DTM*m*

Memorização Dinâmica de *Traces*
com Reuso de Valores de Instruções
de Acesso à Memória

por Luiz Marcio Viana

Conteúdo

- **Objetivo**
- Memorização Dinamica de *Traces* – DTM
- Adicionando Instruções de Acesso à Memória ao DTM
- O Mecanismo DTM*m*
- Ambiente de Simulação
- Análise dos Resultados
- Conclusão e Trabalhos Futuros

Objetivo Inicial

Extender a funcionalidade da técnica de reuso de *traces* de instruções denominada *Dynamic Trace Memoization* - DTM, adicionando reuso de valores de instruções de acesso à memória

Etapa #1	Etapa #2	Etapa #3	Etapa #4
SuperSIM	Implementação do DTM	Adição de reuso de valores de instruções de acesso à memória no DTM	DTM*m*

Conteúdo

- Objetivo
- **Memorização Dinamica de *Traces* – DTM**
- Adicionando Instruções de Acesso à Memória ao DTM
- O Mecanismo DTM*m*
- Ambiente de Simulação
- Análise dos Resultados
- Conclusão e Trabalhos Futuros

Memorização Dinâmica de *Traces* - DTM

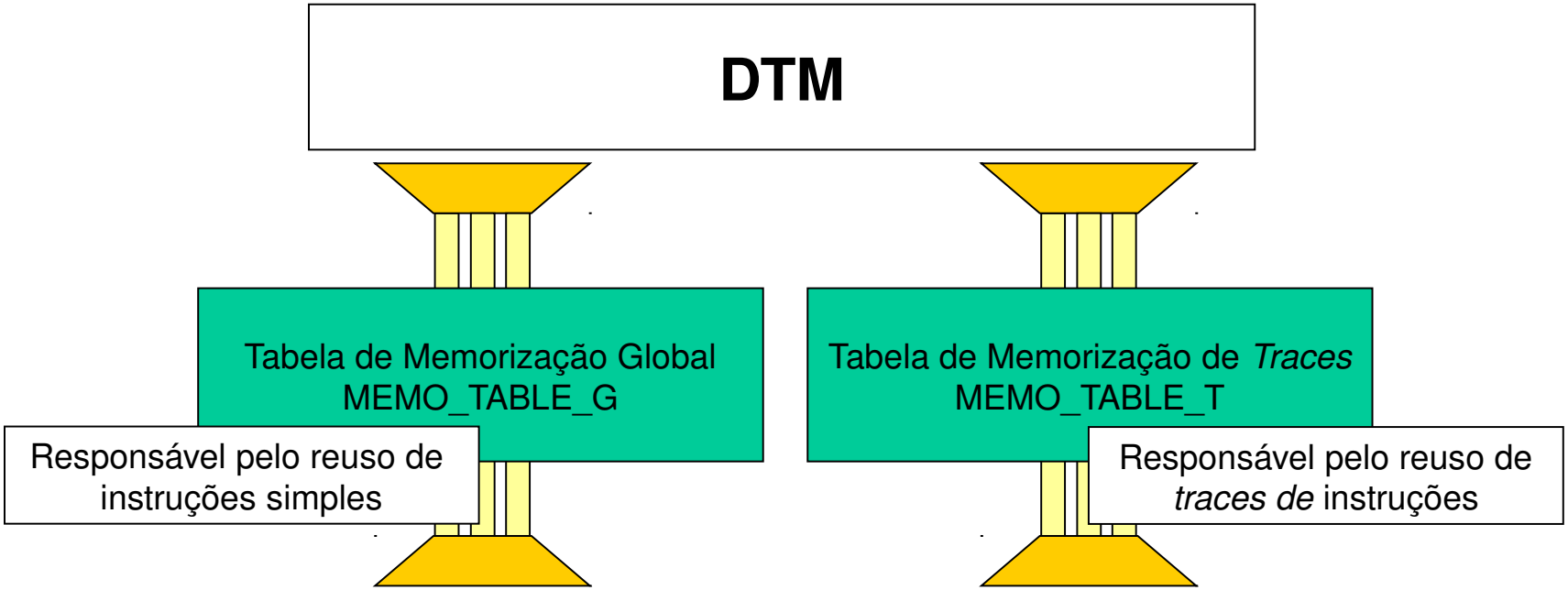
Técnica que explora computação redundante através do reuso de *traces* de instruções.

