폴링(Polling), 인터럽트(Interrupt), DMA(Direct Memory Access)

[폴링(Polling)]

호스트와 입/출력 하드웨어 사이의 **프로토콜**은 복잡하지만 기본적인 **핸드셰이킹**(hand shaking) 개념은 간단하다.(중앙 집중적 기능을 하는 컴퓨터와 그 컴퓨터의 주변장치인 입출력 장치 사이에 정보를 주고 받는 방식은 복잡하지만 기본적인 개념인 핸드셰이킹은 간단하다.)

제어기는 상태 레지스터의 busy 비트를 통해 자신의 상태를 나타낸다.(지금 자신이 명령을 수행 중인지 아니면 명령을 기다리는 중인지.) 제어기는 바쁠 때는 busy 비트를 1 로 설정하고 다음 명령을 받아들일 준비가 되어 있을 경우에는 busy 비트를 0 으로 설정한다.

호스트는 다음과 같은 방법으로 핸드셰이킹을 통해 제어기와 협력하면서 **포트**를 통해 출력을 쓴다.

호스트가 반복적으로 (소거될 때까지) busy 비트를 검사한다.(이것이 폴링이다.)

호스트가 명령 레지스터에 쓰기 비트(write bit)를 설정하고 출력(data out)레지스터에 출력할 바이트를 쓴다.

1. 호스트가 명령어 준비 완료 비트(command ready bit)를 설정한다.
2. (입/출력 하드웨어) 제어기가 명령어 준비 완료 비트가 설정된 것을 알아 차렸을 때 **자신**의 busy 비트를 설정한다.
3. (입/출력 하드웨어) 제어기는 명령어 레지스터를 읽고 write 명령어임을 알게 된다. 출력 레지스터를 읽어 해당 바이트를 가져와 해당 하드웨어 장치로 출력한다.
4. (입/출력 하드웨어) 제어기는 명령어 준비 완료 비트를 소거(command-ready==0)하고 입/출력이 끝났음을 알리기 위해 busy 비트를 소거한다.

이 루프는 매 바이트마다 반복되며 1 단계에서 호스트는 폴링(polling)을 하게 된다.

즉 폴링은 매주기마다 명령어가 무엇인지 묻는 방식이다.

코딩 하기 쉽지만 비효율적이라는 단점이 있다.(cpu 가 계속 busy 인지 아닌지를 체크해야 하니깐 다른 일을 하지를 못한다.)

- 포트란? 장치와 컴퓨터 시스템간의 통신을 위한 연결점을 의미한다.
- 제어기(controller)란? 포트, 버스, 장치의 작동을 지시하는 칩을 말한다. 단일 칩으로 구현된 것은 직렬 포트 제어기, 회로 기판으로 구현된 것은 SCSI 제어기, 장치에 내장된 제어기 등으로 나누어진다.

- 입출력 포트: [그림 12.2] 호환성 PC에서 장치 입출력 포트의 위치
  → 4개의 레지스터로 구성
  ① status 레지스터: 장치의 상태
    ex) 판독 가능 여부, 에러 유발 여부 등
  ② control 레지스터: 장치 모드의 변경
    ex) 전이중(full-duplex) 혹은 반이중(half-duplex) 통신의 선택, 패리티 검사, 워드 길이 선택,
      전송 속도 선택 등
  ③ data-in 레지스터: 입력용 자료 레지스터
  ④ data-out 레지스터: 출력용 자료 레지스터

- 핸드셰이킹(HandShaking)이란? 호스트와 제어기 사이의 교류를 위한 프로토콜을 말한다.

  제어기는 status 레지스터의 busy 비트를 통해 상태를 설정한다. 명령을 수행 중이라면 busy == 1, 지시 받은 명령을 완료하고 다음 명령을 기다리는 상태라면 busy == 0

  호스트는 command 레지스터의 command-ready 비트를 사용한다. 제어기에게 명령을 내린 상태라면 command-ready == 1, 제어기가 명령을 수행하여 완료된 상태라면 command-ready == 0 (즉 제어기에게 다음 명령을 내릴 수 있다는 의미)

  호스트는 요청한 명령이 완료될 때까지 busy 비트를 매 주기마다 반복적으로 판독한다. (이것이 바로 폴링이다.) 이 때 사용 중 대기(busy-waiting)이 발생한다. 이는 중앙처리장치(CPU)의 비효율성을 초래한다. 그래서 제어기가 중앙처리장치에 통보하는 방식인 인터럽트 방식이 훨씬 효과적이다.

  출력의 과정은 아래와 같다.

