컴퓨터 하드웨어의 구성

1. CPU(중앙 처리 장치)

연산을 담당하는 장치

1. 메인 메모리

램이라는 저장장치로 구성되며 컴파일이 완료된 프로그램 코드가 올라가서 실행되는 영역 즉, 프로그램 실행을 위해 존재하는 메모리

1. 입/출력 버스

컴퓨터를 구성하는 구성요소 사이에서 데이터를 주고 받기 위해 사용되는 경로

데이터의 종류와 역할에 따라 나뉜다.

1. 어드레스 버스
2. 데이터 버스
3. 컨트롤 버스

CPU에 대한 이해

ALU, 컨트롤 유닛, 레지스터, 버스 인터페이스 + 클럭

1. ALU

CPU 내부에서 연산을 담당하는 블록

사칙연산 같은 산술연산

AND OR 같은 논리연산 수행

1. 컨트롤 유닛

프로그래머가 작성한 프로그램을 컴파일하면 실행파일이 생성되는데 이 실행파일에는 CPU에게 일을 시키기 위한 명령어 들이 저장되어 있다.

컨트롤 유닛은 명령어를 해석하고, 그 결과에 따라 적절한 신호를 CPU의 다른 블록에 보내는 일을 한다.

ALU는 단순 산술,논리 연산만 할 줄 알기 때문에 누군가가 명령어를 해석해 줘야 하는데 그것이 컨트롤 유닛이다.

1. 레지스터

레지스터란 CPU 내부에 존재하는 2진 데이터를 저장하기 위한 저장장치이다.

CPU의 종류에 따라 개수와 형태가 다양하다 가장 기본적인 레지스터는 8개가 있다.

CPU내에서 임시적으로 데이터를 저장하기 위한 조그만 메모리 공간이 필요한데 이를 레지스터라 부른다.

* 이런 메모리 공간이 왜 필요한가?

만약 CPU에 명령어가 들어왔는데 ALU와 컨트롤 유닛이 작업중에 있다면?

명령어를 대기시키기 위해 임시로 명령어를 저장할 공간이 필요하기 때문이다.

1. 버스 인터페이스

서로 데이터를 주고 받기 위한 매개체 역할을 해준다.

이 버스에 접근해서 데이터를 전송하기도 하고 입력 받기도 하는데 버스나 비행기도 이용방법을 알아야 탑승이 가능한 것처럼 I/O버스의 통신방식을 이해하지 못하면 데이터를 보낼 수도, 받을 수도 없다.

즉, CPU 내에는 I/O 버스의 통신방식을 이해하고 있는 무엇인가가 있어야 한다.

이것이 바로 버스 인터페이스이다.

이 버스 인터페이스 장치는 버스가 어떻게 데이터를 전송하는지, 그에 대한 프로토콜 혹은 통신방식을 알고 있는 녀석이다.

1. 클럭 신호

클럭 신호는 CPU를 구성하는 요소는 아니지만 타이밍을 제공하는 중요한 역할을 한다.

CPU의 클럭 속도가 높으면 초당 처리하는 명령어의 개수가 많아지므로 컴퓨터의 전체적인 성능은 좋아진다.

* 클럭 발생기에 의해 발생되는 클럭 신호는 CPU를 구성하는 요소 요소에 제공되어 이 신호에 맞춰 CPU가 일을 하는데 왜 CPU는 클럭 신호에 맞춰 일을 해야 하는가?

컴퓨터 시스템은 동기화가 필요하기 때문이다.

* 동기화가 필요한 이유는 무엇인가?

만약에 출력장치가 데이터를 가져가는 속도가 더 빠르다면, 출력장치는 이미 한번 가져간 데이터를 다시 가져 다가 출력하는 문제점을 드러낼 것이고, 연산장치가 더 빠르다면, 버퍼를 덮어쓰게 되어 연산결과의 일부가 출력되지 않는 문제점이 발생할 것이기 때문인데 이를 해결하기 위해 속도가 느린 장치의 장단에 맞춰 동기화 해준다.

그림그리기

폰 노이만의 아키텍처 (프로그램 내장 방식)

결과론 적으로 폰 노이만의 컴퓨터 구조는 “프로그램이 존재하고, 이 프로그램은 컴퓨터 내부에 저장되어 순차적으로 실행되어야 한다.”이다.

프로그램 실행 과정

전처리기

컴파일러

어셈블러

링커

실행파일의 생성 단계이다.

1. 전처리기에 의한 치환작업

전처리기는 #으로 시작하는 지시자의 지시에 따라 소스코드를 적절히 변경한다.

