HW #1 2017069598 박상지

\*본 과제는 교수님의 영상과 pdf를 참고하여 과제를 하였습니다.

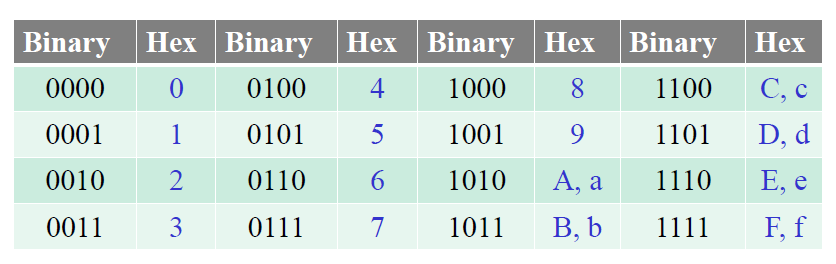
topic 0-3 : data

컴퓨터라는 불리는 기계는 기본적으로 이진 형태의 숫자로 데이터(자료)를 표현하고 저장한다. 0-3 PDF 에서는 컴퓨터에서는 데이터(정보)를 어떻게 표하는지, 그리고 정보를 어떻게 다루는지에 data 와 processing에 대해 배운다.

data는 기본적으로 과학적인 컴퓨팅, 즉 어려운 방정식을 해결하기 위해 다루어 졌으며 데이터는 기본적으로 숫자를 가리켰고 integer, floating-point numbers으로 데이터를 표현하였다. 2차 세계 대전이 끝난 후 컴퓨터는 방대한 데이터를 저장하고 계산하는 것에 대해 중점적으로 다루어 졌으며, characters or text, 즉 문자를 저장하는 게 많아 졌다. 1990년에는 인터넷이 생김으로써 지금까지의 방대한 데이터와 오디오, 이미지, 비디오와 같은 다수의 데이터가 생겼으며 현재는 지금까지의 데이터를 오늘날에 어떻게 사용하면 효율적인지에 대해 중점을 두고 있다.

Number Systems

수의 체계에서는 잠재적인 자리의 값이 있는데, 390의 십진수를 보면 0은 10^0의 위치이고 9는 10^1의 위치 3은 10^2의 위치를 나타낸다. 이진수도 십진수와 같은 수의 체계로 잡혀 있으며, 단지 0과 1로 표현을 할 뿐이다. 컴퓨터에서는 대부분의 하드웨어가 트랜지스터로 이루어져 있으며, 트랜지스터는 on/off를 0과1로 표현을 한다. 즉 컴퓨터는 이진수로 데이터를 표현을 한다. 하지만 이진수로 표현되는 데이터를 사용자가 사용하기에는 읽기/쓰기가 불편하기 때문에 편의를 위해서 사용자는 Hexadecimal Notation 즉 32-bit 메인 메모리를 읽을 때에는 좀더 쉽게 4bit를 나누어서 Hex로 표현을 한다.

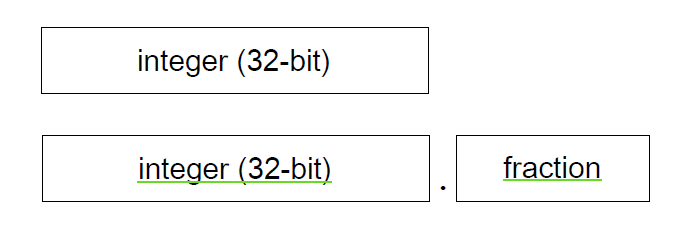


Negative Numbers

그렇다면 컴퓨터는 음수의 값을 어떻게 표현을 할까? 컴퓨터는 2의 보수를 사용하여 음수의 값을 표현을 하며, 2의 보수를 사용하여 ALU 연산을 더욱 빠르게 할 수 있다.

