LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Tests for Monsoon Instruction Subset 0

Computation Structures Group Memo 307
April 1990

Jonathan Young

This report describes research done at the Laboratory of Computer Science of the Massachusetts Institute of Technology. Funding for the Laboratory is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-84-K-0099.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Tests for Monsoon Instruction Subset 0

Jonathan Young

April 1990

1 Introduction

This document describes a series of tests which are designed to test increasingly more complicated portions of the Monsoon Macroarchitecture Specification [2]. Our goal is to run these tests on the hardware simulator, on MINT and other software emulators, and eventually on the actual hardware when it arrives.

This is an evolving document. The tests currently described herein test only those instructions in instruction subset 0 (IS0) [1].

2 The tests

All tests begin execution with a single token in the pipeline and terminate with the pipeline idle. Our goal has been to create small tests (no more than 30 instruction and 100 pipeline beats); only the RTS test exceeds this (106 pipeline beats).

To each test corresponds a "base number" n. The code for each test generally starts at `#xn0`, the frame pointer at `#xn0`, and I-structure memory (if used) at `#xn00`, the initial token for the ISTR test should be ⟨`L,#x50,#x50`⟩⟨pointer to `#x500`⟩.

Test	Base	Instructions tested
FORK-JOIN	1	one fork, one join, identity
FLOATS	2	fadd, fsub, fmul, fdiv, fltp, feqp
BOOLS	4	and, or, xor, ash and .nl1
ISTR	5	faif, istr, %%istr
SEND	6	ct, aoct
	7	(reserved for SEND)
RTS	8	stake, id.st1, id.rnl1, ap
MISC	#xA	remov, swt

Note that the graphs presented at the end of this document depend crucially on the ability to predict which of two tokens will be executed first – in particular, we assume that the second token (dest2) will be the next token executed after a fanout (`id.m2`) instruction.

The graphs also assume that the token queues are configured as a *stack*. On the gate-level simulator this assumption was not the case.

3 FORK-JOIN

This tests the basic execution of the machine: monadic and dyadic matching, the identity alu operation, and the generation of 0, 1, or 2 output tokens. The first instruction split the input token into two, which are joined by the second instruction and terminated by `stop`, the third insruction. Note that since `stop` is the last instruction in the current microcode, this also enforces the declaration of unused opcodes and second-level-decode entries.

4 FLOATS

This test pushes a known value (5.5) through 4 floating-point ALU operations (`fadd`, `fsub`, `fmul`, and `fdiv`) in parallel. We also exercise different matching and output modes: first, we match with an absolutely-addressed constant and issue one token (`.n1`); second, we issue 2 tokens (`.n2`); third, we do a frame-relative dyadic match which issues 2 tokens (`.n2`). Finally, we do a frame-relative dyadic match which issues only 1 token; these 4 tokens are fed into two floating compares which in turn feed a logical **and**. The result should be boolean false.

5 BOOLS

This test exercises the floating-point comparison and (integer) boolean operators.

6 ISTR

This test executes several two-phase memory transations - three I-fetches and two I-stores, to be precise. We test I-fetch from both present and empty locations, as well as I-store to both empty and deferred. (We do not test the full functionality of the `%%istr` instruction, only that need for subset 0.)

7 SEND

This test uses the `ct` and `aoct` instructions to test a rudimentary function-call sequence.

8 RTS

This tests the instructions needed to implement a protypical run-time system on the Monsoon machine which allocates storage (i.e. get-context and get-aggregate) but cannot deallocate it. The instructions particularly needed are `stake` (take on read-only, spin on empty) and `id.stic` ("put").

This test executes two `get-context` and two `get-aggregate` operations in parallel; one of each spins for several cycles while the critical section of the other one is executed.

Note that because of the limitations of IS0, we are still taking advantage of the pointer/continuation duality.

9 MISC

This final test exercises "everything else". We test dyadic matching when the left token arrives first, and we test the switch (`swt`), fix (`fix`), and remove (`remov`) opcodes. We also test loading a global constant.