1. 컴파일러에 의한 번역

소스코드를 어셈블리 코드로 번역된다.

1. 어셈블러에 의한 바이너리 코드 생성

1과 0으로만 구성되는 코드이며 어셈블리 코드는 컴퓨터에 의해 실행 되기 전에 바이너리 코드로 번역 되어야 하는데 컴퓨터는 오로지 1과 0 만을 이해하기 때문이다.

* 어셈블러란 어셈블리 코드를 CPU가 이해할 수 있는 바이너리 코드로 바꾸어 주는 프로그램이다.

1. 링커에 의한 연결과 결합

프로그램 내에서 참조하는 함수나 라이브러리들을 하나로 묶는 작업을 한다고 할 수 있다.

이 과정이 끝나면 실제로 실행 가능한 실행파일이 생성된다..

만들어진 실행파일에는 컴퓨터에게 일을 시키기 위한 명령어가 들어 있다.

그림그리기

명령어들은 CPU에 의해 순차적으로 실행되는데, 메모리상에서 실행되는 것이 아니라 CPU 내부로 하나씩 이동한 다음 실행하게 된다.

이는 3단계로 진행되는데

1단계 : Fetch -> 메인 메모리에 존재하는 명령어를 CPU로 가져오는 작업

2단계 : Decode -> 가져다 놓은 명령어를 CPU(컨트롤 유닛)가 해석하는 단계인데 쉽게 말하면 무슨 일을 하라는 명령어인지 분석하는 단계이다.

3단계 : Execution -> 해석된 명령어의 명령대로 CPU(ALU)가 실행하는 단계이다.

문제

1. 명령어 실행을 위해서 제일 먼저 하는 일이 Fetch인데 어떤 경로를 통해서 명령어의 Fetch가 진행되는가?

?

1. 명령어를 CPU 안에 가져다 놓을 때 어디에 저장하는가?

IR(instruction Register)에 저장된다.

1. CPU 안에 가져 다 놓은 명령어는 Decode 단계에서 해석되는데, 이는 CPU안에 존재하는 누구에 의해서 진행되는가?

컨트롤 유닛이다.

1. 마지막 단계인 Execution은 누구에 의해 진행되는가?

산술 및 논리 연산을 하는 Execution의 주체는 ALU이다.

데이터 이동의 기반이 되는 버스 시스템

데이터의 종류에 따라 어드레스 버스, 데이터 버스, 컨트롤 버스로 구성된다.

45p 그림

1. 데이터 버스

데이터를 이동하기 위해 필요한 버스로 써 여기서 말하는 데이터는 명령어나 피연산자가 될 수 있다.

1. 어드레스 버스

주소값을 이동하기 위해 필요한 버스인데 그렇다면 왜 주소값을 이동해야만 하는가?

만약 CPU가 0x1234번지에 저장되어 있는 데이터 4바이트를 읽으려 한다고 가정해보자. 그렇다면 메모리 영역에 주소값 0x1234를 먼저 전달해야 하기 때문이다.

이 때 사용되는 것이 어드레스 버스이다.

1. 컨트롤 버스

CPU가 원하는 바를 메모리에 전달할 때 사용된다.

쉽게 말해 CPU와 메모리가 서로 특별한 사인을 주고받는 용도로 사용 되는 버스이다.

그렇다면 왜 특별한 사인이 필요한가?

CPU는 메모리로부터 데이터를 가져오기도 하지만 반대로 메모리에 데이터를 저장하기도

한다 그렇다면 둘 사이에는 보낼 것인지 받을 것인지에 대한 사인이 필요하게 된다.

사례적으면서 다시생각해 보기

문자셋의 종류와 특성

1. SBCS (Single Byte Character Set)

문자를 표현하는 데 있어서 1바이트만을 사용하는 방식이다.

1. MBCS (Multi~)

다양 한 바이트 수를 사용해서 문자를 표현하는 방식으로 어떤 문자는 1바이트로 표현하고 어떤 문자는 2바이트로 표현하는 방식인데 이 때문에 유니코드는 MBCS방식이라고 헷갈릴 수 있지만 아니다 아스키코드에서 정의하는 문자를 표현할 땐 1바이트 유니코드일 땐 2바이트로 처리한다.

1. WBCS(Wide~)

유니코드는 WBCS에 해당하는 데 모든 문자를 2바이트로 처리한다.