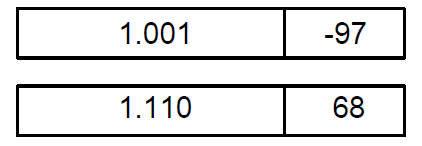
Fixed-point representation

이진수 체계를 사용하는데 적은 bit값을 사용하여 유효 값과 소수 값을 표현을 한다. 하지만 소수 값까지 표현을 할 경우 유효 값의 표현 범위가 상대적으로 감소하기 때문에 하드웨어를 추가하여 표현을 한다. 정수 값은 32-bit으로 표현을 하고 나머지 소수 값을 하드웨어를 추가하여 10bit를 소수 값으로 표현한다. 하지만 10bit는 상대적으로 작기 때문에 우리는 그보다 큰 값을 연산을 할 경우 근사값으로 표현을 한다. 그래서 컴퓨터의 실수 연산을 근사 연산이라고 표현한다.



Floating-Point Numbers

우리는 컴퓨터에서 절대 값이 매우 크거나 작을 경우 floating-point으로 표현을 하는데 32-bit를 예를 들면 유효 숫자로 표현할 수 있는 것은 제외한 나머지 값을 지수 값을 표현하는데 저장한다.



Memory Model

단일 메모리의 주소는 8bit(1byte)으로 표현되며, 32-bit memory일 경우에는 단일 메모리 4개가 4열 종대로 이루어져 있다. 그러므로 시작 주소 값이 0000번지이면 다음 주소는 0004이다. 만약 16-bit memory일 경우에는 2열 종대이기 때문에 다음 주소는 0002번지이다.

different types of data

텍스트, 오디오, 이미지, 비디오 같은 멀티 미디어의 데이터를 저장할 때에도 이진수의 방식으로 저장되어진다.

텍스트일 경우에는 초기에는 7bit로 128개의 형태들을 대표하는 ASCII 로 표현을 하였지만 나중에는 ASCII의 확장된 8bit 로 다른 언어를 그룹을 지어서 수용을 하였고, 오늘날에는 Unicode를 사용하여 세계의 언어를 표현하는 국제화를 일으켰다. Unicode의 문자들은 1byte에서 4byte로 되어있다.

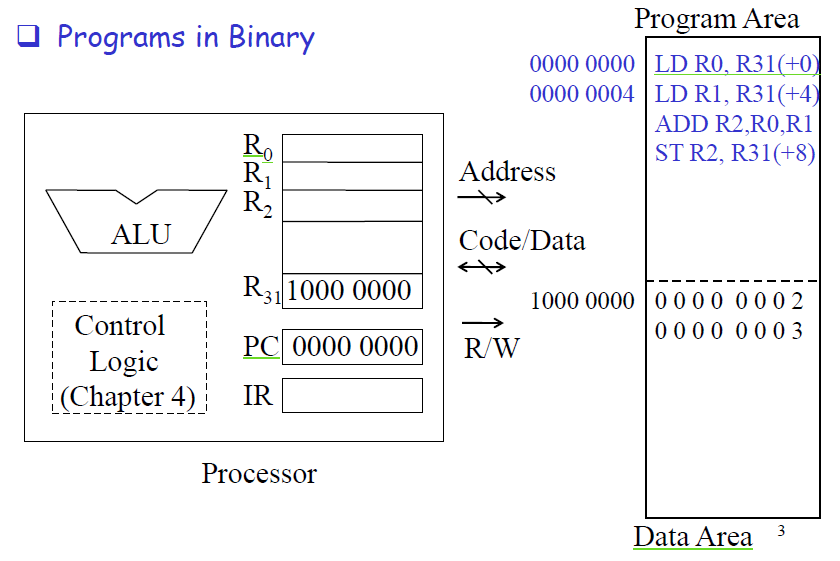
오디오를 저장할 때에는 원음을 일정한 시간의 간격마다의 값을 저장하여 그 사본을 잘 만들어서 원음으로 표현을 하였다.

이미지를 저장할 때에는 색깔을 24 bit per pixel로 저장하여 8bit씩 빨강 초록 파랑으로 나타내어 Full HD를 예로 들면 가로, 세로 1920\*1080 pixel에 3byte(24bit per pixel (red, green, blue))를 곱하여 이미지를 표현하였다.