10 Testing Limitations

The simulation of only 600 instructions on our machine must inherently leave something out. In particular, we did not test the FALU operations in any comprehensive manner, but only enough to establish that the correct operation was being performed on the correct (e.g. not flipped) arguments. We do not test every second-level-decode of every macroinstruction; we intentionally take advantage of the microcode structure to test matching modes and ALU operations separately.

11 Appendix: Test Graphs

The dataflow graphs for the tests, annotated to include each token expected during execution, appear on the following pages.

12 Acknowledgements

This document is the combined efford of several people. I am particularly indebted to Madhu Sharma, who drew the pictures and debugged my code, and to Ralph Tiberio, who first executed these graphs on the hardware simulator.

References

[1] M. J. Beckerle and J. H. Young. Monsoon instruction subsets. Computation Structures Group Memo 306a, Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge MA, 1988.

[2] G. M. Papadopoulos and K. R. Traub. Monsoon macroarchitecture reference manual. Computation Structures Group Memo 306, Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge MA, 1990.

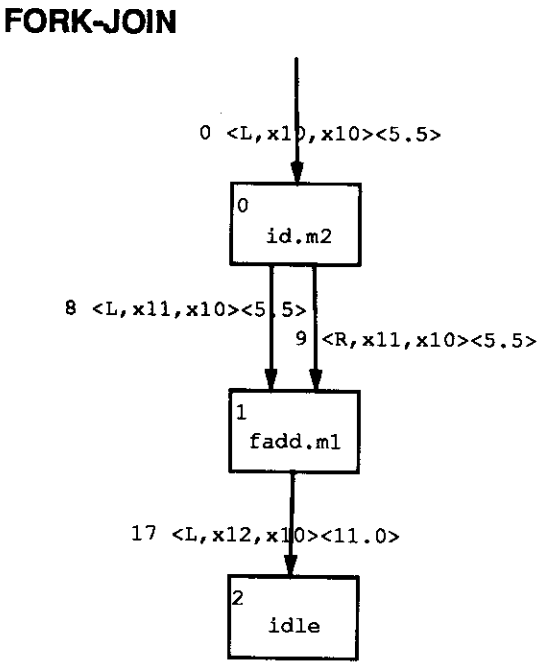

Figure 1: Fork-Join