Direct 모드와 InDirect 모드

Direct Addressing Mode

- 메모리에 있는 값을 그대로 레지스터로 LOAD하는 모드

Indirect Addressing Mode

- 메모리에 있는 값이 주소값이며, 그 주소값을 참조해 메모리에 있는 값을 레지스터로 LOAD하는 모드

Direct모드의 문제점

Direct 모드에서는 하나의 명령어에 여러 정보를 담다 보니 표현하는 데이터 크기에 제한이 따른다는 문제점이 있는데 왜냐하면 Direct모드로 메모리를 접근할 경우, 할당 된 비트 수 안에서 표현 가능한 범위의 메모리 영역만 접근이 가능하기 때문이다.

Indirect 모드

메모리 접근 방법에 다양성을 부여한다.

명령어에서 지정하는 번지에 저장된 값을 주소 값으로 참고하여

<추후에 그림 그려보며 이해해보기>

프로세스

프로세스란 간단하게 실행 중에 있는 프로그램을 의미한다.

게임을 실행하기 위해 exe파일을 실행하게 된다면 이 순간 클릭한 프로그램의 실행을 위해 메모리할당이 이뤄지고, 이 메모리 공간으로 바이너리 코드가 올라가게 된다. 이순간부터 프로그래므은 프로세스라고 불린다.

메모리의 영역

1. 정적 세그먼트
2. Code

제어문, 함수 , 상수들이 저장되는 영역

작성한 코드가 들어가는 부분으로 프로세스가 종료될 때까지 유지된다.

1. Data

초기화 된 데이터는 data에 저장되고

초기화 되지 않은 데이터는 BSS(Block Stated Symbol)에 저장된다.

data영역이 런타임 이전에 초기화 하는 것이라면, bss는 런타임 이후 초기화 하는 것

이 영역 또한 프로세스의 종료까지 유지된다.

2. 동적 세그먼트

1. Heap

프로그래머가 동적으로 사용하는 영역으로써 malloc, free 또는 new, delete에 의하여

할당, 또는 반환되는 영역이다.

2. Stack

지역변수, 매개변수, 복귀 번지 등이 저장되어 있는 프로그램이 자동으로 사용하는

임시메모리로써 함수 호출 시 생성, 종료시 반환된다. LIFO정책을 사용한다.

프로세스의 스케줄링과 상태 변화

Q : CPU는 하나인데, 어떻게 여러 개의 프로그램이 동시에 실행 가능한 것인가?

A : 프로세스의 스케줄링

하나의 CPU가 여러 개의 프로세스를 번갈아 가면서 실행해야 하는데 다행히도 CPU는 아주 빠르기 때문에 하나의 CPU가 여러 프로세스를 고속으로 번갈아 가며 실행시킬 경우, 프로그램 사용자들은 CPU가 동시에 여러 개의 프로그램을 실행시킨다고 느끼게 된다.

결국 멀티 프로세스 운영체제에서 여러 개의 프로세스가 실행되는 것 처럼 보이는 이유는 여러 개의 프로세스들이 CPU 할당 시간을 나누기 때문이다.

스케줄링의 기본 원리

프로세스의 CPU 할당 순서 및 방법을 결정짓는 일을 가리켜 스케줄링 이라하며, 이 때 사용되는 알고리즘을 가리켜 스케줄링 알고리즘 이라 한다. 또 한 스케줄링 알고리즘을 적용해서 실제로 프로세스를 관리하는 운영체제 요소를 가리켜 스케줄러라 한다.

스케줄러는 소프트웨어적으로 구현되어 있는 요소로서, 운영체제를 구성하는 중요한 요소 중에 하나이다.

멀티 프로세스

일반적으로, 프로그램이 실행되는 과정에서 많은 시간을 I/O에 할당하게 되는데 여기서 I/O라 함은 데이터의 입력 및 출력을 말하는 것인데, 파일 입출력만을 뜻하는 것이 아니라 네트워크 프로그램의 경우 데이터 송수신 또한 I/O에 해단된다.

프로세스의 상태 변화

상태변화 그림

상황 1 : Start에서 Ready 상태로의 전이를 보여준다.

Start는 프로세스의 생성을 의미하며 프로세스는 생성과 동시에 Ready 상태로 들어간다.

여기서 Ready 상태에 있는 프로세스는 CPU에 의해 실행되기를 희망하는 상태이다.

여기서 “프로세스가 생성되었으면 바로 실행 되어야 하는 것이 아닌가?”라는 의문이 들 수 있는데 멀티 프로세스 운영체제이기 때문에, 현재 CPU에 의해서 임의의 프로세스가 실행 중에 있을 것이고 새로운 프로세스가 생성되었다고 해서 무조건 현재 실행하던 프로세스를 멈출 수는 없기 때문에 Ready상태로 두는 것이다.

