DTM*m*

Dynamic Trace Memoization with Reuse of Memory Access Instruction’s Values

Luiz Marcio Faria de Aquino Viana, M.Sc.

E-mail: lmarcio@cos.ufrj.br

Phone: +55-21-99983-7207

Content

Main Objective

Extend the functionalities of the instruction trace reuse technique denomitated *Dynamic Trace Memoization* - DTM, adding the capability to reuse of memory access instruction’s values

Step #1	Step #2	Step #3	Step #4
SuperSIM (Development of Sparc V7 - Simulator)	DTM Implementation	Adding the reuse of memory access instruction’s values to DTM	DTM*m*

Content

Dynamic Traces Memoization - DTM

Technique to explore redundant computation with the reuse of instructions' traces.

Dynamic Traces Memoization - DTM

Initial Definitions

❑ **Instructions' Traces** – sequence of dynamic instructions of a program;

❑ **Redundant Trace –** trace formed by redundant instructions, instructions which are being executed with the same input values;

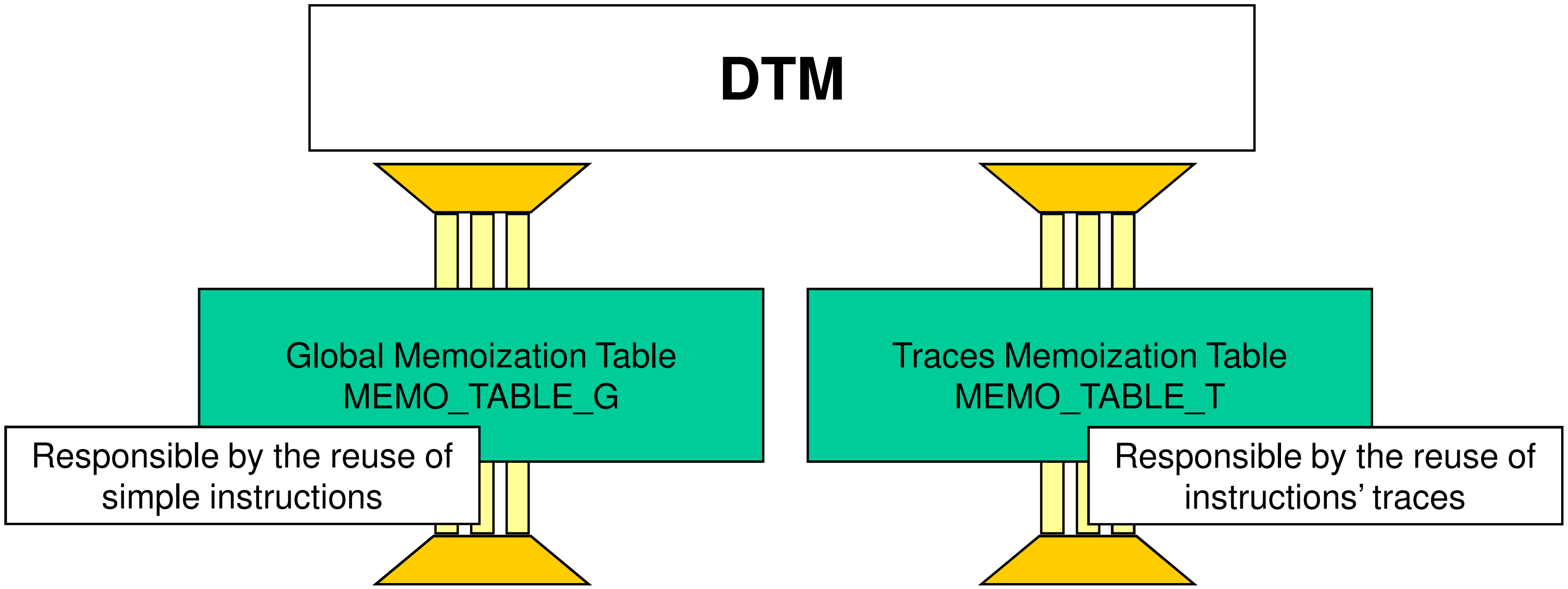
❑ **Valid Instructions –** instructions pertencent to a subset of processor's instructions which are accepted by DTM;

- In this subset not are included instructions of memory access, system calls and float point instructions;

❑ **Input Context** – set formed by input registers, which the values are provided by instructions extern the trace, and by their respective values;

❑ **Output Context –** set formed by output registers modifyed by the trace and their respective values;

Dynamic Traces Memoization - DTM

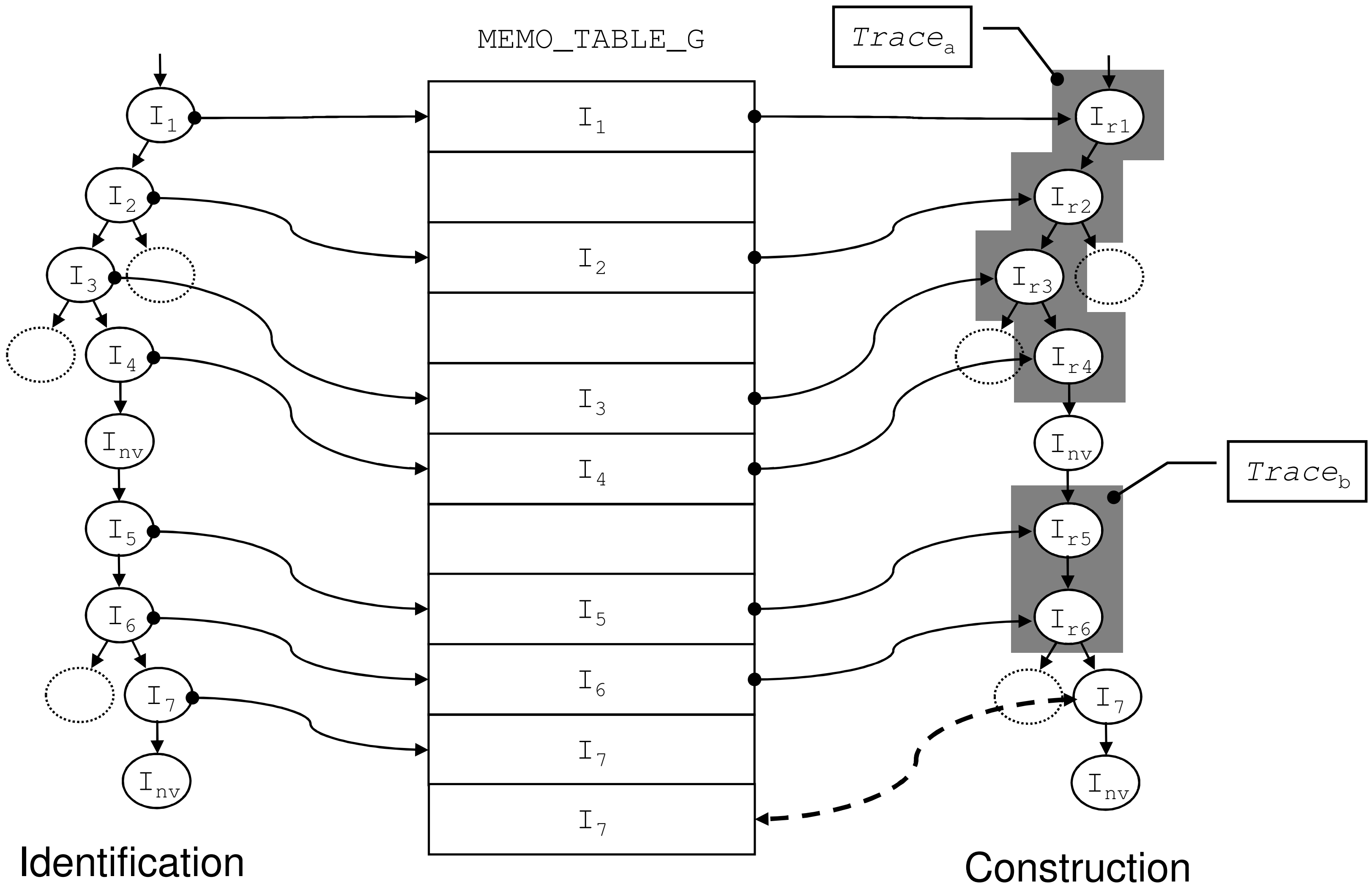

DTM
Global Memoization Table
MEMO_TABLE_G
Traces Memoization Table
MEMO_TABLE_T
Responsible by the reuse of simple instructions
Responsible by the reuse of instructions' traces

❑ There are two levels of reuse – reuse of simple instructions and traces reuse

- Less volatility of traces reuse table
- Pré- qualification of instructions which will form the traces

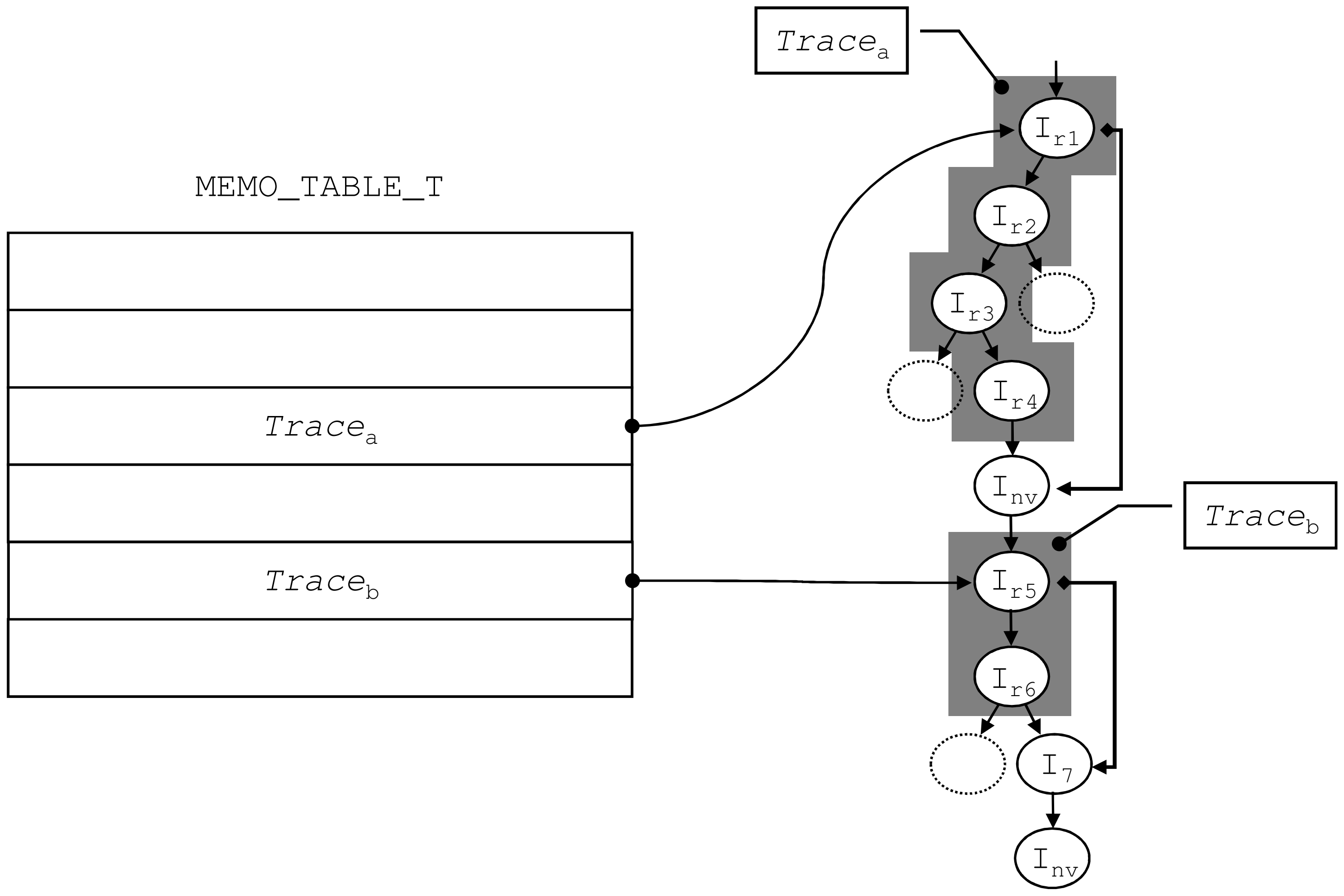
Dynamic Traces Memoization - DTM

Process of Traces Reuse (Identification, Construction and Reuse)



Dynamic Traces Memoization - DTM

Process of Traces Reuse



Reuse

Dynamic Traces Memoization - DTM

Format of Global Memoization Table's Entry – MEMO_TABLE_G

pc	jmp	brc	btaken	sv1	sv2	res/npc
30b	1b	1b	1b	32b	32b	32b

pc – instruction address

jmp – flag of inconditional branch

brc – flag of conditional branch

btaken – flag of conditional brach **taken** or **not_taken**

sv1 – value of source operand #1

sv2 – value of source operand #2

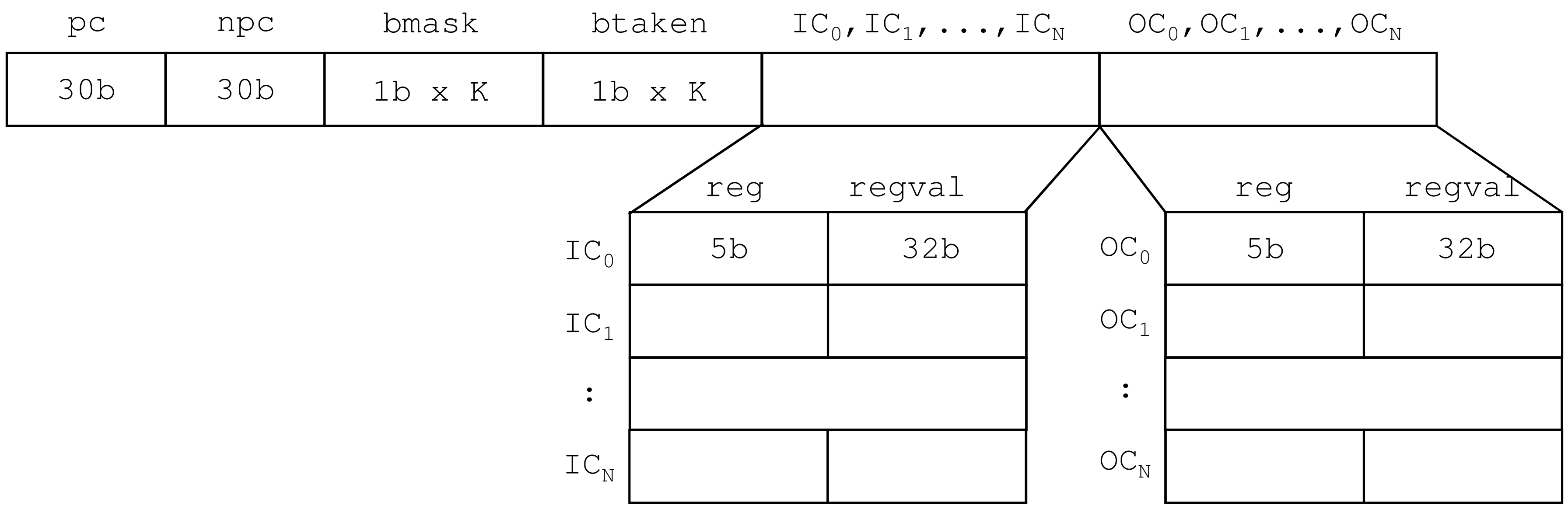
res/npc – operation result / destination address

```
:
0x755c      sethi %hi(0xfffac00), %o2
0x7560      or %o2, 0x3cf, %o1
0x7564      add %o0, %o1, %o0
:
```

0x1d57	0	0	0	–	–	0xfffac00
0x1d58	0	0	0	0xfffac00	–	0xfffafcf
0x1d59	0	0	0	0xffffff0	0xfffafcf	0xfffafc0

