Software Performance Estimation for Pipeline and Superscalar Processors

Chu-Yi Huang and Daniel D. Gajski

Technical Report# 95-20
June 16, 1995

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
Phone: (714)824-8059
FAX: (714)824-4056
Email: chu-yih@ics.uci.edu

Abstract

When moving toward hardware/software codesign, software estimation provides important information in chosing hardware implementation (ASIC) or software implementation (software running on processor). This report analizes the pipeline stall and superscalar interlock phenomenon and their influence on software performance. Simple processor profile is proposed to count these two effects. Based on generic estimation model, our estimator can produce accurate estimation without large computation time and precious resource, such as compilers or simulators for each processor. The accuracy and flexibility make our approach suitable for design automation tools.

Contents

1 Motivation 1

2 Problem Analysis 2
- 2.1 Software Estimation . . . 2
- 2.2 Pipeline Stall and Superscalar Interlock . . . 3

3 Estimation Model 7
- 3.1 Pipeline Processor . . . 7
- 3.2 In-Order Issue Superscalar Processor . . . 8
- 3.3 Out-Of-Order Issue Superscalar Processor . . . 10
- 3.4 Software Performance Estimator . . . 11

4 Experiment 12

5 Conclusion and Future Work 13

6 Acknowledgements 14

A Technology File for SPARC (partial) 15

B Technology File for Intel Pentium 21

C Technology File for PowerPC 27

1 Motivation

In system level design, designers have to decide to implement the specification in hardware (ASIC), software (running on processor), or combination of these two. Performance and cost are the most important decision factors. To explore the space of hardware/software codesign, design tools have to provide the performance and cost information.

To provide metrics for hardware/software codesign, both hardware and software estimator are needed. Hardware estimator provides performance and cost metrics for implementing the specification in ASIC. Software estimator provides performance and cost metrics for implementing specification in software running on processor.

This paper discuss the software performance estimation, even though the tool can produce the cost metrics (such as code size, data size), too. Please reference [4] for cost estimation technique. For hardware estimation, please reference [5].

Approach	Work Flow	Flexibility	Computation Time	Accuracy
In–Circuit Simulation	Specification → Machine Code → Processor	Low	Low	High
Software Simulation	Specification → Machine Code → Simulator	Mid	High	High
Software Estimation	Specification, Processor Profile → Estimator	High	Low	Mid

Table 1: Comparision of Software Estimation Approach.

Software performance can be measured in three ways: in-circuit simulation, software simulation, and software estimation. The comparison of these three approaches are summaried in Table 1. In hardware simulation, system specification was compiled into target machine code and performance was measured by running the code on target processor. This approach need one in-circuit simulator and one compiler for each processor. Though

the measured metrics is accurate but the flexibility of the tool is low due to the huge resource requirement.

In software simulation, system specification was compiled into target machine code and performance was measured by running the code in software simulator of the target processor. This approach needs one compiler and one simulator for each processor. In this approach, the estimation accuracy is high (the same as in-circuit simulation) and the flexibility is higher because software simulator is more accessible than in-circuit simulator. But software simulator is very time consuming.

In software estimation, the estimator calculates the performance of the system specification based on a profile of each processor. No compiler, software simulator, or in-circuit simulator are needed. This approach only consume less computation time and need less resource, one technology file for each target processor and one estimator only. The accuracy is lower than the previous two approaches.

From this comparison, we know the software estimator is more suitable for design automation tools because its flexibility and speed allow design space exploration. In this paper, we are trying to develop an estimation method for pipeline and superscalar processors that can produce accurate enough metrics for system level design tools.

2 Problem Analysis

2.1 Software Estimation

Software estimation can be divided into two steps: flow analysis and basic block estimation. In flow analysis, system specification was divided into several basic blocks, as the example in Figure 1. Basic block is a straight-line code which has no branches. Every branch among basic blocks is associated with a probability that this branch will be taken. The execution frequency of each basic block can be calculated based on the graph of basic block and branch probability [5].

After the execution time of each basic block was measured in the second step, the execution time of the whole specification can be calculated by following equation:

$$execution(S) = \sum_{b_i \in S} execution(b_i) \times freq(b_i) \tag{1}$$

where b_i is basic block.

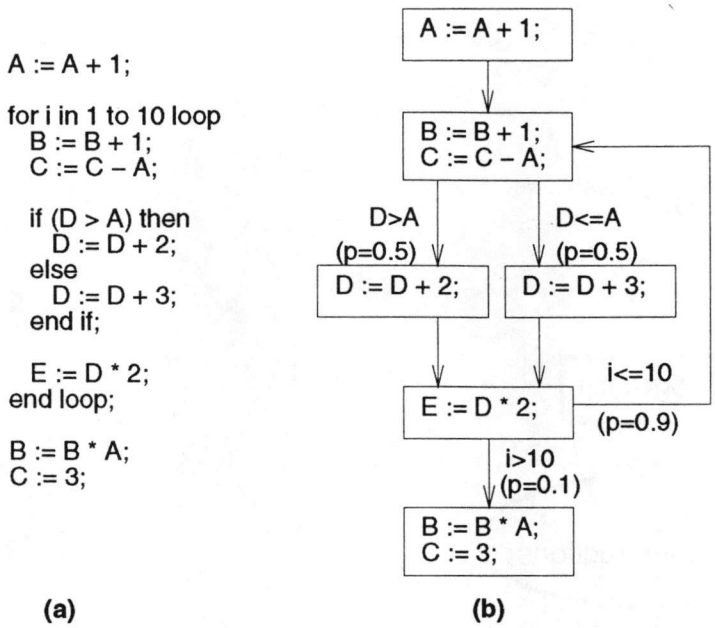


Figure 1: (a) VHDL program, (b) basic block graph.

This report takes the same approach and focus on the basic block estimation for pipeline and superscalar processors. In basic block estimation, we adapt the generic estimation model [4] which was shown in Figure 2. The system specification was compiled into generic three address instructions. For each processor, there is a technology file providing the timing and instruction size of each generic instruction. Then the execution time and code size of basic block can be calculated as follows:

$$execution(B) = \sum_{I_j \in B} time(I_j) \tag{2}$$

$$size(B) = \sum_{I_j \in B} size(I_j) \tag{3}$$

where I_j is the generated generic instruction.

2.2 Pipeline Stall and Superscalar Interlock

Though generic estimation model has shown good results in estimation for non-pipelined processor in [4], the previous two equations can not measure the performance of pipeline or superscalar processor very well. The overlap time of pipeline instruction can be reflected

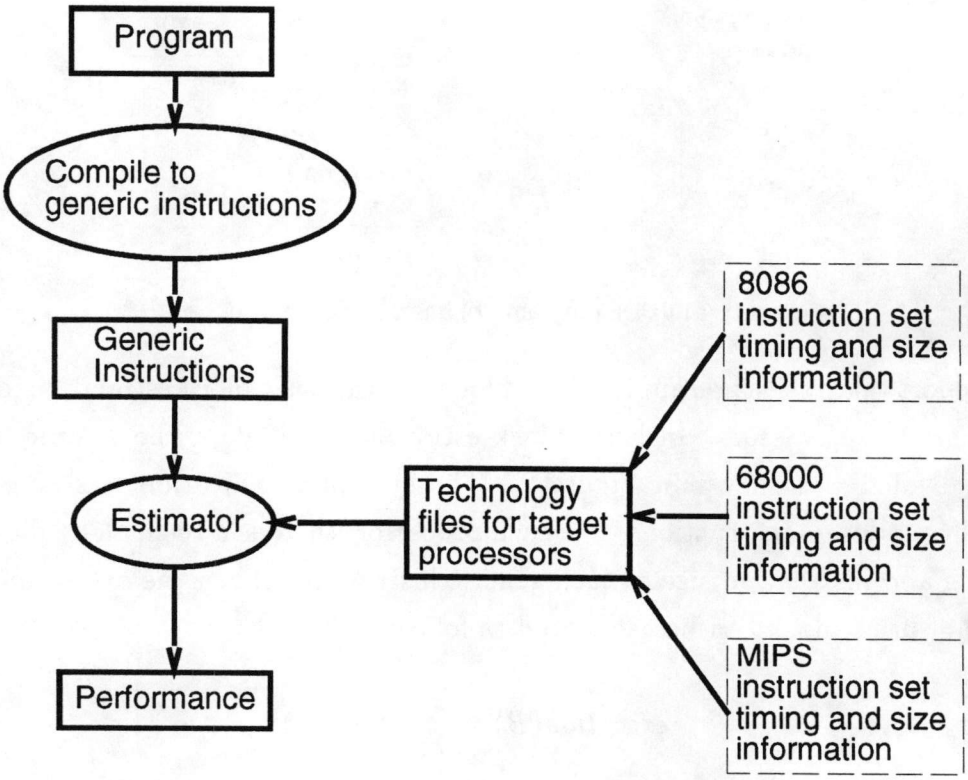


Figure 2: Generic Estimation Model.

in following equation:

$$execution(B) = \sum_{I_j \in B} (time(I_j) - pipe\_depth + 1) + pipe\_depth - 1 \qquad (4)$$

But the result is not accurate due to pipeline stall and superscalar interlock phenomenon.