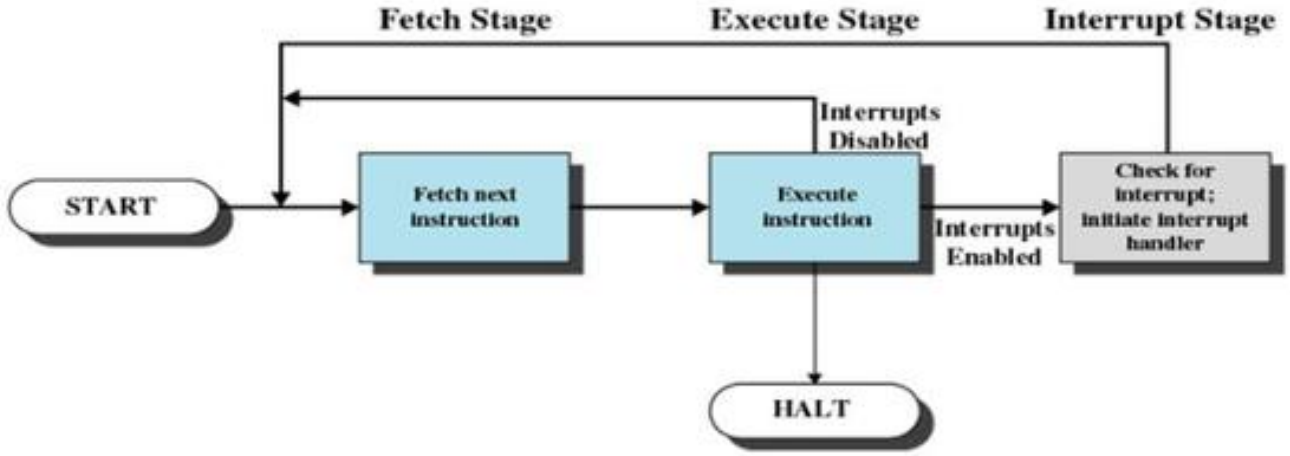
  1. 호스트는 요청한 명령이 완료될 때까지 될 때까지 busy 비트를 반복적으로 판독한다.
  2. 호스트는 command 레지스터의 write 비트를 설정하고 data-out 레지스터에 한 바이트를 쓴다. <즉 다음에 수행할 명령어와 그 명령어(쓰라는 명령어)에 맞는 데이터<이 내용을 쓰라는 명령일 때 그 내용>를 일단 올려놓은 다음 스위치를 on 시켜서 수문을 열어 물을 흘러 내리게 하는 것과 동일한 아이디어. 수문을 여는 것은 다음 단계임. 지금은 물살의 방향과 물의 양을 일단 설정하는 것임>
  3. 호스트는 command-ready 비트를 설정한다. <수문을 연다.>
  4. 제어기가 command-ready 비트가 설정되었음을 알아차리면 busy 비트를 설정한다. <지금 물이 흘러내려가고 있는 중이다 즉 명령어를 수행하는 중임을 알리는 busy 를 1 로 설정한 다음 제어기는 명령을 수행하기 시작한다. 즉 명령을 수행하기에 앞서 먼저 지금은 업무 중 혹은 지금은 외출 중이라는 표시를 문 앞에 달아놓는 셈이다.>
  5. 제어기는 command 레지스터를 판독하여 명령이 write 임을 파악하고, data-out 레지스터를 판독하여 해당 바이트를 가져와 장치에 출력한다. <명령 수행>
  6. 제어기는 command-ready 비트를 클리어 하고, 입출력 성공을 알리기 위해 status 레지스터의 error 비트를 클리어 한 후 busy 비트를 클리어 한다. <요청하신 명령을 다 완료했습니다. 즉 주문하신 음식 나왔습니다 라고 고객안내방송 내보내는 것임. 명령 수행 성공했으면 error 비트를 없애고 busy 비트도 없애서 지금은 바쁘지 않음, 이제 재실 중이라는 표시를 한다.)

- 호스트(host)란? 일종의 컴퓨터이다. 중앙 집중적 기능을 수행하는 컴퓨터. 목적지이자 원천. 즉 호스트는 클라이언트가 되기도 하고 서버가 되기도 한다.
- 프로토콜(protocol)이란? 정보를 주고받는 방식. 정보의 포맷구성, 포맷의 송수신방법 등

[인터럽트(Interrupt)]

CPU 가 일일이 모든 장치를 폴링하지 않고 용건이 있는 장치가 직접 요청을 통보해오는 방식이다. 입/출력 장치가 CPU 에게 자신의 상태 변화를 통보하는 하드웨어 기법이다. CPU 하드웨어는 인터럽트 요청 라인(interrupt request line)이라고 불리는 선을 하나 갖는데 CPU 는 매 명령어를 끝내고 다음 명령어를 수행하기 전에 인터럽트 요청 라인을 검사한다.