Dynamic Traces Memoization - DTM

Format of Traces Memoization Table’s Entry – MEMO_TABLE_T



pc – instruction adress

npc – instruction adress of trace’s following instruction

bmask – flags of branch instructions

btaken – flags of branch instructions **taken** / **not_taken**

$IC_0, IC_1, ..., IC_N$ – trece’s input context

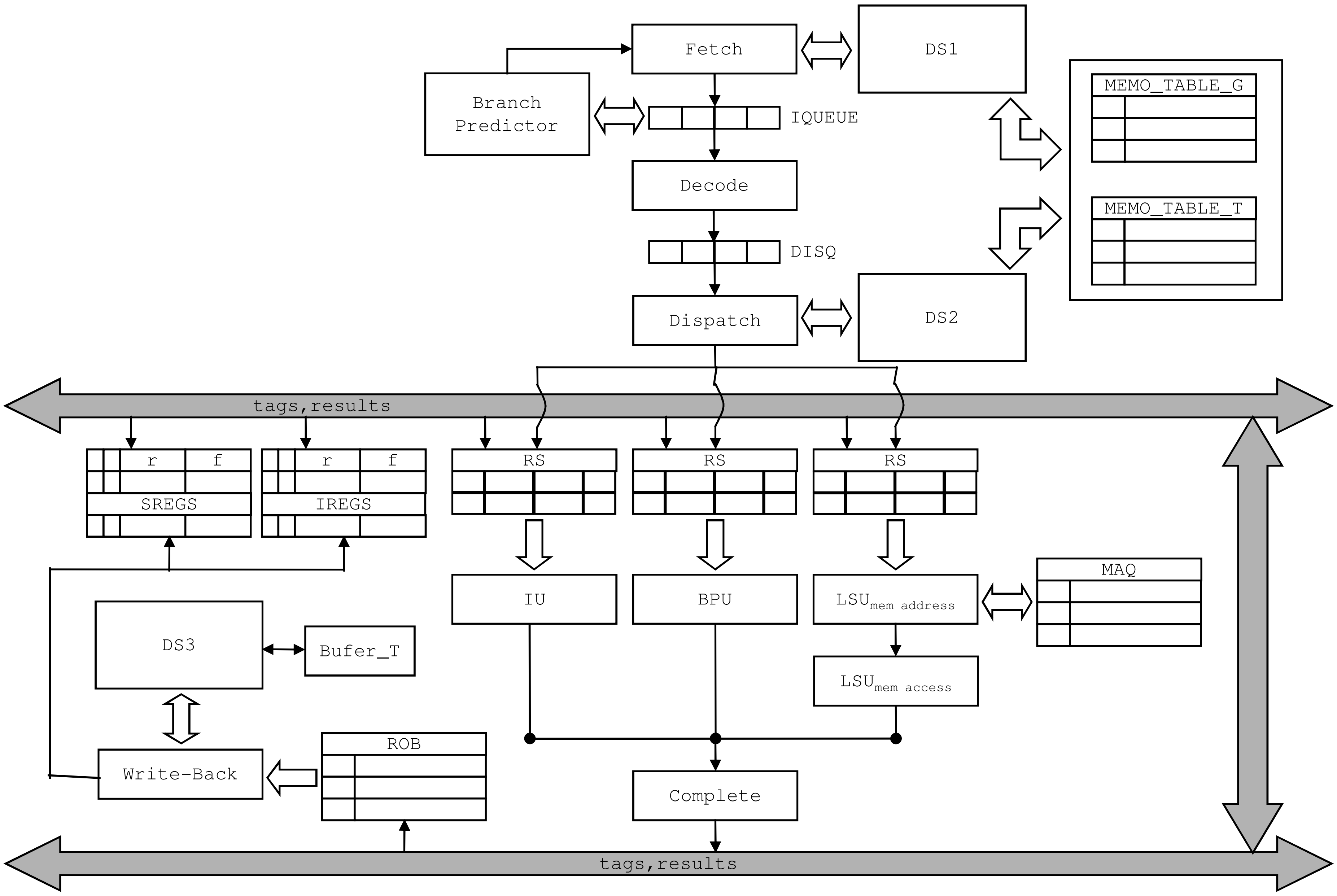
reg – address of source register

regval – value of source register

$OC_0, OC_1, ..., OC_N$ – trace’s output context

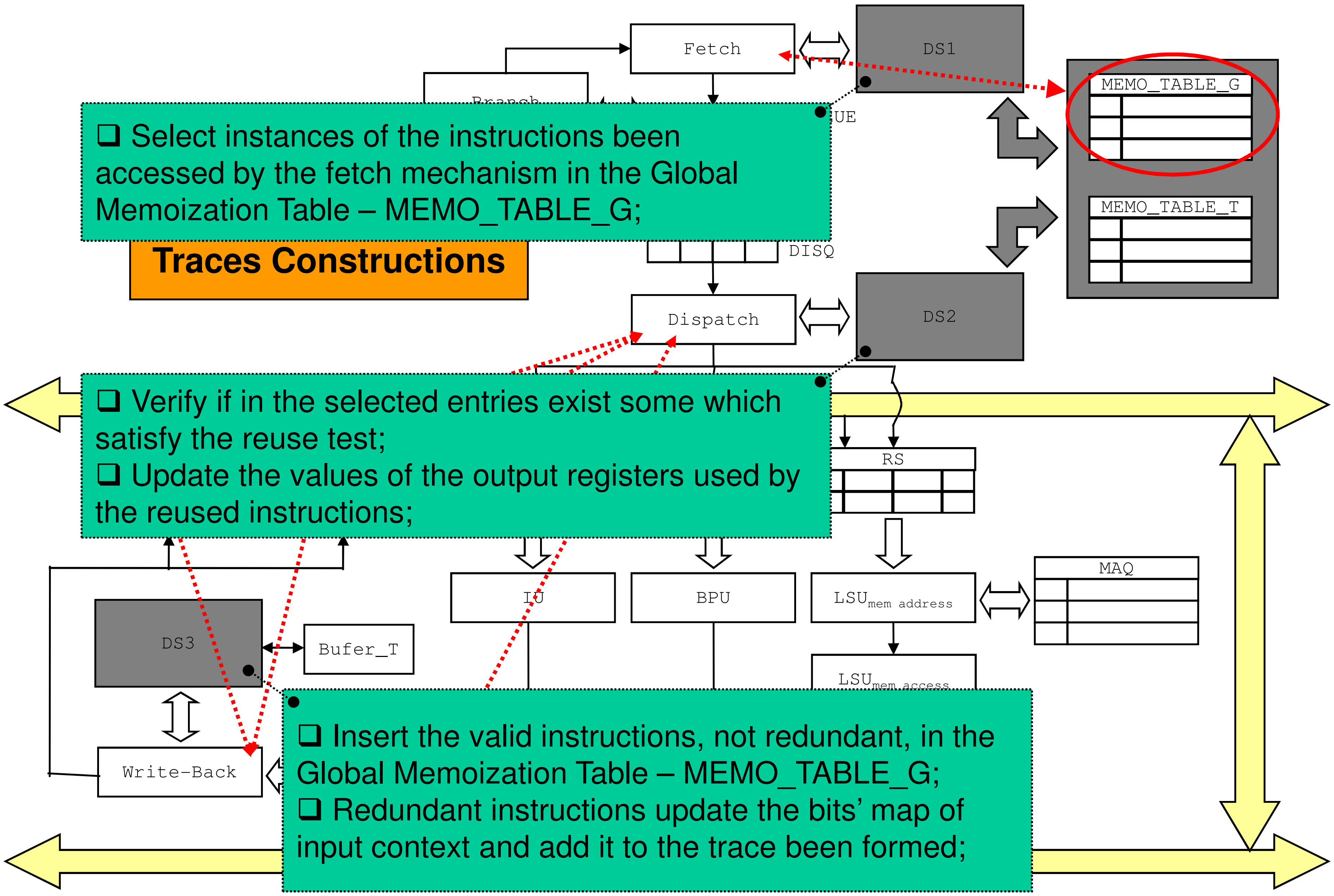
reg – address of destination register

regval – value of destination register



Fetch
DS1
Branch Predictor
IQUEUE
MEMO_TABLE_G
Decode
MEMO_TABLE_T
DISQ
Dispatch
DS2
tags,results
r
f
SREGS
r
f
IREGS
RS
RS
RS
IU
BPU
LSU$_{mem\ address}$
MAQ
DS3
Bufer_T
LSU$_{mem\ access}$
ROB
Write-Back
Complete
tags,results

Dynamic Traces Memoization - DTM



❑ Select instances of the instructions been accessed by the fetch mechanism in the Global Memoization Table – MEMO_TABLE_G;

❑ Verify if in the selected entries exist some which satisfy the reuse test;

❑ Update the values of the output registers used by the reused instructions;

❑ Insert the valid instructions, not redundant, in the Global Memoization Table – MEMO_TABLE_G;

❑ Redundant instructions update the bits' map of input context and add it to the trace been formed;

Dynamic Traces Memoization - DTM

	MEMO_TABLE_G			
	pc	sv1	sv2	res
➡ 100. cmp r2,r1,0x05	0x100	0x0006	–	0x0001
➡ 104. bne r2,0x124	0x104	0x0001	–	0x124
➡ 124. add r4,r3,0x02	0x124	0x0003	–	0x0005
➡ 128. sll r2,r4,0x04	0x128	0x0005	–	0x0050
➡ 12c. or r4,r4,r2	0x12c	0x0005	0x0050	0x0055
➡ 130. sll r2,r4,0x08	0x130	0x0055	–	0x5500
➡ 134. or r4,r2,r4	0x134	0x5500	0x0055	0x5555

contexto de entrada							contexto de saída						
r0	r1	r2	r3	r4	..	r31	r0	r1	r2	r3	r4	..	r31
	1								1				
	1								1				
	1		1						1		1		
	1		1						1		1		
	1		1						1		1		
	1		1						1		1		
	1		1						1		1		

Buffer_T

pc	npc	bmask				btaken				IC_{r0}	IC_{v0}	IC_{r1}	IC_{v1}	IC_{r2}	IC_{v2}	OC_{r0}	OC_{v0}	OC_{r1}	OC_{v1}	OC_{r2}	OC_{v2}
0x100	0x104									r1	0x0006					r2	0x0001				
0x100	0x124	1				1				r1	0x0006					r2	0x0001				
0x100	0x128	1				1				r1	0x0006	r3	0x0003			r2	0x0001	r4	0x0005		
0x100	0x12c	1				1				r1	0x0006	r3	0x0003			r2	0x0050	r4	0x0005		
0x100	0x130	1				1				r1	0x0006	r3	0x0003			r2	0x0050	r4	0x0055		
0x100	0x134	1				1				r1	0x0006	r3	0x0003			r2	0x5500	r4	0x0055		
0x100	0x138	1				1				r1	0x0006	r3	0x0003			r2	0x5500	r4	0x5555		

Dynamic Traces Memoization - DTM

	MEMO_TABLE_G			
	pc	sv1	sv2	res
➡ 100. cmp r2,r1,0x05	0x100	0x0006	–	0x0001
➡ 104. bne r2,0x124	0x104	0x0001	–	0x124
➡ 124. add r4,r3,0x02	0x124	0x0003	–	0x0005
➡ 128. sll r2,r4,0x04	0x128	0x0005	–	0x0050
➡ 12c. or r4,r4,r2	0x12c	0x0005	0x0050	0x0055
➡ 130. sll r2,r4,0x08	0x130	0x0055	–	0x5500
➡ 134. or r4,r2,r4	0x134	0x5500	0x0055	0x5555

contexto de entrada

r0	r1	r2	r3	r4	..	r31
	1					
	1					
	1		1			
	1		1			
	1		1			
	1		1			
	1		1			

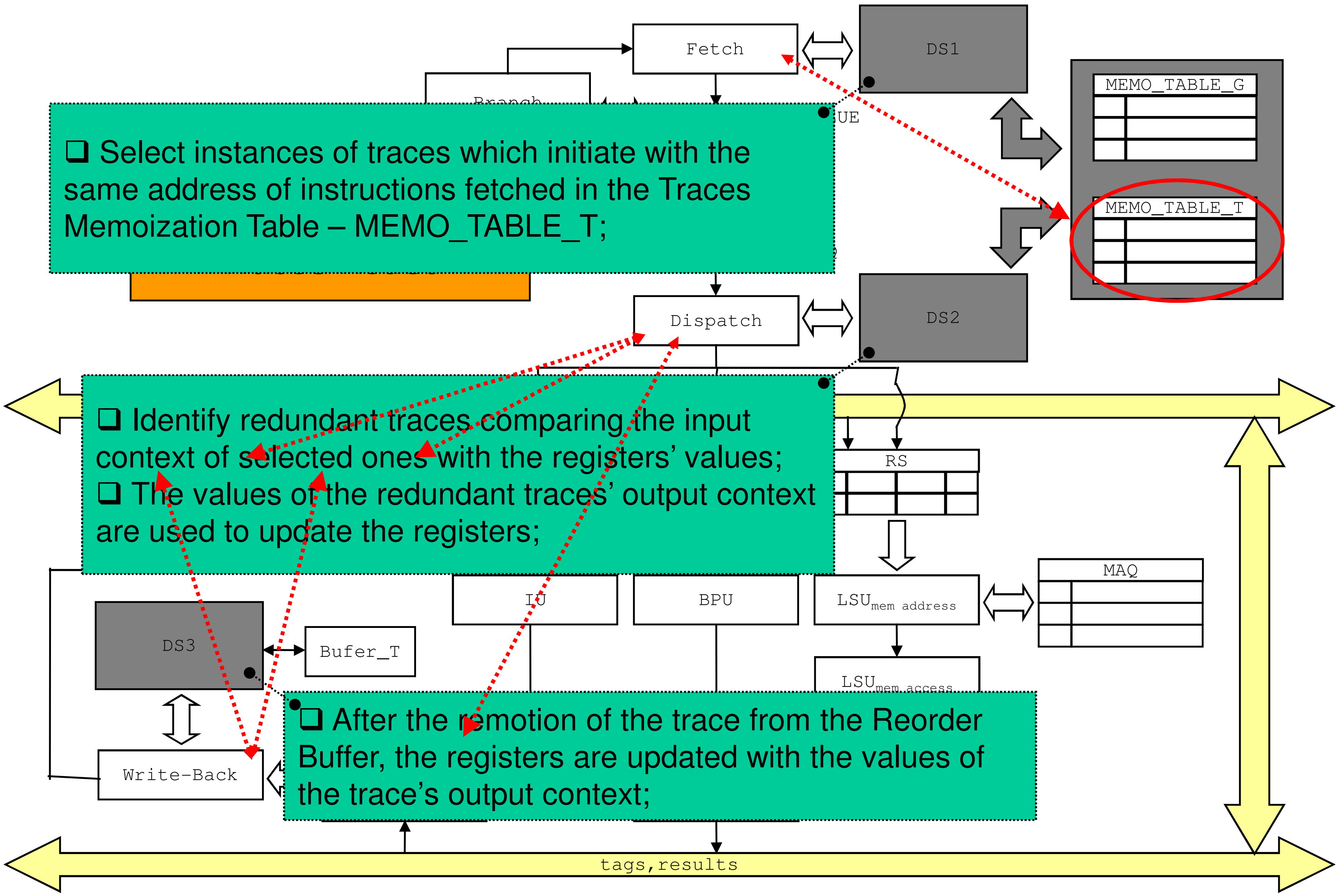
contexto de saída

r0	r1	r2	r3	r4	..	r31
		1				
		1				
		1		1		
		1		1		
		1		1		
		1		1		
		1		1		

Buffer_T

pc	npc	bmask				btaken				IC_{r0}	IC_{v0}	IC_{r1}	IC_{v1}	IC_{r2}	IC_{v2}	OC_{r0}	OC_{v0}	OC_{r1}	OC_{v1}	OC_{r2}	OC_{v2}
0x100	0x104									r1	0x0006					r2	0x0001				
0x100	0x124	1				1				r1	0x0006					r2	0x0001				
0x100	0x128	1				1				r1	0x0006	r3	0x0003			r2	0x0001	r4	0x0005		
0x100	0x12c	1				1				r1	0x0006	r3	0x0003			r2	0x0050	r4	0x0005		
0x100	0x130	1				1				r1	0x0006	r3	0x0003			r2	0x0050	r4	0x0055		
0x100	0x134	1				1				r1	0x0006	r3	0x0003			r2	0x5500	r4	0x0055		
0x100	0x138	1				1				r1	0x0006	r3	0x0003			r2	0x5500	r4	0x5555		

Dynamic Traces Memoization - DTM



❑ Select instances of traces which initiate with the same address of instructions fetched in the Traces Memoization Table – MEMO_TABLE_T;

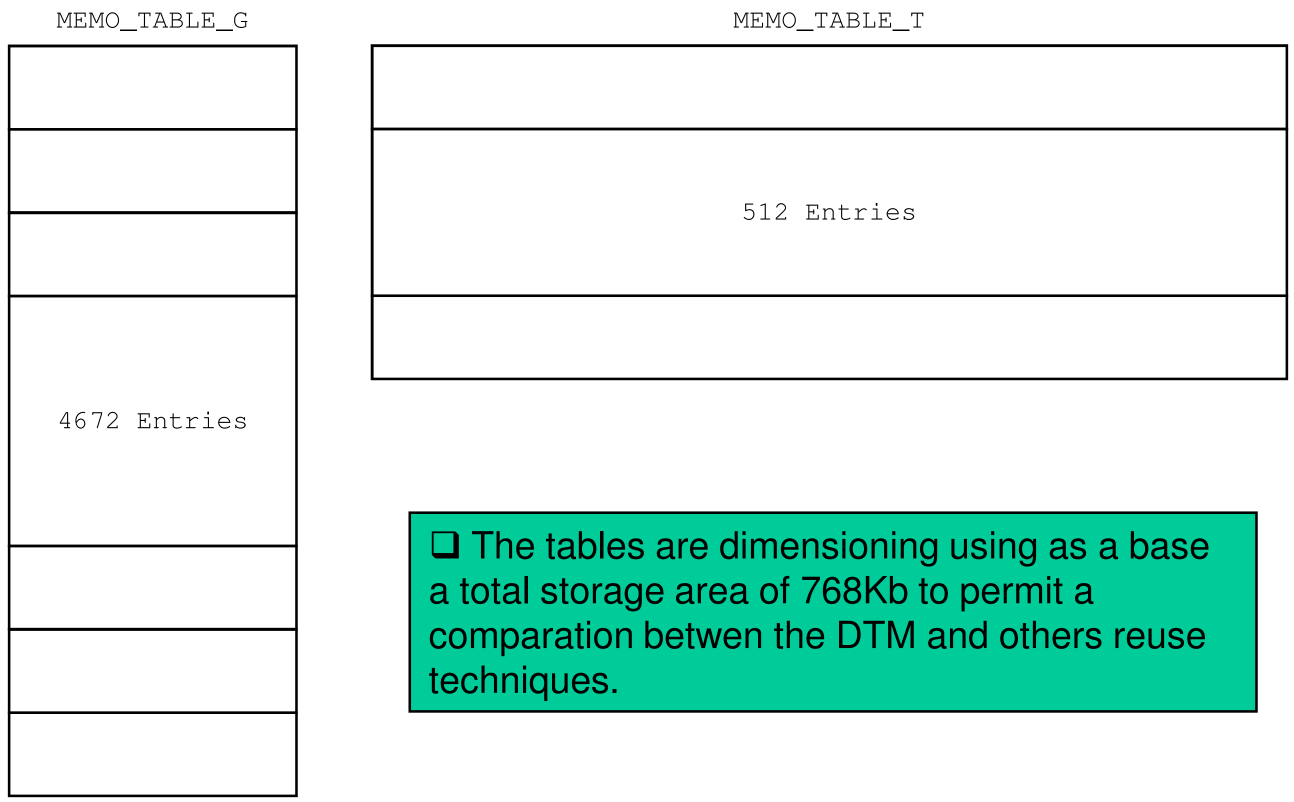
❑ Identify redundant traces comparing the input context of selected ones with the registers' values;