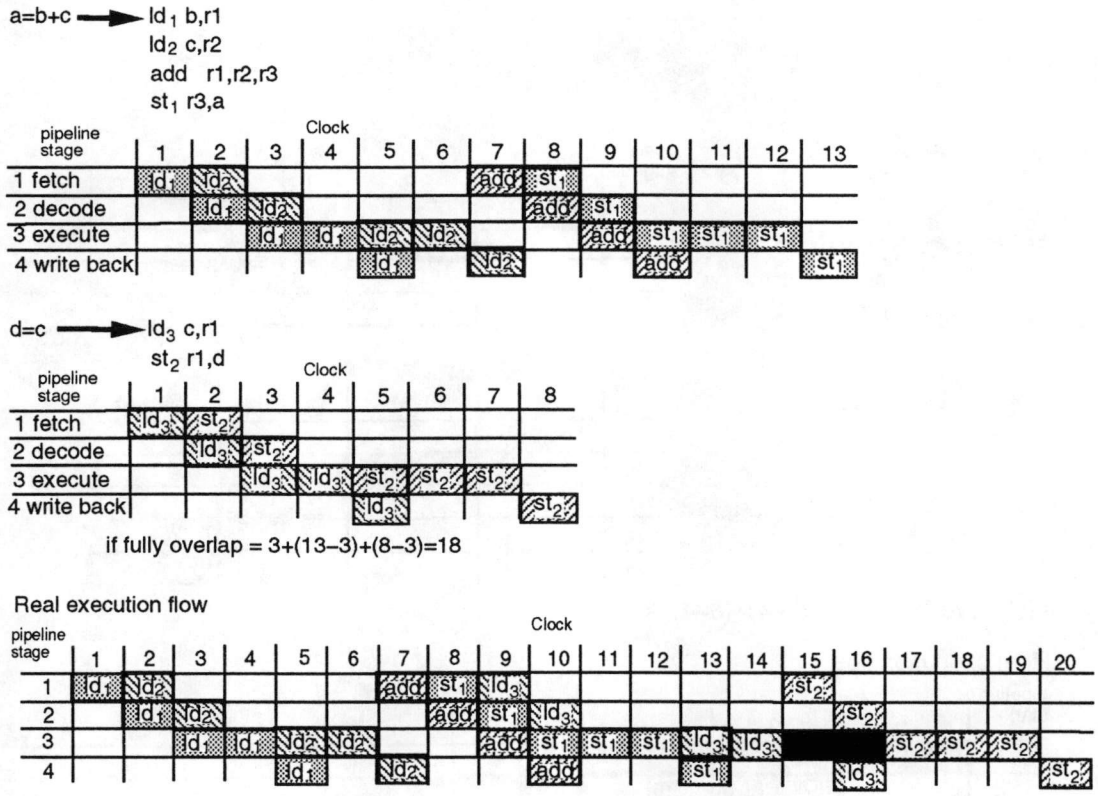


Figure 3: Pipeline Stall Example in SPARC.

Pipeline processor can issue one instruction per clock cycle in ideal situation. When one instruction depend on the result or content a same resource, such as bus, of previous instructions that are still in the pipe, this instruction can not be issued and pipeline was stalled. Figure 3 shows an example of pipeline stall in SPARC processor. When executing the second generic instruction, instruction st_2 can not be fetched until clock cyle fifteen because it content the bus with instruction st_1 and ld_3. Two new pipeline stalls (marked by black) happened in clock cycle 15 and 16. The execution time will be 20 clock cycle, not the 18 clock cyle measured by Equation 4.

Superscalar processor can issue more than one instruction per clock. Processor check

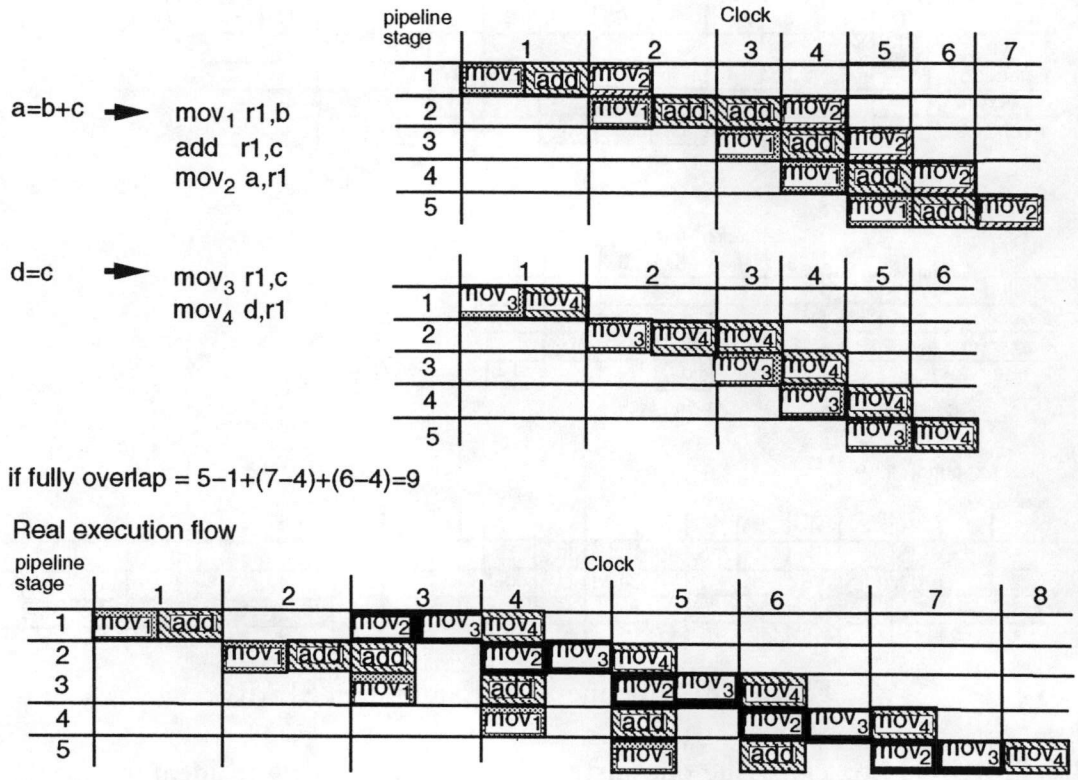
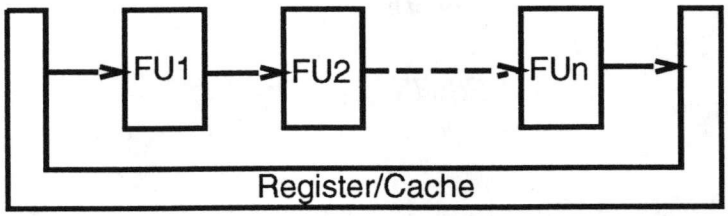


Figure 4: Execution sequence in Pentium.

the issue rule to decide a new instruction can be issued or not. Instructions depend on the result or content same resources (such as bus or ALU) of previous instructions which are not available yet can not be issued. This is called superscalar interlock and prevent the processor achieving maximal performance. Figure 4 shows an execution sequence in Pentium superscalar processor. In this example, instruction mov_2 and mov_3 (marked by black) can be executed at the same time that save one execution clock cycle if the two generic instruction are executed continuously.

