# CPU의 구조와 기능

\*CPU가 수행하는 세부적인 동작

1. 명령어 인출

-> 기억장치로 부터 명령어를 읽어온다.

2. 명령어 해독

-> 수행해야 할 동작을 결정하기 위하여 명령어를 해독한다.

3. 데이터 인출

-> 명령어 실행을 위하여 데이터가 필요한 경우에는 기억장치 혹은 IO장치로 부터

그 데이터를 읽어온다.

4. 데이터 처리

-> 데이터에 대한 산술적 혹은 논리적 연산을 수행한다.

5. 데이터 저장

-> 수행한 결과를 저장한다.

1~2 => 모든 명령어들에 대하여 공통적으로 수행

3~5 => 필요한 경우에만 수행

\*1.CPU의 기본 구조

산술논리연산장치(Arithmetic and Logical Unit : ALU)

레지스터 세트(Register set)

제어 유니트(Control unit)

ALU

=> 각종 산술 연산들, 논리 연산들을 수행하는 회로들로 이루어진 하드웨어 모듈

합, 차, 곱, 나눔 and AND, OR, NOT 연산 등.

레지스터

=> CPU 내부에 위치한 기억장치 (엑세스 속도가 가장 빠르다.)

but, 많은 수의 레지스터들을 CPU에 포함시키기 어려움 (적다)

제어 유니트

=> 프로그램 코드(명령어)를 해석하고 그것을 실행하기 위해 제어 신호들을

순차적으로 발생하는 하드웨어 모듈이다.

CPU 내부버스

=> 데이터와 제어 신호들의 전송 통로 (CPU내부에서만)

\*2.명령어 실행

명령어 사이클

=> 한 개의 명령어를 실행하는 데 필요한 전체 과정

명령어 인출, 명령어 실행

인출 사이클, 실행 사이클

-명령어 실행을 위해 기본적으로 필요한 CPU 내부 레지스터

1. 프로그램 카운터 (PC)

=> 다음에 인출될 명령어의 주소를 가지고 있는 레지스터.

각 명령어가 인출된 후에는 자동적으로 1이 증가되며, 분기명령어가 실행되는 경우에는

그 목적지 주소로 갱신된다.

2. 누산기(Accumulator : AC)

=> 데이터를 일시적으로 저장하는 레지스터

이 레지스터의 비트 수 => CPU에서 한번에 연산처리할 수 있는 비트 수(단어 길이)

3. 명령어 레지스터 (IR)

=> 가장 최근에 인출된 명령어가 저장되는 레지스터

4. 기억장치 주소 레지스터 (MAR)

=> 프로그램 카운터에 저장된 명령어의 주소가 시스템 주소 버스로 출력되기 전에

일시적으로 저장되는 주소 레지스터 => CPU외부 주소 버스 선들과 직접 접속됨

5. 기억장치 버퍼 레지스터 (MBR)

=> 기억장치에 저장될 데이터 혹은 기억장치로부터 읽혀진 데이터가 일시적으로 저장되는 버퍼 레지스터

==> CPU외부 데이터 버스 선들과 직접 접속된다.

\*2.1 인출 사이클

==> 기억장치의 지정된 위치로부터 명령어를 읽어오는 과정

CPU => 프로그램 명령어 => 프로그램 카운터가 가리키는 => 기억장치의 위치로부터 명령어를 인출해온다.

마이크로 연산 => CPU 클록의 한 주기 동안 수행되는 기본적인 동작

\*2.2 실행 사이클

==> CPU가 인출된 명령어 코드를 해독(decode)하고, 그 결과에 따라 필요한 연산을 수행한다.

CPU가 수행하는 연산들은 다음과 같다.

1. 데이터 이동 : CPU가 기억장치 간 혹은 CPU와 I/O 장치 간에 데이터를 이동한다.

2. 데이터 처리 : 데이터에 대하여 산술 혹은 논리 연산을 수행한다.

3. 데이터 저장 : 연산결과 데이터 혹은 입력장치로부터 읽어들인 데이터를 기억장치에 저장

4. 프로그램 제어 : 프로그램의 실행 순서를 결정한다.