0-4 topic : To start Program Execution

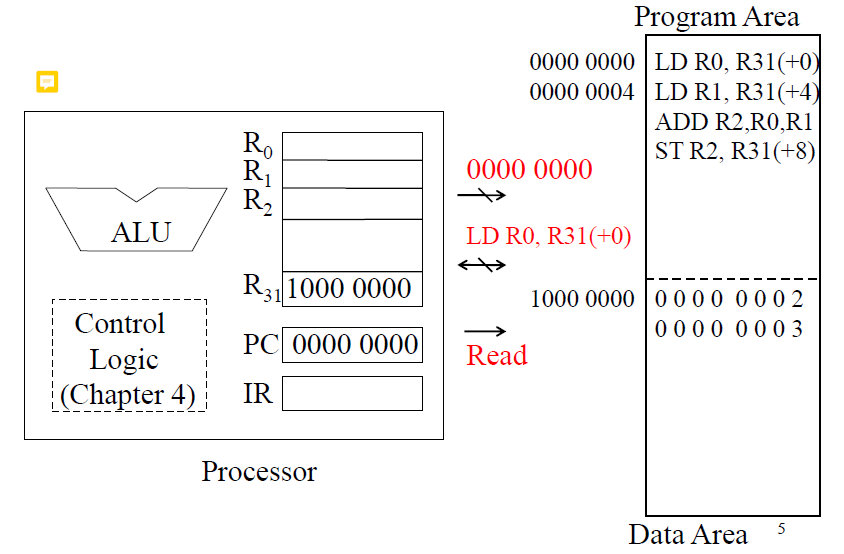
프로세서에서는 ALU와 Control Logic이 존재하면 32의 레지스터와 실행 주소를 가리키는 PC(program counter)가 있으면 PC가 가리키는 주소의 명령을 저장하는 IR(instruction register)가 존재한다. 메모리에는 program area라는 32-bit의 4열 종대로 놓여져 있는 명령어들이 존재하고 data area라는 핵사(HEX)로 표현한 데이터 값들이 존재한다.

컴퓨터는 명령을 수행할 때 3단계로 이루어진다.