3 Estimation Model

3.1 Pipeline Processor



Technology File

- Timing for each generic instruction
- Resource Contention Stall Table (RCStall)

	reg add	assign
reg add	0	0
assign	1	1

- Data Dependency Stall Table (DDStall)

Figure 5: Pipeline Processor Model.

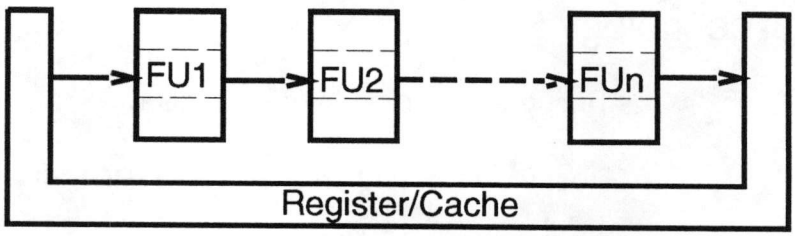
The hardware model of pipeline processor was depicted in Figure 5. Pipeline processor was profiled as a sequence of function unit which can handle one instruction at a time. In order to calculate the pipeline stall, two pipeline stall tables were included in addition

to the execution time of each generic instruction. As the example in Figure 3, pipeline stalls in each generic instruction were already covered in the execution time of generic instruction. Pipeline stalls between generic instructions were stored in pipeline stall table. Pipeline stall table is a two dimension array. Both the x and y dimension are generic instructions. And the array value is the number of pipeline stalls instroduced when the instruction indexd by x is following the instruction indexed by y. The two pipeline stall tables have the same format, but having different value. Data dependent pipeline stall table, *DDStall*, is used when an instruction depend on the result of its previous one instruction, while resource contention pipeline stall table is used for instructions that have no data depedency. The execution time of a basic block can be formulated as follows:

$$execution(B) = \sum_{I_j \in B}(time(I_j) + stall(I_j - 1, I_j) - pipe\_depth + 1) + pipe\_depth - 1 \quad (5)$$

$$\text{where } stall = \begin{cases} RCStall(I_{j-1}, I_j) & \text{if } I_j \text{ dose not depend on } I_{j-1} \\ DDStall(I_{j-1}, I_j) & \text{if } I_j \text{ depend on } I_{j-1} \end{cases}$$

3.2 In-Order Issue Superscalar Processor



Technology File

- Mapping of generic instruction to machine instruction
- Timing information for machine instruction
- Dispatch Rule: 2 add, 1 mul, 1 load
  1 add, 1 mul, 1 load, 1 branch

Figure 6: In-Order Issue Superscalar Processor Model.

The estimation model of in-order issue superscalar processor was shown in Figure 6. This model adds a new feature: each function unit can handle more than one instruction.

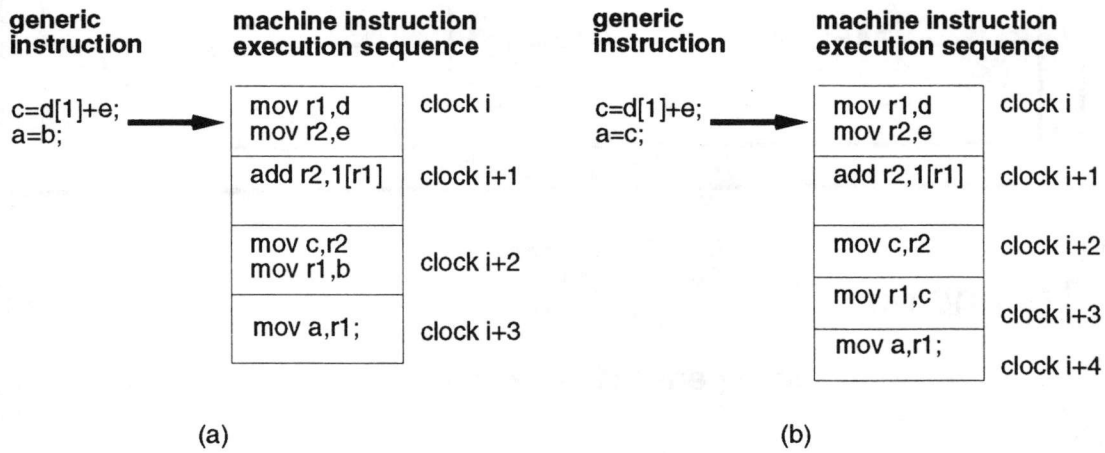


Figure 7: Grouping machine instruction in Pentium.

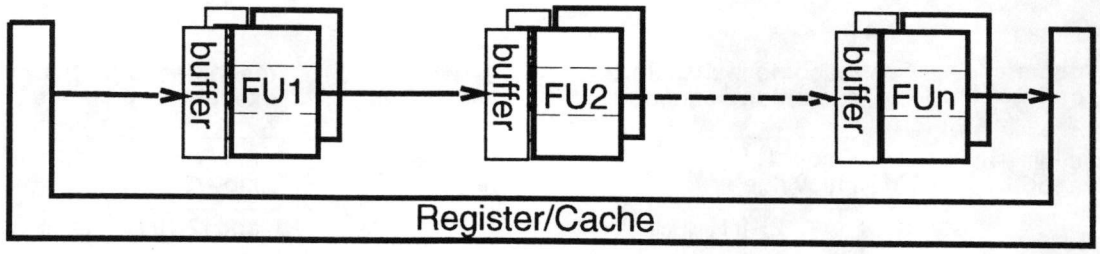
The technology file for this type of processor changes a lot. First, the mapping of generic instruction to machine instruction should be included. When a processor executing the machine instruction of a generic instruction, it fetches more than one machine instructions and check against issue rule. If these instruction satisfy the issue rule, they are grouped together and execute at the same time. The grouping of machine instruction of a same generic instruction is not always the same in different execution instances. For example, *mov c, r2* is grouped with *mov r1, b* in Figure 7(a) while *mov c, r2* in Figure 7(b) consist a group by itself even though they belong to the same generic instruction, an assignment statment. So the mapping of generic instruction to machine instruction should be provided, that the estimator can check the issue rule at machine instruction level.

Second, the execution time of each machine instruction, not generic instruction, should be included. After checking the issue rule, the estimator can determine the parallel instruction group and calculate the performance as follows:

$$\begin{array}{rl} execution(B) = & pipe\_depth - 1 + \sum_{MIG_j \in B} time(MIG_j) \\ \text{where} & MIG_j \text{ is a parallel machine instruction group,} \\ time(MIG_j) = & MAX_{MI_{jk} \in MIG_j} time(MI_{jk}) \\ \text{where} & MI_{jk} \text{ is a machine instruction of group } MIG_j. \end{array} \tag{6}$$

Of course, the issue rule is included in the technology file.

3.3 Out-Of-Order Issue Superscalar Processor



Technology File

- Mapping of generic instruction to machine instruction
- Timing information for machine instruction
- Dispatch Rule
- Number of FUs and buffers

Figure 8: Out-Of-Order Issue Superscalar Processor Model.

Out-of-order issue processor can execute more than one instruction at one time and rearrange the instruction order. When one instruction was stalled, the processor will look ahead several instructions to find a free instruction that can be executed now. So a look ahead buffer was inserted in front of each function unit, as shown in Figure 8.

In this model, each pipeline stage may has more than one function unit and each function unit may handle multiple instructions at the same time. In most cases, there is only one function unit that can handle multiple instructions in fetch and decode stage. In execution stage, multiple function units exist and each can handle one or multiple instructions. When one function unit is busy, other function units can execute instructions from buffer.