❑ The values of the redundant traces' output context are used to update the registers;

❑ After the remotion of the trace from the Reorder Buffer, the registers are updated with the values of the trace's output context;

Dynamic Traces Memoization - DTM

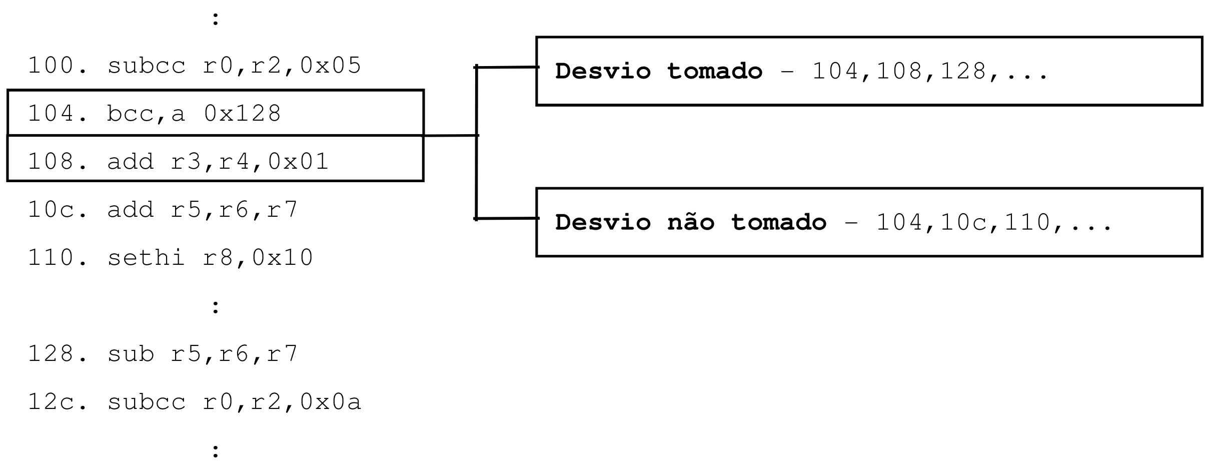
Size of Reuse Tables



❑ The tables are dimensioning using as a base a total storage area of 768Kb to permit a comparation betwen the DTM and others reuse techniques.

Dynamic Traces Memoization - DTM

Implementation Details - Delay Slot (Sparc v7)



```
              :
100. subcc r0,r2,0x05
104. bcc,a 0x128
108. add r3,r4,0x01
10c. add r5,r6,r7
110. sethi r8,0x10
              :
128. sub r5,r6,r7
12c. subcc r0,r2,0x0a
              :
```

Desvio tomado – 104,108,128,...
Desvio não tomado – 104,10c,110,...

❑ Instructions in the delay slot are not reused because they introduce ambiguities in the program’s execution flow;

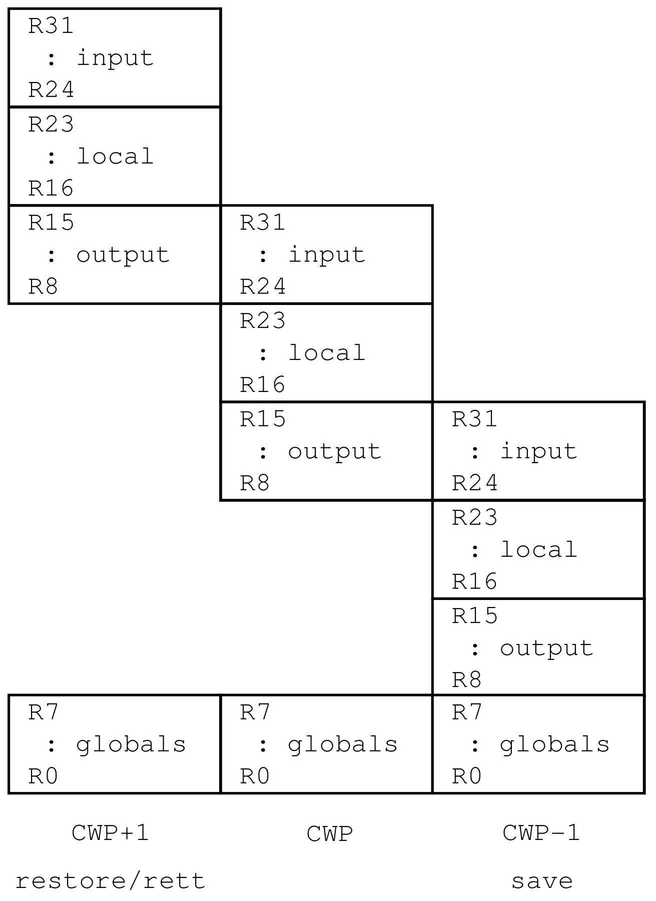
❑ Traces do not contain instructions in the delay slot because they could produce ambiguity in the program’s execution flow if the correspondent branch instruction is not present in the trace;

❑ Instructions in the delay slot are necessarially included in the trace been formed and finalize them, but they are not reused;

- Branches with annul bit enabled introduce problems with the reuse of traces which include branch instructions been executed speculativemently;

Dynamic Traces Memoization - DTM

Implementation Details – Registers' Window (Sparc v7)



```
          :
0x728. save %r30,-64,%r30
0x72c. orcc %r24,%r25,%r0
0x730. bge 0x754
0x734. xor %r24,%r25,%r18
          :
0x768. retl
0x76c. restore
          :
```

Dynamic Traces Memoization - DTM

Implementation Details – Registers' Window (Sparc v7)

include save and store instructions
x
do not include save and store instructions

Include save and store	**Do not include save and store**
This abordage add to each instruction or trace an unique identifyer to each register used for the input and output context.	This abordage store the relative address of the registers into the registers' window in the trace's input and output context.
❑ Project simplicity; ❑ Permit the inclusion of save and store instructions in the traces; ❑ Could grow significantly the size of trace's input and output context; ❑ Do not explore the redundance existent in recursive procedures;	❑ Add capability to reuse present inside recursive f ❑ Do not permit the inclusion and store instructions into the tr ❑ Grow the project complexity;

Content

Adding Memory Access Instructions to DTM

Motivations

❑ Grow the median size of the traces;

- Experiments show which 13% of the traces are ended by access memory instructions, and 36% of them by not redundant instructions;
- Access memory insttructions has great value lacality;

❑ Grow the avaiability of sources operands;

- The frequent use of load instructions at the begin of a computation;

❑ Memory access instructions has significant latency, principaly when they don't be catch by the cache;

DTM*inv*

DTM*upd*

Adding Memory Access Instructions to DTM

Initial Conditions

I/O ADDRESS

❑ Machines that use instructions dedicated to input and output operations;

- In these architectures, instructions dedicated to input and output operations are excluded from the valid DTM instruction set;

❑ Machines that use memory input and output mapping;

- The mechanisms presented use a pair of registers that delimit the memory areas dedicated to these operations;

I/O BUFFERS

❑ Reading of input and output buffer areas introduces inconsistency between the values contained in the reuse tables and the values present in memory;

- The memory regions used by input and output devices are delimited;

❑ There are additional instructions for enabling and disabling the mechanism, as well as appropriate instructions for flushing the reuse tables;

Adding Memory Access Instructions to DTM

Modifications Applyed at the Global Memoization Table's Entries – MEMO_TABLE_G

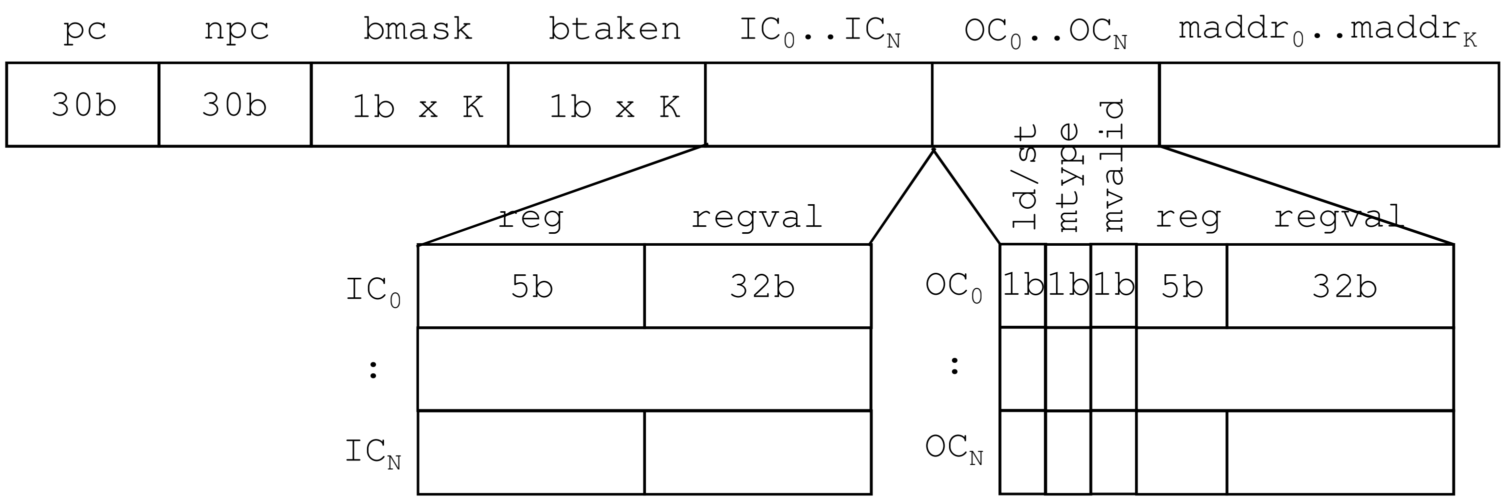
pc	jmp	brc	btaken	sv1	sv2	maddr	mem valid	res/npc
30b	1b	1b	1b	32b	32b	32b	1b	32b

pc – instruction address	jmp – flag of inconditional branch
brc – flag of conditional branch	btaken – flag of branch **taken** or **not_taken**
sv1 – value of source operand #1	sv2 – value of source operand #2
maddr – memory access address	mem valid – flag of valid memory value for load

res/npc – operation result / branch target address / value to be read or write in the memory

Adding Memory Access Instructions to DTM

Modifications Applyed at the Traces Memoization Table's Entries – MEMO_TABLE_T



pc – instruction address

npc – flag of the instruction after the trace

bmask – flags of branches instructions

btaken – flags of branches **taken**/**not_taken**

$IC_0, ..., IC_N$ – trace's input context

reg – source register's address

regval – source register's value

$OC_0, ..., OC_N$ – trace's output context

reg – destination register's address

regval – destination register's value

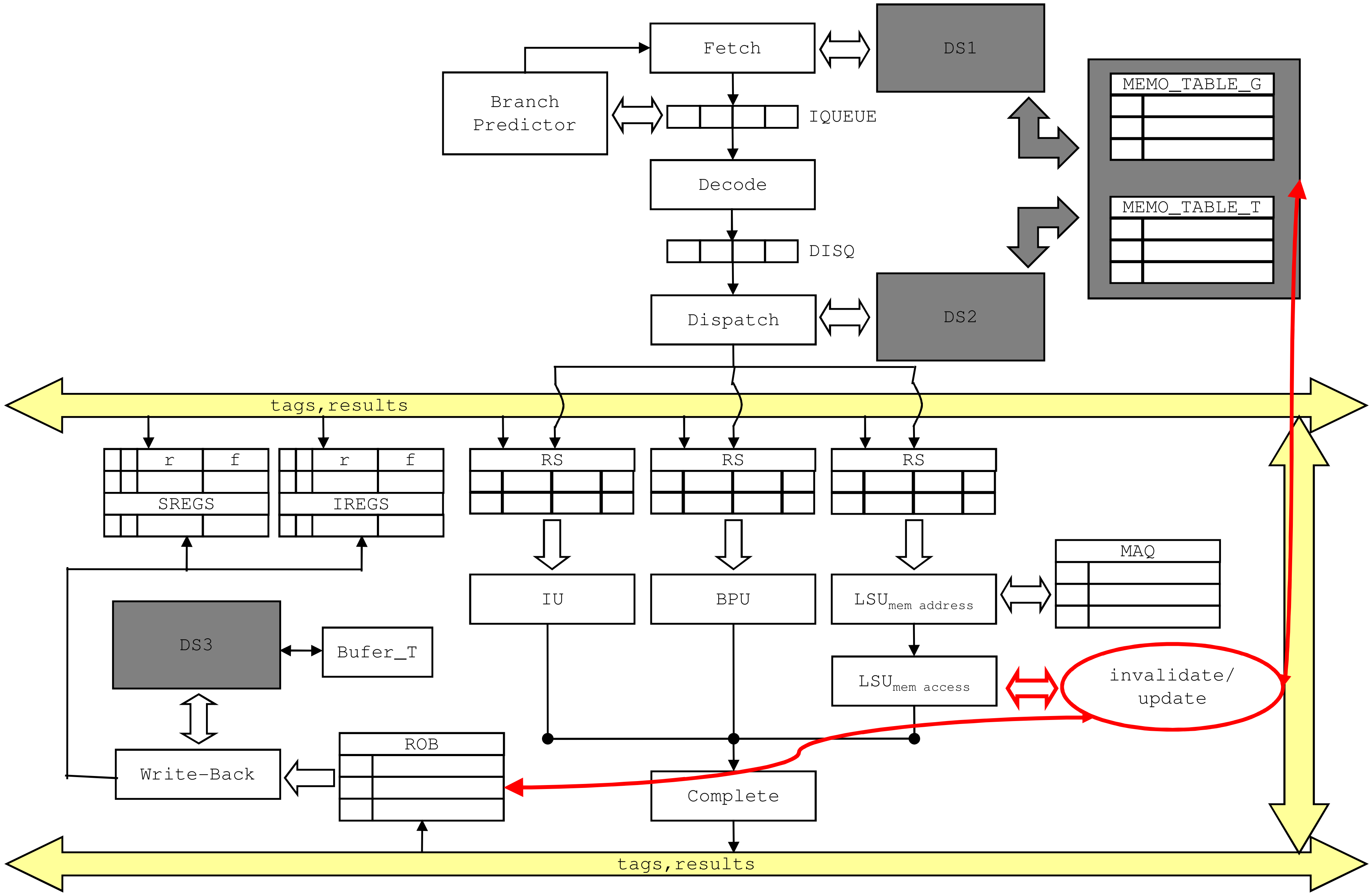
ld/st – flag of memory access instructions

mtype – flag of memory access type **load/store**

mvalid – flag of valid memory value for reuse

$maddr_0, ..., maddr_K$ – address of memory address

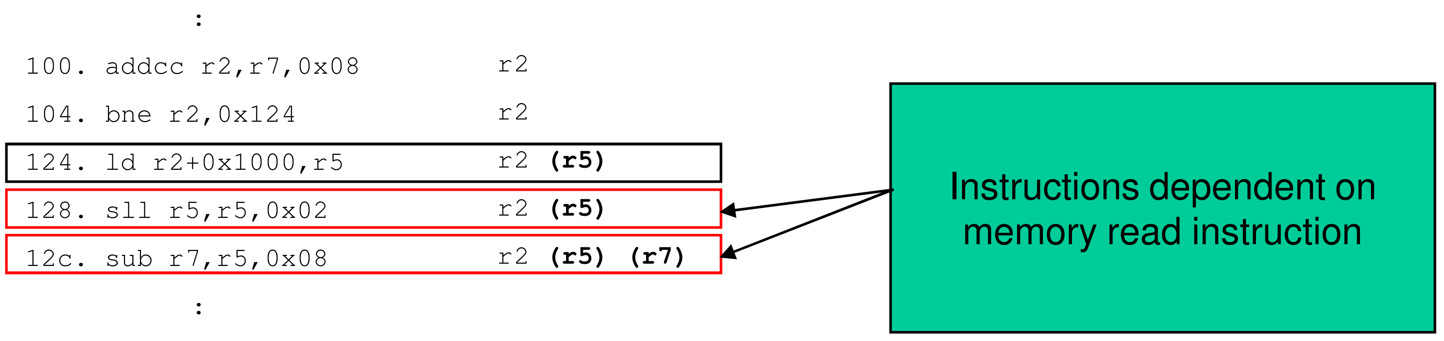
Adding Memory Access Instructions to DTM