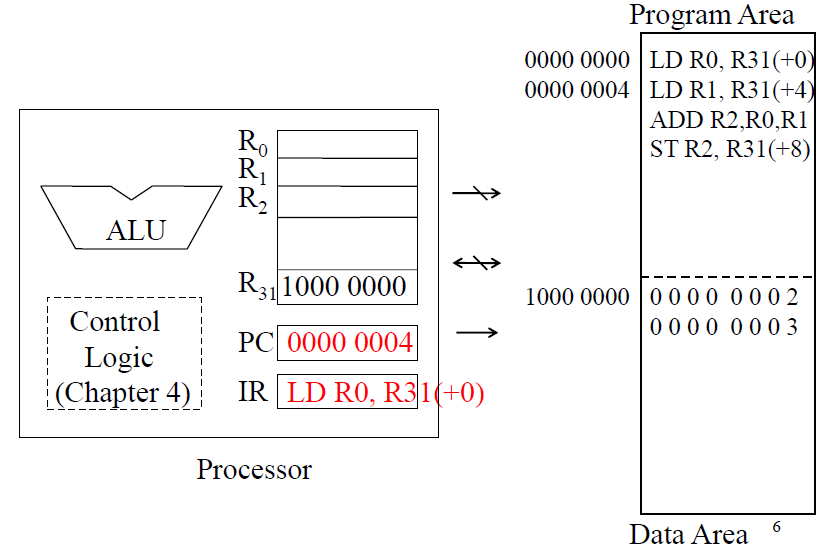


Fetch-Decode-Execute

1. Instruction Fetch(IF)

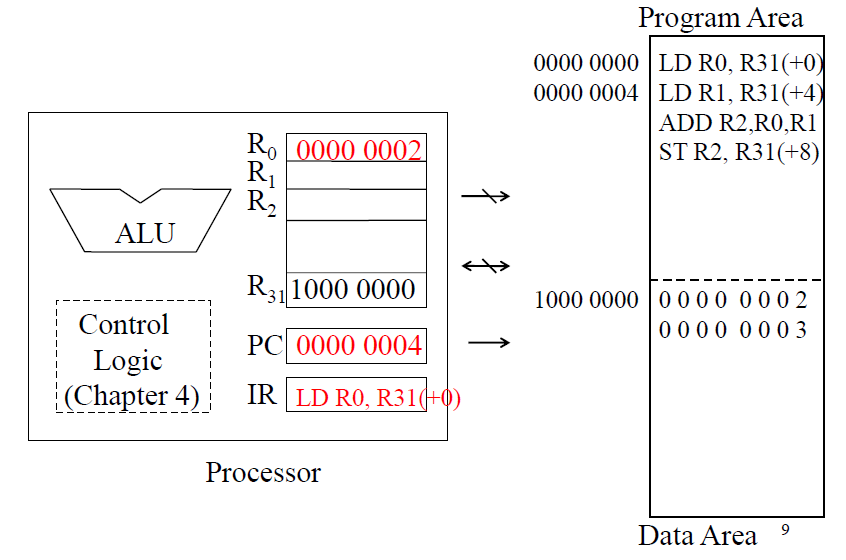


1번째 단계 fetch에서는 PC가 가리키는 0번지를 먼저 확인을 한다. 0번지에는 현재 LD R0, R31(+0)이라는 명령을 IR에 저장을 한다. 그 다음에 PC값을 4bit을 증가시켜준다.



1. Instruction Decode(ID)

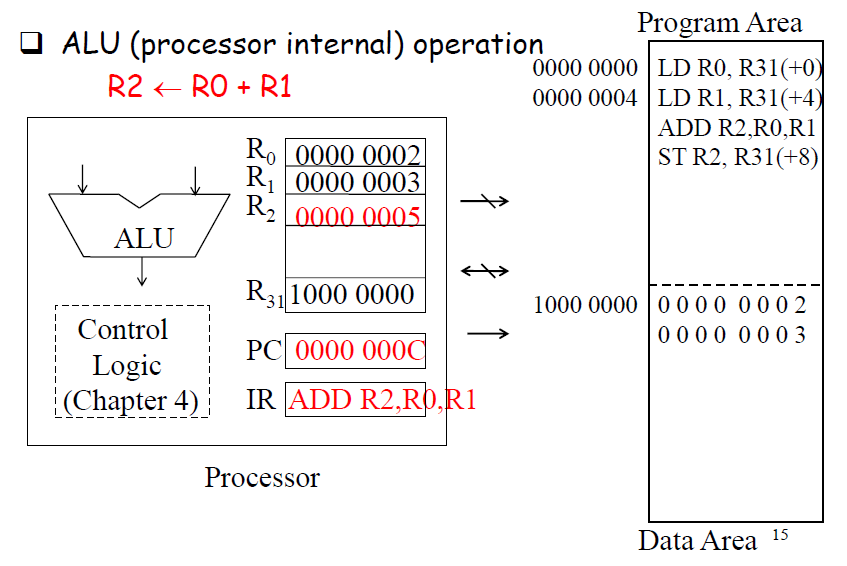
decode 단계에서는 IR에 있는 명령어들을 해석하는 단계이며, 현재 IR에는 LD R0, R31(+0)이다. R31 레지스터가 가리키는 주소의 +0만큼 증가시켜 LD(메모리 읽기)하여 R0 레지스터에 저장을 한다.



여기서 LD, R0, 10000000 명령어를 사용하지 않고 LD, R0, R31(0) 명령어를 사용한 이유는 10000000 명령어를 사용하면 해당 명령어를 32bit로 사용해야 하고 이것은 RISC 방식과 다르며, 만약 이렇게 쓴다면 32bit을 2word로 나누어서 사용해야 하고 그러면 fetch를 2번 반복해야 하기 때문에 해당 명령어를 사용하는데 컴퓨터의 성능이 감소하게 된다.