The technology file for this type of processor has the number of function unit and size of buffer in each pipeline stage in addition to the technology file of in-order issue processor. In estimation, the estimator schedule and bind machine instructions to function unit according to issue rule, data dependency, and function unit availability. Then the

performance is calculated as follows:

$$\begin{array}{lll} execution(B) = & MAX_{\text{all } FU_i} stop\_time(FU_i) & \\ \text{where} & FU_i \text{ is a function unit,} & \\ stop\_time(FU_i) = & \sum_{MI_{ij} \in FU_i} time(MI_{ij}) & \\ \text{where} & MI_{ij} \text{ are machine instructions executed by } FU_i. & \end{array} \tag{7}$$

3.4 Software Performance Estimator

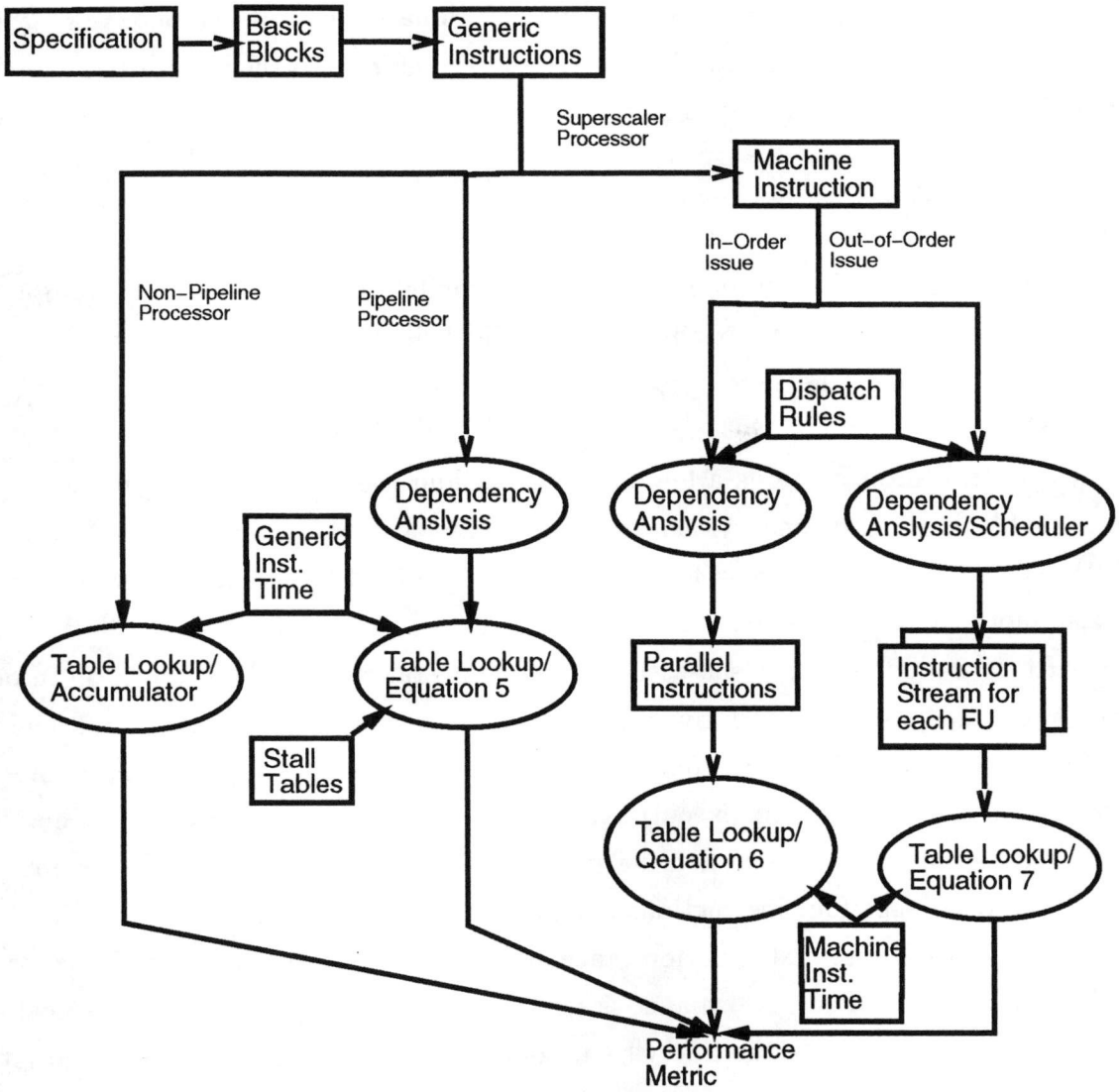


Figure 9: Software Performance Estimation Flowchart.

Figure 9 shows the flowchart of software performance estimator. Specification was divided into basic blocks, first. Then each basic block was compiled into generic instructions. If it is a non-pipelined processor, look up the timing in technology file. Sum up these timing to get the execution time. If it is a pipeline processor, check the data dependency and look up pipeline stall table. Calculate execution time by Equation 5. Otherwise the generic instructions were mapped to machine instruction. If it is a in-order issue superscalar processor, check the data dependency and issue rule to determine parallel instructions. Then calculate execution time by Equation 6. If it is a out-of-order issue superscalar processor, check data dependency and issue rule to schedule instruction into different function units. Then Calculate execution time by Equation 7.

4 Experiment

To test the accuracy of the proposed estimation model, we have run three examples on three processors. The three example are elliptical filter [10], medical system [9], and MPEG decoder [3]. The three processors are SPARC [1] [2], Intel Pentium [8], and PowerPC [7].

The elliptical filter contains a few basic blocks and most of its statements are inside one basic block. The medical system contains many basic blocks (more than thirty) and each basic block only contains a few statements. The MPEG decoder has large number of basic blocks and statements.

SPARC is a pipeline processor with four pipeline stages. Intel Pentium is an in-order issue superscalar processor with five pipeline stages. It can issue at most two instruction per clock cycle. PowerPC 604 is a out-of-order issue superscalar processor with six pipeline stages. Even though it has six executing function units, two single cycle integer unit, one multiple cycle integer unit, one floating point unit, one load/store unit, and one branch unit, it can only issue four instructions per cycle at most.

Table 2 shows the estimated performance and real performance of these three program on three processors. The performance is measured by clock cycle. To get the real performance, we first translate the SpecCharts [6] specification of these three system into C language. Then the C programs are compiled into machine instruction of each processors without optimization. The real performance is calculated based on these machine

	SPARC			Pentium			PowerPC 604		
	est. perf.	actual perf.	error	est. perf.	actual perf.	error	est. perf.	actual perf.	error
Elliptic Filter	887	868	2.1%	187	171	9.3%	473	431	9.7%
Medical System	792	754	5.0%	424	434	2.3%	574	571	0.5%
MPEG Decoder	428.5K	466.2K	-8.0%	169.2K	165.3K	2.3%	139.8K	128.8K	8.5%

Table 2: Experiment Results.

instructions assuming no cache miss.

From Table 2, we show that our estimation model can produce an accurate result on different application types and different processors. All estimation error are less than 10 percent.

5 Conclusion and Future Work

In this report, we have shown the software performance estimation that can produce accurate result with a few resource and computation time. This flexible approach was very suitable for design tools of system level hardware/software codesign.

Compiler optimization and cache miss are two issues about software estimation that are not counted in this estimation. When implementing the specification in software, most program will be compiled with optimization option. The speedup of optimized code depends on application, compiler, and processor. A simple solution is to find out the statistic speedup ratio by running many examples. The estimated performance can multiply this ratio to get the optimized performance metrics. This approach has been used in [4]. But it is very hard to find a good ratio for wide range of applications.

Another possible approach is to profile the optimization in technology file. For the most common optimization technique, put variables in register, the technology can has another version that assumes variables are in registers. The optimization ratio can be approximated by the ratio of register number and live variable number. This accuracy and techniques for profiling other optimization method need further researches.

Cache miss has similar effect as optimization and can be approximated by a ratio. This

simple solution has the same problem as optimization's. To accurately estimate the cache miss number, the system should have some sort of mapping between variable and memory address. With this information, the memory address distance between instruction can be calculated and cache miss can be predicted more precisely.

6 Acknowledgements

The authors would like to thank Jie Gong and Alfred Baehrenz Thordarson for providing the specification of experiment applications. We also want to thank Jin-Her Lin and Yirng-An Chen for compiling the specification into Pentium and PowerPC machine code.