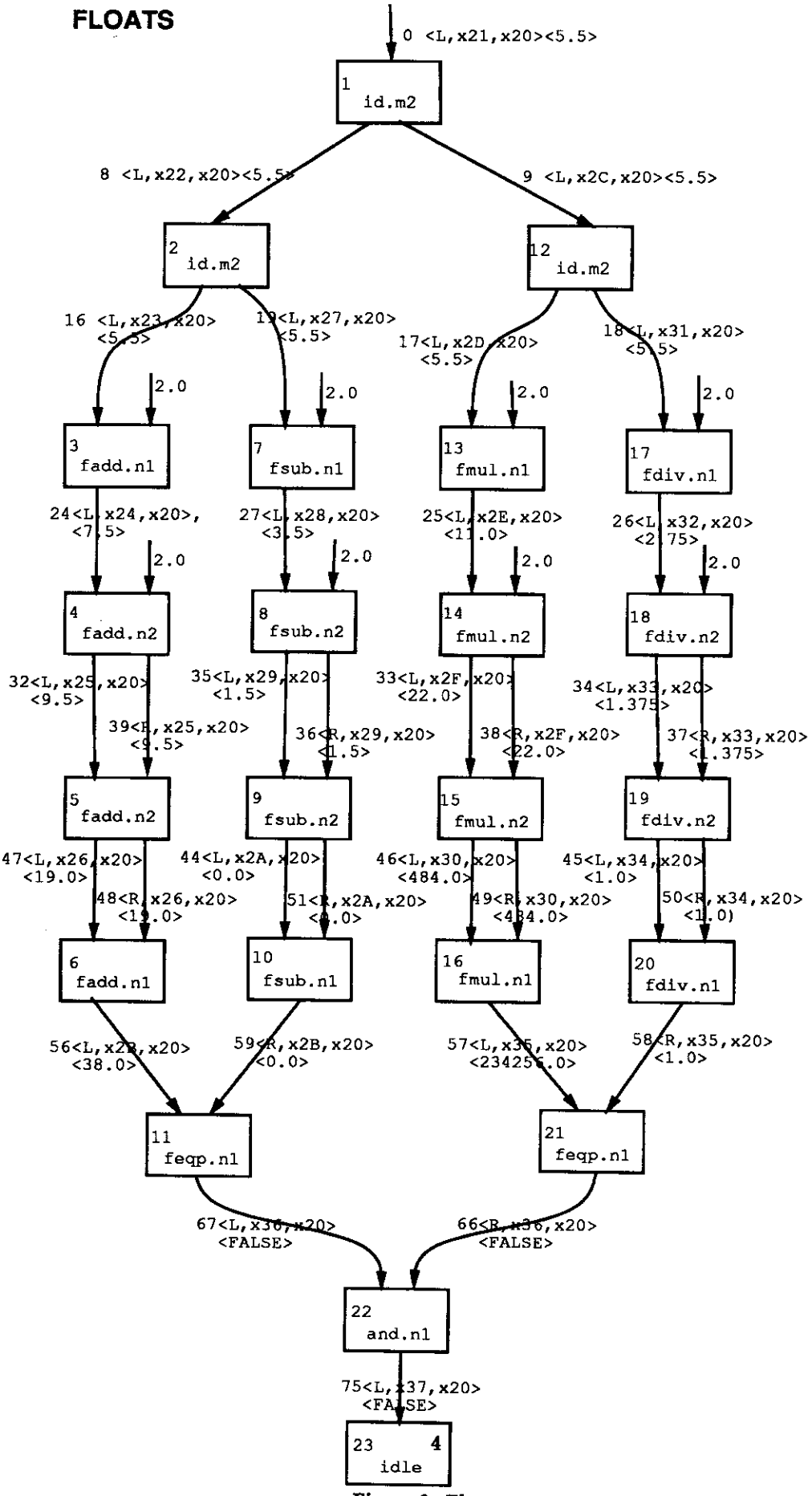

Figure 2: Floats

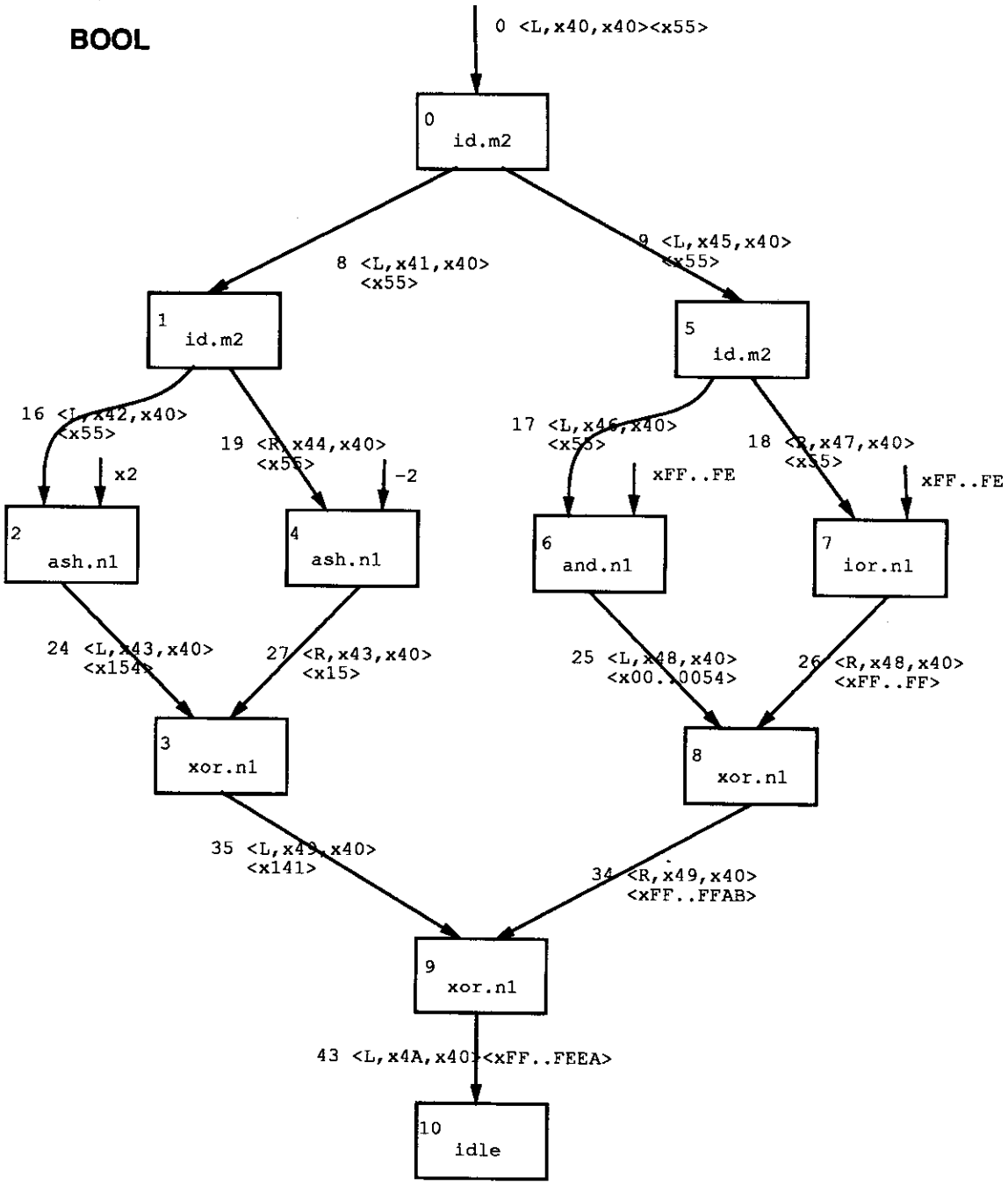

Figure 3: Bool

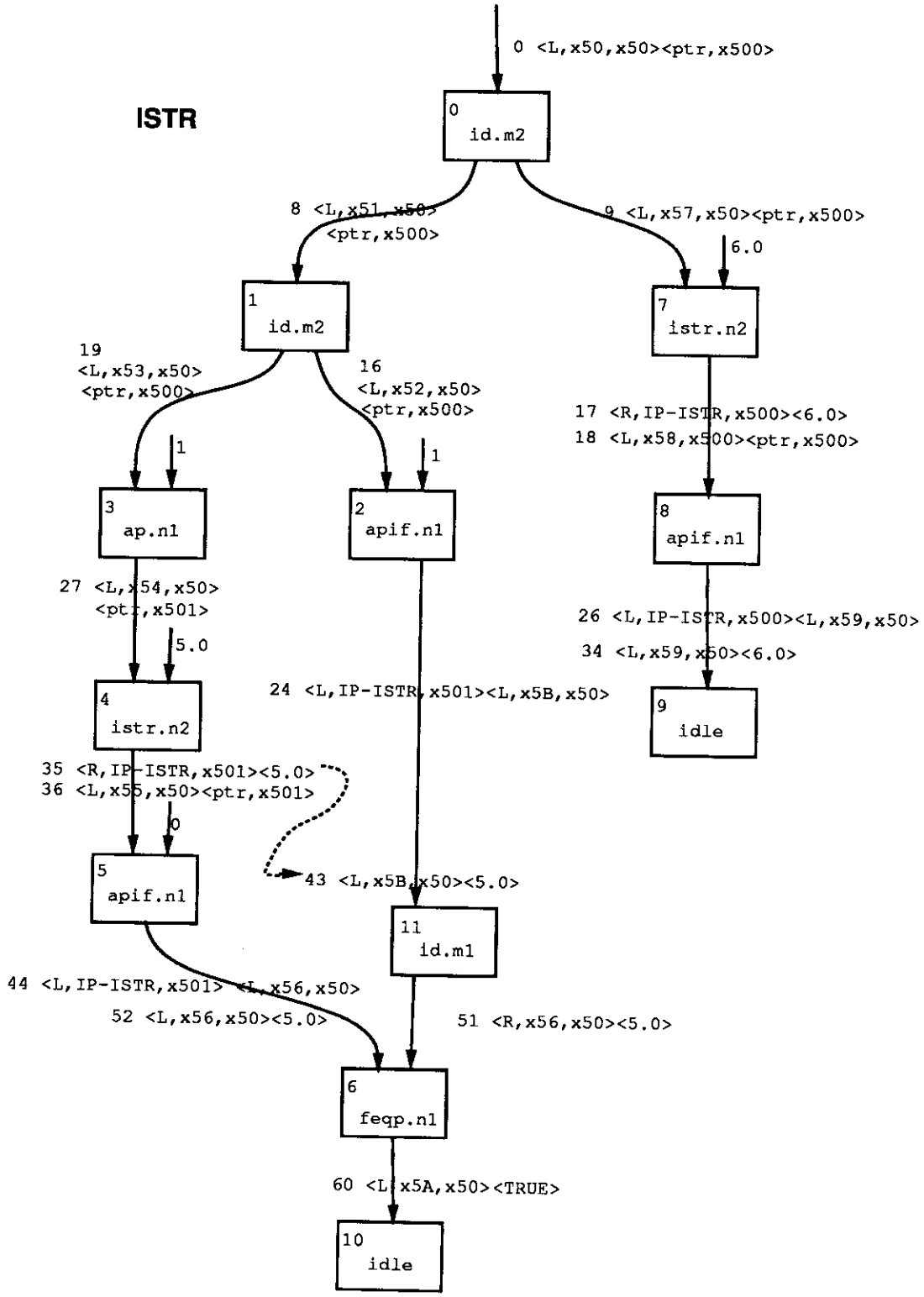

Figure 4: I-structures

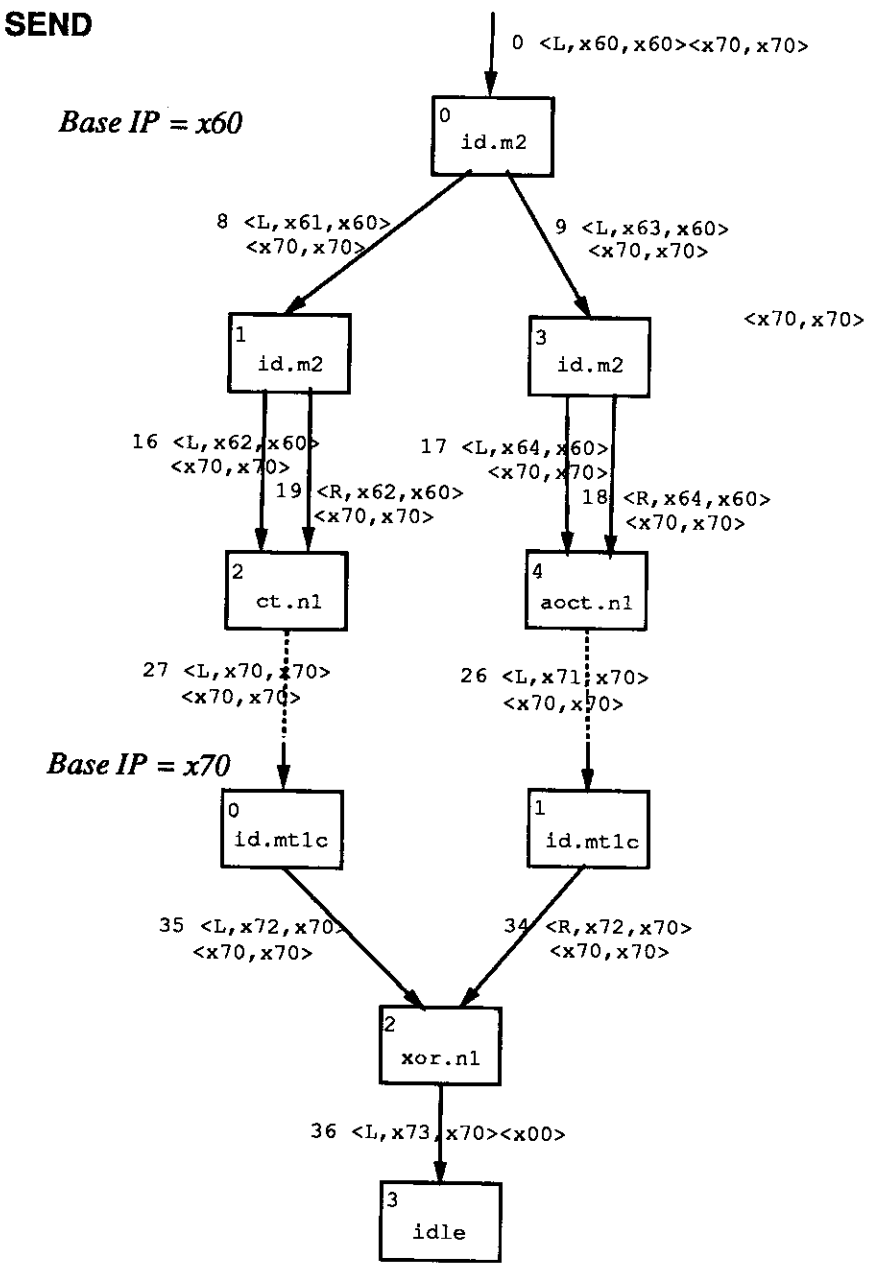

Figure 5: Send

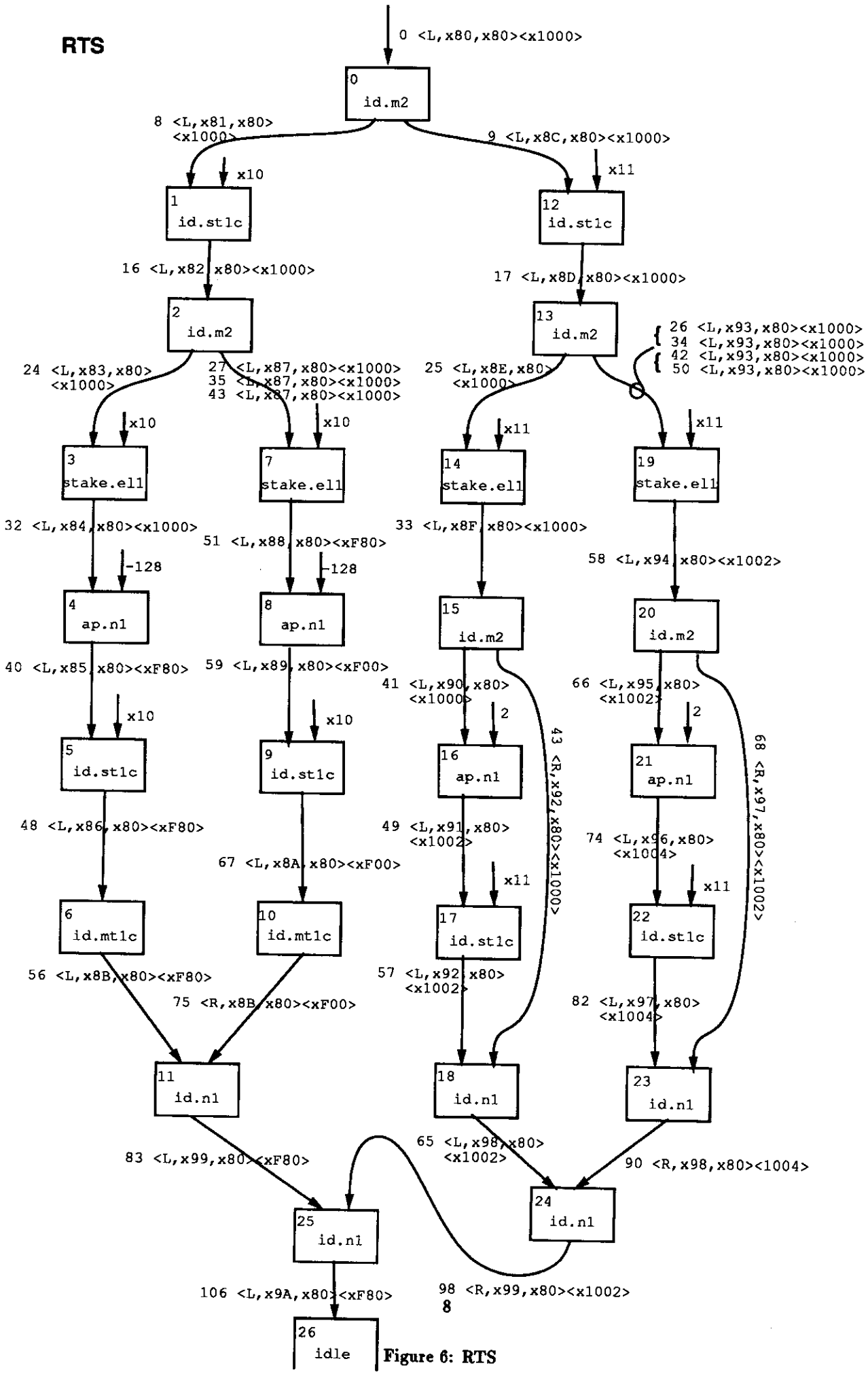
0 <L,x80,x80><x1000>
0 id.m2
8 <L,x81,x80> <x1000>
9 <L,x8C,x80><x1000>
x10
x11
1 id.st1c
12 id.st1c
16 <L,x82,x80><x1000>
17 <L,x8D,x80><x1000>
2 id.m2
13 id.m2
26 <L,x93,x80><x1000>