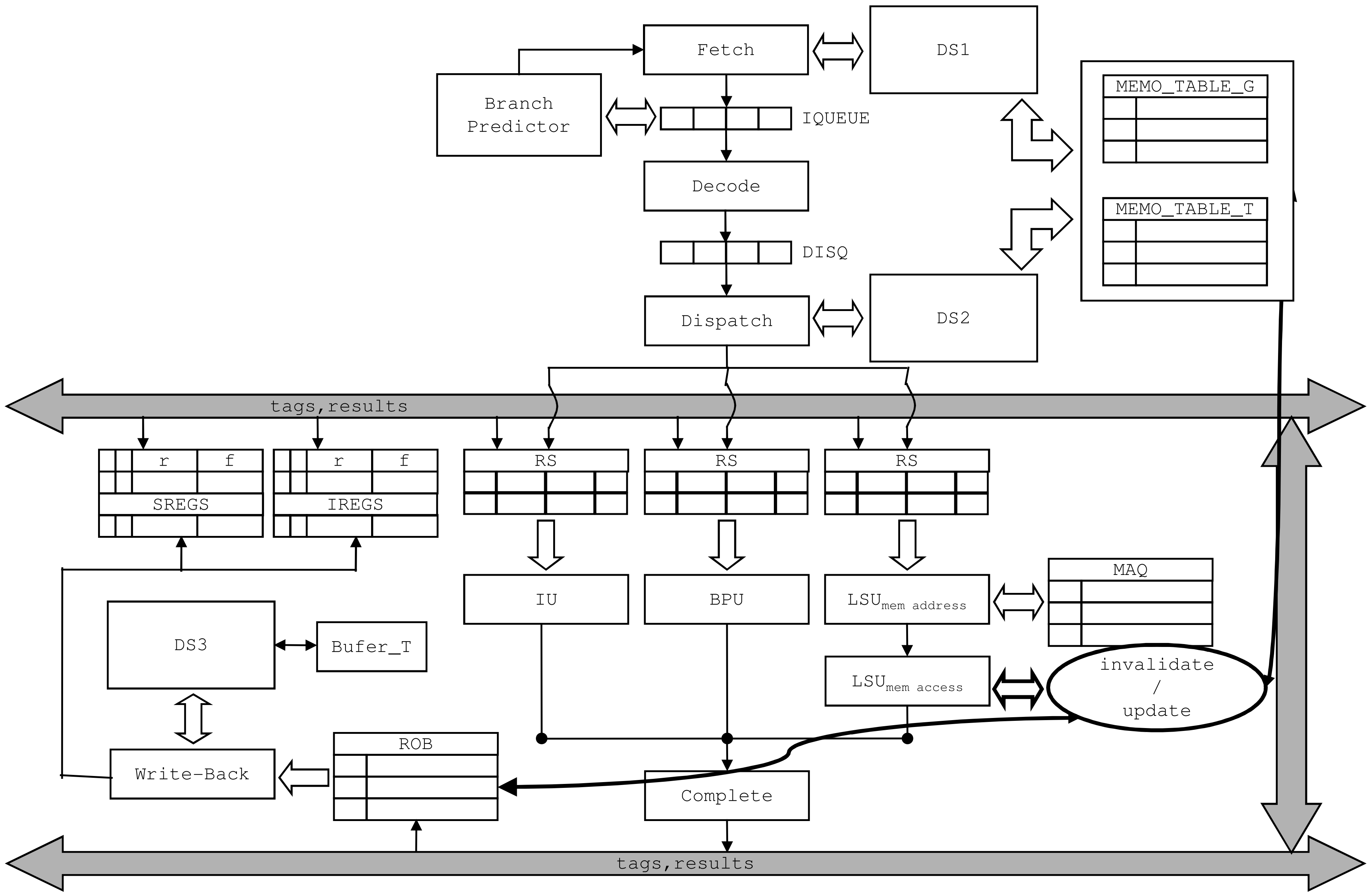
Adding Memory Access Instructions to DTM

Implementation Details

- ❑ Reading instructions are necessarily the last trace instructions;
  - • Reading instructions can have their values modified by instructions external to the trace;
  - • The inclusion of instructions dependent on a memory reading instruction in the same trace can lead to total invalidation of the trace if the contents of the memory are modified;
- ❑ Only one read instruction in memory is allowed per trace;
  - • Avoids the need to update and invalidate the Buffer_T in the event of a write access to memory at an address used by a reading instruction present in the trace being formed;
  - • Reduces the cost of implementing a mechanism for invalidating and anticipating values for a large number of fields with memory access values;



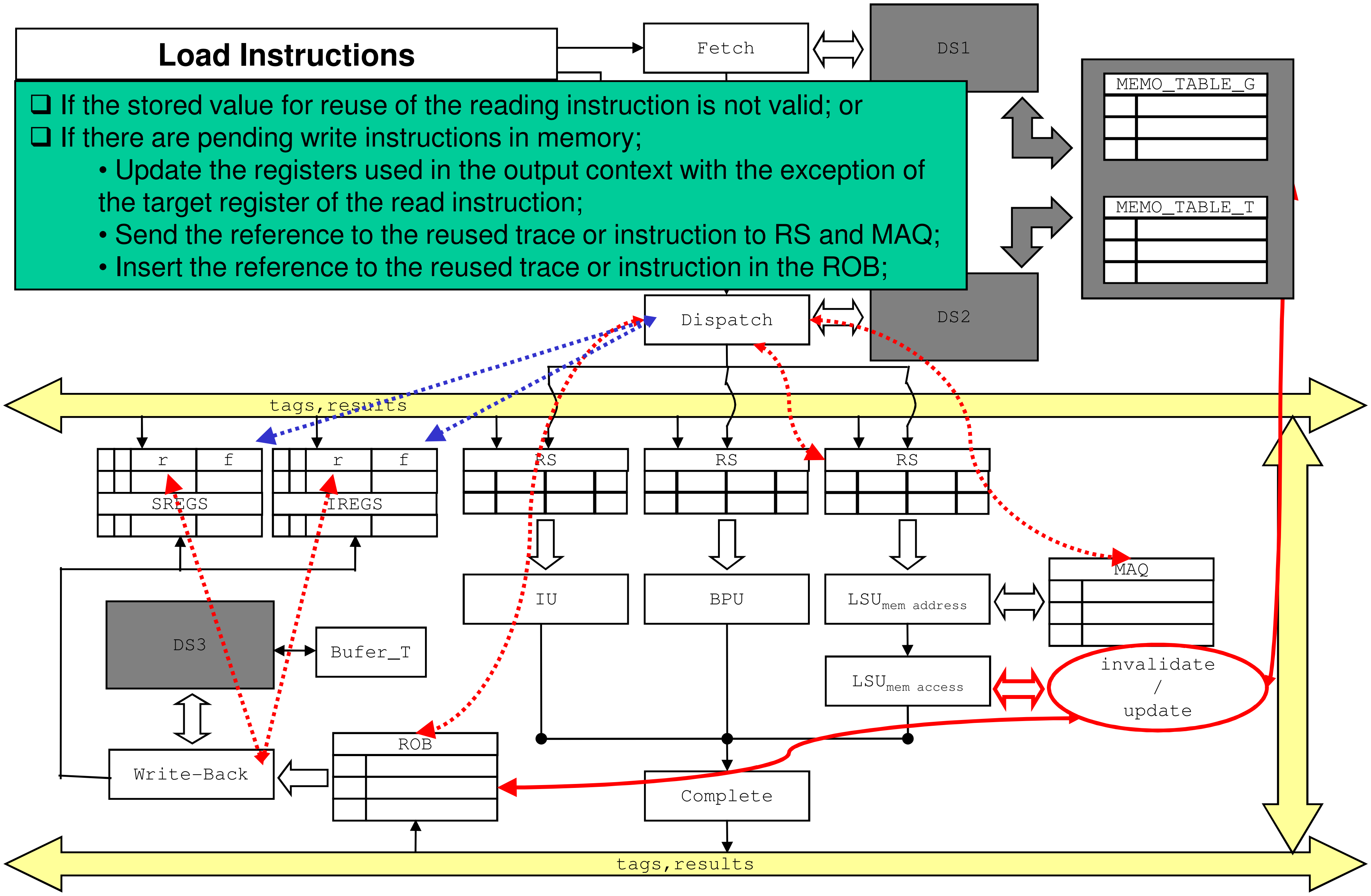
```
          :
100. addcc r2,r7,0x08        r2
104. bne r2,0x124            r2
124. ld r2+0x1000,r5         r2  (r5)
128. sll r5,r5,0x02          r2  (r5)
12c. sub r7,r5,0x08          r2  (r5)  (r7)
          :
```


Fetch
DS1
Branch Predictor
IQUEUE
MEMO_TABLE_G
Decode
MEMO_TABLE_T
DISQ
Dispatch
DS2
tags,results
r
f
SREGS
IREGS
RS
RS
RS
IU
BPU
LSU$_{mem\ address}$
MAQ
DS3
Bufer_T
LSU$_{mem\ access}$
invalidate / update
Write-Back
ROB
Complete
tags,results

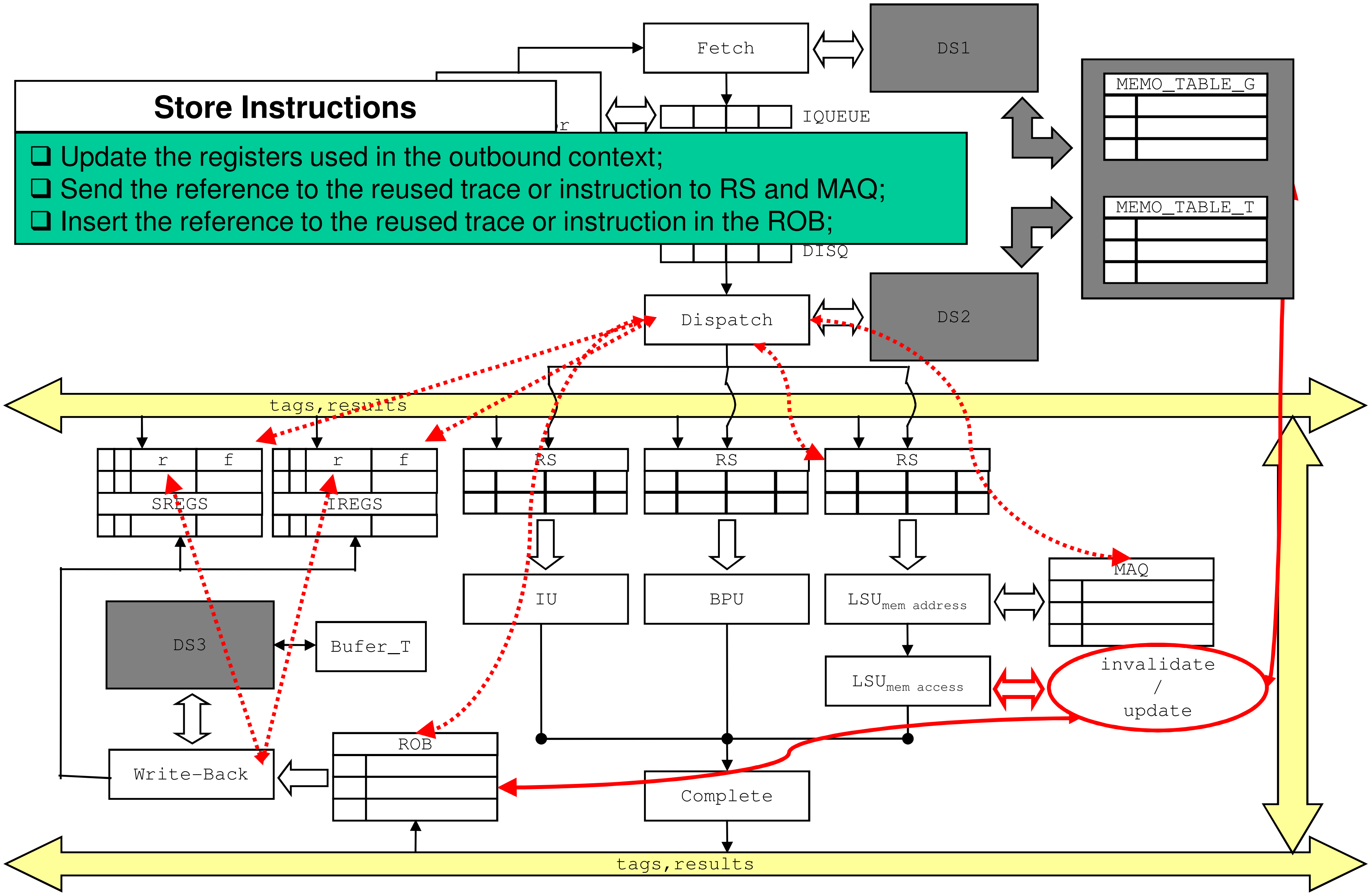
Adding Memory Access Instructions to DTM

Load Instructions

- ❑ If the stored value for reuse of the reading instruction is not valid; or
- ❑ If there are pending write instructions in memory;
  - • Update the registers used in the output context with the exception of the target register of the read instruction;
  - • Send the reference to the reused trace or instruction to RS and MAQ;
  - • Insert the reference to the reused trace or instruction in the ROB;



Adding Memory Access Instructions to DTM



Store Instructions

- ❑ Update the registers used in the outbound context;
- ❑ Send the reference to the reused trace or instruction to RS and MAQ;
- ❑ Insert the reference to the reused trace or instruction in the ROB;

Content

The DTM*m* Mechanism

Initial Objectives

- ❑ Reduce the cost of invalidate/update mechanisms;
- ❑ Avoid increasing the size of the trace memorization table's entries;

Store Instructions

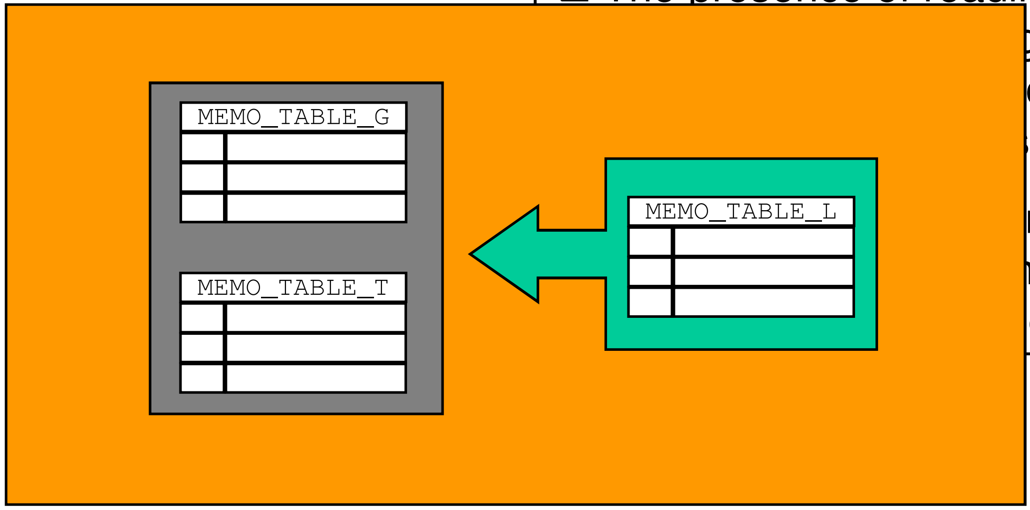
❑ When a memory write instruction has the same input values, we can ensure that the memory access address and the value to be stored are the same;

Load Instructions

❑ When a memory read instruction has the same input values, we can assure that the memory access address stored is the same, but we cannot say anything about the value that will be read;

❑ The presence of reading instructions
gs the need to
or invalidate/
e tables;

nisms for reusing
the reuse tables
ore complex;