References

[1] *CMOS BiCMOS Data Book*, Cypress Semiconductor, 1989

[2] M. Slater ed., *A Guide to RISC Microprocessor*, Academic Press, 1992

[3] A.B. Thordarson and D.D. Gajski, *Comparison of Manual and Automatic Behavioral Synthesis on MPEG Algorithm*, Technical Report ICS-05-09, UC-Irvine, 1995

[4] J. Gong, D.D. Gajski, and S. Narayan, *Software Estimation from Executable Specification*, Technical Report ICS-93-5, UC-Irvine, 1993

[5] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, ch. 7.3, 1994

[6] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, ch. 3.5, 1994

[7] *PowerPC 604, RISC Microprocessor User's Manual*, Motorola Inc., 1994

[8] *Pentium Family User's Manual*, Intel Corporation, 1994

[9] A. Wu, *A Microprocessor-based ultrasonic system for measuring bladder volumes*, Master Thesis in Electrical and Computer Engineering at University of Arizona, Tuscon, 1985.

[10] N. Dutt and C. Ramachandran, *Benchmarks for the 1992 high level synthesis workshop*, UC Irvine, Dept. of ICS, Technical Report 92-107, 1992.

A Technology File for SPARC (partial)

```
# Lines starting with '#' in the beginning of this file are comments.
#
# The number in the first line after the comment is processor type and pipeline
# depth. The processor type is encode as:
# 1 for pipeline, 2 for in_order_issue superscaler and
# 3 for out_order_issue superscaler.
#
# If this metric is for pipeline processor, the format after the
# processor_type is as follows:
#
# row_number, column_number
# row_number x column_number matrix for data dependent stall
# row_number x column_number matrix for resource conflict stall
#
# All these number are separated by space or new_line.
#
# If this metric file is for out_order_issue superscaler processor,
# the format after the processor_type is as follows:
#
# max_issue fu_type queue_size_type1 num_fu1
# ..............
#                       queue_size_typen num_fun
# In this part, max_issue is the max of concurrently issued instructions.
# fu_type is the number of function unit type.
# queue_size_type1 to queue_size_typen is the size of
# the queue in front of the function unit executing
# this type of machine instruction. num_fux is the number of copies
# of this fu type.
#
```

```
# exec_time type COMMENTS
# exec_time type COMMENTS
# ..........
# exec_time type COMMENTS
# XXXXXXXXXXXXXXXXXXXXXX
# Each line says the execution time and instruction type of a
# machine instruction. The first line is for the
# machine instruction whose id is 1. Instruction type
# is used for determine parallel issue instruction.
# Typically, instructions are grouped ina same type
# if they are executed by a same function unit.
# Anything after the two numbers in a line are comments.
# This part ends with a line starting with nonnumeric
# symbol.
#
# id_1 depend_1.1 depend_1.2 ... id_a depend_a.1 depend_a.2 X
# id_1 depend_1.1 depend_1.2 ... id_b depend_b.1 depend_b.2 X
# ........................................
# id_1 depend_1.1 depend_1.2 ... id_n depend_n.1 depend_n.2 X
# Each line says the mapped machine instruction for a generic
# instruction. The first line is for the first generic
# instruction. Each generic instruction must be mapped to one
# and only one line. 'depend_x_x' says dependency among these
# machine instructions for a same generic instruction.
# if 'depend_x_x' is -1, it means this machine instruction
# depends on the previous machine instruction. Every
# machine instruction can have two depedency at most.
# If there is no depedency, set 'depend_x_x' greater or
# equal to 0. EACH LINE MUST ENDS WITH A NONNUMERIC SYMBOL.
#
# number_1 type_1    .... number_a type_a XXX
# number_1 type_1    .... number_b type_b XXX
```

```
# ......................
# number_1 type_1   .... number_n type_n XXX
# Each line is a grouping rule. Up to the number of instructions
# in the specified types can be issued in parallel. Each
# line ends with a nonnumeric symbol. This part is the end
# of this metric file.
#
# If this file is for an in_order_issue superscaler processor,
# the format is the same as the one for out_order_issue
# superscaler processor without the first part, that is
# the following line:
# f_queue_size instr_type queue_size_type1 ... queue_size_typen
#
#
#
#
# This file is for SPARC processor whose architecture is
# in-order-issue superscaler.
1 4
138 138

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV1

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV2
```

```
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV3

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV4

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV5

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV6

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV7

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
```

```
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV8

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV9

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV10

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV11

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV12

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV13

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV14

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV15

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0  DIV16

1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1
1 1 0 0 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1  1 1 1 1 0 0 1 1 1 1  1 1 1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 1 1 1  0 1 1 1 1 0 DIV17

1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1
1 1 0 0 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1  1 1 1 1 0 0 1 1 1 1  1 1 1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 1 1 1  0 1 1 1 1 0 DIV18
```