34 <L,x93,x80><x1000>
42 <L,x93,x80><x1000>
50 <L,x93,x80><x1000>
24 <L,x83,x80> <x1000>
27 <L,x87,x80><x1000>
35 <L,x87,x80><x1000>
43 <L,x87,x80><x1000>
25 <L,x8E,x80> <x1000>
3 stake.el1
7 stake.el1
14 stake.el1
19 stake.el1
32 <L,x84,x80><x1000>
51 <L,x88,x80><xF80>
33 <L,x8F,x80><x1000>
58 <L,x94,x80><x1002>
-128
4 ap.n1
8 ap.n1
15 id.m2
20 id.m2
40 <L,x85,x80><xF80>
59 <L,x89,x80><xF00>
41 <L,x90,x80> <x1000>
66 <L,x95,x80> <x1002>
2
5 id.st1c
9 id.st1c
16 ap.n1
21 ap.n1
43 <R,x92,x80><x1000>
68 <R,x97,x80><x1002>
48 <L,x86,x80><xF80>
49 <L,x91,x80> <x1002>
74 <L,x96,x80> <x1004>
67 <L,x8A,x80><xF00>
17 id.st1c
22 id.st1c
6 id.mt1c
10 id.mt1c
56 <L,x8B,x80><xF80>
57 <L,x92,x80> <x1002>
75 <R,x8B,x80><xF00>
82 <L,x97,x80> <x1004>
11 id.n1
18 id.n1
23 id.n1
65 <L,x98,x80> <x1002>
83 <L,x99,x80><xF80>
90 <R,x98,x80><1004>
24 id.n1
25 id.n1
106 <L,x9A,x80><xF80>
98 <R,x99,x80><x1002>
26 idle