The DTM*m* Mechanism

Modifications Applyed at the Global Memoization Table's Entries – MEMO_TABLE_G

pc	jmp	brc	btaken	sv1	sv2	maddr	res/npc
30b	1b	1b	1b	32b	32b	32b	32b

pc – instruction address

jmp – flag of inconditional branch

brc – flag of conditional branch

btaken – flag of branch **taken/not_taken**

sv1 – source operand value #1

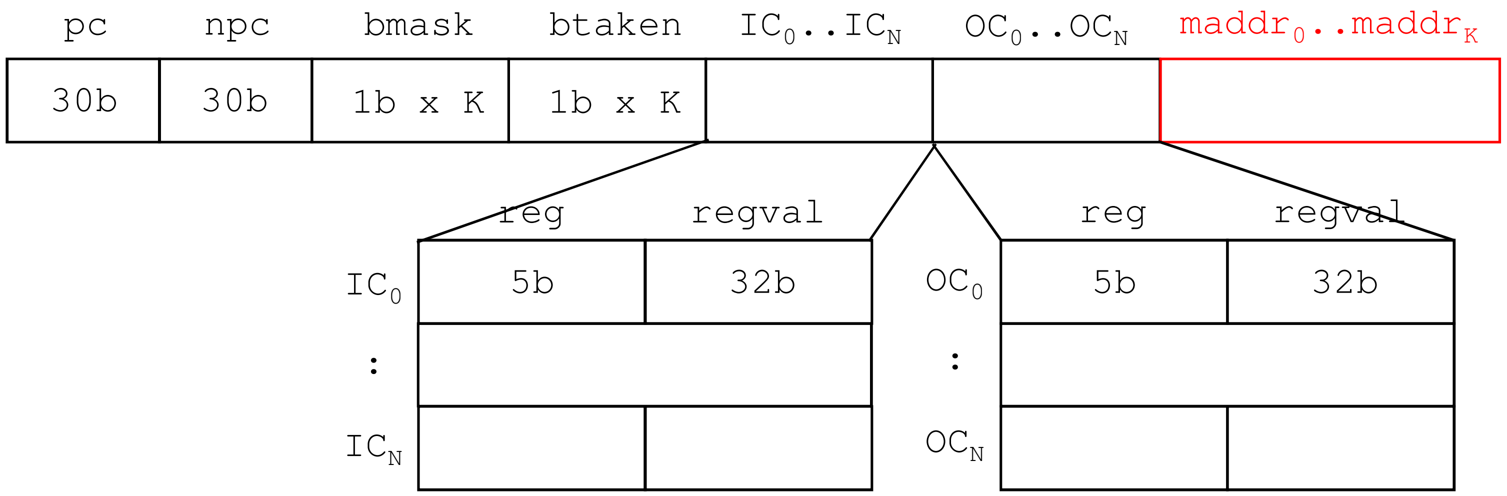
sv2 – source operand value #2

maddr – memory access address

res/npc – operation result/branch target destination

The DTM*m* Mechanism

Modifications Applyed at the Trace Memoization Table's Entries – MEMO_TABLE_T



pc – instruction address

npc – address of the next instruction after the trace

bmask – flag of branch instructions

btaken – flag of branch **taken/not_taken**

$IC_0, ..., IC_N$ – trace's input context

reg – source registers' address

regval – source registers' values

$OC_0, ..., OC_N$ – trace's output context

reg – destination registers' address

regval – destination registers' values

$maddr_0, ..., maddr_K$ – memory access address

The DTM*m* Mechanism

Load Instruction Table’s Entries – MEMO_TABLE_L

pc	sv1	sv2	maddr	mem valid	res
30b	32b	32b	32b	1b	32b

pc – instruction address

sv1 – source operand value #1

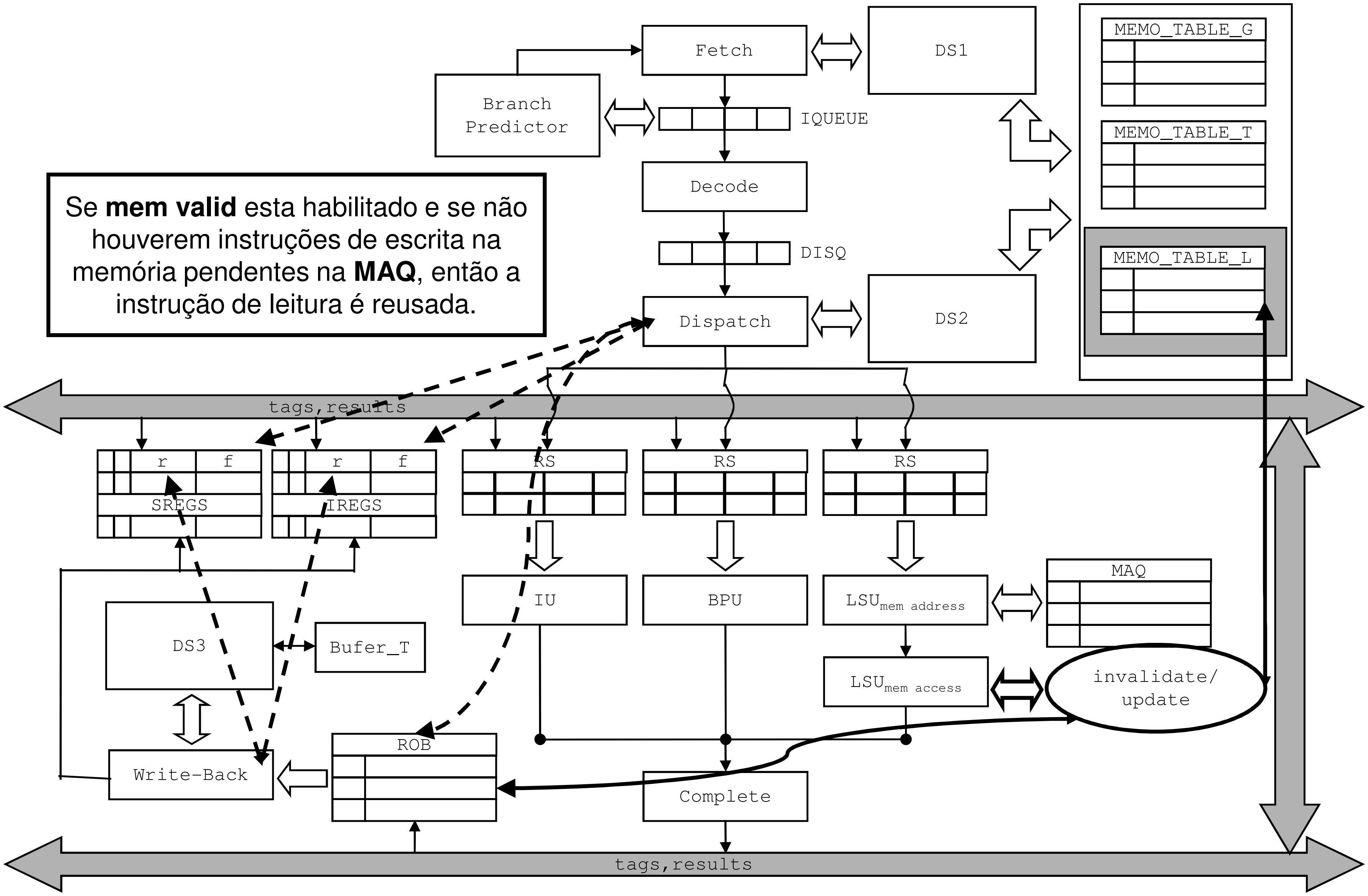
sv2 – source operand value #2

maddr – memory access address

mem valid – flag of valid value for load

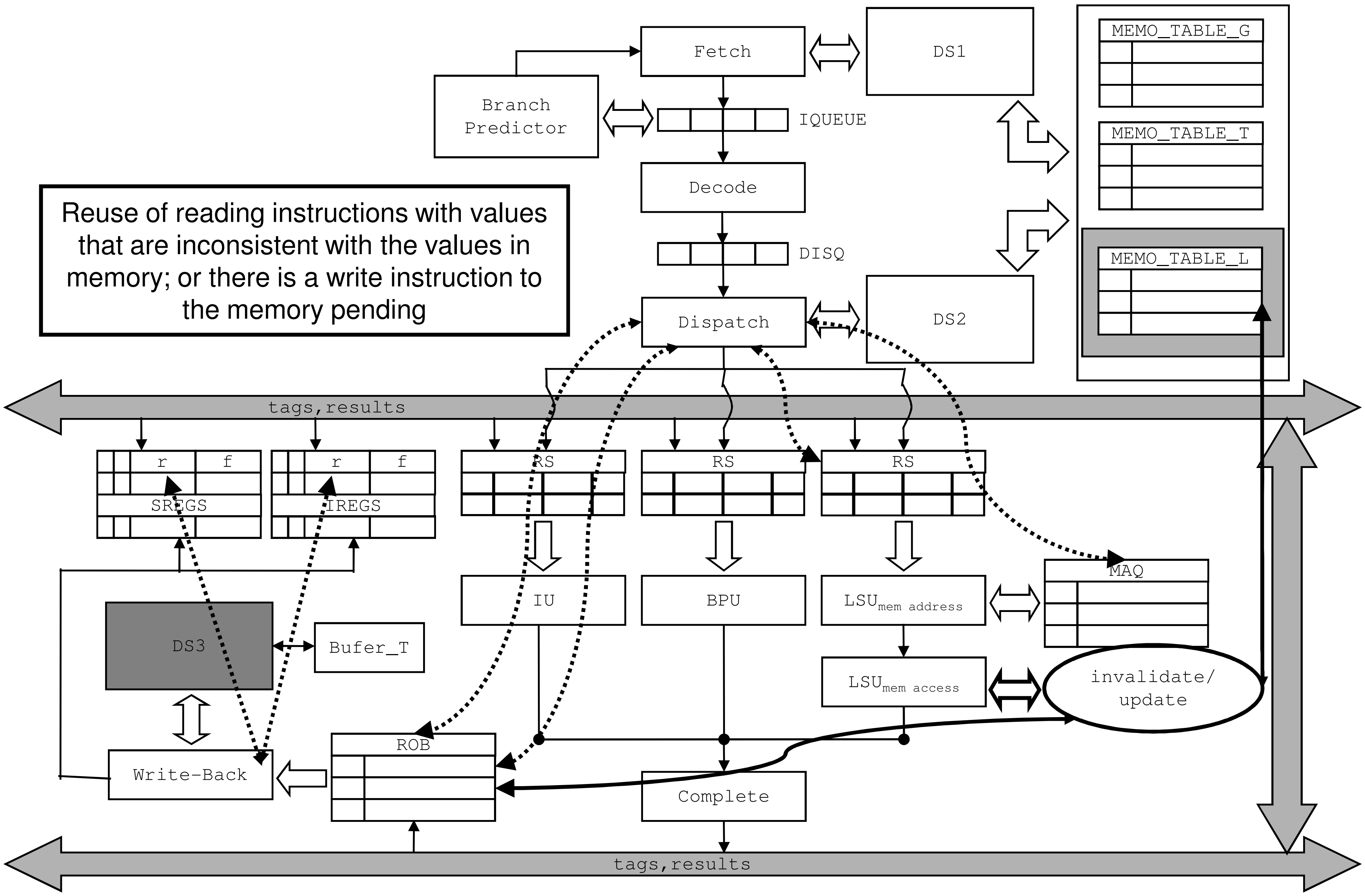
res – operation result

Se **mem valid** esta habilitado e se não houverem instruções de escrita na memória pendentes na **MAQ**, então a instrução de leitura é reusada.



The DTM*m* Mechanism

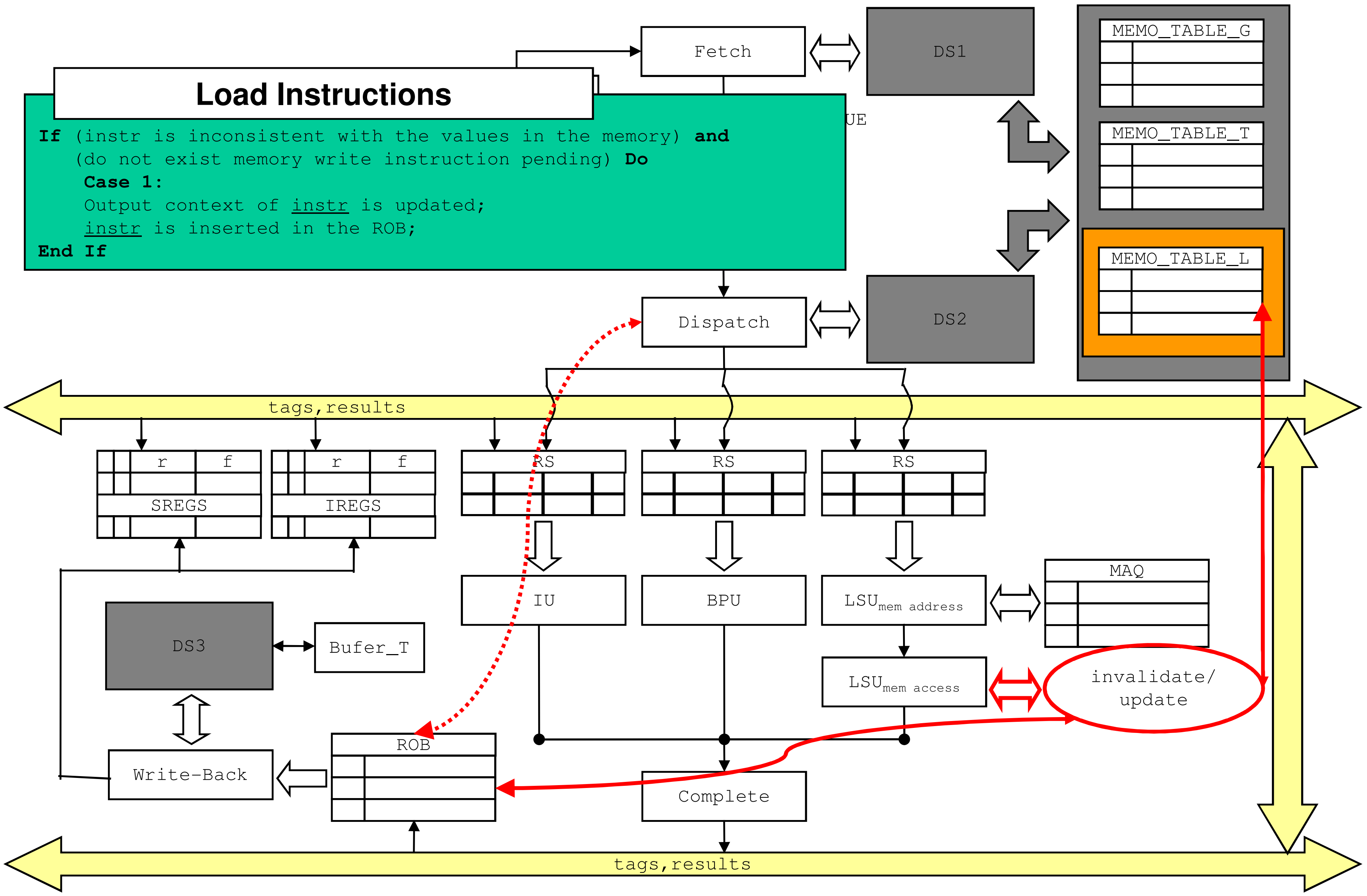
Reuse of reading instructions with values that are inconsistent with the values in memory; or there is a write instruction to the memory pending



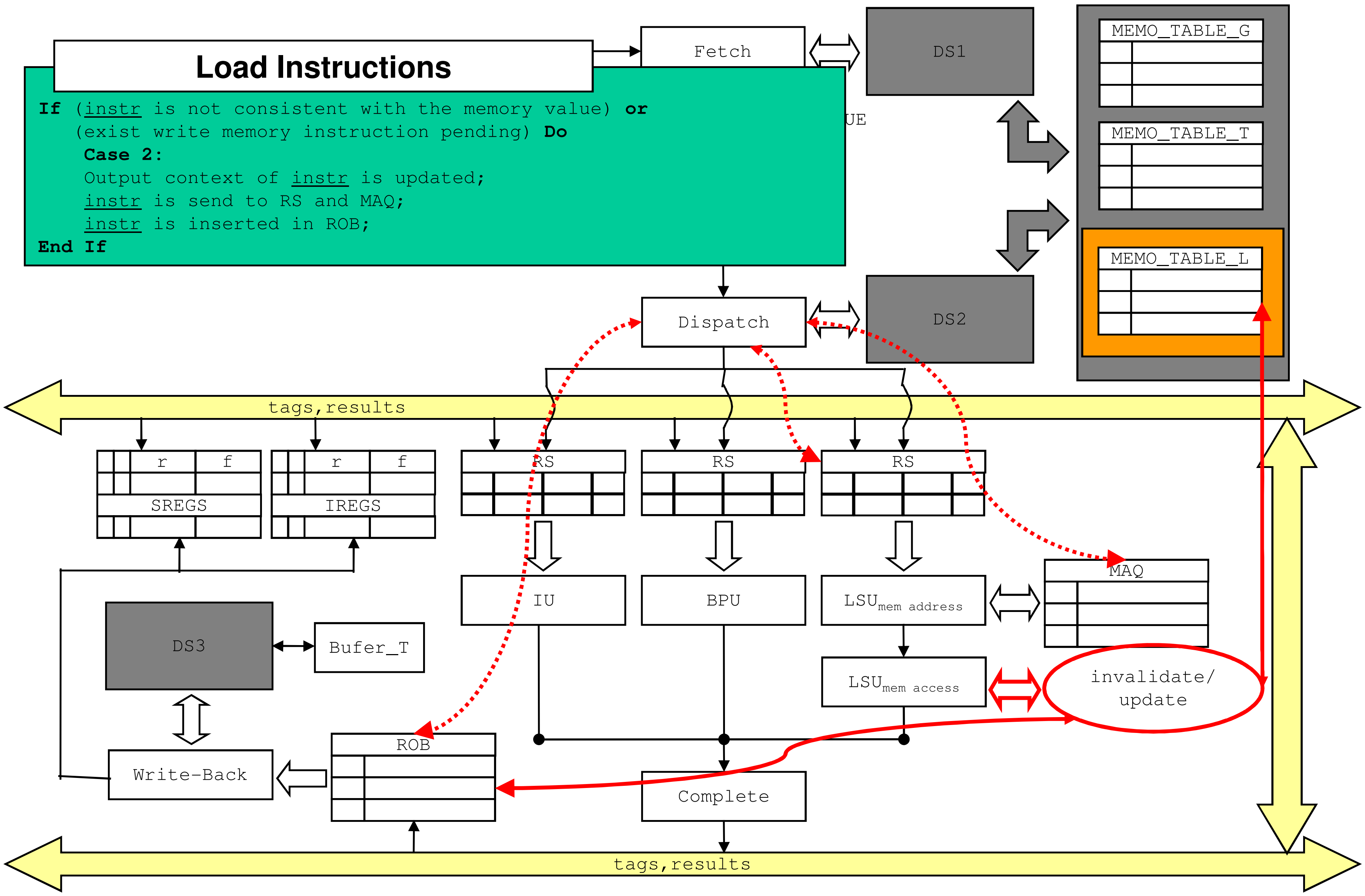
The DTM*m* Mechanism

Load Instructions

```
If (instr is inconsistent with the values in the memory) and
   (do not exist memory write instruction pending) Do
    Case 1:
    Output context of instr is updated;
    instr is inserted in the ROB;
End If
```



