Computer Organization

Assignment 2

2018-14245 김익환

May 23, 2020

1. Count the number of executed instructions in `hash` and `hash_rv32z`. Which one executes more instructions?

```
./emu-rv32i hash
9988e12d
00fd5bcb

Instructions Stat:
LUI     = 119
JAL     = 19
JALR    = 7
BNE     = 115
BGE     = 18
BLTU    = 16
LW      = 891
LBU     = 260
SB      = 34
SW      = 490
ADDI    = 461
ANDI    = 48
SLLI    = 16
SRLI    = 113
SUB     = 16
XOR     = 226
SRL     = 16
LI*     = 55
MUL     = 113

Five Most Frequent:
1) LW   = 891 (29.91%)
2) SW   = 490 (16.45%)
```

```
3) ADDI = 461 (15.47%)
4) LBU  = 260 (8.73%)
5) XOR  = 226 (7.59%)

>>> Execution time: 3148800 ns
>>> Instruction count: 2979 (IPS=946074)
>>> Jumps: 162 (5.44%) - 26 forwards, 136 backwards
>>> Branching T=136 (91.28%) F=13 (8.72%)

./emu-rv32z hash_rv32z
9988e12d
00fd5bcb

Instructions Stat:
LUI     = 4
JAL     = 19
JALR    = 7
BNE     = 115
BGE     = 18
BLTU    = 16
LW      = 439
LBU     = 260
SB      = 34
SW      = 153
ADDI    = 348
ANDI    = 48
SLLI    = 16
SUB     = 16
SRL     = 16
LI*     = 55
hash_init       = 2
hash_update     = 113
hash_digest     = 2

Five Most Frequent:
1) LW   = 439 (26.98%)
2) ADDI = 348 (21.39%)
3) LBU  = 260 (15.98%)
4) SW   = 153 (9.40%)
5) BNE  = 115 (7.07%)

>>> Execution time: 1715100 ns
```

```
>>> Instruction count: 1627 (IPS=948632)
>>> Jumps: 162 (9.96%) - 26 forwards, 136 backwards
>>> Branching T=136 (91.28%) F=13 (8.72%)
```

Listing 1: Instructions Stat

Listing 1의 instruction count를 보면, `./emu-rv32i hash`는 총 2979개의 instruction을 수행하였고, `./emu-rv32z hash_rv32z`는 총 1627개의 instruction을 수행하였다. `./emu-rv32i hash`가 더 많은 instruction을 수행하였다. 구체적으로 많은 수의 LW, SW, ADDI, LBU, XOR 등의 instruction이 `hash_init`, `hash_update`, `hash_digest`의 instruction으로 대체된 것을 관찰할 수 있다.

2. We designed the custom `RV32Z` in hopes it would accelerate the performance of hash computation. Can you relate your answer in 1 to justify why `RV32Z`-based hash computation would be faster than `RV32IM`-based hash computation?

   다음의 식이 성립한다.

$$\text{latency} = \frac{\text{CPI} \times \text{number of instructions}}{\text{frequency}}$$

   latency가 짧을수록 performance가 좋다. latnecy는 number of instructions에 비례하므로 instruction의 수가 적은 `RV32Z`가 hash computation에 유리하다고 볼 수 있다. 하지만 이 가정은 CPI가 동일할 때 성립한다.

```
hash_value ^= *key;
hash_value *= 0x5bd1e995;
hash_value ^= hash_value >> 15;
```

Listing 2: Hash Computation

`hash_update`의 경우 Listing 2의 코드를 수행하는데, 이를 단순히 생각하면 ALU를 통한 execution을 3번 수행해야 하므로 CPI가 일반적인 arithmetic instruction에 비해 최대 3배가 될 수도 있다. 하지만 이를 carry lookahead와 같은 방법으로 hash computation dedicated hardware를 최적화하여 CPI를 줄일 수 있다면 `RV32IM`보다 hash computation performance가 확실히 더 좋아질 것이라고 말할 수 있다. CPI를 줄이지 않더라도 구현된 `RV32Z`에서는 `hash_value`를 저장하는 register를 따로 사용하므로 LW와 SW의 사용횟수가 현저히 줄어 performance가 증가하였다.