LOAD => 기억장치에 저장되어 있는 데이터를 CPU 내부 레지스터인 AC로 적재하는 명령어

STA => AC 레지스터의 내용을 기억장치에 데이터 저장을 위한 명령어

ADD => 기억장치에 저장된 데이터를 AC 레지스터의 내용과 더하고, 그 결과를 다시 AC레지스터에 저장하는 명령어

JUMP => 분기(branch)명령어라고도 한다. 현재 PC내용이 가리키는 위치가 아닌 다른 위치의 명령어로 실행 순서를 바꾸도록 해주는 명령어

기계어

(연산 코드)(오퍼 랜드)

프로그램 실행 과정 : 인출 => 실행의 연속이다.

\*2.3 인터럽트 사이클

인터럽트란 ?

=> 대부분의 컴퓨터들은 프로그램 처리 중에 CPU로 하여금 순차적인 명령어 실행을 중단하고 다른 프로그램을 처리하도록 요구할 수 있는 메커니즘을 제공해주는데 메커니즘을 인터럽트라고 부른다.

=> 정상적인 프로그램 처리의 흐름을 방해하는 동작이나.. 정전과 같은 긴급한 상황에 대처하거나 외부 장치들과의 상호 작용을 위해서 반드시 필요한 기능이다.

인터럽트 요구가 들어오게 되면, CPU는 원래 프로그램 수행을 중단하고, 요구된 인터럽트를 처리해주기 위한 프로그램을 먼저 수행한다. ==> 이것을 인터럽투 서비스 루틴(Interrupt service routine : ISR)이라고 한다.

인터럽트에 대한 처리가 끝나면, CPU는 원래의 프로그램으로 복귀하여 그 수행을 계속한다.

이를 위해 CPU는 아래와 같은 동작들을 수행해야 한다.

1. 인터럽트 요구 신호가 들어오면, 다음에 실행할 명령어의 주소를 가리키는 현재의 PC내용을 스택에 저장한다. 이것은 인터럽틀 처리를 완료한 후에 복귀할 주소를 저장해두기 위한 절차이다.

2. 해당 인터럽트 서비스 루틴을 호출하기 위하여 그 루틴의 시작 주소를 PC에 적재한다. 이때 시작 주소는 인터럽트를 요구한 장치로부터 전송되거나 미리 정해진 값으로 결정된다.

==> 이 과정을 인터럽트 사이클이라고 한다.

하지만 AC에 저장된 값이 IRS로 인해 나중에 다시 수행할때 예상하지 못한 결과값이 나올 수가 있다.

그래서 스택에 레지스터 내용을 저장하였다가, 루틴 마지막 단계에서 원래 프로그램으로 복귀하기 직전에 다시 그들을 복원 시켜주는 절차가 필요하다

==> 다중 인터럽트

\*2.4 간접 사이클

=> 연산에서 사용될 데이터의 주소가 기억장치의 어떤 위치에 저장되어 있고, 명령어에 포함되어 있는 주소는 그 위치를 가리키고 있는 것이다. 이런 경우에 실행 사이클이 시작되기 전에, 그 데이터의 실제 주소를 기억장치로부터 읽어오는 과정이 먼저 수행되어야 한다. 이 과정을 간접 사이클이라고 한다.

인출 사이클과 실행 사이클 사이에 위치한다.

#명령어 파이프라이닝

CPU의 성능은 컴퓨터시스템의 프로그램 처리 시간에 직접 영향을 주기 때문에, 그 속도를 향상시키기 위하여 여러 가지 방법들이 사용되고 있다. 그 중 하나가 명령어 파이프라인이다.

명령어 실행 => 독립적으로 분할 => 동시 처리 => CPU성능 향상

\*3.1 2단계 명령어 파이프라인

인출 사이클, 실행 사이클을 독립적인 모듈(서로 분리되어 작성되는 성질)로 구성하여 각 모듈이 서로 다른 명령어를 동시에 처리하는 방법이다.