1. 입/출력 하드웨어 제어기가 이 요청 라인에 신호를 보낸다.
2. CPU 가 각종 레지스터 값과 상태 정보를 저장한 다음 메모리 상의 인터럽트 핸들러 루틴으로 이동 한다.
3. 인터럽트 핸들러는 인터럽트의 발생 원인을 조사하고 필요한 작업을 수행 한 후 CPU 를 인터럽트 이전 상태로 되돌리는 명령을 수행 한다.
4. 핸들러는 입/출력 장치를 서비스함으로써 이 인터럽트를 처리해 준다.



Instruction Cycle with Interrupts

대부분의 CPU 들은 두 종류의 인터럽트 요청 라인을 가진다.

- 마스크 불가 인터럽트(nonmaskable interrupt) : 회복 불가능한 에러(메모리 에러와 같은 이벤트)를 위해 사용 (미사일 시스템, 정밀한 안전을 요하는 시스템에 사용됨)
- 마스크 가능 인터럽트(maskable interrupt) : 필요 시(중요한 처리를 하는 중엔 인터럽트로 넘어가 버리면 곤란한 경우) 인터럽트 기능을 잠시 중단시켜 놓을 수 있는 기능으로 장치 제어기가 서비스를 요청하기 위해 사용한다. 인터럽트 되어서는 안 되는 주요 명령 시퀀스를 수행하기 전에 CPU 가 끌 수 있다. 인터럽트가 걸려서는 안 되는 명령을 수행 중일 땐 중앙처리장치에 의해 해당 인터럽트는 해제된다.

인터럽트 기법은 보통 주소라고 하는 하나의 작은 정수를 받아 들이는데 이 정수는 특정 인터럽트 핸들링 루틴을 선택하기 위해 사용되며 대부분의 구조에서 이 주소는 인터럽트 벡터라고 불리는 테이블의 오프셋으로 사용된다.

컴퓨터는 인터럽트 벡터 내에 있는 주소들보다 더 많은 수의 장치를 갖고 있다. 이러한 문제를 해결하기 위해 인터럽트 사슬(chaining) 기술을 사용한다. 인터럽트 사슬화에는 인터럽트 벡터의 각 원소들이 여러 인터럽트 핸들러들로 이루어진 리스트의 헤더를 가리키고 있다. 인터럽트가 발생하면 해당 핸들러가 찾아질 때까지 리스트 상의 핸들러들을 하나씩 검사하게 된다.

인터럽트 기법은 인터럽트 우선순위 수준(interrupt priority level)의 구현을 가능하게 한다. 이 기법은 CPU 가 모든 낮은 우선순위 인터럽트를 일일이 마스크 오프(mask off)시키지 않더라도 자동적으로 높은 우선순위 인터럽트가 낮은 순위 인터럽트의 실행을 선점(preempt) 할 수 있게 한다.

[DMA (Direct Memory Access)]

CPU 가 상태 비트를 반복적으로 검사하면서 1 바이트(혹은 워드)씩 제어기 레지스터에 옮기는 입/출력 방식을 PIO(Programmed I/O)라고 한다. 많은 컴퓨터들은 CPU 의 PIO 작업 중 일부를 DMA(Direct Memory Access) 제어기라고 불리는 특수 처리기에 위임함으로써 CPU 의 일을 줄여 준다.

디스크와 같이 많은 양의 데이터를 전송하는 경우에 적합한 방식인데, 이 때 범용의 중앙처리장치가 이 일을 담당하게 하는 것보다는 직접 메모리 접근 제어기라 불리는 별도의 처리기가 담당하게 한다.

DMA 전송을 시작시키기 위해서 호스트는 메모리에 DMA 명령 블록을 쓴다.(이것이 바로 DMA 초기화이다.) 이 블록에는 전송할 자료가 있는 곳의 포인터와 전송할 장소에 대한 포인터, 그리고 전송될 바이트 수가 기록된다. 그리고 CPU 는 이 DMA 명령 블록의 주소를 DMA 에게 알려주고 자신은 다른 작업을 계속 한다.(CPU 는 할 일에 대한 명세표와 위임장(=블록)을 작성 한 다음 자기 대신 일을 수행할 DMA 에게 전해준다.) DMA 는 CPU 의 도움(개입, 참견) 없이도 자신이 직접 버스를 통해 DMA 명령 블록을 접근하여 입/출력을 수행한다.

