1H-3

Speeding Processor up by Employing Instruction Register

Mochamad ASRI † Takayuki MATSUMURA ‡ Kenji KISE ‡
Undergraduate School of Computer Science , Tokyo Institute of Technology†
Graduate School of Information Science and Engineering, Tokyo Institute of Technology‡

1 Introduction

Many embedded system have substantially different design constraint than desktop computing applications. Those constraints are space, power, and performance which often have conflicting design requirement. It is well-known that engineers face formidable problems to improve one constraint without negatively affecting others when designing processors.

In order to address each design issue, identification of inefficiencies in some parts of program execution is needed. Considering that I-Fetch logic consumes approximately 36 % of total processor power, optimizing I-Fetch mechanism is one of the natural target for embedded processors[1].

Instruction Register (IR) was proposed in order to improve I-Fetch mechanism [2]. By integrating IR into the architecture, Hines et al deliberately succeed in reducing power consumption as well as code size. However, the speed issue is still beyond the discussion scope. In this paper, we try to search possibilities on the existing proposed method further out to aim more efficient I-Fetch mechanism that leads to processor speed up.

2 Instruction Register

Current I-Fetch mechanism is inefficient at least in two aspects : All instructions have usually the same length even though most instructions use only a fraction of available length; Almost all references are accessed from the same storage(IC hit or ROM access) even though only a small subset of instructions account for the majority of the reference[2].

Integration of IR into architecture was originally proposed by Hines et al. The idea is, by storing frequently referenced instructions into small size of IR, the conventional power-consuming IC/Memory-based instructions reference could be avoided. Thus, power requirements consumed for referencing frequent instructions are possible to be reduced.

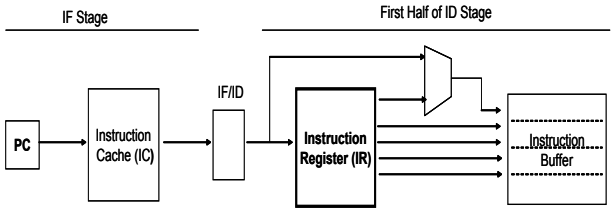

Figure 1. IR Integrated on I-Fetch Mechanism

Figure 1 shows how Hines et all integrated 32-entry IR into the corresponding architecture. If the instruction fetched from the IC is a packed instruction, instruction index fields select from one to five IR entries to write to the instruction buffer. Based on experiment by using MiBench as a benchmark, on average, about 66.51 % of all instructions executed can be stored in a 32-entry IR, assuming it can be loaded with the 32 most common instructions at the start of execution[2].

3 Expanding Instruction Register Entry

To seek the possibility of processor's speed up by using IR, we try to scrutinize how the percentage of all instructions executed that can be stored in IR goes if the number of IR entries are expanded further out.

We use SimMips[3], a MIPS system processor simulator, to conduct the investigation. We regard distinct instruction as 32-bit union. In other words, even so there are two same kind of *ADDI* instructions, for instance *ADDI R2, R3, 5* and *ADDI R2, R3, 8* , we treat them as distinct instruction since the 32-bit binaries of both instructions are different. We adopt SPEC-CINT2006 as a benchmark for the simulation.

Figure 2 shows the relationship between the number of IR entry and the percentage of all instructions executed that can be stored in IR. From the figure, the percentage of instruction fetches to the most common instructions significantly grows as the number of entry increases. Moreover, when IR entry's number is set to 2048, in most of applications except *gcc*, the percentage

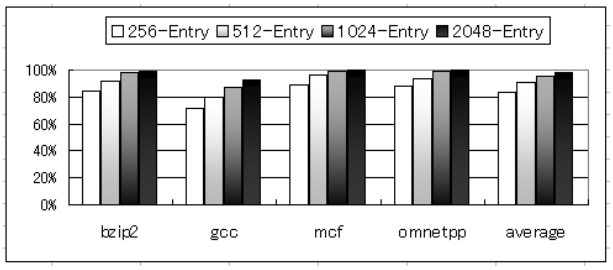

Figure 2. IR Entry Size and Its Coverage Percentage

Table 1. Comparison of IR and IC Hit Rate

Size	IR	1-way IC	2-way IC
1KB	83%	82%	84%
2KB	90%	86%	88%
4KB	96%	91%	93%
8KB	97%	96%	97%

shares above 97% of total program instructions. Complex instructions used in *gcc* could be considered as a factor that led IR to only have 93% of percentage when its entry's number is set to 2048.

