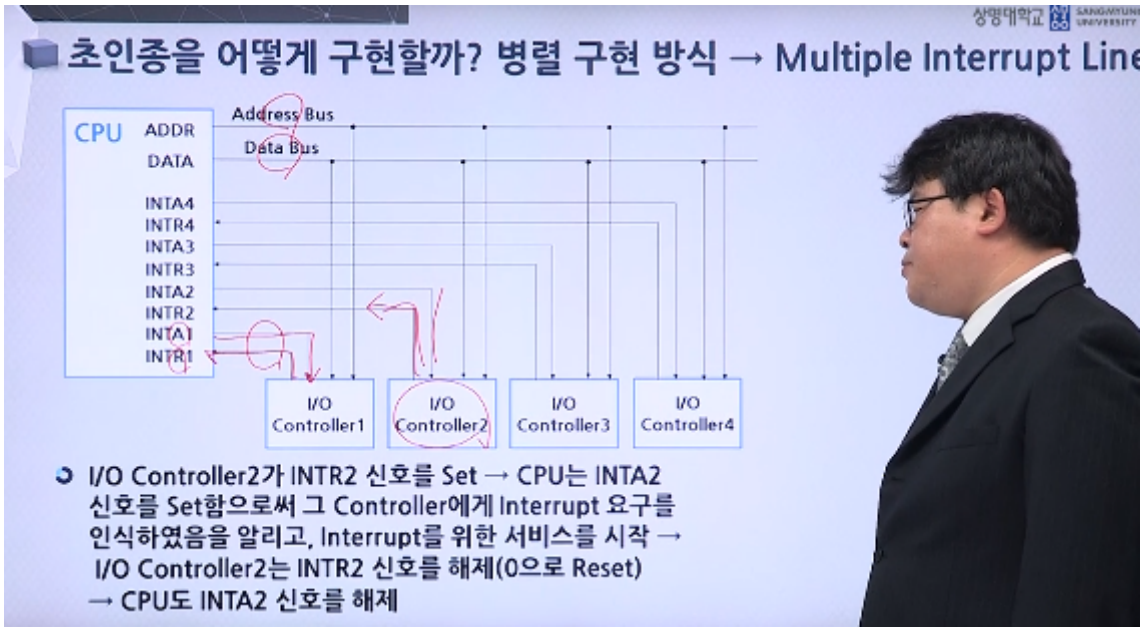
512 프린트를 위한 데이터 레지스터

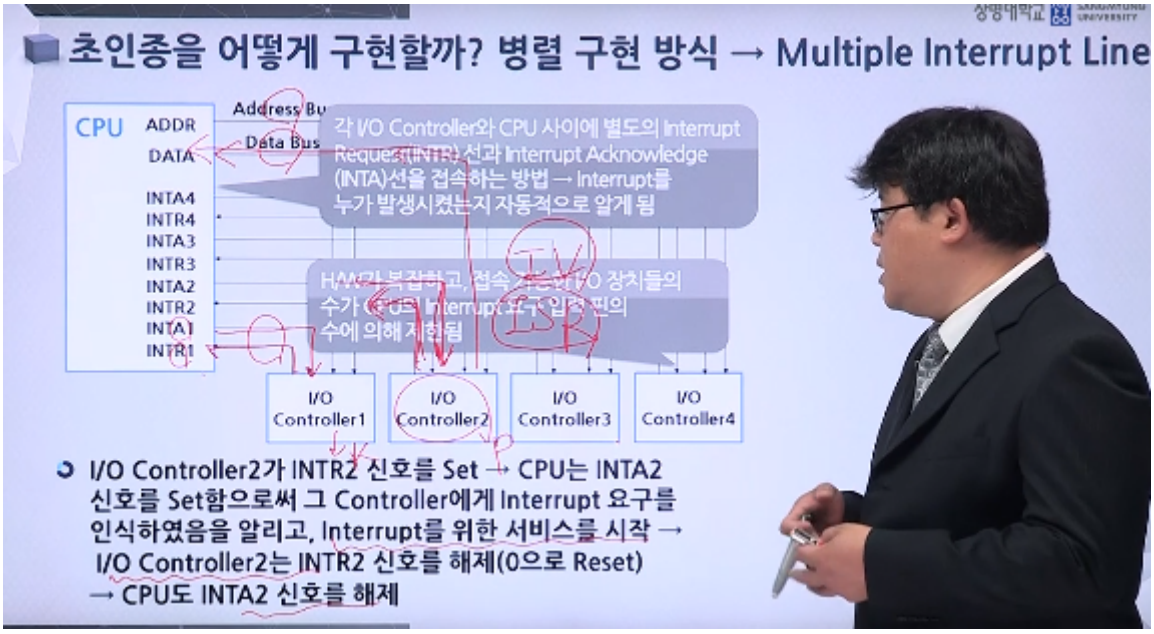


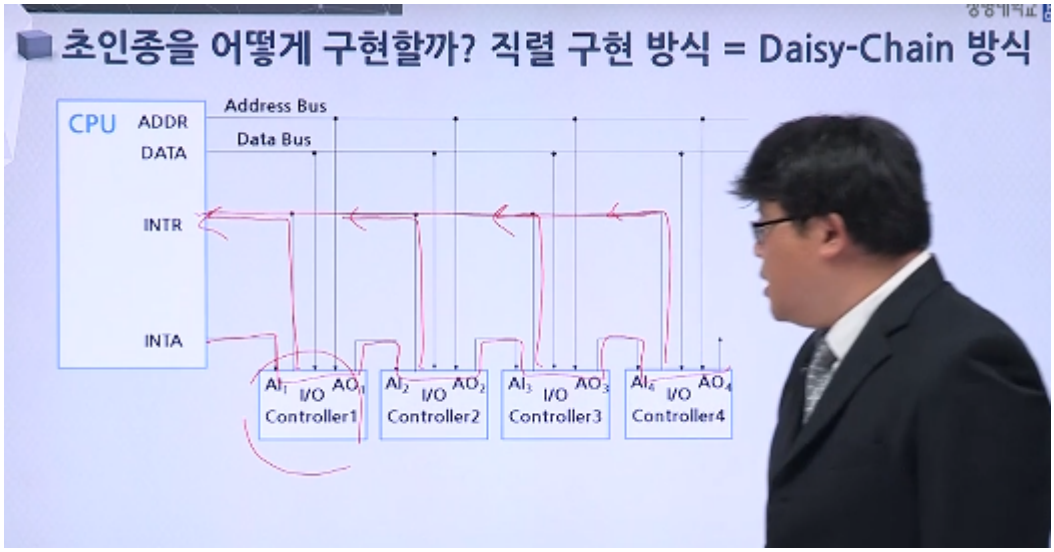

Programmed I/O에서 Data Register와 Status/Control Register를 어떻게 Addressing할 것인가? - Isolated I/O
Address
0
1024
MM을 위한 Address Space
Address
0
1024
I/O를 위한 Address Space
Address Bit: 10 Bits → 기억장소의 수: 1024
1024개 Address : MM에 별도 할당
1024개 Address : I/O장치들에 별도 할당
MM Read/Write 신호와 별도로 별도의 I/O Read/Write 신호 필요
MM Address 영역과는 별도로 I/O Controller Register들의 Address 공간을 할당하는 방식 → MM Address공간이 줄어들지 않음
I/O 장치의 Data Register Address : 0 번지