B Technology File for Intel Pentium

```
2 5

1 1 id=1, Reg = Reg ALU Reg/Const, ALU means integer +, -, &, |
2 1 id=2, Reg = Reg ALU Mem, ALU means integer +, -, &, |
3 1 id=3, Mem = Reg/Const ALU Mem, ALU means integer +, -, &, |
1 1 id=4, MOV, mov memory to/from register/const
10 1 id=5, Reg = Reg MUL Reg/Mem
46 1 id=6, Reg = Reg DIV Reg/Mem
1 1 id=7, Reg = Reg COMPARE Reg/Const
2 1 id=8, Reg = Mem COMPARE Reg/Const
1 1 id=9, NOP
1 1 id=10, conditional/unconditional JUMP
4 2 id=11, RETURN plus pop, leave overhead
4 2 id=12, CALL plus push overhead
###End of information for instruction execution time and instruction type ###

4 0 0 1 -1 0 Reg = Const ALU Const
4 0 0 1 -1 0 Reg = Const ALU Reg
4 0 0 1 -1 0 Reg = Reg   ALU Const
4 0 0 1 -1 0 Reg = Reg   ALU Reg
4 0 0 1 -1 0 Reg = Mem ALU Const
4 0 0 1 -1 0 Reg = Const ALU Mem
4 0 0 2 -1 0 Reg = Mem ALU Reg
4 0 0 2 -1 0 Reg = Reg ALU Mem
4 0 0 2 -1 0 Reg = Mem ALU Mem
4 0 0 4 -1 0 1 -1 0 Reg = IndirectMem ALU Const
4 0 0 4 -1 0 1 -1 0 Reg = Const ALU IndirectMem
4 0 0  4 0 0 2 -1 -2 Reg = IndirectMem ALU Reg
4 0 0 4 0 0  2 -1 -2 Reg = Reg ALU IndirectMem
```

```
4 0 0 4 0 0 2 -1 -2 Reg = IndirectMem ALU Mem
4 0 0 4 0 0 2 -1 -2 Reg = Mem ALU IndirectMem
4 0 0 4 0 0 4 -2 0 2 -1 -2 Reg = IndirectMem ALU IndirectMem
4 0 0 3 -1 0 Mem = Const ALU Const
1 0 0 4 -1 0 Mem = Const ALU Reg
1 0 0 4 -1 0 Mem = Reg ALU Const
4 0 0 1 -1 0 4 -1 0 Mem = Reg ALU Reg
4 0 0 1 -1 0 4 -1 0 Mem = Mem ALU Const
4 0 0 1 -1 0 4 -1 0 Mem = Const ALU Mem
4 0 0 2 -1 0 4 -1 0 Mem = Mem ALU Reg
4 0 0 2 -1 0 4 -1 0 Mem = Reg ALU Mem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Mem = Mem + Mem
4 0 0 4 -1 0 1 -1 0 4 -1 0 Mem = IndirectMem ALU Const
4 0 0 4 -1 0 1 -1 0 4 -1 0 Mem = Const ALU IndirectMem
4 0 0 4 0 0 2 -1 -2 4 -1 0 Mem = IndirctMem ALU Reg
4 0 0 4 0 0 2 -1 -2 4 -1 0 Mem = Reg ALU IndirctMem
4 0 0 4 0 0 2 -1 -2 4 -1 0 Mem = IndirectMem ALU Mem
4 0 0 4 0 0 2 -1 -2 4 -1 0 Mem = IndirectMem ALU Mem
4 0 0 4 -1 0 4 0 0 2 -1 -2 4 -1 0 Mem = IndirectMem ALU IndirectMem
4 0 0 1 -1 0 Reg = Empty ALU Const
4 0 0 1 -1 0 Reg = Empty ALU Reg
4 0 0 1 -1 0 Reg = Empty ALU Mem
4 0 0 4 -1 0 1 -1 0 Reg = Empty ALU IndirectMem
4 0 0 1 -1 0 4 -1 0 Mem = Empty ALU Const
1 0 0 4 -1 0 Mem = Empty ALU Reg
4 0 0 1 -1 0 4 -1 0 Mem = Empty ALU Mem
4 0 0 4 -1 0 1 -1 0 4 -1 0 Mem = Empty ALU IndirectMem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Reg = Const MUL Const
4 0 0 1 -1 0 4 -1 0 Reg = Const MUL Reg
4 0 0 1 -1 0 4 -1 0 Reg = Reg MUL Const
4 0 0 5 -1 0 4 -1 0 Reg = Reg MUL Reg
4 0 0 1 -1 0 4 -1 0 Reg = Mem MUL Const
```

```
4 0 0 1 -1 0 4 -1 0 Reg = Const MUL Mem
4 0 0 5 -1 0 4 -1 0 Reg = Mem MUL Reg
4 0 0 5 -1 0 4 -1 0 Reg = Reg MUL Mem
4 0 0 5 -1 0 4 -1 0 Reg = Mem MUL Mem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Reg = IndirectMem MUL Const
4 0 0 4 0 0 1 -1 -2 4 -1 0 Reg = Const MUL IndirectMem
4 0 0 4 0 0 5 -1 -2 4 -1 0 Reg = IndirectMem MUL Reg
4 0 0 4 0 0 5 -1 -2 4 -1 0 Reg = Reg MUL IndirectMem
4 0 0 4 0 0 5 -1 -2 4 -1 0 Reg = IndirectMem MUL Mem
4 0 0 4 0 0 5 -1 -2 4 -1 0 Reg = Mem MUL IndirectMem
4 0 0 4 -1 0 4 0 0 5 -1 -2 4 -1 0 Reg = IndirectMem MUL IndirectMem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Mem = COnst MUL Const
4 0 0 1 -1 0 4 -1 0 Mem = COnst MUL Reg
4 0 0 1 -1 0 4 -1 0 Mem = Reg MUL Const
4 0 0 5 -1 0 4 -1 0 Mem = Reg MUL Reg
4 0 0 1 -1 0 4 -1 0 Mem = Mem MUL Const
4 0 0 1 -1 0 4 -1 0 Mem = COnst MUL Mem
4 0 0 5 -1 0 4 -1 0 Mem = Mem MUL Reg
4 0 0 5 -1 0 4 -1 0 Mem = Reg MUL Mem
4 0 0 5 -1 0 4 -1 0 Mem = Mem MUL Mem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Mem = IndirectMem MUL Const
4 0 0 4 0 0 1 -1 -2 4 -1 0 Mem = Const MUL IndirectMem
4 0 0  4 0 0 5 -1 -2 4 -1 0 Mem = IndirectMem MUL Reg
4 0 0  4 0 0 5 -1 -2 4 -1 0 Mem = Reg MUL IndirectMem
4 0 0 4 0 0 5 -1 -2 4 -1 0 Mem = IndirectMem MUL Mem
4 0 0 4 0 0 5 -1 -2 4 -1 0 Mem = Mem MUL IndirectMem
4 0 0 4 0 0 4 -2 0 5 -1 -2 4 -1 0 Mem = IndirectMem MUL IndirectMem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Reg = Const DIV Const
4 0 0 6 -1 0 4 -1 0 Reg = Const DIV Reg
4 0 0 1 -1 0 4 -1 0 Reg = Reg DIV Const
4 0 0 6 -1 0 4 -1 0 Reg = Reg DIV Reg
4 0 0 1 -1 0 4 -1 0 Reg = Mem DIV Const
```

```
4 0 0 6 -1 0 4 -1 0 Reg = Const DIV Mem
4 0 0 6 -1 0 4 -1 0 Reg = Mem DIV Reg
4 0 0 6 -1 0 4 -1 0 Reg = Reg DIV Mem
4 0 0 6 -1 0 4 -1 0 Reg = Mem DIV Mem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Reg = IndirectMem DIV Const
4 0 0 4 0 0 6 -1 -2 4 -1 0 Reg = Const DIV IndirectMem
4 0 0 4 0 0 6 -1 -2 4 -1 0 Reg = IndirectMem DIV Reg
4 0 0 4 0 0 6 -1 -2 4 -1 0 Reg = Reg DIV IndirectMem
4 0 0 4 0 0 6 -1 -2 4 -1 0 Reg = IndirectMem DIV Mem
4 0 0 4 0 0 6 -1 -2 4 -1 0 Reg = Mem DIV IndirectMem
4 0 0 4 -1 0 4 0 0 6 -1 -2 4 -1 0 Reg = IndirectMem DIV IndirectMem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Mem = COnst DIV Const
4 0 0 6 -1 0 4 -1 0 Mem = COnst DIV Reg
4 0 0 1 -1 0 4 -1 0 Mem = Reg DIV Const
4 0 0 6 -1 0 4 -1 0 Mem = Reg DIV Reg
4 0 0 1 -1 0 4 -1 0 Mem = Mem DIV Const
4 0 0 6 -1 0 4 -1 0 Mem = COnst DIV Mem
4 0 0 6 -1 0 4 -1 0 Mem = Mem DIV Reg
4 0 0 6 -1 0 4 -1 0 Mem = Reg DIV Mem
4 0 0 6 -1 0 4 -1 0 Mem = Mem DIV Mem
4 0 0 4 0 0 1 -1 -2 4 -1 0 Mem = IndirectMem DIV Const
4 0 0 4 0 0 6 -1 -2 4 -1 0 Mem = Const DIV IndirectMem
4 0 0  4 0 0 6 -1 -2 4 -1 0 Mem = IndirectMem DIV Reg
4 0 0  4 0 0 6 -1 -2 4 -1 0 Mem = Reg DIV IndirectMem
4 0 0 4 0 0 6 -1 -2 4 -1 0 Mem = IndirectMem DIV Mem
4 0 0 4 0 0 6 -1 -2 4 -1 0 Mem = Mem DIV IndirectMem
4 0 0 4 0 0 4 -2 0 6 -1 -2 4 -1 0 Mem = IndirectMem DIV IndirectMem
4 0 0 7 -1 0 4 -1 0 Reg = Const CMP Const
7 0 0 4 -1 0 Reg = Const CMP Reg
7 0 0 4 -1 0 Reg = Reg CMP Const
7 0 0 4 -1 0 Reg = Reg CMP Reg
8 0 0 4 -1 0 Reg = Mem CMP Const
```

```
8 0 0 4 -1 0 Reg = COnst CMP Mem
8 0 0 4 -1 0 Reg = Mem CMP Reg
8 0 0 4 -1 0 Reg = Reg CMP Mem
4 0 0 8 -1 0 4 -1 0 Reg = Mem CMP Mem
4 0 0 8 -1 0 4 -1 0 Reg = IndirectMem CMP Const
4 0 0 8 -1 0 4 -1 0 Reg = Const CMP IndirectMem
4 0 0 8 -1 0 4 -1 0 Reg = IndirectMem CMP Reg
4 0 0 8 -1 0 4 -1 0 Reg = Reg CMP IndirectMem
4 0 0 4 0 0  8 -1 -2 4 -1 0 Reg = IndirectMem CMP Mem
4 0 0 4 0 0  8 -1 -2 4 -1 0 Reg = Mem CMP IndirectMem
4 0 0 4 0 0 4 -2 0 8 -1 -2 4 -1 0 Reg = IndirectMem CMP IndirectMem
4 0 0 7 -1 0 Mem = Const CMP Const
7 0 0 Mem = Const CMP Reg
7 0 0 Mem = Reg CMP Const
7 0 0 Mem = Reg CMP Reg
8 0 0 Mem = Mem CMP Const
8 0 0 Mem = Const CMP Mem
8 0 0 Mem = Mem CMP Reg
8 0 0 Mem = Reg CMP Mem
4 0 0 8 -1 0 Mem = Mem CMP Mem
4 0 0 8 -1 0 Mem = IndirectMem CMP Const
4 0 0 8 -1 0 Mem = Const CMP IndirectMem
4 0 0 8 -1 0 Mem = IndirectMem CMP Reg
4 0 0 8 -1 0 Mem = Reg CMP IndirectMem
4 0 0 4 0 0 8 -1 -2 Mem = IndirectMem CMP Mem
4 0 0 4 0 0 8 -1 -2 Mem = Mem CMP IndirectMem
4 0 0 4 0 0 4 -2 0 8 -1 -2 Mem = IndirectMem CMP IndirectMem
4 0 0 Reg = Const
4 0 0 Reg = Reg
4 0 0 Reg = Mem
4 0 0 4 -1 0 Reg = IndirectMem
4 0 0 Mem = Const
```

```
4 0 0  Mem = Reg
4 0 0 4 -1 0 Mem = Mem
4 0 0 4 -1 0 4 -1 0 Mem = IndirectMem
4 0 0 4 -1 0 IndirectMem = Const
4 0 0 4 -1 0 IndirectMem = Reg
4 0 0 4 0 0 4 -1 0 IndirectMem = Mem
4 0 0 4 0 0 4 -2 0 4 -1 0 IndirecMem = IndirectMem
9 0 0 NOP
10 0 0 CJUMP
10 0 0 JUMP
11 0 0 RET
12 0 0 CALL
9 0 0 DEFAULT; End of Mapping; follows are issue rules


2 1 Less or equal to Two type 1 instrcutions
1 2 One type 2 instruction
```