인출 단계, 실행 단계로 나눌 수 있다.

작동방식

첫번째 클록 주기 => 명령어 A를 인출한다.

두번째 클록 주기 => 명령어 A를 실행한다. 명령어 B를 인출한다.

(반복..)

실행될 명령어를 미리 인출하는 것 => 명령어 선인출, 인출 중복

이와 같이 명령어 실행 하드웨어를 두 단계로 분리시킨 것을 2-단계 명령어 파이프라인이라고 부른다.

\*2.3.2. 4-단계 명령어 파이프라인

2-단계 명령어 파이프라인에는 단점이 존재하는데 속도 향상을 위해서는 인출과 실행 시간이 같다는 조건이 필요한데 실제로 실행단계 속도가 인출단계 속도보다 빠르기 때문에 속도 향상의 조건에 위배된다. 그래서 좀 더 단계를 분할하여 처리 시간이 같아지도록 하는 파이프라인을 4-단계 명령어 파이프라인이라고 한다.

4단계로 나눌 수 있다.

* 명령어 인출(IF) : 명령어를 기억장치로부터 인출한다.
* 오퍼랜드 해독(ID) : 해독기를 이용하여 명령어를 해석한다.
* 오퍼랜드 인출(OF) : 기억장치로부터 오퍼랜드를 인출한다.
* 실행(EX) : 지정된 연산을 수행하고, 결과를 저장한다.

-T(파이프라이닝을 통한 전체 명령어 실행 시간)

T = k(파이프라인 단계 수) + (N(실행할 명령어의 수) – 1)

-C(파이프라이닝을 사용하지 않은 경우 전체 명령어 실행 시간)

C = k \* N

-Sp(파이프라이닝을 이용함으로써 얻을 수 있는 속도 향상)

Sp = C / T

하지만.. 때때로 파이프라인 단계를 거치지 않는 경우도 존재한다.

(예 : 조건 분기, 기억장치 충돌, 오퍼랜드 인출 불필요 등)

이러한 경우에는 파이프라이닝을 사용하여도 그에 비례한 속도향상을 얻기 힘들 수 있다.

그래서 사용하는 것이 ‘슈퍼파이프라이닝(superpipelining)’이다.

* 분기 예측 : 분기가 일어날 것인지를 예측하고, 그에 따라 어느 경로의 명령어를 인출할 지를 결정하는 확률적 방법이다. 분기 결과들을 저장하여 두는 분기 역사 표(branch history table)를 참조하여 예측하는 방법이 가장 널리 사용되고 있다.
* 분기 목적지 선인출 : 조건 분기가 인식되면, 분기 명령어 다음 명령어 뿐만 아니라, 조건이 만족될 경우에 분기하게 될 목적지의 명령어도 함께 인출하는 방법이다. 이 방법의 사용을 위해서는 분기 목적지 명령어를 별도로 저장해둘 수 있는 기억 장소가 추가되어야 한다.
* 루프 버퍼 : 이 버퍼는 파이프라인 명령어 인출 단계에서 포함되어 있는 작은 고속 기억장치인데, 가장 최근에 인출된 일정 개수의 명령어들이 순서대로 저장되어 있다. 만약 분기가 발생하면, 하드웨어는 먼저 분기 목적지의 명령어가 버퍼에 있는지를 검사한다. 만약 있다면, 다음 명령어는 버퍼로부터 인출된다. 하지만 반복 루프의 크기가 크다면 버퍼 메모리에 명령어를 담지 못하는 경우가 생길 수도 있다. 그러므로 작은 반복 루프에 처리하기 좋은 방법이다.
* 지연 분기 : 프로그램 내의 명령어들을 재배치함으로써 파이프라인의 성능을 개선하는 방법이다. 즉, 분기 명령어의 위치를 적절히 조정하여 원래보다 나중에 실행되도록 재배치함으로써, 그로 인한 성능 저하를 최소화하는 것이다.

조건 분기 명령어를 사용하는 조건들은 CPU 내부의 ‘상태 레지스터(status register)’에 저장되어 있다. 각 조건의 상태를 나타내는 비트를 ‘플래그(flag)’라고 한다.