Status/Control Register Address : 1 번지
→ b0: READY Status Bits, b7: WRITE Control Bits

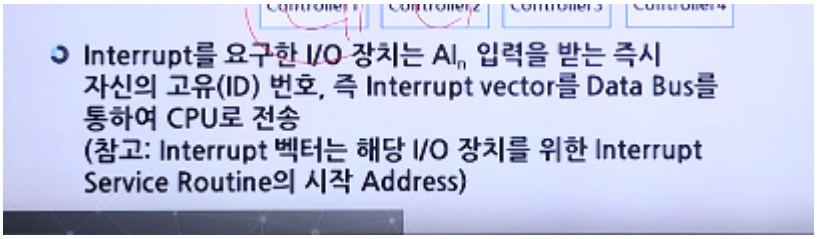

Programmed I/O에서 Data Register와 Status/Control Register를 어떻게 Addressing할 것인가? - Isolated I/O
Isolated I/O Program
TEST: IN 1; Status/Control Register의 내용을 읽는다.
ANI 01H READY Status Bit를 제외한 모든 Bit들을 0으로 Clear한다.
JZ TEST; 만일 READY Status Bit가 0이라면 TEST로 Jump한다.
LOAD 100; Print 할 Data를 MM로 부터 읽어온다.
OUT 0; Print 할 Data를 Data Register에 쓴다.
LOAD 80H; AC에 Binary 10000000을 Load한다.
OUT 1; Write Control Bit를 Setting한다.

