Software Performance Estimation for Toshiba TLCS-R3900

En-Shou Chang
Daniel D. Gajski

Technical Report #96–19
May 20, 1996

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
(714) 824-8059

echang@ics.uci.edu

Abstract

This report contains information about software performance of Toshiba TLCS-R3900 RISC processor evaluated by a software estimation technique proposed by J. Gong et. al. This technique decomposes the program into basic block then evaluates total execution time by analysis execution flow. The execution time of basic block is computed by compiling subprogram into generic instructions then mapping to real instruction. In addition, we analyze the pipeline stall phenomenon for TLCS-R3900. A processor profile is proposed to count the effects. Based on this generic estimation model, our estimator can produce accurate estimation without large computation time and precious resource, such as compilers or simulators for each processor.

1 Introduction

Software performance can be measured in three ways[1]: in-circuit simulation, software simulation, and software estimation. The comparison of these three approaches are summarized 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.

Through the discussion above, we know the software estimator is more suitable for design automation tools because its flexibility and speed allow design space exploration. In this report, we develop an estimation method that can produce accurate enough metrics for system level design tools for Toshiba 32-bit RISC microprocessor TLCS-R3900 family[2].

2 Software performance estimation

Software performance estimation can be divided into two steps: (1) flow analysis and (2) 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[1].

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:

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

where b_i is basic block.

This report takes the same approach in [1] to do software performance estimation for TLCS-R3900 processor. In basic block estimation, we adapt the generic estimation model[3], which is 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 of basic block can be calculated as follows:

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

where I_j is the generated generic instruction.

3 Estimation model for TLCS-R3900

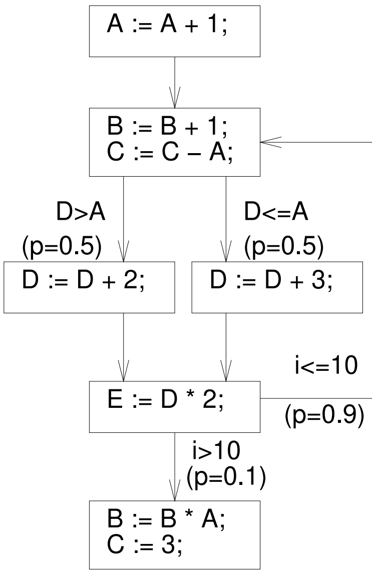
According to the specification of TLCS-R3900 in[2], we can model TLCS-R3900 as a 5-stage pipeline pro-

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: Comparison of software estimation approaches

```
A := A + 1;

for i in 1 to 10 loop
   B := B + 1;
   C := C − A;

   if (D > A) then
      D := D + 2;
   else
      D := D + 3;
   end if;

   E := D * 2;
end loop;

B := B * A;
C := 3;
```

(a)



(b)