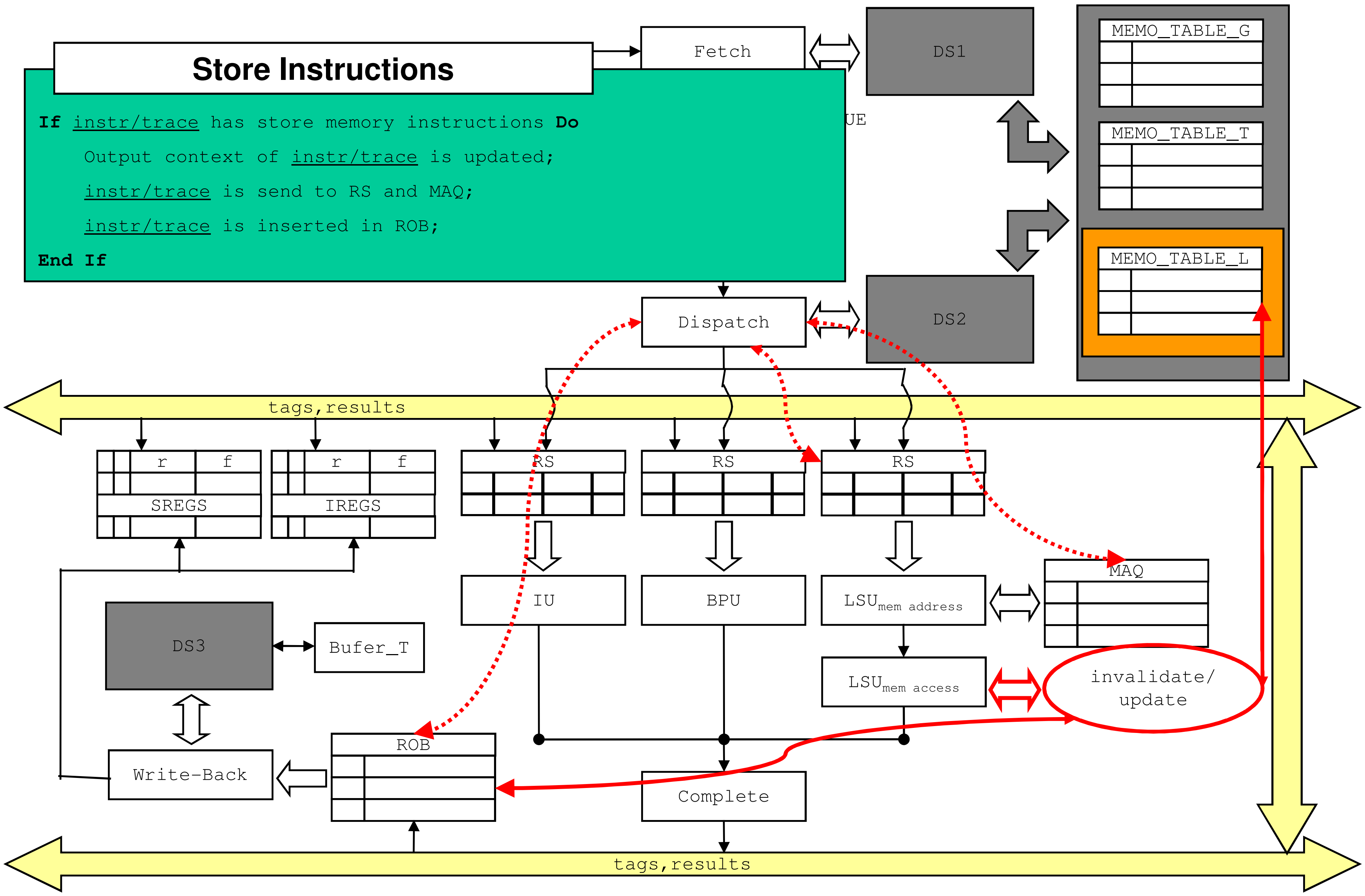
The DTM*m* Mechanism



Load Instructions

```
If (instr is not consistent with the memory value) or
   (exist write memory instruction pending) Do
    Case 2:
    Output context of instr is updated;
    instr is send to RS and MAQ;
    instr is inserted in ROB;
End If
```

The DTM*m* Mechanism



Store Instructions

```
If instr/trace has store memory instructions Do
    Output context of instr/trace is updated;
    instr/trace is send to RS and MAQ;
    instr/trace is inserted in ROB;
End If
```

Content

Simulation Enviroment - SuperSIM

Implemented Resources

❑ Run the compiled programs for the **Sparc v7** processor;

❑ Branch prediction, speculative execution, future registers, reservation stations, reordering buffer;

❑ Step-by-step execution, complete program execution and execution of a specified number of instructions;

❑ Reports for monitoring the running program, main memory, BTB, instruction queue, dispatch queue, registers window, reservation stations, execution units, memory access queue, reordering buffer, memorization tables and statistics;

❑ Runs on SUN Solaris 7 and INTEL Linux platforms;

Simulator Limitations

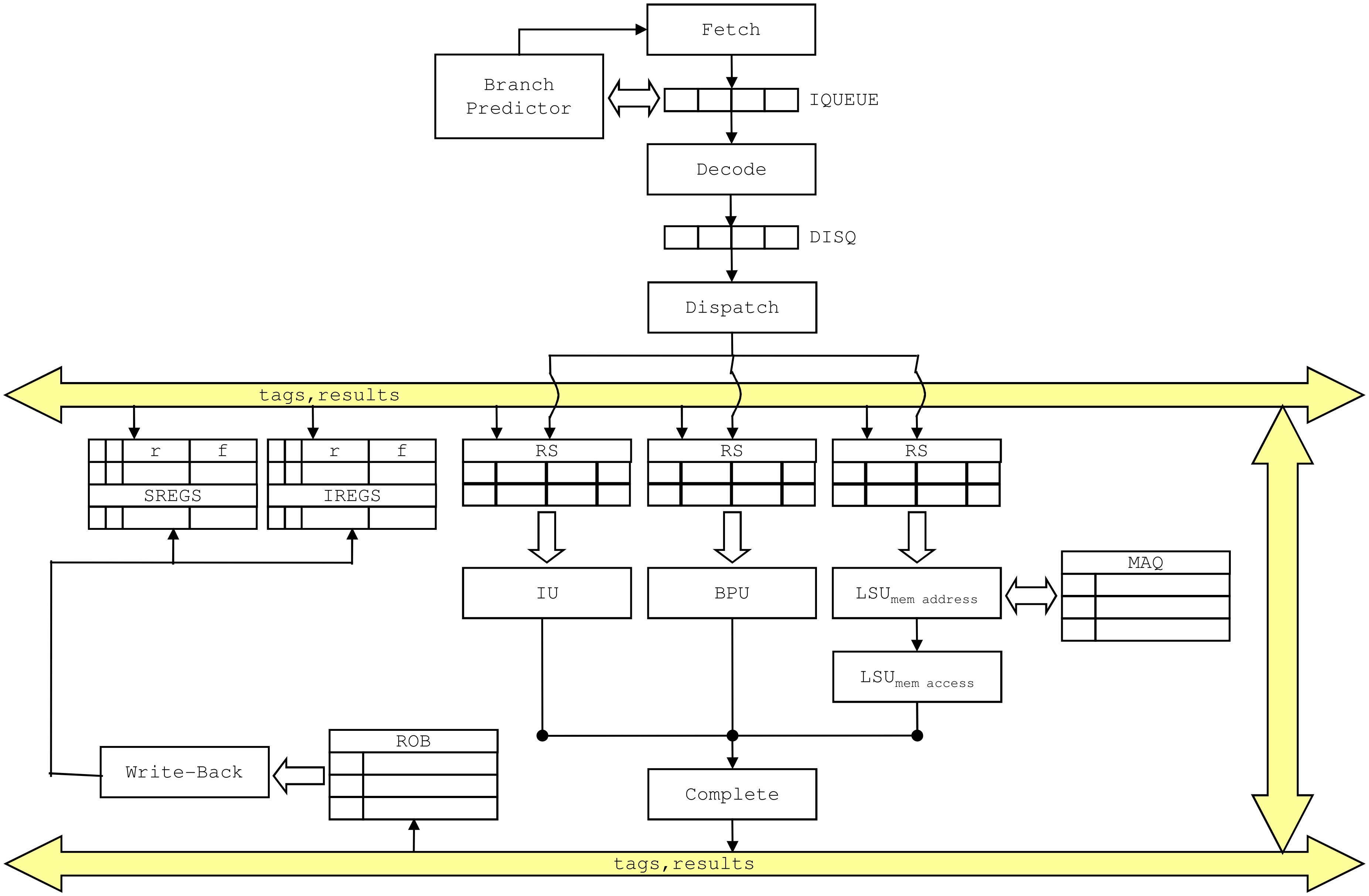
❑ Does not implement floating-point registers and does not support read and write operations in memory using floating-point registers;

❑ Does not implement floating point unit;

❑ Does not implement overflow and underflow control of registers' window;

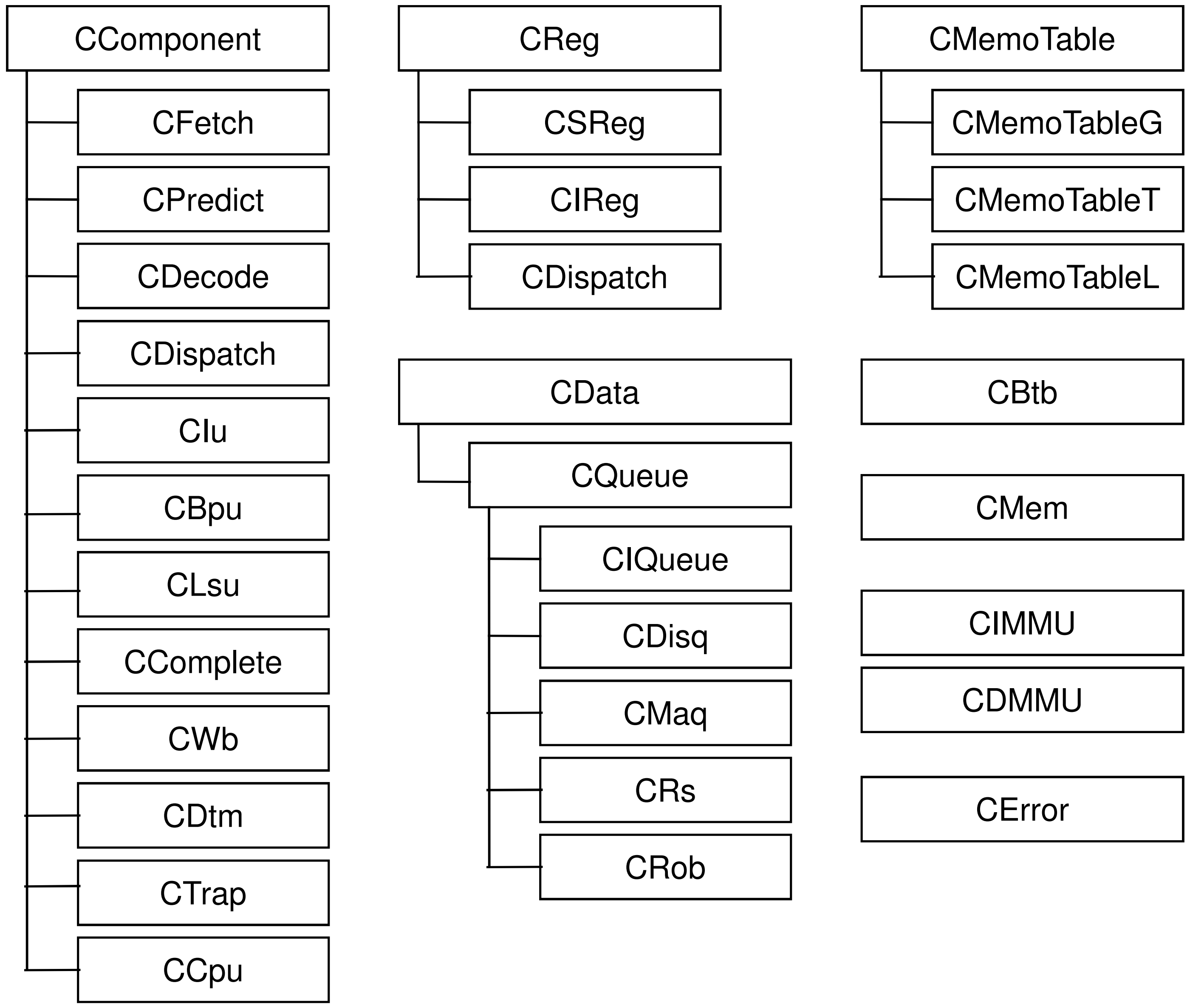
❑ Implements parcially the double deviation chains - delayed control-transfer couples defined in **Sparc v7** architecture;

Simulation Enviroment - SuperSIM



Simulation Enviroment - SuperSIM

Data Model



Content

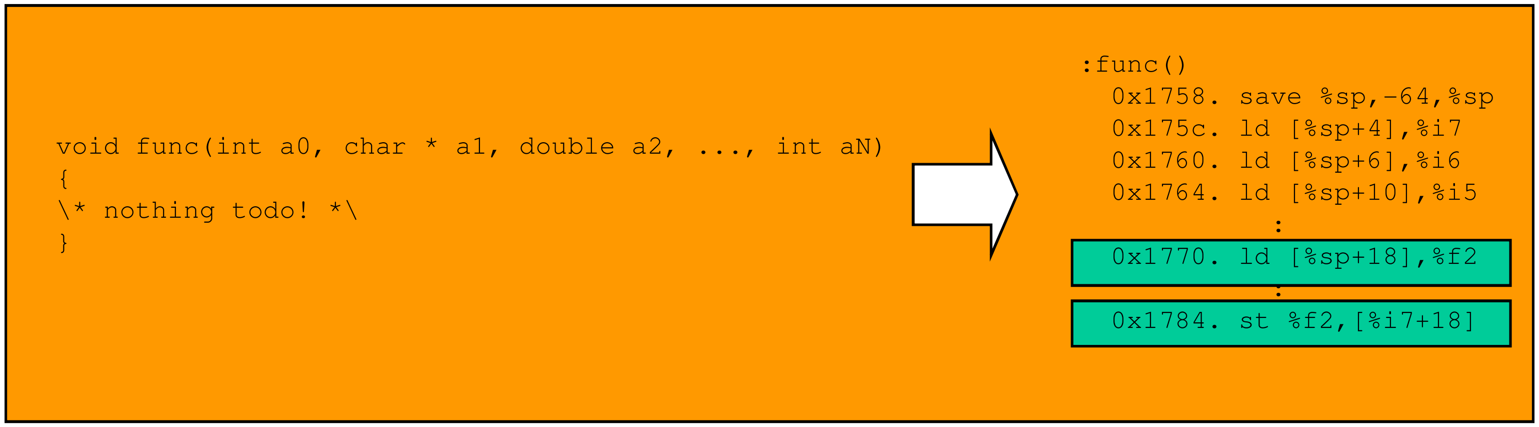
Results' Analisys

Programa	**Entrada**	**Total de Instruções**
go	9stone21 (ref)	100.000.000 (não finalizado)
m88ksim	ctl.raw (test)	100.000.000 (não finalizado)
compress	test.in (train)	76.978.452 (finalizado)
li	deriv.lsp (ref)	100.000.000 (não finalizado)
ijpeg	vigo.ppm (ref)	100.000.000 (não finalizado)
vor		o)

❑ All programs were compiled using gcc-2.5.2 with -O optimization, -static static library option and disabled register window usage -mflat;

Results’ Analisys

The GCC-2.5.2 compilation effect



```
void func(int a0, char * a1, double a2, ..., int aN)
{
\* nothing todo! *\
}
```

```
:func()
  0x1758. save %sp,-64,%sp
  0x175c. ld [%sp+4],%i7
  0x1760. ld [%sp+6],%i6
  0x1764. ld [%sp+10],%i5
                :
  0x1770. ld [%sp+18],%f2
                :
  0x1784. st %f2,[%i7+18]
```

Results' Analisys

	Simplescale Tool Set	SuperSIM
Instruction fetch	4 instructions per cycle. Only one branch taken per cycle. May cross the border of the cache.	4 instructions per cycle. Only one branch taken per cycle. There is no limit imposed by the cache.
Instruction cache	16 Kb, associative – 2 per set, 32 bytes per line, latency of 6 cycles for cache miss in L1 and 20 cycles for cache miss in L2.	Not implemented, adopted 100% cache hit.
Branch prediction	Bimodal, 2k entries, can predict multiple deviations simultaneously.	Bimodal, 1k entries, can predict multiple deviations simultaneously.
Speculative execution mechanism	Execution of up to four instructions per cycle out of order, reorder buffer with 16 entries and memory access queue with 8. Loads are executed after all previous store addresses are known. Loads are served by stores that access the same address if both are in the memory access queue.	Execution of up to four instructions per cycle out of order, reorder buffer with 16 entries and memory access queue with 16. Loads are executed after all preceding stores has been executeds. Load instructions aren't served by stores.
Architectural registers	32 integers registers, 32 floating point registers, implements spetial registers hi, lo and fcc.	Registers' window with 520 integer registers, 8 global registers and 32 x 24 overlaped registers, 2 spetial registers PSR and Y.
Functional units	4 ALUs of integers, 2 load/store units, 4 floating point adders, 1 mult/div of integers, 1 mult/div of floating point.	3 ALUs to solve integer operations, including multiplication. 2 branch units and 2 loads/stores units.
Latency of functional units	ALU-integers/1, load/store/1, integer mult/3, integer div/20, fp adders/2, fp mult/4, fp div/12, fp sqrt/24.	All integers instructions has latency of 1. Load/store instructions have latency of 2.
Data cache	16Kb, associative-2 per set, 32 bytes per line, latency of 6 cycles for cache miss in L1 and 20 cycles for cache miss in L2.	Not implemented, adopted 100% cache hit.