Based on the result, since on average IR shares high percentage of total program, we are pondering that instead of combining IC and IR, we can take off IC and simply employ 1024 or 2048-entry IR into the architecture. One advantage that can be exploited is conventional 32-bit used for referencing instruction will be reduced to only 11 or 12-bit (Figure 3), in case we apply 1024-entry or 2048-entry of IR.

Furthermore, if the percentage of all program instruction in IR is not lower than the IC hit rate, we can simply disintegrate IC from the architecture so that the memory hierarchy turns to be simpler and hardware cost could possibly be diminished.

We inspect further to compare the percentage of all program instructions in IR and IC hit rate. Table 1 shows the comparison (average value of 4 benchmark applications : *bzip2, gcc, mcf, omnetpp*). From the table, we can conclude that the percentage of all program instructions in IR on 1024 or 2048-entry is not lower than 1-way and 2-way IC hit rate.

Although it will become much more complicated with binaries since we change the architectural resources, we can unravel it by using binary translation. The binary is first altered, afterwards we sign a flag to instructions that refer to IR (Figure 4). By storing beforehand the most frequent instructions in IR, we can implement IR in the architecture without using IC.

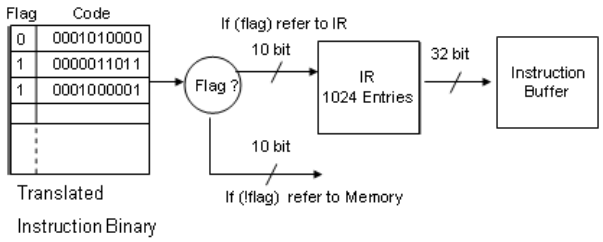

Figure 3. Referencing Mechanism of IR

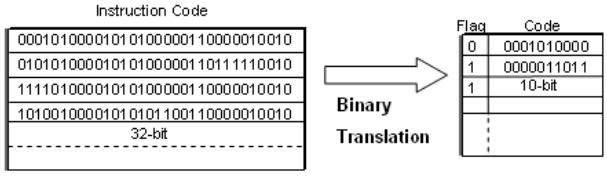

Figure 4. 32-bit to 11-bit Binary Translation

4 Conclusion

We investigated the possibility of speeding processor up by using Instruction Register. By expanding the number of IR entry, we found that when the number of entry is set to 1024 or 2048, on average it covers 96-97% of all program instructions.

Moreover, we found that the percentage of all program instruction in IR is higher than 1-way and 2-way IC hit rate, especially in case of 1024 and 2048-entry. It means it is quite possible replacing IC by IR to get simpler memory hierarchy, lesser hardware cost and lesser power consumption .

Yet, there are many areas of future works to provide comprehensive discussion on accomplishing speed up fruition. It will include branch predictor implementation, speed up technique and evaluation.

References

[1] Montanaro, J et al: A 160-mhz, 32-b, 0.5-W CMOS RISC Microprocessor. Digital Tech. J., 9(1):49-62, 1997

[2] Hines, S. and Green, J. and Tyson, G. and Whalley, D: Improving Program Efficiency by Packing Instructions into Registers. Proceedings of the 32th Annual International Symposium on Computer Architecture (ISCA) 2005.

[3] Naoki, F. and Miyoshi, T. and Kenji, K: SimMips : A MIPS System Simulator. Workshop on Computer Architecture Education(WCAE) held in conjunction with MICRO-42 , pp. 32-39 (December 2009).