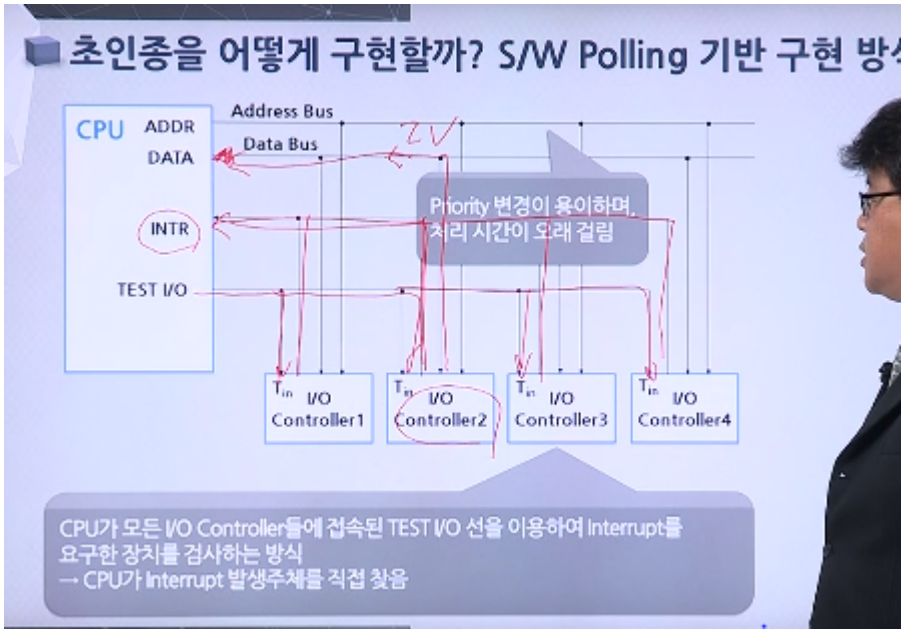

Interrupt Driven I/O의 동작 원리: CPU가 Printer로 Data를 출력하는 과정을 중심으로
Memory
1. Print Data
2. Print Data와 Print Command
3. Data Print
CPU가 다른 일을 할 수 있는 시간
장점
CPU는 Interrupt가 발생할 때만 Status를 검사하므로 남은 시간에 다른 일을 할 수 있음
Ready Status Bit
Write Control Bit
Data Register
0
1
1010
1
0
xxxx
1
0
xxxx
xxxx
단점
모든 Data는 항상 MM → CPU → Printer Controller → Printer, CPU가 배제될 수는 없을까?
Printer
CPU
Printer Controller