Results' Analisys

	DTMmips	**DTMsparc**
contexto de entrada	6 entries	7 entried
contexto de saída	6 entries	7 entries
tamanho max dos traces	16 instructions	Unlimited
Número max de desvios	10 branches	10 branches
heurística	Simple instruction repetition.	Simple instruction repetition.
conjunto de seleção	Aritmetrics instructions, logics, branches, system calls and returns, memory access address calculations.	Aritimetrics instructions, logics, branches, system calls and returns, memory address calculations amd memory access values.
política de atualização das tabelas de reuso	LRU	LRU
tabelas de reuso	MEMO_TABLE_G – 4672 entradas, associativa. MEMO_TABLE_T – 512 entradas, associativa.	MEMO_TABLE_G – 4672 entradas, associativa. MEMO_TABLE_T – 512 entradas, associativa. MEMO_TABLE_L – 512 entradas, associativa.

Results’ Analisys

Used Metrics

- ❑ Percentual Reuse:

% reuse = ir / itot ; ir – number of reused instructions
; itot – number of executed instructions

- ❑ Percentual acceleration:

acceleration = IPC_t / IPC_{base} ; IPC_t – instructions executed per cycle
; with the mechanism
; IPC_{base} – instructions executed per
; cycle in base archtecture

- ❑ Arithmetric Mean:

AM = $(\sum^{i}_{n=0} S_i) / n$; n – total of computed values
; S_i – computed values

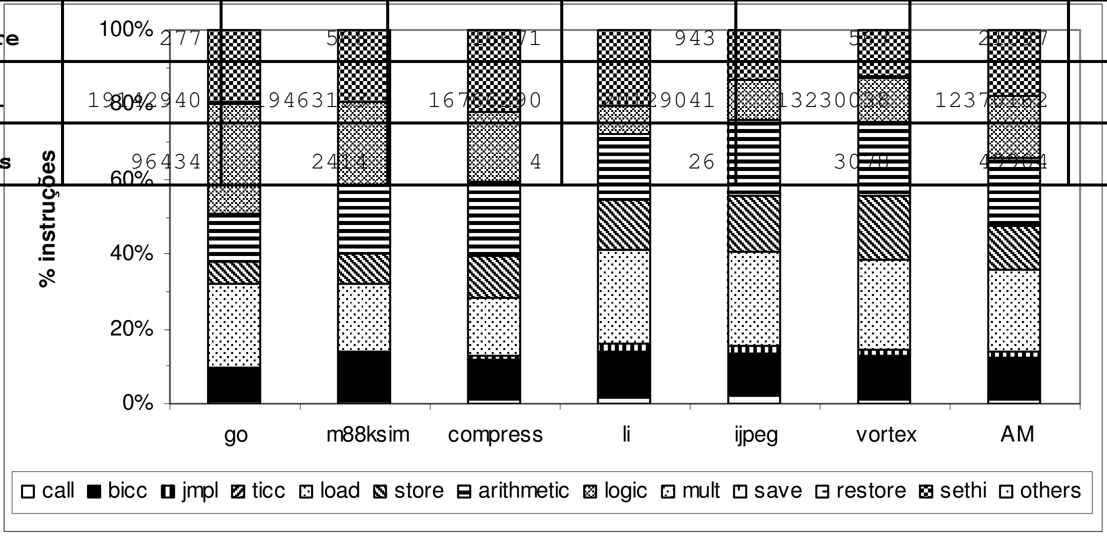
- ❑ Harmonic Mean:

HM = $n (\sum^{i}_{n=0} 1/S_i)^{-1}$; n – total of computed values
; S_i – computed values

Análise dos Resultados

Distribution of Instructions Type in the Programs

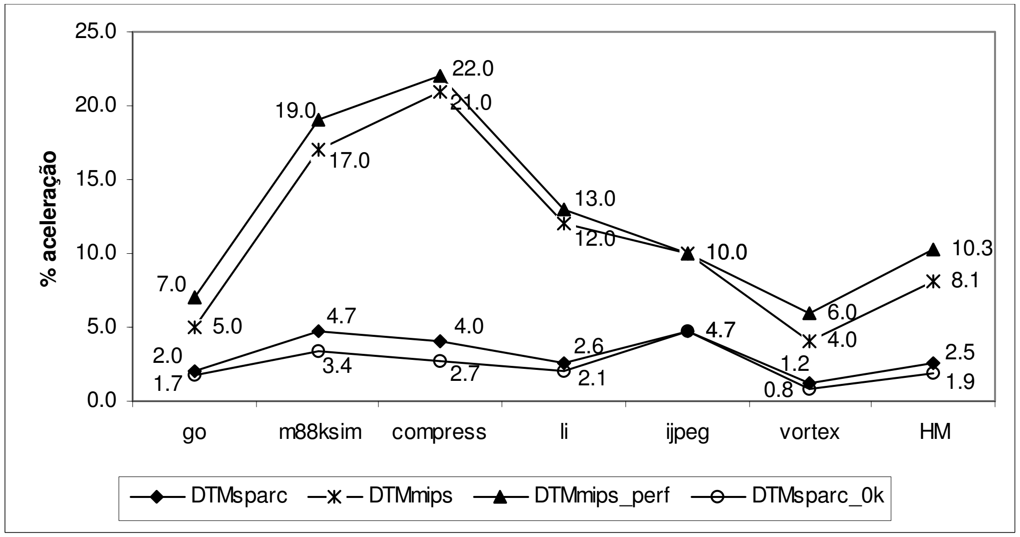
	go	m88ksim	compress	li	ijpeg	vortex	AM
call	431747	733152	939892	1647111	2205754	1119034	1179448
bicc	8509903	12592476	8031639	12185567	11028625	11947490	10715950
jmpl	451801	814256	940050	2002746	2206465	1134519	1258306
ticc	12	65	12	203	546	4224	844
load	22677213	17947340	12084545	25117843	24989669	24552343	21228159
store	6092792	7958462	8263676	13822452	15428577	16871675	11406272
arithmetic	12701672	18469412	15563675	17263697	19833903	19553774	17231022
logic	29518067	22011463	14355316	7829303	11038960	12143797	16149484
mult	376865	6691	10	125	33251	210884	104638
save	277	568	16672	943	567	21097	6687
restore	277	568	16671	943	567	21097	6687
sethi	19142940	19463155	16766290	20129041	13230038	12370162	16850267
others	96434	2414	4	26	3078	49904	25310

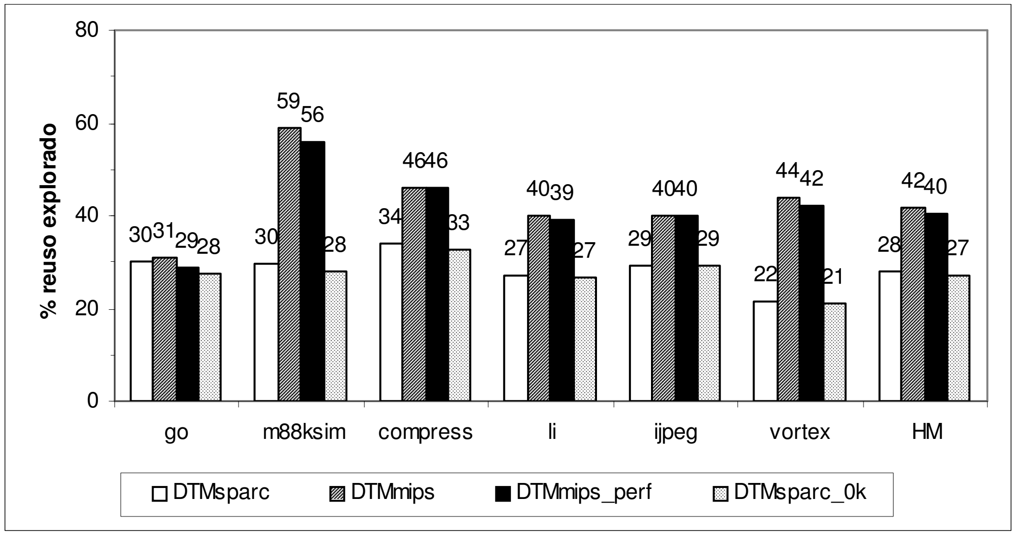


- ❑ Memory access instructions represent an average of 34% of instructions executed*;*
- ❑ The frequency of save and restore instructions is very low and had no effect on the compilation applied;
- ❑ Memory read instructions represent 22% of instructions executed;
- ❑ SETHI represent about 19% of the instructions executed;

Results' Analisys

Results of DTM Implementation in Sparc v7



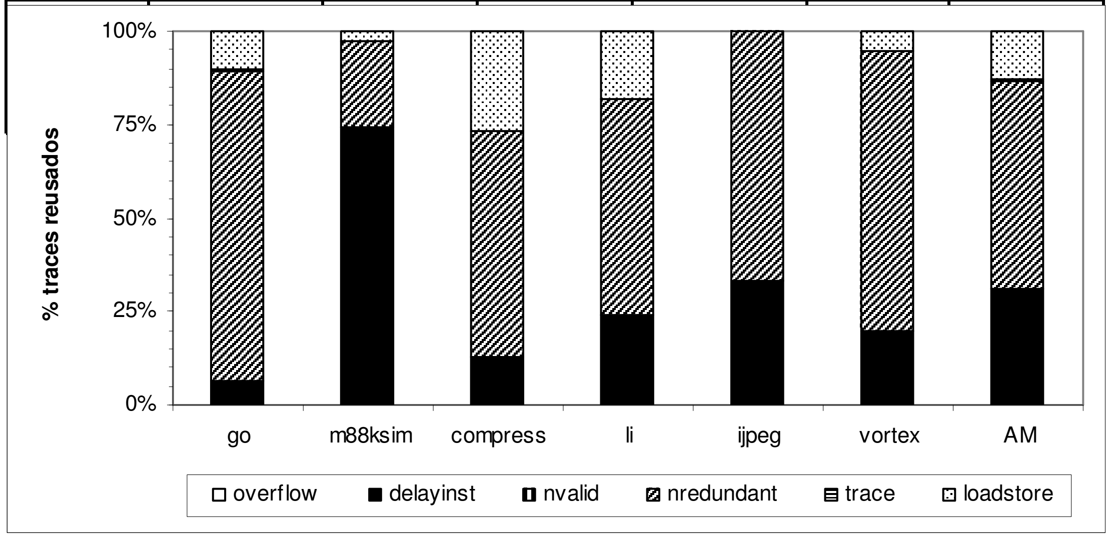


- ❑ DTMmips obtained an average acceleration 5.6% above the result presented by DTMsparc*;*
- ❑ DTMmips obtained an average result 14% higher than the exploited reuse presented by DTMsparc;
- ❑ DTMsparc obtained an average result 0.6% above the result obtained by DTMsparc_0k (without MEMO_TABLE_T);
- ❑ Reuse explored by DTMsparc and DTMsparc_0k obtained similar results;

Results' Analisys

Results of DTM Implementation in Sparc v7

	overflow	delayinst	nvalid	nredundant	trace	loadstore
go	0.00	6.22	0.00	83.07	0.39	10.32
m88ksim	0.00	74.30	0.00	22.97	0.00	2.73
compress	0.00	13.10	0.00	59.99	0.00	26.91
li	0.00	24.15	0.00	57.43	0.00	18.42
ijpeg	0.00	33.33	0.00	66.67	0.00	0.00

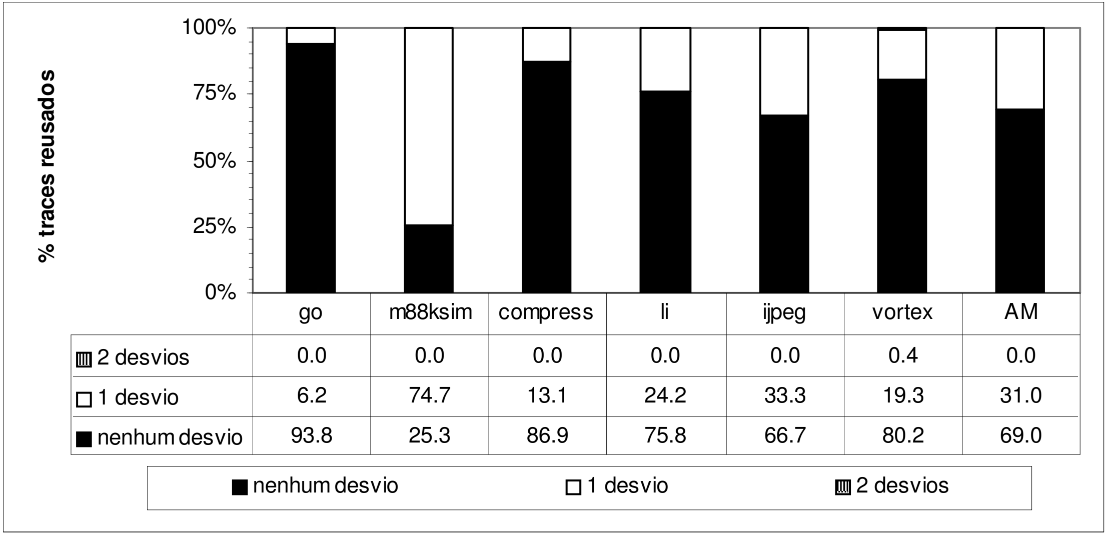


- ❑ Traces completed by instructions in the delay-slot represented up to 74% for m88ksim and obtained a harmonic mean of 31%*;*
- ❑ In the compress 27% of the traces were finished by memory access instructions;

Results’ Analisys

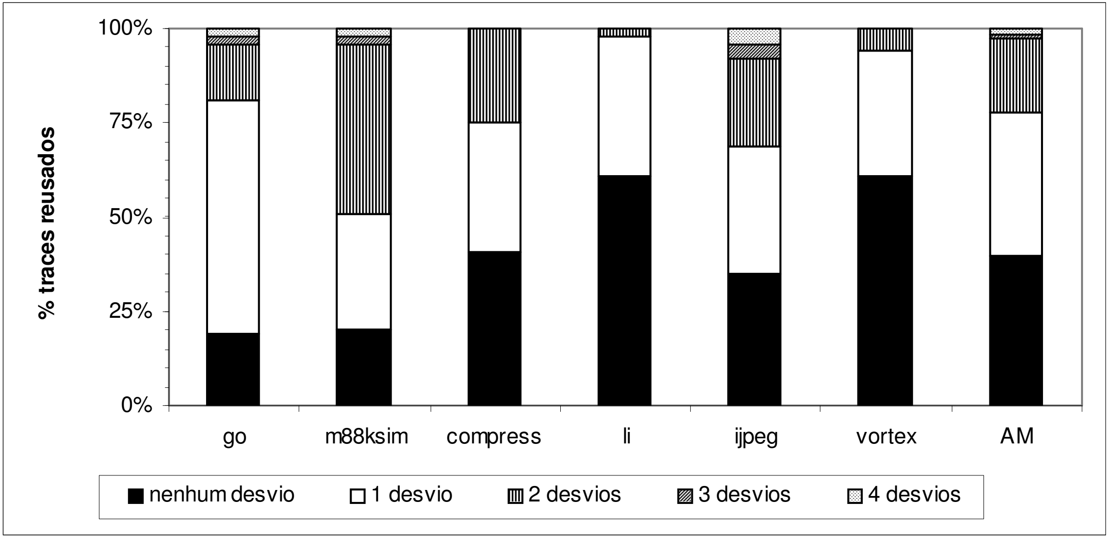
Results of DTM Implementation in Sparc v7

DTM*sparc*



	go	m88ksim	compress	li	ijpeg	vortex	AM
▥ 2 desvios	0.0	0.0	0.0	0.0	0.0	0.4	0.0
□ 1 desvio	6.2	74.7	13.1	24.2	33.3	19.3	31.0
■ nenhum desvio	93.8	25.3	86.9	75.8	66.7	80.2	69.0