Figure 6: RTS

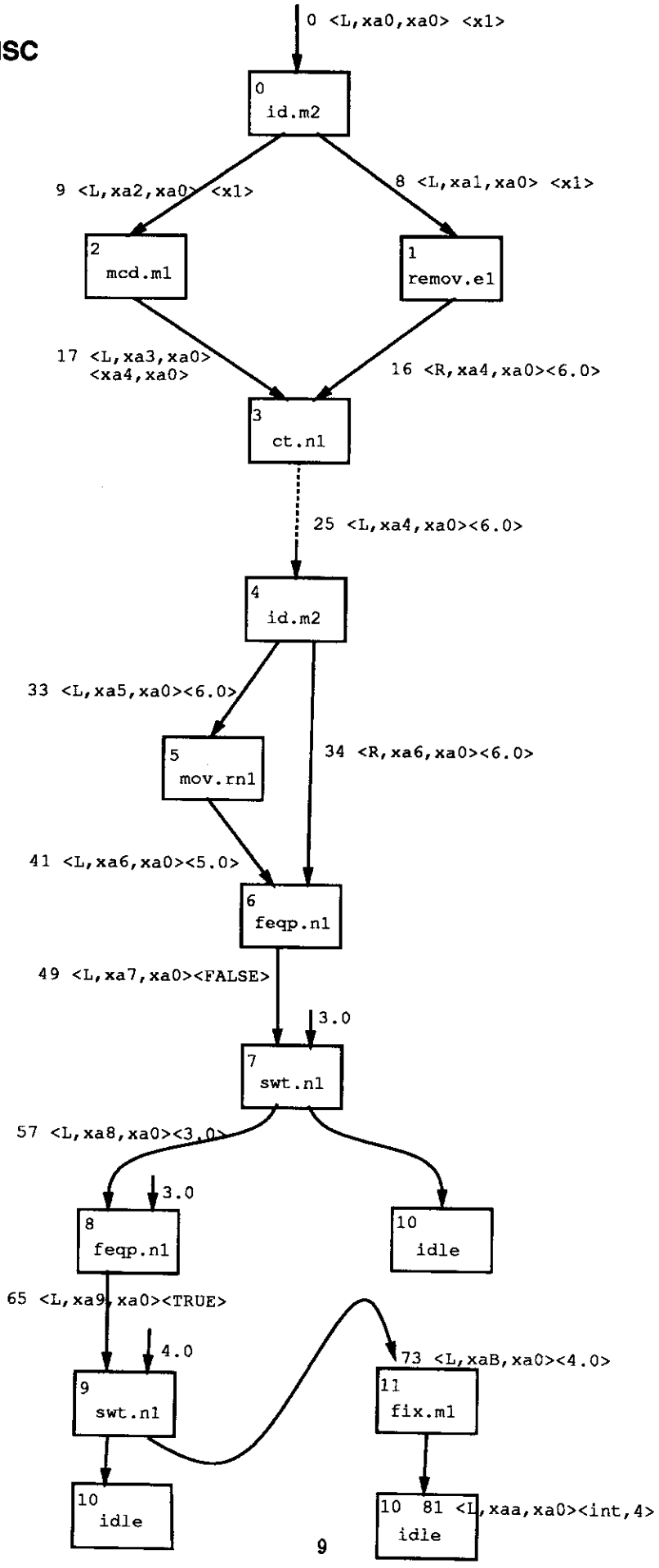

Figure 7: Misc.