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

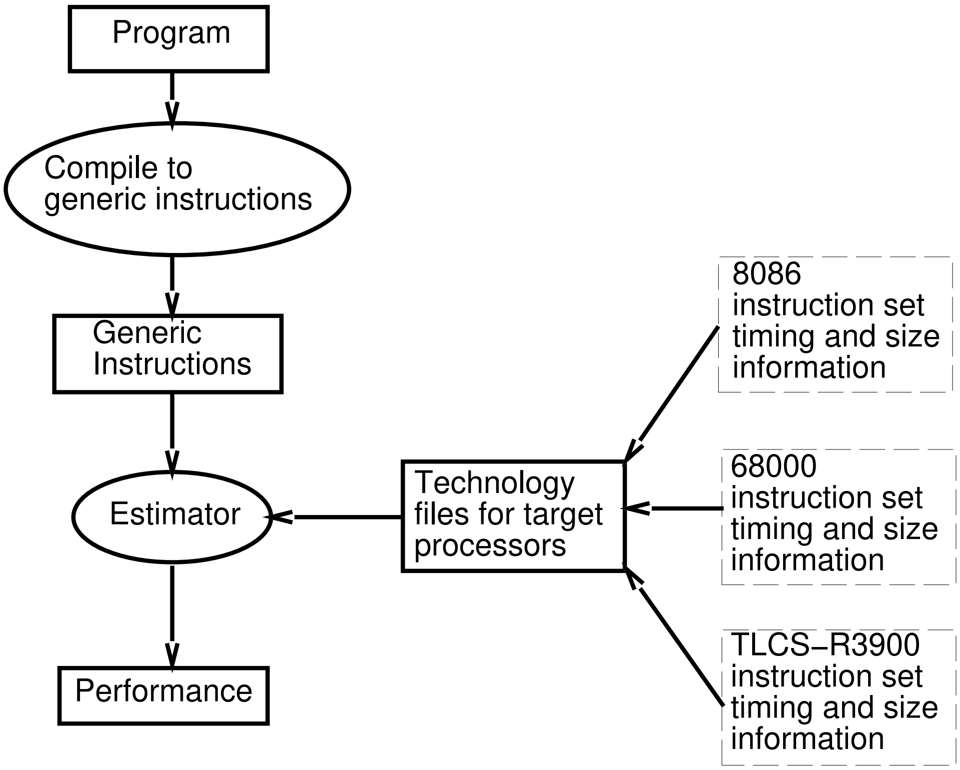


Figure 2: Generic estimation model

cessor with certain exceptions.

Since the generic estimation model has shown good results in estimation for non-pipelined processor in[3], we now extend it to measure the performance of TLCS-R3900, which is a pipeline processor. Fundamentally, the overlap time of pipeline instruction can be reflected in

$$execution(B) = \sum_{I_j \in B} (time(I_j) - pipe\_depth + 1) + pipe\_depth - 1 \quad (3)$$

, but the result is not accurate due to pipeline stall.

TLCS-R3900 can issue one instruction per clock cycle in ideal situation. When one instruction depend on the result of previous instructions or content a same resource of previous instructions that are still in the pipeline, this instruction would be paused and pipeline was stalled. Figure 3 shows an example of pipeline stall from TLCS-R3900 user manual[2].

The hardware model we use for TLCS-R3900 was depicted in Figure 4. The TLCS-R3900 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.

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 introduced when the instruction indexed 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 instruction, while resource contention pipeline stall table is used for instructions that have no data dependency but require the same resource at the same time. The execution time of a basic block can be formulated as Eq.(3).

Both technology files, for generic instruction timing and for pipeline stall insertion, are attached in Appendix A and B respectively.

4 Software performance for CPU core

Performance of three typical programs on TLCS-R3900 are estimated in this section. We also quote software performance estimation of some commercial products from[4] for comparison.

The elliptical filter[5] contains a few basic blocks and most of its statements are inside one basic block. The medical system[6] contains many basic blocks (more than thirty) and each basic block only contains a few statements. The MPEG decoder[7] has large number of basic blocks and statements.

Table 2 shows the estimated performance of these programs on four processors. The performance is measured by clock cycle.

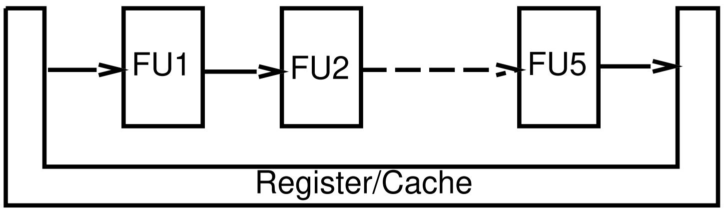
5 Practical performance estimation for TLCS-R3900

In general, commercial products use slow but inexpensive DRAM as main memory and increase system performance by way of using cache. There is certain amount of build-in cache with TLCS-R3900 processor family. It take only one clock cycle to access data from cache. But, it would take several clock cycle to access data if there is a cache-missing. Since the cache misses depend on characteristics and size of program, replacement policies, size, and levels of cache, the only feasible way known is that user provides the miss ratio and cost. Table 3 shows some estimation of practical

LW r2,20(r0)	F	D	E	M	W		
ADD r3,r1,r2		F	D	ES	E	M	W

`F` : Fetch
`D` : Decode
`E` : Execution
`M` : Memory access
`W` : Write to register
`ES` : Pipeline stall

Figure 3: An example of pipeline stall



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 4: Hardware model of TLCS-R3900

$$execution(B) = \sum_{I_j \in B} (time(I_j) + stall(I_j - 1, I_j) - pipe\_depth + 1) \; + \; pipe\_depth - 1 \tag{4}$$

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

This table
is not authorized
to be put on public domain
by project sponsor.

Table 2: Software performance of 4 CPU cores

program	performance estimated		hardware specification
	without cache-missing	with cache-missing	
elliptic filter	370	444	clock period = 50MHz
medical system	1031	1237	DRAM access time = 70ns
MPEG decoder	496.7K	596.0K	cache miss-ratio = 5%
elliptic filter	370	666	clock period = 50MHz
medical system	1031	1856	DRAM access time = 70ns
MPEG decoder	496.7K	894.1K	cache miss-ratio = 20%
elliptic filter	370	370	clock period = 20MHz
medical system	1031	1031	DRAM access time = 40ns
MPEG decoder	496.7K	496.7K	cache miss-ratio = 20%

Table 3: Estimation of practical performance of TLCS-R3900

software performance of TLCS-R3900.