C Technology File for PowerPC

```
3 6
4 5  2 2   2 1   2 1   2 1   2 1

1 1 id=1, Reg = Reg ALU Reg/Const, ALU means integer +, -, &, |
1 4 id=2, CALL plus push overhead
1 1 id=3, NOP
2 2 id=4, load
4 3 id=5, Reg = Reg MUL Reg/Mem
20 3 id=6, Reg = Reg DIV Reg/Mem
1 1 id=7, Reg = Reg COMPARE Reg/Const
1 4 id=8, conditional/unconditional JUMP
3 2 id=9, store
1 4 id=10, RETURN plus pop, leave overhead
###End of information for instruction execution time and instruction type ###

4 0 0 4 0 0 1 -1 -2 Reg = Const ALU Const
1 0 0 Reg = Const ALU Reg
4 0 0 1 -1 0 Reg = Reg   ALU Const
1 0 0 Reg = Reg   ALU Reg
4 0 0 4 0 0 4 -2 0 1 -1 -2 Reg = Mem ALU Const
4 0 0 4 0 0 4 -2 0 1 -1 -2 Reg = Const ALU Mem
4 0 0 4 -1 0 1 -1 0 Reg = Mem ALU Reg
4 0 0 4 -1 0 1 -1 0 Reg = Reg ALU Mem
4 0 0 4 0 0  4 -2  0 4 -2 0 1 -1 -2 Reg = Mem ALU Mem
4 0 0 4 0 0 1 -2 0 1 -1 -2 Reg = IndirectMem ALU Const
4 0 0 4 0 0 1 -2 0 1 -1 -2 Reg = Const ALU IndirectMem
4 0 0  1 -1 0 1 -1 0 Reg = IndirectMem ALU Reg
4 0 0 1 -1 0 1 -1 0 Reg = Reg ALU IndirectMem
4 0 0 4 0 0 1 -2 0  4 -2 0 1 -1 -2 Reg = IndirectMem ALU Mem
```

```
4 0 0 4 0 0 1 -2 0  4 -2 0 1 -1 -2 Reg = Mem ALU IndirectMem
4 0 0 4 0 0 1 -2 0 1 -2 0 1 -1 -2 Reg = IndirectMem ALU IndirectMem
4 0 0 4 0 0 4 0 0 1 -2 -3 9 -1 -2 Mem = Const ALU Const
4 0 0 4 0 0 1 -2 0 9 -1 -2 Mem = Const ALU Reg
4 0 0 4 0 0 1 -2 0 9 -1 -2 Mem = Reg ALU Const
4 0 0 1 0 0 9 -1 -2 Mem = Reg ALU Reg
4 0 0 4 0 0 4 -2 0 4 0 0 1 -2 -3 9 -1 -2 Mem = Mem ALU Const
4 0 0 4 0 0 4 -2 0 4 0 0 1 -2 -3 9 -1 -2 Mem = Const ALU Mem
4 0 0 4 0 0 4 -2 0 1 -1 -2 9 -1 -3 Mem = Mem ALU Reg
4 0 0 4 0 0 4 -2 0 1 -1 -2 9 -1 -3 Mem = Reg ALU Mem
4 0 0 4 0 0 4 -2 0 4 -2 0 4 0 0 1 -2 -3 9 -1 -2 Mem = Mem + Mem
4 0 0 4 0 0 4 0 0 1 -3 0 1 -1 -3 9 -1 -3 Mem = IndirectMem ALU Const
4 0 0 4 0 0 4 0 0 1 -3 0 1 -1 -3 9 -1 -3 Mem = Const ALU IndirectMem
4 0 0 4 0 0 1 -2 0 1 -1 -3 9 -1 -3 Mem = IndirctMem ALU Reg
4 0 0 4 0 0 1 -2 0 1 -1 -3 9 -1 -3 Mem = Reg ALU IndirctMem
4 0 0 4 0 0 4 -2 0 1 -2 0 4 0 0 1 -2 -3 9 -1 -2 Mem = IndirectMem ALU Mem
4 0 0 4 0 0 4 -2 0 1 -2 0 4 0 0 1 -2 -3 9 -1 -2 Mem = IndirectMem ALU Mem
4 0 0 4 0 0 1 -2 0 1 -2 0 4 0 0 1 -2 -3 9 -1 -2 Mem = IndirectMem ALU IndirectMem
4 0 0 1 -1 0 Reg = Empty ALU Const
1 -1 0 Reg = Empty ALU Reg
4 0 0 4 -1 0 1 -1 0 Reg = Empty ALU Mem
4 0 0 1 -1 0 1 -1 0 Reg = Empty ALU IndirectMem
4 0 0 4 0 0 1 -2 0 9 -1 -2 Mem = Empty ALU Const
4 0 0 1 0 0 9 -1 -2 Mem = Empty ALU Reg
4 0 0 4 0 0 4 -2 0 1 -1 0 9 -1 -3 Mem = Empty ALU Mem
4 0 0 4 0 0 1 -2 0 1 -1 0 9 -1 -3 Mem = Empty ALU IndirectMem
4 0 0 1 -1 0 Reg = Const MUL Const
1 0 0 Reg = Const MUL Reg
1 0 0 Reg = Reg MUL Const
5 0 0 Reg = Reg MUL Reg
4 0 0 4 -1 0 1 -1 0 Reg = Mem MUL Const
4 0 0 4 -1 0 1 -1 0 Reg = Const MUL Mem
```

```
4 0 0 4 -1 0 5 -1 0 Reg = Mem MUL Reg
4 0 0 4 -1 0 5 -1 0 Reg = Reg MUL Mem
4 0 0 4 0 0 4 -2 0 4 -2 0 5 -1 -2 Reg = Mem MUL Mem
4 0 0 1 -1 0 1 -1 0 Reg = IndirectMem MUL Const
4 0 0 1 -1 0 1 -1 0 Reg = Const MUL IndirectMem
4 0 0 1 -1 0 5 -1 0 Reg = IndirectMem MUL Reg
4 0 0 1 -1 0 5 -1 0 Reg = Reg MUL IndirectMem
4 0 0 4 0 0 4 -2 0 1 -2 0 5 -1 -2 Reg = IndirectMem MUL Mem
4 0 0 4 0 0 4 -2 0 1 -2 0  5 -1 -2 Reg = Mem MUL IndirectMem
4 0 0 4 0 0 1 -2 0  1 -2 0 5 -1 -2 Reg = IndirectMem MUL IndirectMem
4 0 0 4 0 0 1 -2 0 9 -1 -2 Mem = COnst MUL Const
4 0 0 1 0 0 9 -1 -2 Mem = COnst MUL Reg
4 0 0 1 0 0 9 -1 -2 Mem = Reg MUL Const
4 0 0 5 0 0 9 -1 -2 Mem = Reg MUL Reg

4 0 0 4 0 0 4 -2 0 1 -1 0 9 -1 -3 Mem = Mem MUL Const
4 0 0 4 0 0 4 -2 0 1 -1 0 9 -1 -3 Mem = COnst MUL Mem
4 0 0 4 0 0 4 -2 0 5 -1 0 9 -1 -3 Mem = Mem MUL Reg
4 0 0 4 0 0 4 -2 0 5 -1 0 9 -1 -3 Mem = Reg MUL Mem
4 0 0 4 0 0 4 -2 0 4 -2 0 4 0 0 5 -2 -3 9 -1 -2 Mem = Mem MUL Mem
4 0 0 4 0 0 1 -2 0 1 -1 0 9 -1 -3 Mem = IndirectMem MUL Const