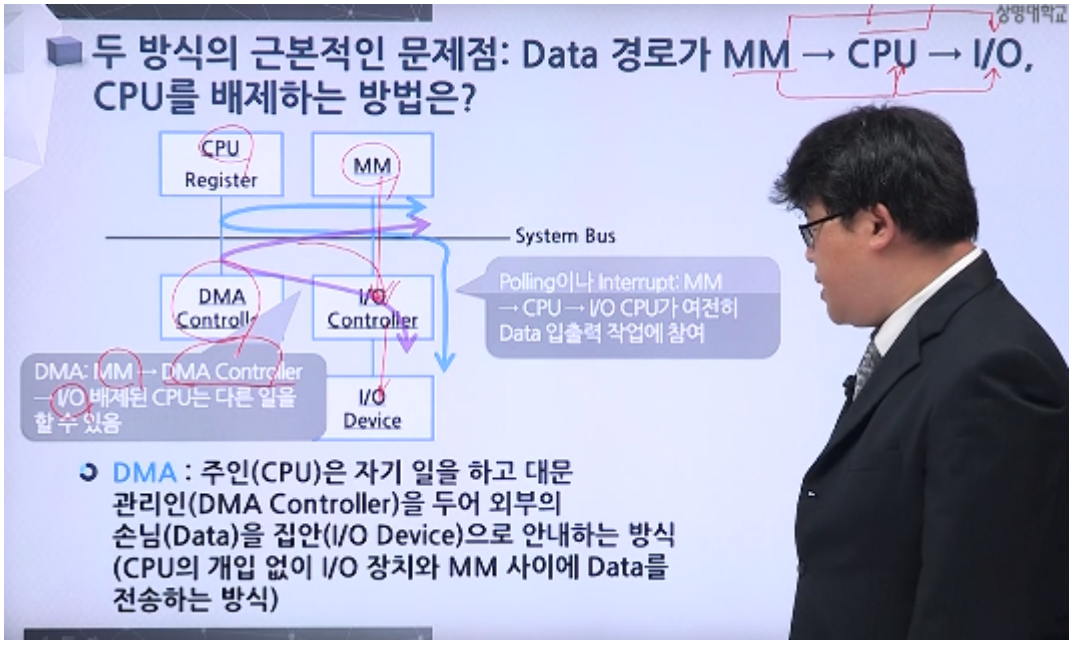

초인종을 어떻게 구현할까? 병렬 구현 방식 → Multiple Interrupt Line
CPU
ADDR
DATA
INTA4
INTR4
INTA3
INTR3
INTA2
INTR2
INTA1
INTR1
Address Bus
Data Bus
I/O Controller1
I/O Controller2
I/O Controller3
I/O Controller4
I/O Controller2가 INTR2 신호를 Set → CPU는 INTA2 신호를 Set함으로써 그 Controller에게 Interrupt 요구를 인식하였음을 알리고, Interrupt를 위한 서비스를 시작 →
I/O Controller2는 INTR2 신호를 해제(0으로 Reset)
→ CPU도 INTA2 신호를 해제

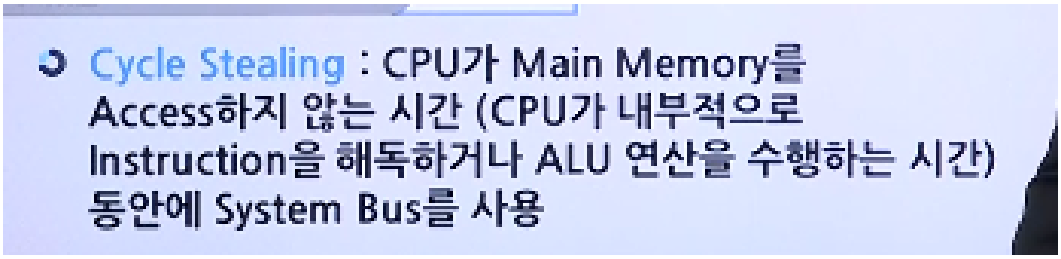

초인종을 어떻게 구현할까? 병렬 구현 방식 → Multiple Interrupt Line
CPU
ADDR
DATA
Address Bus
Data Bus
INTA4
INTR4
INTA3
INTR3
INTA2
INTR2
INTA1
INTR1
각 I/O Controller와 CPU 사이에 별도의 Interrupt Request(INTR) 선과 Interrupt Acknowledge (INTA)선을 접속하는 방법 → Interrupt를 누가 발생시켰는지 자동적으로 알게 됨
I/O Controller1
I/O Controller2
I/O Controller3
I/O Controller4
I/O Controller2가 INTR2 신호를 Set → CPU는 INTA2 신호를 Set함으로써 그 Controller에게 Interrupt 요구를 인식하였음을 알리고, Interrupt를 위한 서비스를 시작 → I/O Controller2는 INTR2 신호를 해제(0으로 Reset) → CPU도 INTA2 신호를 해제

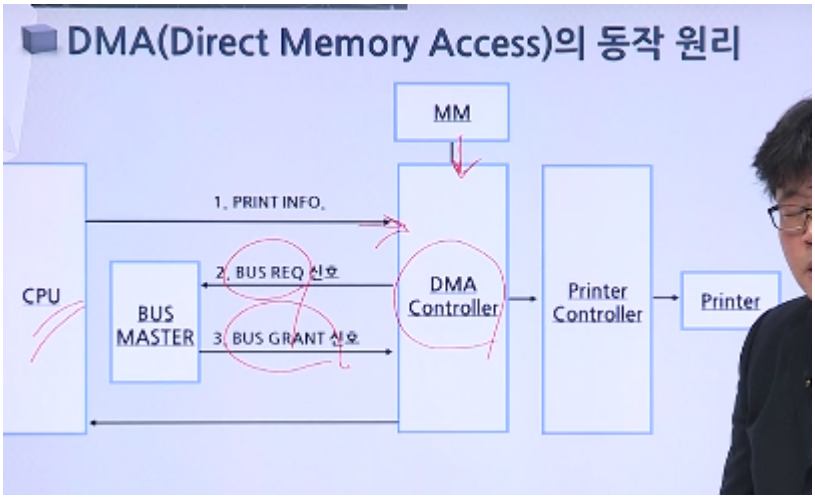

초인종을 어떻게 구현할까? 직렬 구현 방식 = Daisy-Chain 방식
CPU
ADDR
DATA
Address Bus
Data Bus
INTR
INTA
I/O Controller1
I/O Controller2
I/O Controller3
I/O Controller4