6 Concluding remarks

In this report, we have shown the software performance estimation that can produce result with a few resource and computation time for Toshiba TLCS-R3900 processor. 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 on which we didn't do well so far. When implementing the specification in software, most program will be compiled with optimization option. The speed-up of optimized code depends on the application, the compiler, and processor. A feasible solution is to find out a statistically the speed-up ratios by running a certain number of examples. The estimated performance can be multiplied by these ratios to get the optimized performance. Methods that can evaluate the performance of optimized code by profiling the optimization techniques, such as loop-unrolling and register allocation, require further study.

We computed cache overhead by user-provided miss ratio and cost. Since sometimes the user does not know much about the performance of the cache, some methods that can evaluate cache overhead according to profile of cache mechanism and program characteristics require further study, too.

References

[1] D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and design of embedded systems.* New Jersey: Prentice Hall, 1994.

[2] *32-Bit RISC MICROPROCESSOR TLCS-R3900 FAMILY USERS MANUAL.* Toshiba Co., 1995.

[3] J. Gong, D. Gajski, and S. Narayan, "Software estimation from executable specifications." UC Irvine, Dept. of ICS, Technical Report 93-05,1993.

[4] C.-Y. Huang and D. Gajski, "Software performance estimation for pipeline and superscalar processors." UC Irvine, Dept. of ICS, Technical Report 95-20,1995.

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

[6] A. Wu, "A microprocessor-based ultrasonic system for measuring bladder volumes." Master Thesis in Electrical and Computer Engineering at University of Arizona, Tucson, 1985., 1985.

[7] A. Thordarson and D. Gajski, "Comparison of manual and automatic behavioral synthesis on mpeg algorithm." UC Irvine, Dept. of ICS, Technical Report 95-09,1995.

A Technology file for TLCS-R3900

```
## Anything after '#' are comments.
## This is the technology file for Toshiba TLCS-R3900 processor (CY7C601).
## DirectMem means direct memory addressing.
## IndirectMem means indirect memory addressing.
## The mapped instruction:      M Multiplication
##                              A ALU
##                              D Devide  : 34 extra cycle,
##                                          LAB -- 3 extra word fill-in
##                              S Store
##                              X NOP
##                              J Jump    : 1 extra cycle
##                              B Branch  : 1 extra cycle
##                              ? stall
##                              L load
#  OP    DESTINATION    SOURCE1     SOURCE2     time(clock cycles) size(bytes)
   DIV     Register     Constant     Constant   42  24 LL?D
   DIV     Register     Constant     Register   41  20 L?D
   DIV     Register     Register     Constant   41  20 L?D
   DIV     Register     Register     Register   39  16 D
   DIV     Register    DirectMem     Constant   42  24 LL?D
   DIV     Register     Constant    DirectMem   42  24 LL?D
   DIV     Register    DirectMem     Register   41  20 L?D
   DIV     Register     Register    DirectMem   41  20 L?D
   DIV     Register    DirectMem    DirectMem   42  24 LL?D
   DIV     Register  IndirectMem     Constant   43  28 LLL?D
   DIV     Register     Constant  IndirectMem   43  28 LLL?D
   DIV     Register  IndirectMem     Register   43  24 L?L?D
   DIV     Register     Register  IndirectMem   43  24 L?L?D
   DIV     Register  IndirectMem    DirectMem   43  28 LLL?D
   DIV     Register    DirectMem  IndirectMem   43  28 LLL?D
   DIV     Register  IndirectMem  IndirectMem   44  32 LLLL?D
   DIV    DirectMem     Constant     Constant   43  28 LL?DS
   DIV    DirectMem     Constant     Register   42  24 L?DS
   DIV    DirectMem     Register     Constant   42  24 L?DS
   DIV    DirectMem     Register     Register   40  20 DS
   DIV    DirectMem    DirectMem     Constant   43  28 LL?DS
   DIV    DirectMem     Constant    DirectMem   43  28 LL?DS
   DIV    DirectMem    DirectMem     Register   42  24 L?DS
   DIV    DirectMem     Register    DirectMem   42  24 L?DS
   DIV    DirectMem    DirectMem    DirectMem   43  28 LL?DS
   DIV    DirectMem  IndirectMem     Constant   44  32 LLL?DS
   DIV    DirectMem     Constant  IndirectMem   44  32 LLL?DS
   DIV    DirectMem  IndirectMem     Register   44  28 L?L?DS
   DIV    DirectMem     Register  IndirectMem   44  28 L?L?DS
   DIV    DirectMem  IndirectMem    DirectMem   44  32 LLL?DS
   DIV    DirectMem    DirectMem  IndirectMem   44  32 LLL?DS
   DIV    DirectMem  IndirectMem  IndirectMem   45  36 LLLL?DS
   ALU     Register     Constant     Constant    8  12 LL?A
   ALU     Register     Constant     Register    7   8 L?A
   ALU     Register     Register     Constant    7   8 L?A
```