-플래그 종류

1. 부호(S) 플래그 : 직전에 수행된 산술 연산 결과값의 부호 비트로 세트, 0은 양수 1은 음수

2. 제로(Z) 플래그 : 연산 결과값이 0(Zero)이면, 이 플래그가 1로 세트된다.

3. 올림수(C) 플래그 : 덧셈이나 뺄셈에서 올림스나 빌림수가 발생한 경우에 세트된다.

4. 동등(E) 플래그 : 두 수를 비교한 결과가 동일한 경우에 세트된다.

5. 오버플로우(V) 플래그 : 산술 연산 과정에서 오버플로우가 발생한 경우에 세트된다.

6. 인터럽트(I) 플래그 : 인터럽트 가능 상태이면 0, 인터럽트 불가능 상태이면 1로 세트

7. 슈퍼바이저(P) 플래그 : CPU가 슈퍼바이저 모드와 사용자 모드중의 어느 모드에서 프로그램을 실행하고 있는지를 나타내는 데에 사용된다.

S Z C X(사용되지 않는 비트) E V I P

8비트 상태 레지스터의 플래그들

\*2.3.3 슈퍼스칼라

슈퍼스칼라는 CPU의 처리 속도를 더욱 높이기 위하여 내부 두개 혹은 그 이상의 명령어 파이프라인들을 포함시킨 구조를 말한다. 다시 말해 파이프라이닝은 명령어 파이프라인 단계를 여러개로 나눈 것이라면 슈퍼스칼라는 파이프라인 자체를 여러 개로 만든것이다.

주의할 점이 있다면 서로 데이터 의존성이 존재하지 않아야 동시에 사용이 가능하다.

-T(m)(m-way 슈퍼스칼라 프로세서가 N개의 명령어들을 실행하는 데 걸리는 전체 시간)

T(m) = k(파이프라인 단계) + (N(명령어 개수) – m(m-way)) / m

-Sp(슈퍼스칼라 프로세서로 얻을 수 있는 속도 향상)

Sp = T(1) / T(m)

CPU가 동시에 처리할 수 있는 명령어들의 수가 늘어나게 되면, 기억장치와 레지스터에 대한 충돌이 발생하지 않을 수 있도록 하드웨어도 그만큼 추가되어야 한다.

대부분 고성능 프로세서들이 슈퍼스칼라 구조를 채택하고 있다.

\*2.3.4. 듀얼-코어 및 멀티-코어

CPU 코어란?

* 기존의 CPU칩에 포함되던 하드웨어중 명령어 실행에 반드시 필요한 핵심 모듈들을 말한다. 예를들면 ALU, 명령어 파이프라인들로 이루어진 슈퍼스칼라 모듈, 레지스터 세트 등이 있다.

CPU 코어를 여러 개를 하나의 칩에 놓은 것 => 멀티코어 프로세서

멀티-태스킹

* 하나의 프로그램에 포함된 명령어들을 명령어 파이프라인의 수만큼씩 인출하여 동시에 실행하는 슈퍼스칼라 프로세서와는 다르게 독립적인 처리가 가능한 태스크 프로그램들이 CPU 코어들에 의해 동시에 처리되는 것.

스레드

* 독립적으로 실행될 수 있는 최소 크기의 프로그램 단위

멀티 스레딩

* 하나의 CPU코어가 다수의 스레드를 동시에 실행하는 기법

각 코어는 스레드를 처리하는 중의 시스템 상태와 데이터 및 주소 정보를 저장하기 위하여 프로그램 카운터(PC), 스택 포인터(SP), 상태 레지스터, 데이터 레지스터, 주소 레지스터 등으로 이루어진 ‘레지스터 세트(RS)’를 가지고 있다. 코어가 두개의 RS를 가지고 있으면, 스레드를 두개씩 동시에 처리할 수 있다. 두 개의 스레드들이 CPU 코어의 하드웨어 자원들을 공유하면서 동시에 처리되며, 스레드의 상태는 각 RS에 저장된다.