

TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

Efficient Fetch Mechanism by Employing Instruction Register



Mochamad Asri
Tokyo Institute of Technology

TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

What do we do ?

- Propose Efficient Processor Fetch Mechanism

How do we do ?

- Employing Large Entry of Instruction Register
 - Instruction Register is a register file that stores the most frequently executed instructions
- Performing Unique Binary Translation

How did it turn out ?

- Code size could be diminished up to **54%**

TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

Research Background

- The Advancement of Transistor Technology
 - Processors are embedded in many electronics systems
- Strict design constraints in Embedded Processor
 - Power Consumption
 - Performance
 - Area Size
 - Code Size
- In order to address each design issue :
 - Identification of inefficiencies in some parts of mechanism is needed




TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

Re-Thinking Instruction Fetch Mechanism

- Fetch Logic Optimization is one of important design issues
 - In Fact, 36% of Strong ARM Power is consumed by Fetch Logic ¹⁾
 - The Most Energy-Consuming Part !
- Moreover, conventional fetch mechanism is inefficient at least in two aspects :
 - Use the same length
 - Most instructions use only a fraction of available length
 - Access from the same storage (IC hit or ROM access)
 - Only small subset of instructions account for the majority of reference
- How do come up against these inefficiencies ?

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

TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

◆ Instruction Register

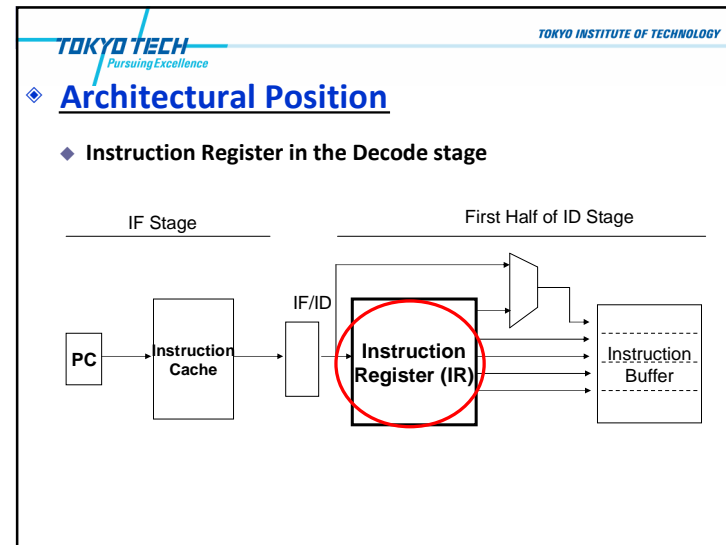
- ◆ Instruction register²⁾ is proposed to reduce those inefficiencies
- ◆ So, What is Instruction Register ?
 - ◆ A Register File that stores **the most frequently executed instructions**

Instruction Register

#	Instruction
0	nop
1	addi r[3], r[3], 4
2	sub r[2], r[9], r[2]
3	addi r[3], r[3], 8
...	...

- ◆ Note that although *addi R3, R3, 4* and *addi R3, R3, 8* apply the same *addi* instruction, they will not be regarded as same instruction since they have different binary code.

[2] S. Hines, J. Green, G. Tyson, and D. Whalley. Improving program efficiency by packing instructions into registers. Proceedings of the 32nd International Symposium on Computer Architecture (ISCA), 2005.



TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

◆ More about Instruction Register

- ◆ Advantages of integrating Instruction Register :
 - ◆ Instruction Packing
 - ◆ Fetch Energy and Code Size Reduction
- ◆ Previous research used 32-entry Instruction Register
 - ◆ Exploitation of :
 - ◆ Instruction packing techniques
 - ◆ Compiler support
 - ◆ Reduction of :
 - ◆ Fetch Energy by **37%**
 - ◆ Code Size by **19%**

TOKYO TECH Pursuing Excellence TOKYO INSTITUTE OF TECHNOLOGY

◆ The Proposed Idea

- ◆ Based on the previous work :
 - ◆ The larger entry number of IR
 - ◆ The bigger percentage of IR-resident instructions
- ◆ With the help of **binary translation** :
 - ◆ Possible to reduce Code Size !

↓

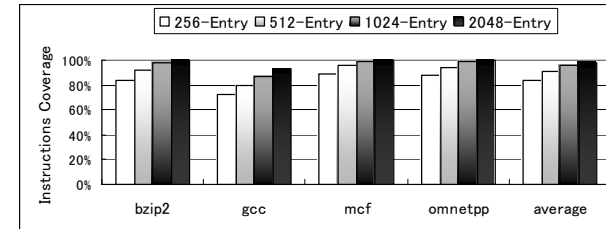
Target:
To realize efficient fetch mechanism
that leads to further reduction of code size !

