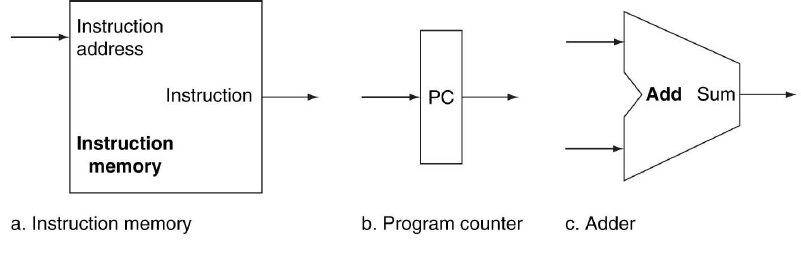
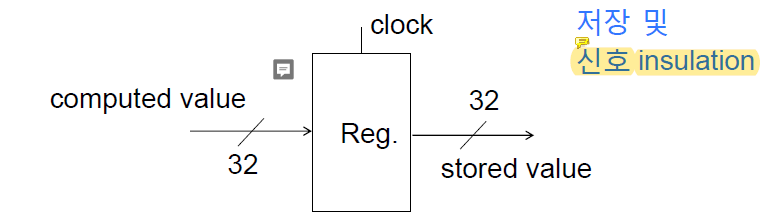
#HW11 2017069598 박상지

the processor – single cycle design

building blocks for IF

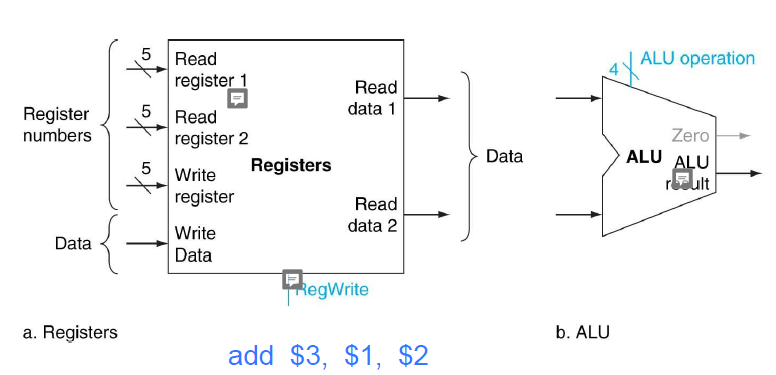


instruction fetch에서는 PC가 가리키는 Instruction 주소를 주어서 해당 instruction에 대한 fetch가 일어난다. (only read) PC가 가리키는 주소를 읽으면 PC값은 자동적으로 +4를 시킨다.

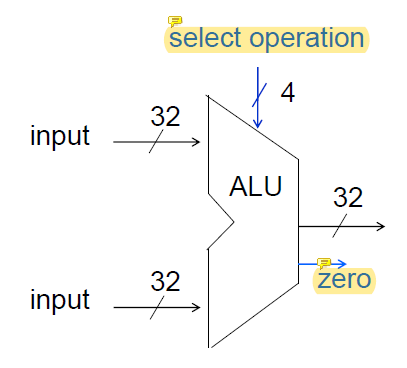


Register에서는 32-bit machine을 사용하면 32-bit 데이터가 input으로 들어오는데, clock이 올라올 때 input 값을 한 주기동안 저장한다. clock이 다시 올라오면 해당 input값에 따라 데이터를 저장한다. 즉 주기가 끝나기 전에 input이 들어와도 레지스터의 값은 바뀌지 않는다. 주기가 끝날 때까지 input가 output이 영향을 주지 않게 해당 신호를 차단을 할 수 있으며 insulation이라 한다.

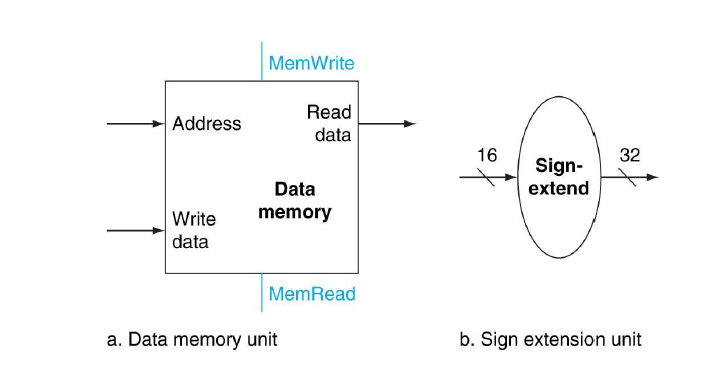
building blocks for r-types



해당 instruction을 예로 처음에 두 개의 레지스터를 읽을 수 있는 channel이 필요하다. 해당 channel은 32-to-1 mux을 사용하며, 32개의 레지스터 중에 하나의 레지스터를 선택을 한다. 더 깊게 들어가면 레지스터를 선택하기 전에 5-to-32 decoder을 통해서 신호에 따른 결과 값이 하나씩 만 있기 때문에 5-to-32 decoder는 5-bit으로 표현되는 레지스터에서 신호를 받은 해당 레지스터만 mux으로 신호를 보낸다. 두 개의 레지스터를 읽으면 Read data 1, 2에 ALU 연산으로 보내는데 ALU연산에서도 마찬가지 어떤 연산인지 확인하기 위해 select operation와 zero signal이 존재한다. select operation에서 add, sub, 등의 signal이 opcode에 따라 결정되며 zero signal은 beq에서 사용되는데 ALU에서 빼기 연산으로 zero의 값이 나오면 jump를 한다. 현재 add의 연산이기 때문에 결과값을 다시 보내기 위해 ALU result를 통해 Write Data에 데이터를 보내고 Write register에 해당한 레지스터에 값을 저장한다. add라는 instruction을 실행시키면 두 개의 레지스터를 더해서 레지스터에 저장을 해야 하는데 여기서 RegWrite 신호를 enable를 해줘야 한다. 반대로 beq는 두 개의 레지스터를 같은 지 아닌지 비교만 하기 때문에 신호를 disable 즉, 0의 값을 줘야 한다. 이것도 역시 opcode에서 결정한다.