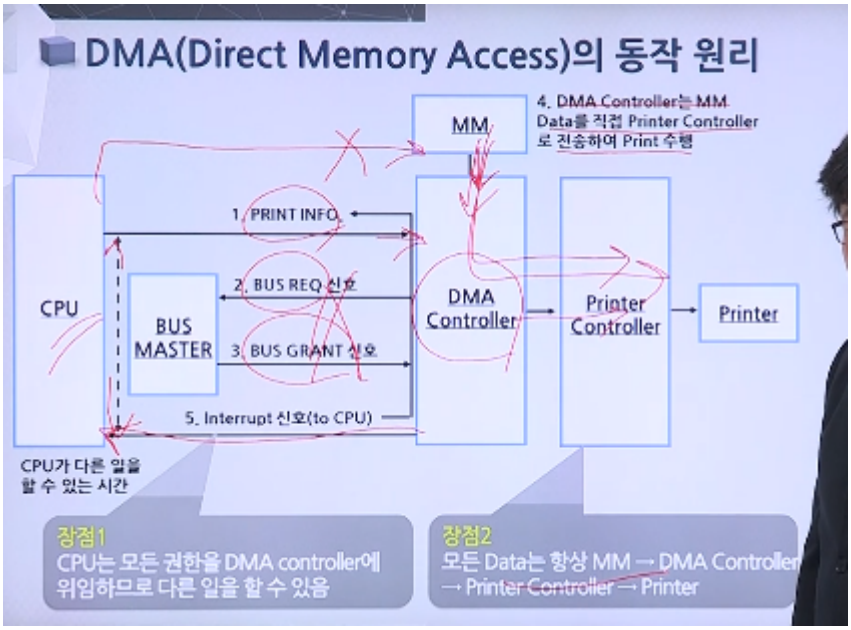

Interrupt를 요구한 I/O 장치는 AIn 입력을 받는 즉시 자신의 고유(ID) 번호, 즉 Interrupt vector를 Data Bus를 통하여 CPU로 전송
(참고: Interrupt 벡터는 해당 I/O 장치를 위한 Interrupt Service Routine의 시작 Address)

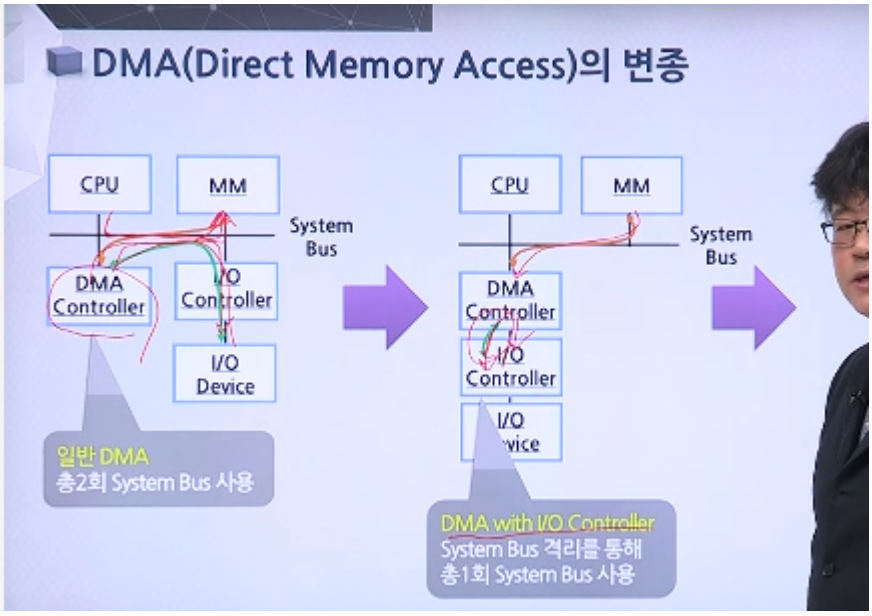

초인종을 어떻게 구현할까? S/W Polling 기반 구현 방
CPU
ADDR
DATA
Address Bus
Data Bus
INTR
TEST I/O
Priority 변경이 용이하며,
처리 시간이 오래 걸림
I/O Controller1
I/O Controller2
I/O Controller3
I/O Controller4
CPU가 모든 I/O Controller들에 접속된 TEST I/O 선을 이용하여 Interrupt를
요구한 장치를 검사하는 방식
→ CPU가 Interrupt 발생주체를 직접 찾음