ALU	Register	Register	Register	5	4	A
ALU	Register	DirectMem	Constant	8	12	LL?A
ALU	Register	Constant	DirectMem	8	12	LL?A
ALU	Register	DirectMem	Register	7	8	L?A
ALU	Register	Register	DirectMem	7	8	L?A
ALU	Register	DirectMem	DirectMem	8	12	LL?A
ALU	Register	IndirectMem	Constant	9	16	LLL?A
ALU	Register	Constant	IndirectMem	9	16	LLL?A
ALU	Register	IndirectMem	Register	9	12	L?L?A
ALU	Register	Register	IndirectMem	9	12	L?L?A
ALU	Register	IndirectMem	DirectMem	9	16	LLL?A
ALU	Register	DirectMem	IndirectMem	9	16	LLL?A
ALU	Register	IndirectMem	IndirectMem	10	20	LLLL?A
ALU	DirectMem	Constant	Constant	9	16	LL?AS
ALU	DirectMem	Constant	Register	8	12	L?AS
ALU	DirectMem	Register	Constant	8	12	L?AS
ALU	DirectMem	Register	Register	6	8	AS
ALU	DirectMem	DirectMem	Constant	9	16	LL?AS
ALU	DirectMem	Constant	DirectMem	9	16	LL?AS
ALU	DirectMem	DirectMem	Register	8	12	L?AS
ALU	DirectMem	Register	DirectMem	8	12	L?AS
ALU	DirectMem	DirectMem	DirectMem	9	16	LL?AS
ALU	DirectMem	IndirectMem	Constant	10	20	LLL?AS
ALU	DirectMem	Constant	IndirectMem	10	20	LLL?AS
ALU	DirectMem	IndirectMem	Register	10	16	L?L?AS
ALU	DirectMem	Register	IndirectMem	10	16	L?L?AS
ALU	DirectMem	IndirectMem	DirectMem	10	20	LLL?AS
ALU	DirectMem	DirectMem	IndirectMem	10	20	LLL?AS
ALU	DirectMem	IndirectMem	IndirectMem	11	24	LLLL?AS
ALU	Register	Empty	Constant	7	8	L?A
ALU	Register	Empty	Register	5	4	A
ALU	Register	Empty	DirectMem	7	8	L?A
ALU	Register	Empty	IndirectMem	9	12	L?L?A
ALU	DirectMem	Empty	Constant	8	12	L?AS
ALU	DirectMem	Empty	Register	6	8	AS
ALU	DirectMem	Empty	DirectMem	8	12	L?AS
ALU	DirectMem	Empty	IndirectMem	10	16	L?L?AS
MUL	Register	Constant	Constant	8	12	LL?M
MUL	Register	Constant	Register	7	8	L?M
MUL	Register	Register	Constant	7	8	L?M
MUL	Register	Register	Register	5	4	M
MUL	Register	DirectMem	Constant	8	12	LL?M
MUL	Register	Constant	DirectMem	8	12	LL?M
MUL	Register	DirectMem	Register	7	8	L?M
MUL	Register	Register	DirectMem	7	8	L?M
MUL	Register	DirectMem	DirectMem	8	12	LL?M
MUL	Register	IndirectMem	Constant	9	16	LLL?M
MUL	Register	Constant	IndirectMem	9	16	LLL?M
MUL	Register	IndirectMem	Register	9	12	L?L?M
MUL	Register	Register	IndirectMem	9	12	L?L?M
MUL	Register	IndirectMem	DirectMem	9	16	LLL?M
MUL	Register	DirectMem	IndirectMem	9	16	LLL?M

