<CPU 와 OS>

60181629_김호남
60181650_염제성
60181664_정 민

1.Central Processing Unit

CPU 는 명령어를 해석하고 실행하는 장치로서 산술계산과 논리연산을 처리하는 기능을 수행한다. CPU 에는 다양한 구성요소가 있는데 그것은 다음과 같다.

1-(1). CPU 구성

1). ALU(Arithmetic(산술계산) and Logic(논리(boolean, and, or) Unit) : 산술연산과 논리연산을 수행하는 중앙처리장치 내부의 회로 장치이다. 산술계산으로 비교연산을 대신하기 때문에 비교하는 연산은 따로 존재하지 않는다.

ALU 는 Data Segment 와 연결되어 있어서 입출력에 접근이 가능하다.

2). CU(Control(제어 : 흐름(순서) 결정) Unit) : 모든 실제 명령어를 실행하는 핵심이다. 기억장치에 있는 명령을 해독하여 각 장치에 동작을 지시한다.
CU 는 Code Segment 와 연결 되어있다.

3). Register : CPU 안에 있는 기억장치를 말하며 Register 는 Main Memory 보다 그 속도가 빠르다.
여기서 Main memory 는 CPU 가 어떠한 명령어를 실행하기 위해서는 무조건 거쳐야 하는 곳이며 현재 실행 중에 있는 프로그램과 이 프로그램이 필요로 하는 데이터를 일시적으로 저장하는 장치이다.

4). Bus : 통신하려고 뚫어 놓은 길이다.
<버스의 종류>.
1) Address bus : 주소 버스를 통해 주기억장치의 어느 위치에서 데이터를 읽을지 또는 어느 위치에 데이터를 쓸지가 정해진다.
2) Data bus : 데이터 버스를 통해 주기억장치로부터 읽거나 주기억장치에 써야 할 데이터가 전송된다.
3) Control bus : 제어 버스를 통해 데이터를 주기억장치에 쓸지 읽을지를 결정하는 정보가 전송된다. 전송되는 정보는 중앙처리장치에서 주기억장치로만 전송되므로 단방향성이다.

5). AC : ALU 가 연산 해야 할 값이나 ALU 가 연산한 결과 값을 저장하는 공간이다.

6). SP : Data Segment 의 시작위치이다.

7). MAR : PC 에서 값을 받아와서 저장하는 장소이다.

8). MBR : MAR 의 주소를 읽어오며, 읽어온 값을 IR 에 보낸다.

7-8). '주소를 읽어 온다.' 라는 것은 MAR 에서 주소를 가져와서 MBR 에 적는 것을 말한다. CU 에 어떤 주소가 들어와 있다. MAR 이 주소를 메모리에게 말하고, 그 주소에 대한 데이터를 MBR 이 읽어온다.

9). IR : 가져온 Instruction 을 두 부분으로 나눠 저장한다.
- opcode : LDA, ADD, STO 등을 저장하며 CU 가 opcode 를 보고 ALU 또는 PC 에 명령을 내린다.
- address : 값을 저장하는 공간이다.

10). Status : ALU 가 계산했는데 자릿수가 넘어가면 Status 에 저장해 놓는다.
Status 는 양수, 음수와 같은 상태를 저장하며, 크기에 대한 비교를 할 수 있는 공간이다.

11). PC(Program Counter) : 실행할 라인(명령어)의 주소이다.
PC 는 Code Segment 의 몇 번째 메모리를 실행할지 적어 놓는다. 즉, PC 는 CU 가 실행할 라인(명령어의 주소)을 적어 놓는다.

12). Clock : CPU 에는 Clock 이 있다. Clock 이 일정한 단위로 looping 을 무한대로 돌고 있다.
Clock 이 일정속도로 CPU 를 계속 실행(looping)시키는데, 그 때마다 CPU 는 명령을 주고, 그 때 모든 디바이스가 동시에 명령을 수행한다.
Clock speed 가 빠르면 CPU 가 빠르다는 것이다. 그래서 CPU 가 Clock 보다 느리면 깨지기 때문에 CPU 가 처리하는 속도 중에 가장 긴 속도가 Clock 의 스피드보다 빨라야 한다.