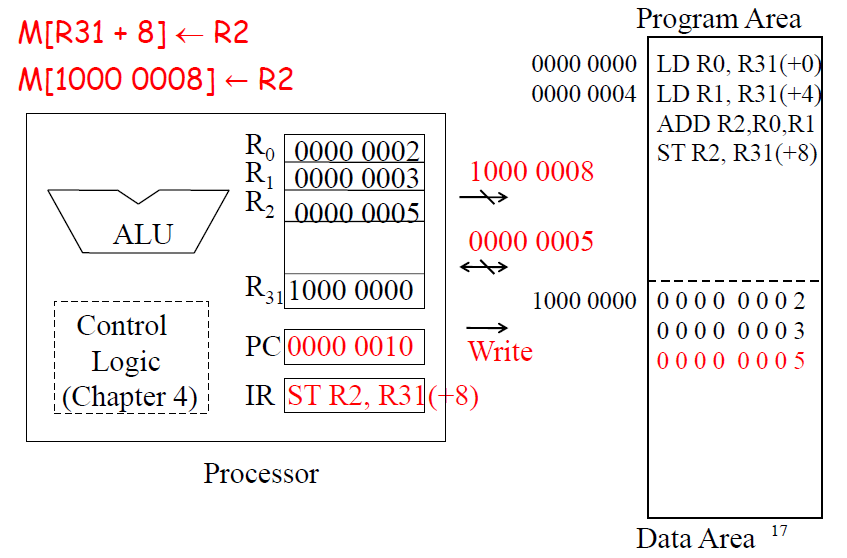
1. Instruction fetch and decode

0번지와 4번지의 명령어들을 수행하면 ADD R2, R0 , R1 명령어를 fetch하게 되는데 해당 명령어를 fetch하고 IR에 저장하고 PC의 값은 4bit을 증가시키고 ADD작업을 수행하기 위해서는 ALU 연산을 사용하여 R1 과 R2의 레지스터에 저장되어 있는 값을 ALU 연산에 넣어서 Control Logic을 수행하여 결과 값 5을 R2에 저장한다.



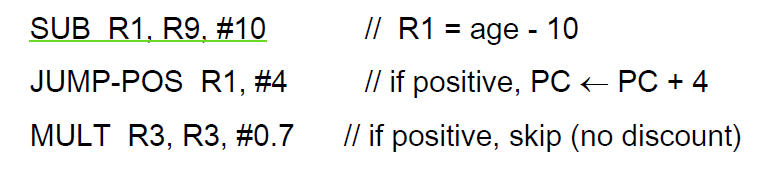
1. instruction fetch, decode, execution

마지막 명령 ST R2, R31(+8)을 가져와서 R31 레지스터에 저장되어 있는 번지 값에 +8bit을 한 번지에 R2의 값을 저장한다.



1. IF

ALU instruction 과 메모리 접근하는 LD, ST에서 다루고 IF 조건문이 마지막으로 남았다. IF(조건문)은 먼저 조건에 해당하는 명령을 임의로 만들어서 레지스터에 저장하여 JUMP-POS 명령을 사용하여 해당 레지스터가 양수와 음수에 따라 해당 PC의 값을 +4를 시켜 만약 음수면 다음 명령어로 들어가고 양수이면 PC + 4을 시켜준다.



위에 예시를 보면 R9에는 어떤 어린이의 나이가 수로 저장되어 있으며 해당 수에서 상수 10을 SUB 명령어를 하여 R1 레지스터에 저장한다. 그 다음 명령어 JUMP-POS에서는 R1이 양수이면 PC +4로 점프를 하고 음수이면 다음 명령어인 MULT 명령어로 들어간다. 해당 MULT 명령어는 R3 레지스터에 저장되어 있는 값은 상수 0.7를 곱하여 R3에 저장하는 명령어이다. 물론 이 명령어는 R1이 양수이면 점프가 되어 실행되지 않는다.

time behavior

시간의 관점으로 바라보면 1GHz 속도의 프로세서는 1초에 1GHZ를 사용하는 것인데 이 의미는 1초를 10^9으로 나누면 1ns가 되는데, 1ns에 instruction (fetch-decode-execution)이 진행된다.

ISA (Instruction Set Architecture)

* ALU : add, sub, mult, div, and , or , not
* Data transfer : load, store
* Jump : jump if =, =/ , >, <, <=, >=

Programs in Binary

지금까지 우리는 모든 명령어를 어셈블(assemble) 언어로 표현을 하였다. 만약 명령어 ADD R1, R2, R3가 있으면 이걸 이진으로 표현을 하면 0000 1100 00010 00100 00110 unused(9)로 표현한다. 어셈블로 표현한 언어와 이진으로 표현한 것은 간단히 1:1 번역이 가능하며, 대게 사용자들은 간단히 표현할 수 있는 어셈블언어를 선호한다.