상황 2 : Ready 상태에서 Running 상태로의 전이를 보여준다.

Ready 상태에 있는 프로세스들은 스케줄러에 의해서 관리되는 프로세스 들이다. 스케줄러는 Ready 상태에 있는 프로세스 중 하나를 선택해서 CPU에 의해 실행될 수 있도록 한다.

결국 Ready상태에 있던 프로세스 중 스케줄러에 의해 선택된 프로세스는 Running 상태가 되어 실행되는 것이다.

상황 3 : Running 상태에서 Ready 상태로의 전이를 보여준다.

일의 중요도에 따라 실행 순서를 달리하기 위해서 프로세스에는 우선순위라는 개념이 존재하는데 예를 들어 프로세스 a의 운선순위가 프로세스 b의 우선순위보다 높도록 프로그램이 되어있고 현재 b가 실행 중 이라고 생각해 보자.

여기서 갑자기 b보다 우선순위가 높은 a가 생성되어 Ready상태가 되었다면 일반적으로 스케줄러는 프로세스 b의 실행을 멈추고, 우선순위가 높은 a를 실행시킨다. 이때 b는 Ready 상태가 되어서 a가 양보해 주기만을 기다린다.

상황 4 : Running 상태에서 Blocked 상태로의 전이를 보여준다.

실행 중에 있는 프로세스가 실행을 멈추는 Blocked 상태로 들어가는 것 인데

일반적으로 데이터 입출력에 관련된 일을 하는 경우에 발생한다.

프로그램 실행의 상당 부분을 데이터 입출력에 소모하게 되고 이 시간에는 CPU에 의해서 프로세스가 더 이상 실행될 수 없다.

따라서 데이터 입출력을 진행 중인 프로세스는 잠시 Blocked 상태로 만들고 Ready 상태에 있는 프로세스 중 하나를 실행 시키는 것이 CPU를 효율적으로 사용하는 것이다.

상황 5 : Blocked 상태에서 Ready상태로의 전이를 보여준다.

Ready 상태는 스케줄러에 의해 선택되어 지금 당장이라도 실행 가능한 상태이고, Blocked 상태는 스케줄러에 의해서 선택될 수 없는 상태이다 .

그렇다면 입출력이 완료된 Blocked 상태에 있는 프로세스는 어떤 상태가 되어야 하겠는가?

당연히 다시 Ready 상태가 되어 스케줄러에 의해 선택되길 기다려야 한다.

컨텍스트 스위칭

앞서 멀티 프로세스 프로그램은 둘 이상의 프로세스 실행을 통해 CPU의 활용도를 높여 성능 향상까지 가져온다는 결론을 내리게 되었는데 이에 반해 단점도 존재한다.

실행 중인 프로세스의 변경은 시스템에 많은 부하를 가져다 주기도 하기 때문이다.

이를 이해하기 위해선 “CPU내에 존재하는 레지스터들은 현재 실행 중에 있는 프로세스 관련 데이터들로 채워진다.”라는 말을 이해해야 한다.

이를 이해했다면 한가지 고민에 빠질 것이다. 실행 중인 프로세스가 변경되면 CPU 내에 존재하는 레지스터들의 값이 변경되기 때문이다.

다른 프로세스가 실행 되기 전에 현재의 데이터들을 어딘가에 저장해야만 하는데 이 데이터들은 다음에 실행 될 때 이어서 진행할 경우 필요하기 때문이다.

그 뿐 아니라 실행되는 프로세스가 새롭게 실행된 프로세스가 아니라면 이전에 실행될 때 레지스터 들이 지니고 있던 데이터들을 어딘가에 저장해 놨을 것이다.

이 데이터들을 다시 레지스터에 복원 시켜 줘야만 한다. 그래야 프로세스 B의 작업이 멈춰진 부분에서부터 이어질 수 있다.

컨텍스트 스위칭이란 현재 실행 중인 레지스터 정보는 메모리에 저장하고, 새로운 프로세스 관련 레지스터 정보는 CPU의 레지스터에 복원 시키는 작업이다.

이 컨텍스트 스위칭은 시스템에 많은 부담을 주는데 이는 레지스터 개수가 많을수록, 프로세스별로 관리되어야 할 데이터 종류가 많을수록 더하다.

이것이 바로 멀티 프로세스 운영체제의 단점이다.