<int x = 8; int y = 16; LDA x; ADD y; STO y; 시나리오>

1. Clock 이 Memory 를 때리면 프로그램이 실행된다. 실행되면 OS 가 Code Segment 의 시작 주소를 PC 에, Data Segment 의 시작 주소를 SP 에 저장한다.

2. PC에 있는 시작 주소(LDA x)를 MAR이 저장한다. Memory가 MAR에 있는 LDA x를 읽어 와서 MBR 에 저장한다. 이를 다시 IR 에 나누어 저장한다. opcode 엔 명령어인 LDA를, address 엔 주소인 x를 저장한다.
3. 이 둘을 CU 가 확인 및 판단한다. 이 때 x 는 주소값이므로 x 가 가리키는 값을 가져와야 하기 때문에 SP 에 있는 8 을 MAR 에 저장한다. 8 을 MBR 로 읽어와서 AC 에 저장한다.
5. 이렇게 x 의 입력이 끝나고 PC 를 1 증가시킨다.
6. ADD y 도 x 와 마찬가지고 IR 에 저장하고 CU 가 명령을 판단한다. 이때 y 역시 값을 가져와야하기 때문에 SP 에 있는 y 값 16 을 MAR 에 저장하고 이를 MBR 로 읽어온다.
7. 그 후 ALU 로 가서 LDA 8 과 ADD 16 을 하고 AC 에 다시 저장한다.
8. 그리고 STO y 앞에와 마찬가지고 MAR, MBR 을 거쳐 IR 에 저장한다.
9. 이 때 STO y 는 저장할 주소이므로 MAR 에 잠시 저장해둔다. 그리고 AC 에 있는 24 는 MBR 에 저장해둔다.
10. 그리고 이 둘을 메모리에 보내 STO y 에 24 를 저장한다.

1-(2). Instruction cycle (Fetching, Decoding & Execution)

CPU 의 실행 순서에는 Fetching, Decoding, Execute 가 있다.

1) Fetching : Instruction 을 IR 로 가져올 때까지의 단계이다.
CU 가 PC 를 보고 MAR 에 메모리가 찾아야할 주소를 올린다. 이후 메모리가 MAR 을 보고 내용물을 MBR 에 저장한 후 그 내용물을 IR 로 가져오는 과정을 말한다.

2) Decoding : 가져온 Instruction 의 opcode 를 해석하고 다음에 실행할 명령어를 확인 및 판단하는 단계이다.
CU 가 값을 AC 에 저장하면 opcode 가 해석해서 해야 할 일을 정한다.

3) Execution : 실행해야 할 명령을 다 실행시키고 난 후 PC 를 1 증가시키는 단계이다.

CU 가 ALU 에게 명령을 내리고 그 결과값을 AC 에 저장하고 PC 를 1 증가시키고 다음 명령을 실행한다.

이러한 CPU 를 관리해주는 것이 바로 OS(Operating System)이다.

2. OS(Operating System)

- OS 는 CPU(Process Management), Memory(Memory Management), Hard Disk(File Management), I/O Device 를 관리한다.
(형광) : 핵심 기능
- 즉, 위의 말한 것들과 연결되어 있는 자원은 관리가 필요한데 이것을 하는 것이 바로 운영체제(OS)이다.
- 운영체제는 여기서 여러 개의 process 가 공유하는 자원들을 어떻게 효율적으로 사용할 수 있는지를 해주는 것이다.(process, 사용자 관점)
- 운영체제는 크게 사용자가 외부에 사람, 내부에 process, 이렇게 2 가지가 있다.