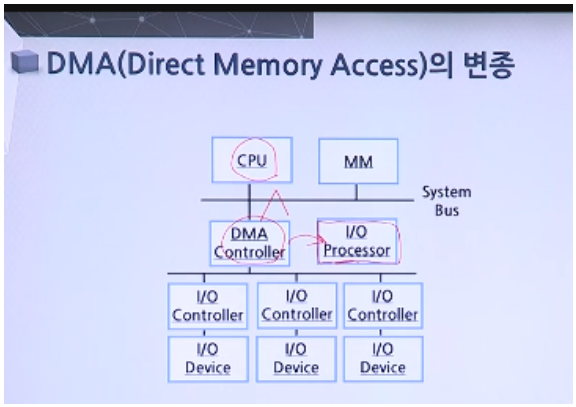

두 방식의 근본적인 문제점: Data 경로가 MM → CPU → I/O,
CPU를 배제하는 방법은?
CPU
Register
MM
System Bus
DMA
Controll
I/O
Controller
I/O
Device
Polling이나 Interrupt: MM
→ CPU → I/O CPU가 여전히
Data 입출력 작업에 참여
DMA: MM → DMA Controller
→ I/O 배제된 CPU는 다른 일을
할 수 있음
DMA : 주인(CPU)은 자기 일을 하고 대문
관리인(DMA Controller)을 두어 외부의
손님(Data)을 집안(I/O Device)으로 안내하는 방식
(CPU의 개입 없이 I/O 장치와 MM 사이에 Data를
전송하는 방식)

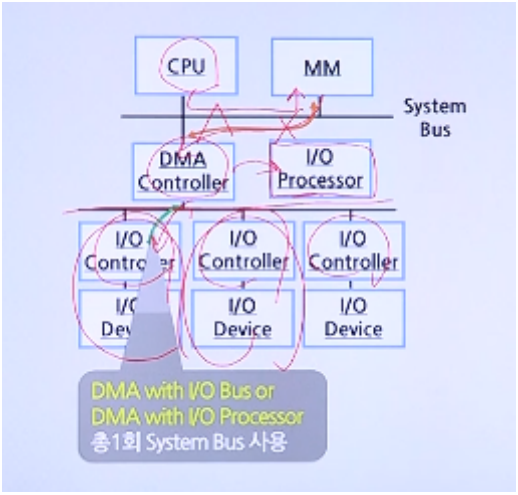

Cycle Stealing : CPU가 Main Memory를
Access하지 않는 시간 (CPU가 내부적으로
Instruction을 해독하거나 ALU 연산을 수행하는 시간)
동안에 System Bus를 사용


DMA(Direct Memory Access)의 동작 원리
MM
1. PRINT INFO.
CPU
BUS MASTER
2. BUS REQ 신호
3. BUS GRANT 신호
DMA Controller
Printer Controller
Printer


DMA(Direct Memory Access)의 동작 원리
MM
4. DMA Controller는 MM Data를 직접 Printer Controller로 전송하여 Print 수행
1. PRINT INFO.
CPU
BUS MASTER
2. BUS REQ 신호
3. BUS GRANT 신호
DMA Controller
Printer Controller
Printer
5. Interrupt 신호(to CPU)
CPU가 다른 일을 할 수 있는 시간
장점1
CPU는 모든 권한을 DMA controller에 위임하므로 다른 일을 할 수 있음
장점2
모든 Data는 항상 MM → DMA Controller → Printer Controller → Printer


DMA(Direct Memory Access)의 변종
CPU
MM
System Bus
DMA Controller
I/O Controller
I/O Device
일반 DMA
총2회 System Bus 사용
CPU
MM
System Bus
DMA Controller
I/O Controller
DMA with I/O Controller
System Bus 격리를 통해
총1회 System Bus 사용


DMA(Direct Memory Access)의 변종
CPU
MM
System Bus
DMA Controller
I/O Processor
I/O Controller
I/O Controller
I/O Controller
I/O Device
I/O Device
I/O Device


CPU
MM
System Bus
DMA Controller
I/O Processor
I/O Controller
I/O Controller
I/O Device
I/O Device
DMA with I/O Bus or
DMA with I/O Processor
총1회 System Bus 사용