4 0 0 4 0 0 1 -2 0 1 -1 0 9 -1 -3 Mem = Const MUL IndirectMem
4 0 0  4 0 0 1 -2 0 5 -1 0  9 -1 -3 Mem = IndirectMem MUL Reg
4 0 0  4 0 0 1 -2 0 5 -1 0  9 -1 -3 Mem = Reg MUL IndirectMem
4 0 0 4 0 0 4 -2 0 1 -2 0 4 0 0 5 -2 -3 9 -1 -2 Mem = IndirectMem MUL Mem
4 0 0 4 0 0 4 -2 0 1 -2 0 4 0 0 5 -2 -3 9 -1 -2 Mem = Mem MUL IndirectMem
4 0 0 4 0 0 1 -2 0  1 -2 0 4 0 0 5 -2 -3 9 -1 -2 Mem = IndirectMem MUL IndirectMem
4 0 0 1 -1 0 Reg = Const DIV Const
4 0 0 6 -1 0 Reg = Const DIV Reg
1 0 0 Reg = Reg DIV Const
6 0 0 Reg = Reg DIV Reg
4 0 0 1 -1 0 Reg = Mem DIV Const
```

```
4 0 0 4 0 0 6 -1 -2 Reg = Const DIV Mem
4 0 0 6 -1 0 Reg = Mem DIV Reg
4 0 0 6 -1 0 Reg = Reg DIV Mem
4 0 0 4 0 0 6 -1 -2 Reg = Mem DIV Mem
4 0 0 1 -1 0 1 -1 0 Reg = IndirectMem DIV Const
4 0 0 4 0 0 1 -2 0 6 -1 -2 Reg = Const DIV IndirectMem
4 0 0 1 -1 0 6 -1 0 Reg = IndirectMem DIV Reg
4 0 0 1 -1 0 6 -1 0 Reg = Reg DIV IndirectMem
4 0 0 4 0 0 1 -2 0 6 -1 -2 Reg = IndirectMem DIV Mem
4 0 0 4 0 0 1 -2 0 6 -1 -2 Reg = Mem DIV IndirectMem
4 0 0 4 0 0 1 -2 0  1 -2 0 6 -1 -2 Reg = IndirectMem DIV IndirectMem
4 0 0 1 -1 0 9 -1 0 Mem = COnst DIV Const
4 0 0 6 -1 0 9 -1 0 Mem = COnst DIV Reg
1 0 0 9 -1 0 Mem = Reg DIV Const
6 0 0 9 -1 0 Mem = Reg DIV Reg
4 0 0 1 -1 0 9 -1 0 Mem = Mem DIV Const
4 0 0 4 0 0 6 -1 -2 9 -1 0 Mem = COnst DIV Mem
4 0 0 6 -1 0 9 -1 0 Mem = Mem DIV Reg
4 0 0 6 -1 0 9 -1 0 Mem = Reg DIV Mem
4 0 0 4 0 0 6 -1 -2 9 -1 0 Mem = Mem DIV Mem
4 0 0 1 -1 0 1 -1 0 9 -1 0 Mem = IndirectMem DIV Const
4 0 0 4 0 0 1 -2 0 6 -1 -2 9 -1 0 Mem = Const DIV IndirectMem
4 0 0  1 -1 0 6 -1 0  9 -1 0 Mem = IndirectMem DIV Reg
4 0 0  1 -1 0 6 -1 0  9 -1 0 Mem = Reg DIV IndirectMem
4 0 0 4 0 0 1 -2 0 6 -1 -2 9 -1 0 Mem = IndirectMem DIV Mem
4 0 0 4 0 0 1 -2 0 6 -1 -2 9 -1 0 Mem = Mem DIV IndirectMem
4 0 0 4 0 0 1 -2 0  1 -2 0 6 -1 -2 9 -1 0 Mem = IndirectMem DIV IndirectMem
4 0 0 7 -1 0 Reg = Const CMP Const
4 0 0 7 0 0 Reg = Const CMP Reg
7 0 0 Reg = Reg CMP Const
7 0 0 Reg = Reg CMP Reg
4 0 0 7 -1 0 Reg = Mem CMP Const
```

```
4 0 0 4 0 0 7 -1 -2 Reg = COnst CMP Mem
4 0 0 7 -1 0 Reg = Mem CMP Reg
4 0 0 7 -1 0 Reg = Reg CMP Mem
4 0 0 4 0 0 7 -1 -2 Reg = Mem CMP Mem
4 0 0 1 -1 0 7 -1 0 Reg = IndirectMem CMP Const
4 0 0 4 0 0 1 -2 0 7 -1 -2 Reg = Const CMP IndirectMem
4 0 0 1 -1 0 7 -1 0 Reg = IndirectMem CMP Reg
4 0 0 1 -1 0 7 -1 0 Reg = Reg CMP IndirectMem
4 0 0 4 0 0  1 -2 0 7 -1 -2 Reg = IndirectMem CMP Mem
4 0 0 4 0 0  1 -2 0 7 -1 -2 Reg = Mem CMP IndirectMem
4 0 0 4 0 0 1 -2 0 1 -2 0 7 -1 -2 Reg = IndirectMem CMP IndirectMem
4 0 0 7 -1 0 Mem = Const CMP Const
4 0 0 7 0 0 Mem = Const CMP Reg
7 0 0 Mem = Reg CMP Const
7 0 0 Mem = Reg CMP Reg
4 0 0 7 -1 0 Mem = Mem CMP Const
4 0 0 4 0 0 7 -1 -2 Mem = COnst CMP Mem
4 0 0 7 -1 0 Mem = Mem CMP Reg
4 0 0 7 -1 0 Mem = Reg CMP Mem
4 0 0 4 0 0 7 -1 -2 Mem = Mem CMP Mem
4 0 0 1 -1 0 7 -1 0 Mem = IndirectMem CMP Const
4 0 0 4 0 0 1 -2 0 7 -1 -2 Mem = Const CMP IndirectMem
4 0 0 1 -1 0 7 -1 0 Mem = IndirectMem CMP Reg
4 0 0 1 -1 0 7 -1 0 Mem = Reg CMP IndirectMem
4 0 0 4 0 0  1 -2 0 7 -1 -2 Mem = IndirectMem CMP Mem
4 0 0 4 0 0  1 -2 0 7 -1 -2 Mem = Mem CMP IndirectMem
4 0 0 4 0 0 1 -2 0 1 -2 0 7 -1 -2 Mem = IndirectMem CMP IndirectMem
4 0 0 Reg = Const
4 0 0 Reg = Reg
4 0 0 Reg = Mem
4 0 0 1 -1 0 Reg = IndirectMem
4 0 0 9 -1 0 Mem = Const
```

```
9 0 0  Mem = Reg
4 0 0 9 -1 0 Mem = Mem
4 0 0 1 -1 0 9 -1 0 Mem = IndirectMem
4 0 0 1 -1 0 9 -1 0 IndirectMem = Const
4 0 0 1 -1 0 9 -1 0 IndirectMem = Reg
4 0 0 4 0 0 1 -2 0 9 -1 -2 IndirectMem = Mem
4 0 0 4 0 0 1 -2 0 1 -2 0 9 -1 -2 IndirecMem = IndirectMem
3 0 0 NOP
8 0 0 CJUMP
8 0 0 JUMP
10 0 0 RET
2 0 0 CALL
9 0 0 DEFAULT; End of Mapping; floowing are issue rule
```



```
2 1   1 2   1 3 type 1
2 1   1 2   1 4 type 2
2 1   1 3   1 4 type 3
1 1   1 2   1 3   1 4 type 4
```