2-(1). Process 와 통신(communication)

프로세스는 실행중인 프로그램이다. 즉 프로세스가 컴퓨터의 주인이며, 컴퓨터는 프로그램을 실행시키기 위한 도구이다.
- 거꾸로 프로그램은 계획이다.
즉, 컴퓨터 프로그램은 CPU 가 실행해야 하는 명령들의 집합이다.
Ex)
- ✧ 음악회 : 연주자
- ✧ TV : 방송국
- ✧ 교육 프로그램 : 선생님

> 주체가 존재해야 한다.

실행중인 프로그램에서, '실행'이란 프로그램을 더블클릭에서 loader 가 메모리에 올리고, 메모리에 올라가면 CPU 서비스를 받는 것이다.
이러한 프로세스를 생성하고 관리 및 삭제하는 것이 프로세스 매니저이다.
이때 하나의 프로세스와 또 다른 프로세스가 공유할 수 있는데 이것을 '통신' 이라고 한다.

통신을 배우기 전에 컨텍스트의 개념부터 알아야 한다.

a.컨텍스트

- CPU 가 여러 가지 일을 동시에 처리하기 위해 필요한 것이 컨텍스트이다.
- n 개의 디바이스가 따로따로 있을 때 이것들을 다 관리하려면, 컨텍스트를 디바이스에 놓으면 좋다. 확장성도 좋고 유연성도 좋아진다.
> 여기서 '컨텍스트'란 무언가의 주체가 쓰는 독립적인 Memory 이다.
- Memory 는 저장하려고 쓰는 장치인데, 저장한다는 것은 특정시점에서 상태를 유지하고 있다는 뜻이 된다. 즉, 상태는 특정시점이 나와야 한다.
new 를 통해서 메모리를 할당하고 값을 가지게 된다. 값을 가지게 됨으로써 특정시점에 있는 값이 상태가 되는 것이다.
Ex)

- ✧ 내가 배가 고프다 > 특정시점의 혈당의 값
- ✧ 나이가 25 살이다. > 모든 세포의 노화정도 값

*특정시점에서 무언가가 가지고 있는 값의 집합 : 상태

- 이렇게 독립적인 Memory 인 컨텍스트를 가진 객체가 데이터를 공유하는 것을 통신이라고 한다.
> 만약 같은 컨텍스트이면 아는 주소를 통해 데이터를 주고받기 때문에 통신이라고 하지는 않는다. 이때 공유가 가능하게끔 하는 것이 필요한데 그것을 네트워크라고 한다.

ex) 독립적인 라인 x=3, y=x 가 있을 때 x 라는 메모리를 통해 두 라인이 통신한다.
즉, 통신은 독립적인 원소들이 서로 공유 메모리를 통해서 데이터를 주고받는 것이다.

b. synchronization (동기화)

이렇게 두 개 이상의 프로세스가 데이터를 공유할 때, 수행하는 시점을 조절하는 것이 synchronization(동기화)이다.
A, B 두 개의 프로세스가 데이터를 공유한다고 가정하면, A 가 데이터를 쓴 후에 B 가 읽어가야 하는데, B 의 입장에서 A 가 데이터를 썼는지 안 썼는지 모르기 때문에 알려줘야 한다. 이렇게 A 가 먼저하고 알려주면 B 가 수행하는 것처럼 수행 시점을 조절해주는 것이다. 동기화문제가 생기면 deadlock(교착 상태)이 발생하며 교착 상태는 해결이 절대 안 되는 상태를 말한다.

2-(2). Process State/Life Cycle

-CPU 가 프로그램을 실행시킬 때, 그 프로세스의 진행상황에 따라 프로세스의 상태가 나뉘어진다.

new : 로더가 메모리에 올린 상태
ready : 프로세스가 메모리엔 올라왔는데 실행되기 전(서비스 전)
waiting : 일을 하는 중간에 잠깐 서있는 것.(서비스 중단)
running : 서비스 중
terminated : 서비스 완료

ex) 은행 안에 들어가면 new, 줄을 서면 ready, 서비스를 받으면 running, 중간에 잠깐 빠지고 다른 일이 끝나기를 기다리면 waiting, 일 다 끝나고 은행 나오면 terminated 이다.