◆ Preliminary Experiment

- ◆ Investigate percentage of IR-resident instructions when the number of IR entry are expanded
 - ◆ To see if it is high or not
- ◆ Evaluate by using SimMips MIPS Processor Simulator³⁾
- ◆ SPECINT2006 (bzip2, gcc, mcf, omnetpp) as the benchmark

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

◆ Preliminary Experiment (Result)



- ◆ On Average, IR-Resident Instructions covers :
 - ◆ About **83%** of total instruction for **256-entry IR**
 - ◆ About **90%** of total instruction for **512-entry IR**
 - ◆ About **96%** of total instruction for **1024-entry IR**
 - ◆ About **97%** of total instruction for **2048-entry IR**

◆ Trade-off Consideration

- ◆ On Average, IR-Resident Instructions covers :
 - ◆ About **83%** of total instruction for **256-entry IR**
 - ◆ About **90%** of total instruction for **512-entry IR**
 - ◆ About **96%** of total instruction for **1024-entry IR**
 - ◆ About **97%** of total instruction for **2048-entry IR**
- ◆ Decided to employ **1024-entry of IR**
 - ◆ It covers almost the same high percentage of instructions with 2048-entry
 - ◆ While at the same time it costs only half hardware of 2048-entry

◆ Instruction Register + Binary Translation

- ◆ It is found that large IR stores most of program instructions
- ◆ Considering this fact :
 - ◆ If we apply 1024-entry IR
 - ◆ Only need 10 bits for reference.
 - ◆ But, how to reduce the code size ?



Let's make the most of binary translation !

TOKYO TECH
Pursuing Excellence

TOKYO INSTITUTE OF TECHNOLOGY

Binary Translation : IR-Resident

- IR-resident case :
 - Translate to 11 bits
 - Express 1024-entry IR with 10 bits
 - Use the remaining 1 bit as a flag bit,
 - Decide whether an instruction is IR-resident or not

IR-resident Instruction Code

00010100001010100000110000010010
01010100001010100000110111110010
11110100001010100000110000010010
101001000010101011001100010000010010
32-bit

Binary Translation

Flag	Code
1	0001010000
1	0000011011
1	10-bit
1	10-bit

TOKYO TECH
Pursuing Excellence

TOKYO INSTITUTE OF TECHNOLOGY

Binary Translation : Non IR-Resident

- Non IR-resident case :
 - Fetch completely 32 bits
 - 32 to 33 bits addressing
 - Divide these bits into 3 parts.
 - Most Significant Bit will be the flag bit

Non IR-resident Instruction Code

0101000010101000011000001001000
0101010000101010000110111110010
1111010000101010000110000010010
10100100001010101100110000010010
32-bit

Binary Translation

Flag	Code
0	0101000010
1	010000110
0	0001001000
0	0101010000
1	010100001
1	011110010

TOKYO TECH
Pursuing Excellence

TOKYO INSTITUTE OF TECHNOLOGY

The Novel Fetch Mechanism

IF (Instruction Fetch) Stage First Half of ID (Instruction Decode) Stage

Translated Binary

Flag	Code
1	0001010000
0	0000011011
1	0001000001
1	0001011001

Flag ?

If (flag) refer to IR

10 bits → IR 1024 Entries → 32 bits → Instruction Buffer

10 or 11 bits → Concatenate → 32 bits → Instruction Buffer

If (!flag) concatenate 3 following codes

- If flag is 0, then concatenate the current 10-bit and 2 following 11-bit codes so that it forms 32-bit instruction
- Else, refer to Instruction Register

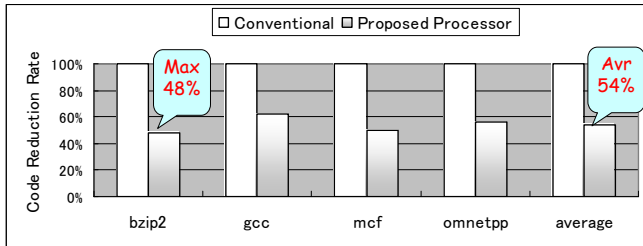
TOKYO TECH
Pursuing Excellence

TOKYO INSTITUTE OF TECHNOLOGY

Simulation Environment

- Processor Simulator
 - Evaluate by using SimMips
- Benchmark
 - SPECINT2006 (bzip2, gcc, mcf, omnetpp)
 - Apply train data set

◆ Experimental Result



- ◆ Code size reduced up to maximum **48%**
- ◆ In average, code is diminished up to **54%**
 - ◆ Previous work only reduced up to **81%**

◆ Conclusion

- ◆ Proposed Efficient Fetch Mechanism
 - ◆ Employing 1024-entry of IR
 - ◆ Performing novel binary translation
- ◆ Succeeded in reducing code size up to **54%**
 - ◆ The previous work reduce only up to **81%**

◆ Future Works

- ◆ Further Evaluation
 - ◆ Power Consumption
 - ◆ Area Consumption
 - ◆ Performance Evaluation
- ◆ FPGA Verification
 - ◆ Implement Processor in FPGA

Thank you !