DMA 제어기와 장치 제어기(Device controller)간의 핸드셰이킹은 DMA-request 와 DMA-acknowledge 라고 불리는 두 개의 선(wire;와이어)을 통해 수행 된다.

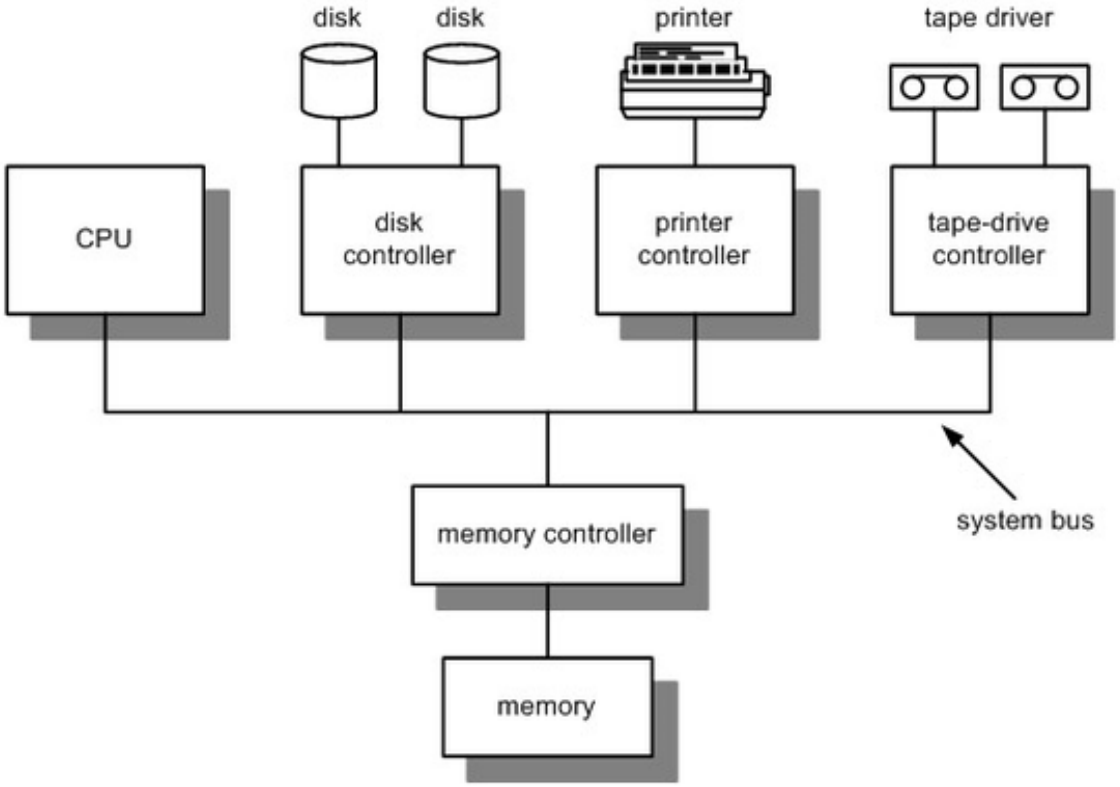
1. 장치 제어기는 전송할 한 워드 분량의 자료가 전송 가능한 상태가 되면

   DMA-request 선에 신호를 보낸다.

2. DMA 제어기가 이 신호를 받으면, 메모리 버스를 점유하고 거기에 원하는 주소를 올려놓고 DMA-acknowledge 선에 신호를 보낸다.

3. DMA-acnkowledge 신호를 받은 해당 장치제어기가 한 워드를 메모리로 전송하고 DMA-request 신호를 제거 한다.

4. 전송이 끝나면 DMA 제어기는 CPU 에게 인터럽트를 건다.



DMA 가 메모리 버스를 점유 중이면 비록 CPU 는 주 캐시와 보조 캐시에 있는 자료는 접근할 수 있지만 일시적으로 주 메모리에 있는 자료는 접근하지 못한다. 이러한 사이클 스틸링(cycle stealing)은 CPU 의 속도를 저하시키지만 입/출력 작업을 DMA 로 넘기는 것은 전체적으로 시스템 성능을 향상 시킨다.

DMA 를 사용할 때 물리적 주소를 사용하지만 DVMA(직접 가상 메모리 접근, direct virtual memory access)을 사용하기도 한다. DVMA 를 사용하면 CPU 나 메모리의 개입이 없어도 가상 주소로 두 개의 메모리 매핑(memory mapped) 장치간에 자료를 전송 할 수가 있게 된다.