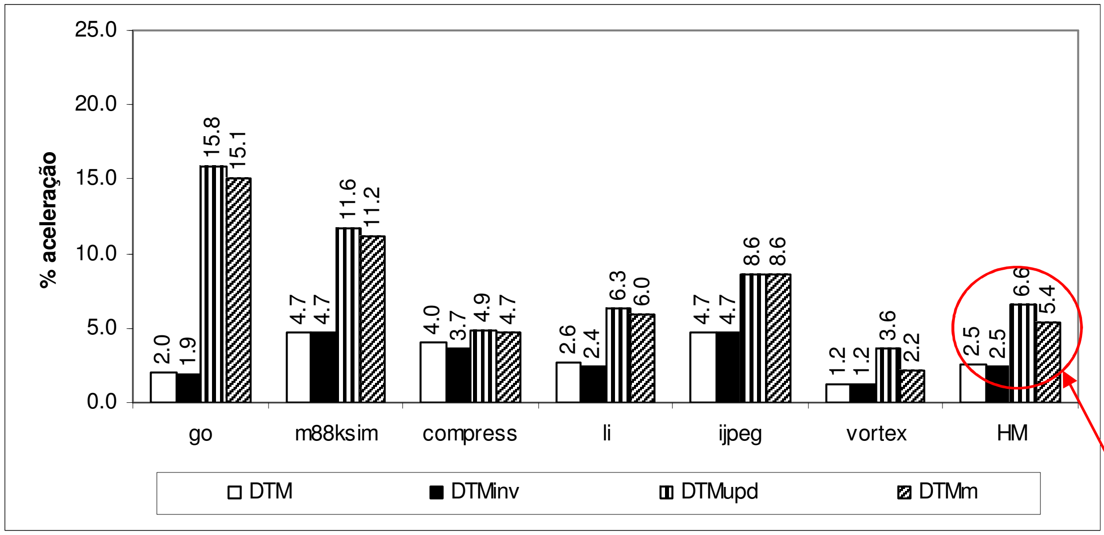
- ❑ In DTMsparc 69% of the reused traces have no deviations and 31% of them have only one deviation;
- ❑ In DTMmips 39% of the traces have no deviation and 38.5% of the traces have only one deviation, and about 22% of the traces have more than one deviation;

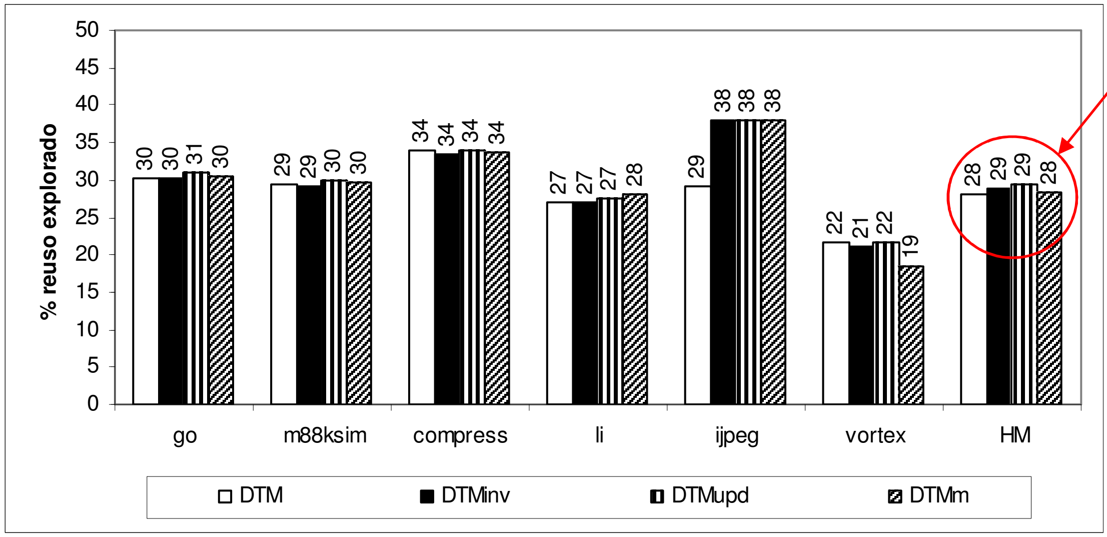


DTM*mips*

Results' Analisys

Results of Implementations of Memory Access Instructions Reuse with DTM





- ❑ The average acceleration of the mechanisms with anticipation of memory values were 4.1% and 2.9% above the value obtained with the DTM;

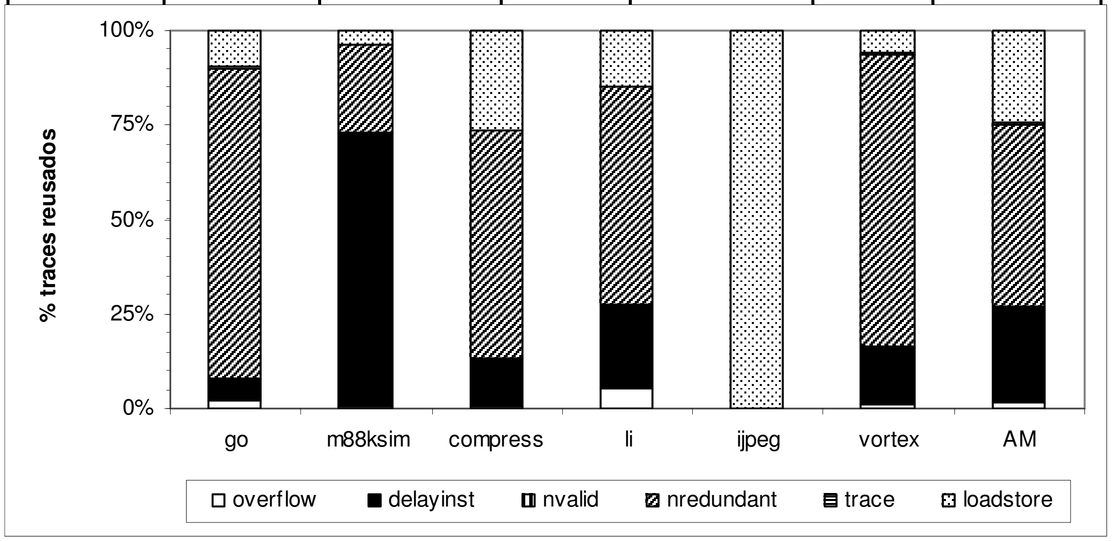
- ❑ The exploited reuse showed an average variation of 1% between the mechanisms;

- ❑ The average acceleration obtained by the mechanism with invalidation of memory values was equal to that obtained with the DTM;

Results’ Analisys

Analisys of the Results Obtained with DTM*inv*

	overflow	delayinst	nvalid	nredundant	trace	loadstore
go	1.97	6.26	0.00	83.61	0.37	9.76
m88ksim	0.32	72.75	0.00	23.48	0.06	3.71
compress	0.45	12.94	0.00	60.44	0.00	26.62
li	5.71	23.44	0.00	60.80	0.00	15.76
ijpeg	0.00	0.00	0.00	0.00	0.00	100.00
vortex	0.95	15.37	0.00	78.42	0.16	6.05



- ❑ DTMinv little explores the reuse of memory reading instructions due to the frequent invalidation of reuse tables;
- ❑ There was an increase in the number of reused traces terminated by overflow of the output context;

Results' Analisys

Analisys of the Results Obtained with DTM*inv*

	2 instrucoes		3 instrucoes		4 instrucoes	
	store	load	store	load	store	load
go	82272	0	58426	333015	0	3624
m88ksim	39133	0	51465	89056	11995	2
compress	24619	0	13830	600920	0	226179
li	141802	0	142255	295577	6	86
ijpeg	0	0	2199272	0	0	0
vortex	166399	0	51416	25008	0	5473
AM	47971	0	410875	219761	2000	38315

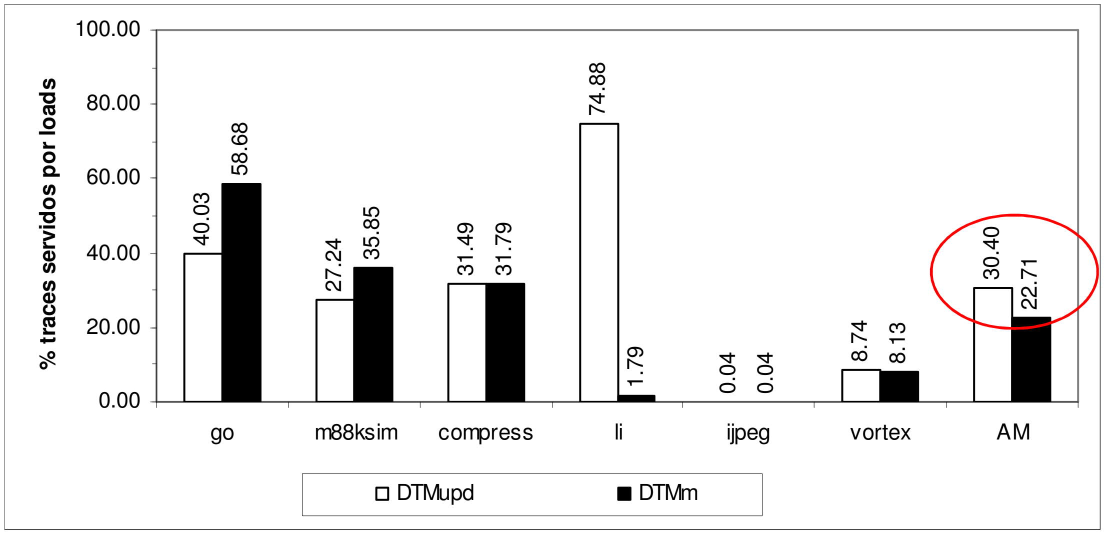
❑ There was an increase in the number of traces with two instructions composed of a write instruction in memory;

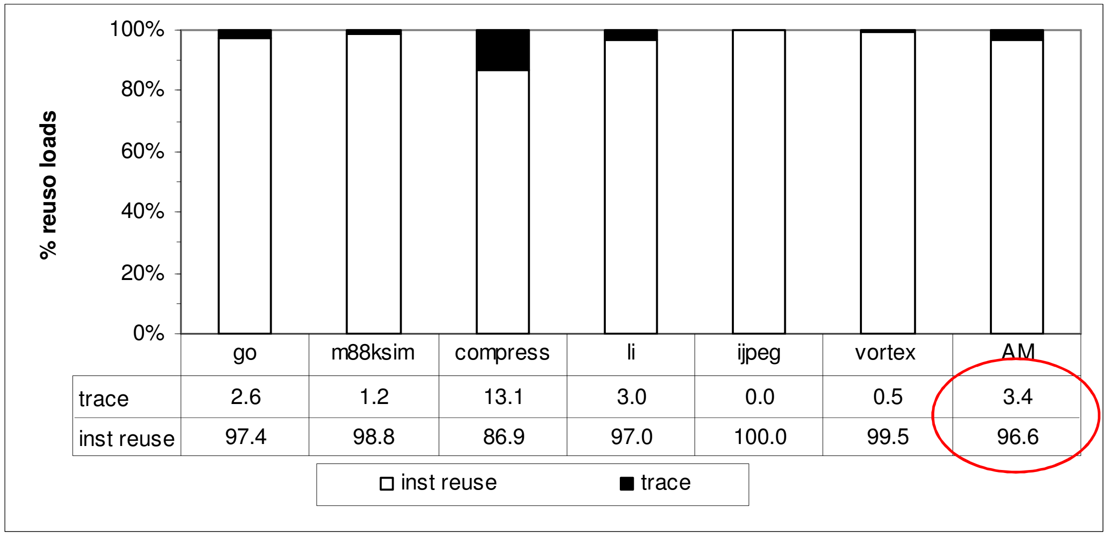
❑ Traces with two instructions and which have memory access instructions are made up of store instructions;

❑ Traces with more than two instructions have a higher frequency of loads;

Results' Analisys

Analisys of the Results of DTM*upd* and DTM*m*



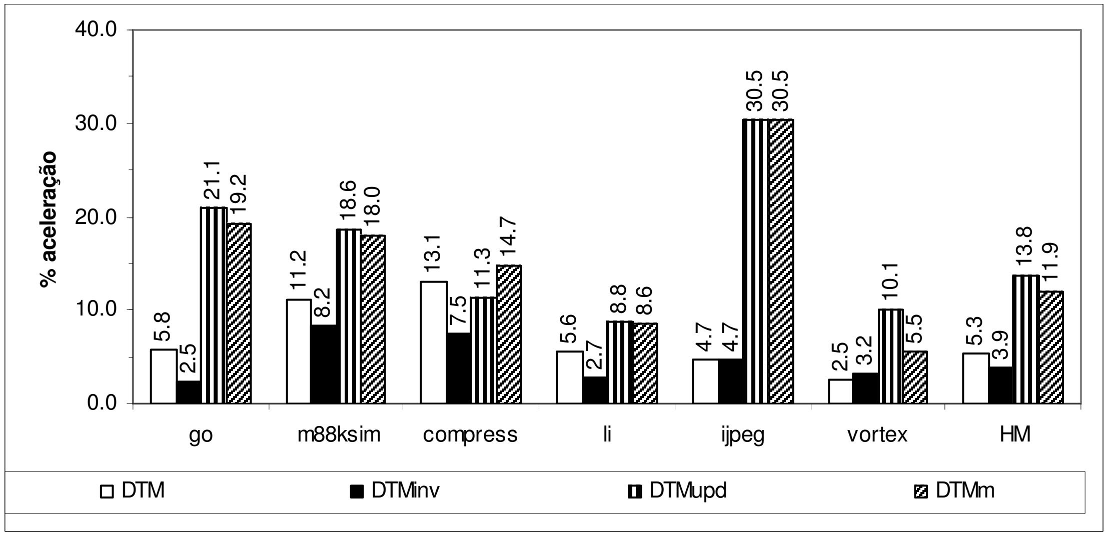


	go	m88ksim	compress	li	ijpeg	vortex	AM
trace	2.6	1.2	13.1	3.0	0.0	0.5	3.4
inst reuse	97.4	98.8	86.9	97.0	100.0	99.5	96.6

- The number of traces served by reused load instructions showed an average result of 30% and 23% for DTMupd and DTMm respectively;
- The reuse of load instructions represents on average 97% of the reuses of this type of instruction;

Results' Analisys

Results Obtained with the Reuse of Instructions Chain and Dependent Traces in the same Cycle



❑ DTMupd and DTMm presented results of 7.2% and 6.5% respectively of harmonic mean over the original implementation of these mechanisms;

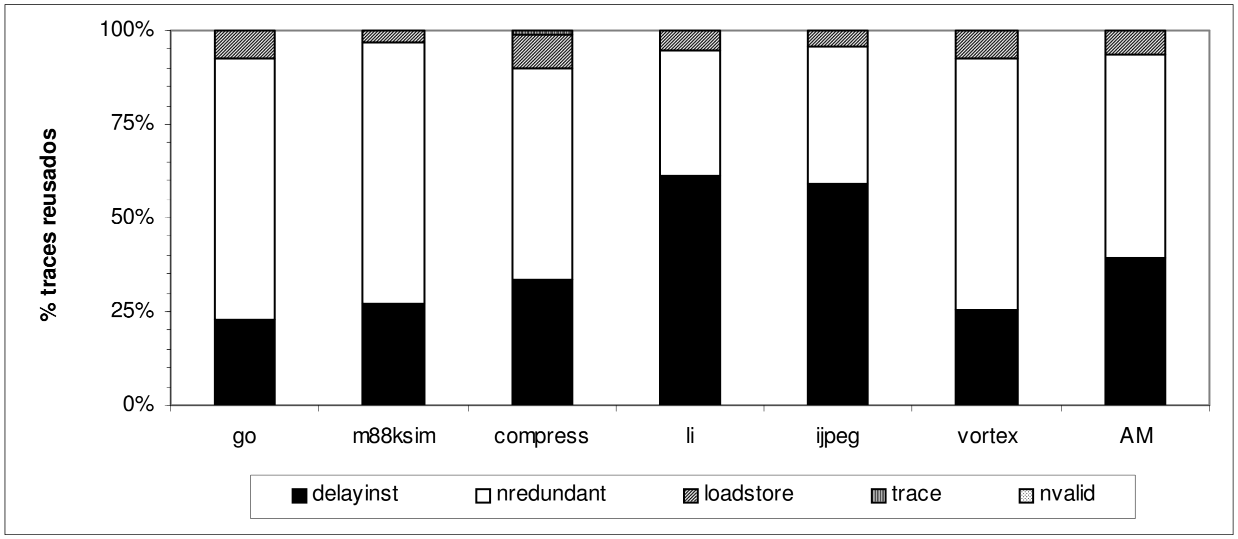
- Excessive number of instructions occupying delay-slot may have excessively fragmented the traces;

- Traces that have a subset of the input context provided by dashes or redundant instructions dispatched in the same cycle;

Results' Analisys

Motivation of Not Formed Traces Finalizartion

	delayinst	nredundant	loadstore	trace	nvalid
go	2298042	7047053	740022	6131	1
m88ksim	3009151	7726553	374041	6974	9
compress	3910282	6593024	1035892	139068	4
li	6494119	3513903	582683	760	4
ijpeg	8079268	5007030	589384	0	3
vortex	2556571	6680095	768659	1419	837
AM	4391239	6094610	681780	25725	143



- ❑ 37% of unformed traces were completed by instructions present in delay-slot;
- ❑ The compress presented a considerable number of unformed traces that are completed by redundant traces;

Content

Conclusions

- The inclusion of memory access instructions in the DTM provides an increase of 4.1% acceleration in harmonic mean;
- The sharing of the fields destined to the output context with the addresses and values of memory instructions increases the frequency of traces finalized by overflow of the output context, besides the inclusion of store instructions do not bring significant gains to the mechanism;
- The experiments demonstrated that the separation of the load instructions brings simplifications to the mechanism, reduces the cost of implementation and maintains performance comparable to that obtained with **DTMupd**;
- The implementation of mechanisms capable of dispatching traces and redundant instructions with data dependencies in the same cycle provides the mechanism with an increase of 7.2% in the hamonic mean of the accelerations;

?