<목적>

인공지능의 convolution 연산에서 덧셈, 곱셈이 반복적으로 사용되고 pooling(max pooling)에서는 비교 연산이 반복적으로 사용된다. 따라서 risc-v를 기반으로 하여 인공지능에서 연산 속도가 향상되도록 custom instruction을 추가하여 하드웨어 설계하였다. int8 quantization을 통해 convolution과 pooling의 input이 8bit integer라고 할 때, vector 개념을 이용하여 기존의 32bit를 4개의 8bit 영역으로 나눔으로써 1 cycle에 4개의 8bit data를 처리할 수 있는 instruction을 설계하였다.

<추가한 instruction의 종류>

vadd

signed 8bit integer data를 병렬적으로 처리하며 두 data를 더한 값을 rd에 저장한다.

R-type의 instruction이다.

테이블이(가) 표시된 사진

자동 생성된 설명

vmul

signed 8bit integer data를 병렬적으로 처리한다.

AI에서 최종적으로 값의 상대적인 크기가 중요하다는 점을 이용해 8bit 곱의 결과인 16bit data의 상위 8bit를 rd에 저장한다.

R-type의 instruction이다.

테이블이(가) 표시된 사진

자동 생성된 설명

mpool

signed 8bit integer data를 병렬적으로 처리하며 두 data 중 큰 값을 rd에 저장한다.

R-type의 instruction이다.

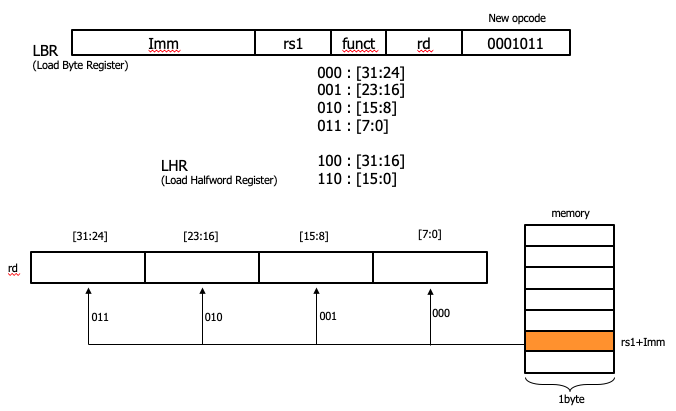
테이블이(가) 표시된 사진

자동 생성된 설명

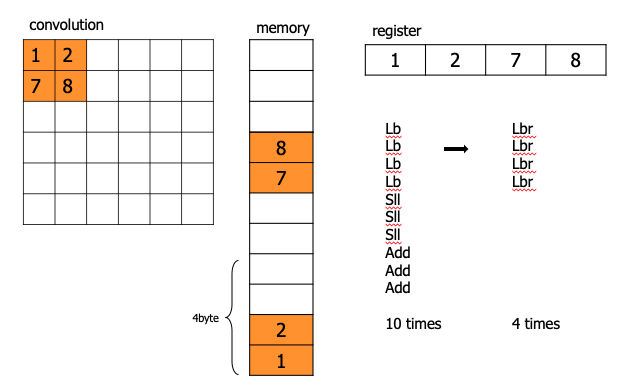
LBR

funct 값에 따라 byte 단위로 register의 원하는 위치에 8bit data를 저장하며 목표 위치 이외의 값은 기존의 값 그대로 유지한다. 새로운 opcode를 사용했으나 load instruction과 동일한 구조를 가지고 있다.

(LHR은 halfword 단위로 저장하는 instruction으로 필요할 경우 LBR에서 funct값을 달리하는 것만으로 구현할 수 있도록 구상하였다.)

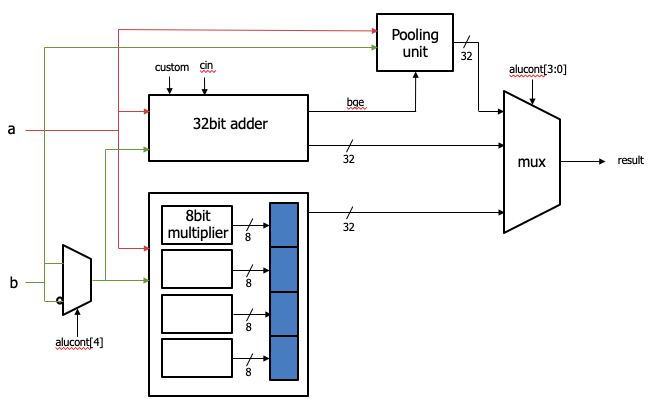


LBR을 만든 이유는 vadd, vmul, mpool instruction을 사용하기 위해서는 register에 계산할 data들을 올바르게 배열하는 과정이 필요하기 때문이다. 기존의 instruction으로 처리할 경우 하나의 vector instructioin을 사용하기 위해서 전처리에만 대략 10 cycles 필요하다는 문제점이 있지만 lbr을 통해서는 4 cycles 안에 해결할 수 있다.