일반적으로 여러 개의 프로세스가 동시에 실행되고 있다고 생각되지만 CPU 가 프로세스보다 매우 빠르기 때문에 그런 것이며 실제로는 순서대로 실행된다. 그렇기 때문에 CPU 의 서비스를 받는 프로세스와 서비스를 기다리는 프로세스가 있다.
여러 개의 프로세스가 실행되는 것을 multi-process(=multi-tasking)이라고 하는데 멀티프로세스를 공평하게 순서대로 실행하려면 time-slice 로 나누어 실행해야 한다.

a. context switching

1. 프로그램을 더블클릭하면 OS 가 알고 메모리에 올린 후에 실행하고 있다고 가정하자.
2. 이때 또 다른 프로그램을 더블클릭해서 메모리에 올렸다고 한다. (= 멀티프로세스)
3. 두 개의 프로그램에 OS 가 가서 실행해줘야 하는데 그것을 time-slice 로 나눠서 한다. 실행할 순서를 OS 가 ready queue 에 적어 놓는다.
4. 처음에 실행(Running)하던 프로그램의 time-slice 가 끝나면 하던 것을 저장하고 다음 프로그램 실행으로 넘어가는데 이것을 context switching 이라고 한다.
5. context switching 이되면 첫 번째 프로그램은 다시 ready 상태로 가고 두 번째 프로그램이 running 상태가 된다.
time-slice 가 끝난 프로그램을 저장하는 저장 공간을 PCB(Process Control Block)이라고 한다.
6. 두 번째 프로그램을 실행하다가 두 번째 프로그램도 time-slice 가 끝나 다시 첫 번째 프로그램으로 돌아오면 PCB 에 저장했던 것을 다시 복구하여 이어서 실행한다.
7. 이후 다 끝나면 다시 레지스터를 다 저장하고 다음 프로그램 실행으로 넘어간다.

이렇게 프로세스 진행을 하다가 time-slice 가 끝났다고 하거나 혹은 중간에 마우스나 키보드 입력, 혹은 프린트 명령어가 들어오는 경우가 있는데 이러한 것을 interrupt 라고 한다.

3. Interrupt

인터럽트는 프로그램이 수행되고 있는 동안에 어떤 조건이 발생하여 수행중인 프로그램을 일시적으로 중지시키게 만드는 것이다.

ex)
1. 실행 중에 프린트하라는 인터럽트(I/O interrupt)가 발생하면 이거를 잠시 wait queue 로 보낸다.

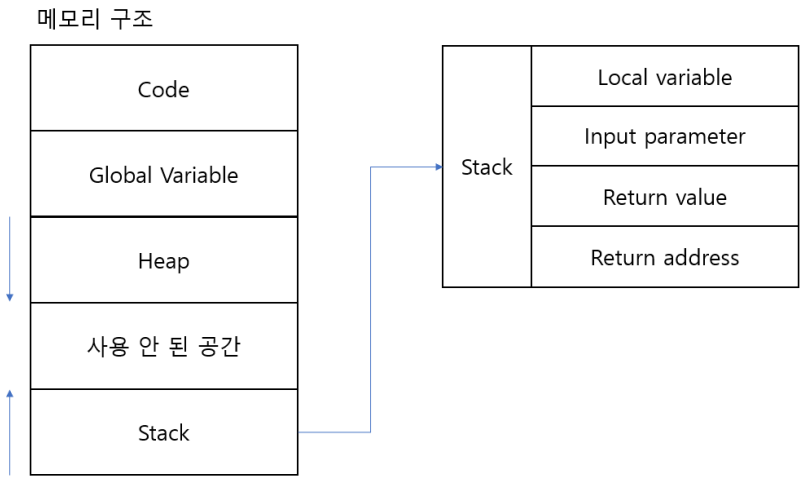
2. 프린트가 OS 에게 프린트 끝난다고 말해서 OS 를 호출하면 다시 원래 있던 자리인 ready queue 로 복귀한다. 그리고 원래 실행하던 것을 이어서 실행한다.
* 이때 인터럽트가 끝났다고 말해주는 것도 인터럽트이며, 일종의 프로세스이다.
> 이렇게 ready queue 에서 wait queue 로, wait queue 에서 ready queue 로 이동시켜주는 것이 바로 scheduling queue 이다.
3. 이렇게 모든 일이 끝나서 프로세스가 죽는 것을 terminated 라고 한다.