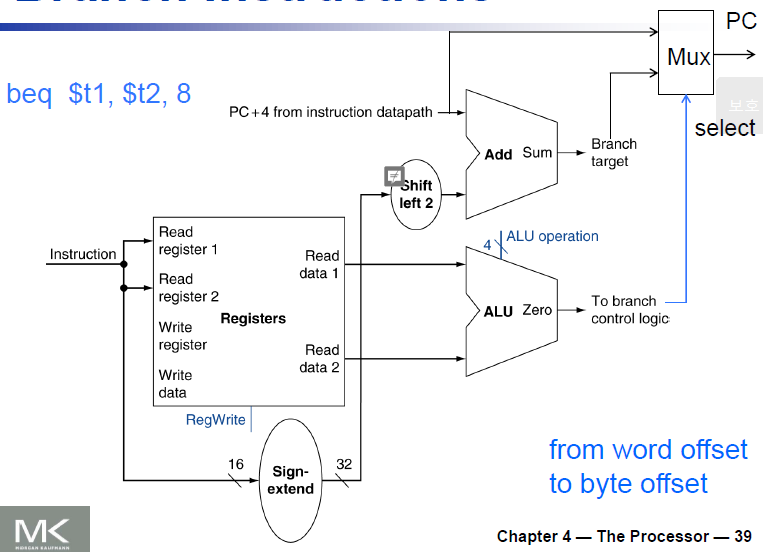


building blocks for lw/sw



load에서는 메모리를 읽기 때문에 MemRead 신호에 1을, store에서는 메모리에 쓰기를 해야 하기 때문에 MemWrite에 신호를 줘야 하고, 저장할 데이터도 받아야 한다. load/store에서는 immediate 값은 16-bit이기 때문에 ALU 연산에 보내기전에 Sign extend에서 32-bit으로 만들어 준다.

Branch Instructions



beq에서 immediate 값을 16-bit에서 32-bit으로 바꾸고 해당 immediate값은 word 단위이기 때문에 byte로 바꿔 주기 위해서 shift를 해준다. 두 개의 레지스터 같으면 ALU 연산에서 0의 값을 받아 Mux로 통해 해당 PC값을 jump를 하고 아니면 PC+4를 받는다.

building the datapath

Datapath란 MIPS 프로세서가 어떤 functional unit으로 구성되어 있나, 그리고 이러한 functional unit들이 어떻게 돌아가는지 나타난 설계이다. datapath 설계는 high-level organization이고 설계를 하면 CPI는 자동적으로 확정이 된다.

Control Design

control signals이란 지금 시작한 instruction의 opcode와 function code에 따라서 값을 결정시킨다. 그에 따라 control signal을 만들어 주는 것은 instruction decode이다.

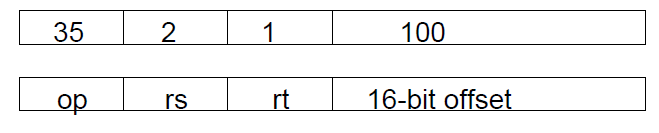
building the datapath

datapath에서 어떤 instruction인지 분석하기도 전에 레지스터 두 개를 읽는다. 왜냐하면 병렬 구조이기 때문이다. 모든 instruction은 레지스터 두 개를 일단 읽어야 한다. load의 경우에도 결과의 레지스터를 읽는 행위도 가능하다. 읽고 쓰든 안쓰드 그만이기 때문에 병렬로 구현해야 시간을 줄 일수 있다.

instruction decode

instruction이 주어지면 그에 해당한 opcdoe와 functional code를 분석해서 control signal을 만들어 주는 것인 instruction decode라 한다. instruction decode와 레지스터 읽는 것을 병렬로 구현한다. 시간 단축을 위해서 이다.

lower ALU Control Unit



Example : lw 1, 100(2)

non-R-type은 opcde만 있고, function code가 존재 하지 않는다. 그렇기 때문에 opcode로 control signal을 설정해줘야 한다. load instruction은 저장할 레지스터와 데이터의 값을 불러들이는 레지스터 그리고 immediate의 값을 받아야한다. load에서는 불러오는 레지스터에 immediate의 값을 더하기 때문에 ALU연산이 필요하다.



RegDst에서 load는 I-format이기 때문에 결과 값을 쓰는 레지스터는 20-16이여야 한다. 그렇기 때문에 신호를 0을 준다. R-format일 경우 레지스터는 11-15을 쓴다. 신호를 1을 준다.

ALUSrc에서는 immediate의 연산을 사용하는지에 대한 신호이기 때문에 R-format은 레지스터간에 연산이기 때문에 0이라는 신호를 주고 load와 store은 각각 1을 준다.

MemtoReg에서는 R-format는 결과 값을 레지스터에 저장해야 하기 때문에 해당 MUX에서 아래 값을 0에 신호를 줘야 한다. 반대로 load에는 Readdata에 신호를 줘야하기 때문에 1의 신호를 받는다.

RegWrite는 R-format은 당연히 결과 값을 레지스터에 저장한다. 그렇기 때문에 1의 신호를 준다. (1 -> enable)