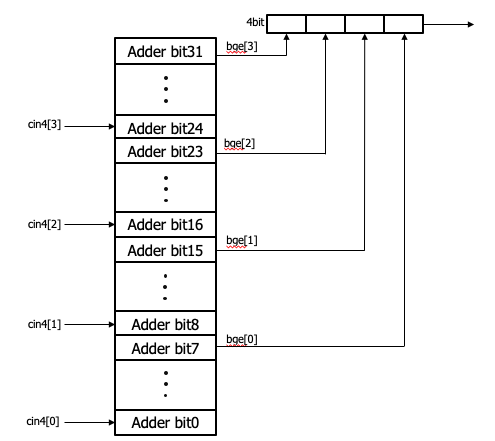


<설계>

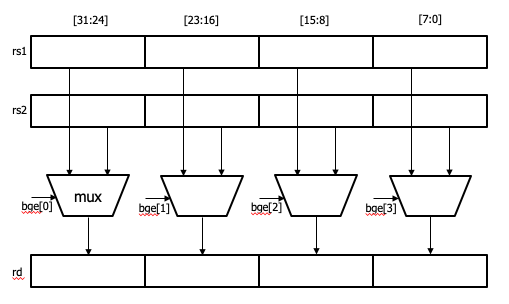
vadd, vmul, mpool의 경우, ALU에 module을 추가하는 것만으로 구현하였다. 추가, 변형된 module은 다음과 같다.



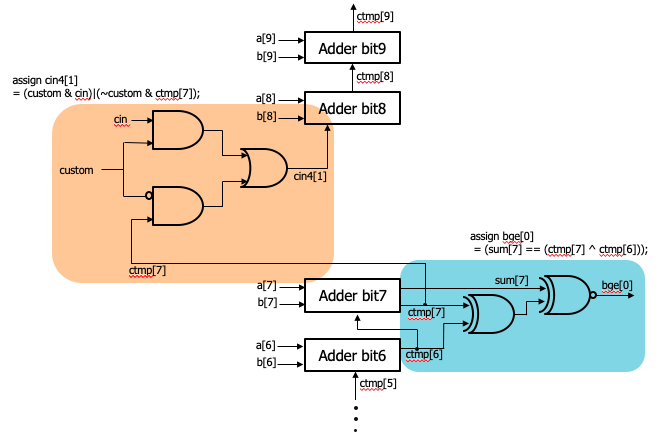
기존의 32bit를 처리하기 위한 32bit adder에 몇 가지 logic을 추가해 vadd, mpool instruction을 구현하였다.



cin4의 값은 현재의 instruction이 custom인지의 여부에 따라 달라지고 bge값은 bge instruction에서 flag를 이용하던 방식 그대로 계산되어 pooling unit에 전달된다.



pooling unit에 전달된 bge는 rs1과 rs2의 byte data 중 하나를 선택하기 위한 mux의 input이 된다.



현재의 instruction이 custom instruction인 경우 cin4는 cin(alucont[4])가 되고 custom instruction이 아닐 경우에는 32bit data를 처리하던 방식대로 이전의 carry\_out 값인 ctmp가 된다. 즉, custom instruction일 때에는 byte 단위로 끊어져 carry가 다음의 1bit adder에 전달되지 않으므로 4개의 8bit adder가 동작하는 것과 같이 작동한다. 또한 alucont[4]가 bit0, bit8, bit16, bit24의 carry\_in이 되도록 하여 끊어진 4개의 byte들이 모두 같은 연산을 수행할 수 있다.

vmul은 16bit adder를 다수 이용해 구현했으며 그 내용은 다음과 같다.

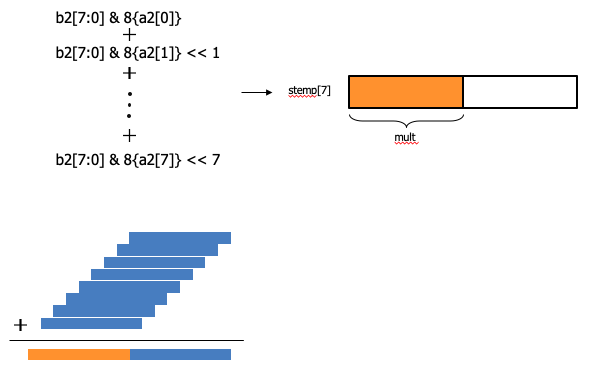
- a 또는 b가 음수인 경우 각각 부호를 바꾸어 a2, b2에 저장한다.

- for n = 0 to 7 -> b2[7:0] & 8{a2[n]} << n 으로 나온 8개의 값들을 모두 더한다.

- 만약 a와 b의 부호가 다를 경우 결과값이 음수여야 하므로 부호를 바꾸어준다.

- 16bit data 중 상위 8bit를 rd에 저장한다.

그림으로 표현하자면 다음과 같다.



LBR은 load instruction과 대부분 비슷하게 작동하지만 rs2를 이용한다는 점과 rd에 값을 넣는 방식이 다르다. LBR에서 임의의 위치를 제외한 나머지 부분의 data는 그대로 유지하기 위해서는 rd의 data를 register에서 가져와야 한다. 따라서 instruction에는 rs2가 없지만 하드웨어 상에는 존재한다는 점을 이용해 rs2에 rd값을 넣어 data를 가져온다. alu에서는 rs1\_data + imm를 계산하고 그 주소로 memory에 접근한다. memory에서 가져온 data와 rd\_data를 funct에 따라 적절히 섞어 rd에 저장한다.