Ex) 메모리가 OS에게 어떠한 일을 부탁하는 것도 인터럽트의 예시이다.
1. OS 만 메모리들의 위치를 알기 때문에 OS 와 메모리가 통신하려면 공유메모리가 필요하다. 그래서 OS 가 편지통 메모리(인터럽트)를 만들어 놓고 다른 메모리들은 이 편지통 메모리의 위치를 알고, OS 에게 할 부탁을 넣어 놓는다
2. CPU 인터럽트가 들어온 것을 확인하면, context switching 하고 인터럽트를 수행한다.
> 인터럽트를 처리하는 루틴들 하나하나를 프로세스라고 봐도 되기 때문에 OS 에는 수많은 프로세스들이 작동하고 있는 것이고 OS 는 n 개의 프로세스가 안에서 돌고 있는 것이다.

- 인터럽트가 들어왔는지 확인하는 방법은 CPU 가 일을 한 번 수행하고 상태를 확인하고 다시 일을 한 번 하고 상태를 확인하는 반복실행을 하는 것인데, 이렇게 내부 상태에 대해 적혀 있는 것이 바로 status register 이다. status register 는 I/O interrupt 가 발생했는지를 루핑 돌면서 확인한다. interrupt 가 들어오면 status register 가 확인한다.
- 또한 인터럽트가 들어왔을 때 어떤 인터럽트인지 대응하는 값을 찾아 명령을 처리하는 것을 ISR(Interrupt Service Routine = Interrupt Handler Routine)이라고 한다.

3-(1). Memory

Memory 의 구조는 Code Segment 와 Data Segment 로 나눌 수 있으며, Data Segment 는 다시 Heap 과 Stack 으로 나눌 수 있다.



- Heap 은 프로그래머가 new 라는 명령어를 써서 메모리 공간이 할당되는 곳으로 동적 메모리가 할당되는 곳이다. 즉, 프로그램이 실행되면서 메모리가 할당된다는 의미이다. New를 통해 선언될 때마다 heap 영역의 아래로 데이터가 쌓인다. 하지만 생성된 객체가 사용되지 않는다면 쓰레기로 간주하여 garbage collector 가 삭제시킨다.

- Heap 과는 다르게 Stack 에는 이미 크기가 정해진 정적 메모리가 저장되며, 전역 변수와 지역 변수가 저장된다. Stack 에 저장된 변수는 선언된 함수 내에서만 사용이 가능하기 때문에 함수를 벗어나면 저장된 변수가 사라진다. 그때 늦게 들어온 데이터가 먼저 삭제된다(last in Frist out). 그렇기 때문에 Stack 의 메모리는 활발하게 공간이 찼다가 비어졌다를 반복한다.

-Stack 에 저장되는 또 다른 것은 activation record(활성화 레코드)가 있는데, OS 가 메인함수를 실행하면 생성되는 것이다. 활성화 레코드는 local variable(함수 내의 지역변수), input parameter, return value(parameter 를 통해 얻은 값 반환), return address(함수가 호출된 곳의 다음 주소)를 저장한다.

```
Ex)   function(int a) {//input parameter
          int x; // local variable
          return x; // return address
}
```