MUL	Register	IndirectMem	IndirectMem	10	20	LLLL?M
MUL	DirectMem	Constant	Constant	9	16	LL?MS
MUL	DirectMem	Constant	Register	8	12	L?MS
MUL	DirectMem	Register	Constant	8	12	L?MS
MUL	DirectMem	Register	Register	6	8	MS
MUL	DirectMem	DirectMem	Constant	9	16	LL?MS
MUL	DirectMem	Constant	DirectMem	9	16	LL?MS
MUL	DirectMem	DirectMem	Register	8	12	L?MS
MUL	DirectMem	Register	DirectMem	8	12	L?MS
MUL	DirectMem	DirectMem	DirectMem	9	16	LL?MS
MUL	DirectMem	IndirectMem	Constant	10	20	LLL?MS
MUL	DirectMem	Constant	IndirectMem	10	20	LLL?MS
MUL	DirectMem	IndirectMem	Register	10	16	L?L?MS
MUL	DirectMem	Register	IndirectMem	10	16	L?L?MS
MUL	DirectMem	IndirectMem	DirectMem	10	20	LLL?MS
MUL	DirectMem	DirectMem	IndirectMem	10	20	LLL?MS
MUL	DirectMem	IndirectMem	IndirectMem	11	24	LLLL?MS
CMP	Register	Constant	Constant	0	0	
CMP	Register	Constant	Register	5	4	L?
CMP	Register	Register	Constant	5	4	L?
CMP	Register	Register	Register	0	0	
CMP	Register	DirectMem	Constant	6	8	LL?
CMP	Register	Constant	DirectMem	6	8	LL?
CMP	Register	DirectMem	Register	5	4	L?
CMP	Register	Register	DirectMem	5	4	L?
CMP	Register	DirectMem	DirectMem	6	8	LL?
CMP	Register	IndirectMem	Constant	7	12	LLL?
CMP	Register	Constant	IndirectMem	7	12	LLL?
CMP	Register	IndirectMem	Register	7	8	L?L?
CMP	Register	Register	IndirectMem	7	8	L?L?
CMP	Register	IndirectMem	DirectMem	7	12	LLL?
CMP	Register	DirectMem	IndirectMem	7	12	LLL?
CMP	Register	IndirectMem	IndirectMem	8	16	LLLL?
MOV	Register	Empty	Constant	5	4	L?
MOV	Register	Empty	Register	5	4	A?
MOV	Register	Empty	DirectMem	5	4	L?
MOV	Register	Empty	IndirectMem	7	8	L?L?
MOV	DirectMem	Empty	Constant	6	8	LS
MOV	DirectMem	Empty	Register	5	4	S
MOV	DirectMem	Empty	DirectMem	6	8	LS
MOV	DirectMem	Empty	IndirectMem	8	12	L?LS
MOV	IndirectMem	Empty	Constant	7	12	LLS
MOV	IndirectMem	Empty	Register	7	8	L?S
MOV	IndirectMem	Empty	DirectMem	7	12	LLS
MOV	IndirectMem	Empty	IndirectMem	8	16	LLLS
NOP	Empty	Empty	Empty	5	4	X
CJUMP	Empty	Empty	Empty	6	4	B
JUMP	Empty	Empty	Empty	6	4	J
RET	Empty	Empty	Empty	8	12	JLA
CALL	Empty	Empty	Empty	8	12	ASJ
DEFAULT	Empty	Empty	Empty	5	4	X

B Pipeline stall file for TLCS-R3900 (partial)

```
# Lines starting with '#' in the beginning of this file are comments.
#
# The number in the first line after the comment is processor type,
#       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 matric for data dependent stall
#               row_number x column_number matric for resource conflict stall
#
# All these number are seperated 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 copyies
#               of this fu type.
#
#               exec_time type COMMENTS
#               exec_time type COMMENTS
#                       ...........
#               exec_time type COMMENTS
#               XXXXXXXXXXXXXXXXXXXXXXX
#       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
```

```
#               instrtion. Each generic instruction must be mapped to one
#               and only one line. 'depend_x_x' says depedency 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 symbl. 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 Toshiba TLCS-R3900 processor whose architecture is
# in-order-issue superscaler.
1 5
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 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  CMP13

0 0 0 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  CMP14

0 0 0 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  CMP15

0 0 0 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
```

```
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  CMP16

0 0 0 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  MOV1

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  MOV2

0 0 0 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  MOV3

0 0 0 1 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1  0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  MOV4

0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0  MOV5

    ..............
    ..............
    ..............
    ..............
    ..............
    ..............

    ..............
```