Memorização Dinâmica de *Traces* - DTM

Definições Iniciais

- **Trace de Instruções -** sequencia dinâmica de instruções de um programa;
- **Trace Redundante –** trace formado por instruções redundantes, isto é, instruções que estão sendo executadas com os mesmos valores de entrada;
- **Instruções Válidas -** instruções pertencentes a um subconjunto de instruções do processador que são aceitas pelo DTM;
  - Neste subconjunto não estão incluídas instruções de acesso à memória, chamadas ao sistema operacional e instruções de ponto flutuante;
- **Contexto de Entrada -** conjunto formado pelos registradores de entrada, cujos valores são fornecidos por instruções externas ao *trace*, e por seus respectivos valores;
- **Contexto de Saída -** conjunto formado pelos registradores que foram modificados pelo *trace* e por seus respectivos valores;

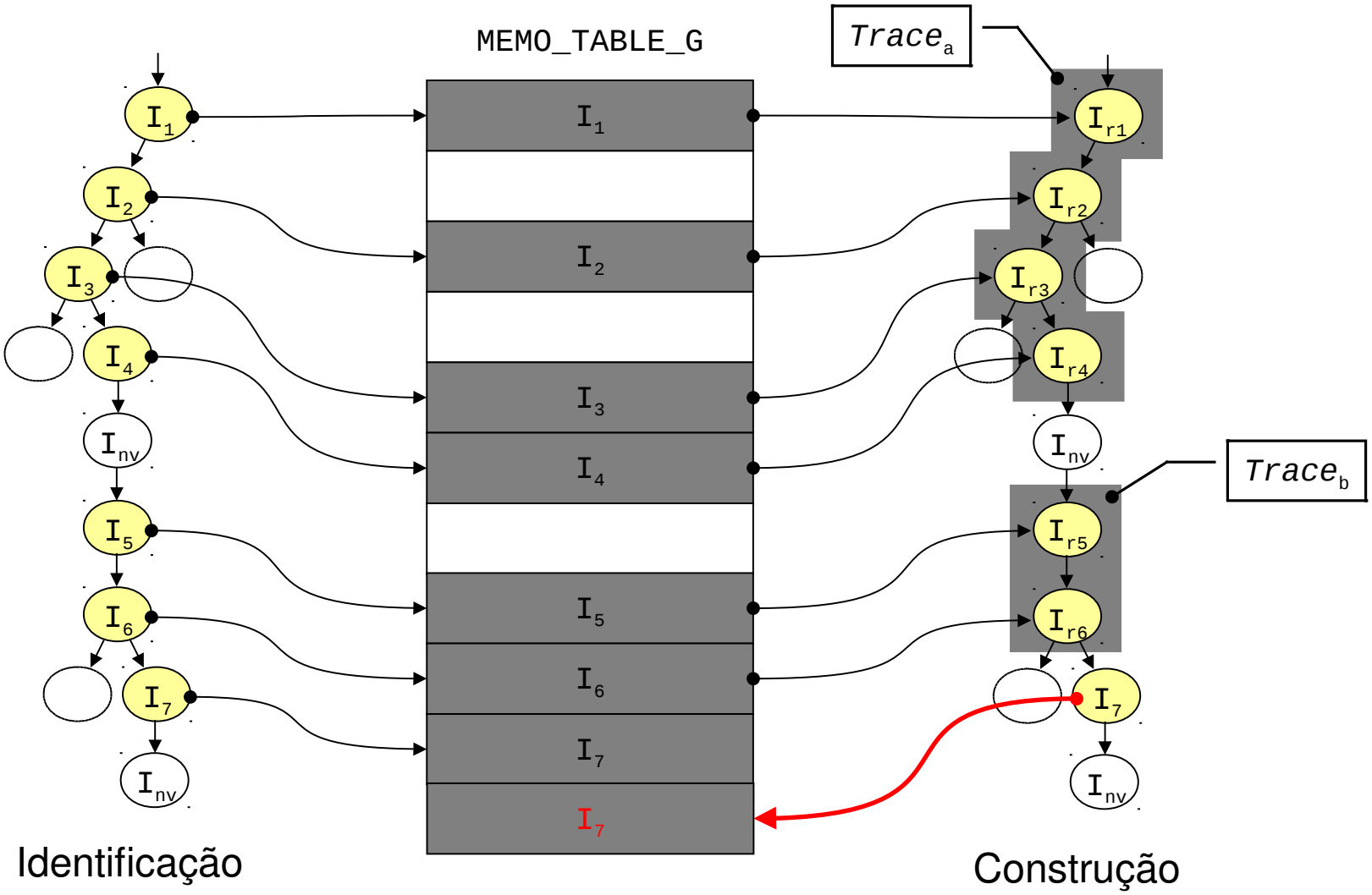
Memorização Dinâmica de *Traces* - DTM



- Implementa dois níveis de reuso – reuso de instruções simples e reuso de *traces*
  - Menor volatilidade da tabela de reuso de *traces*
  - Pré-qualificação das instruções que compõem os *traces*

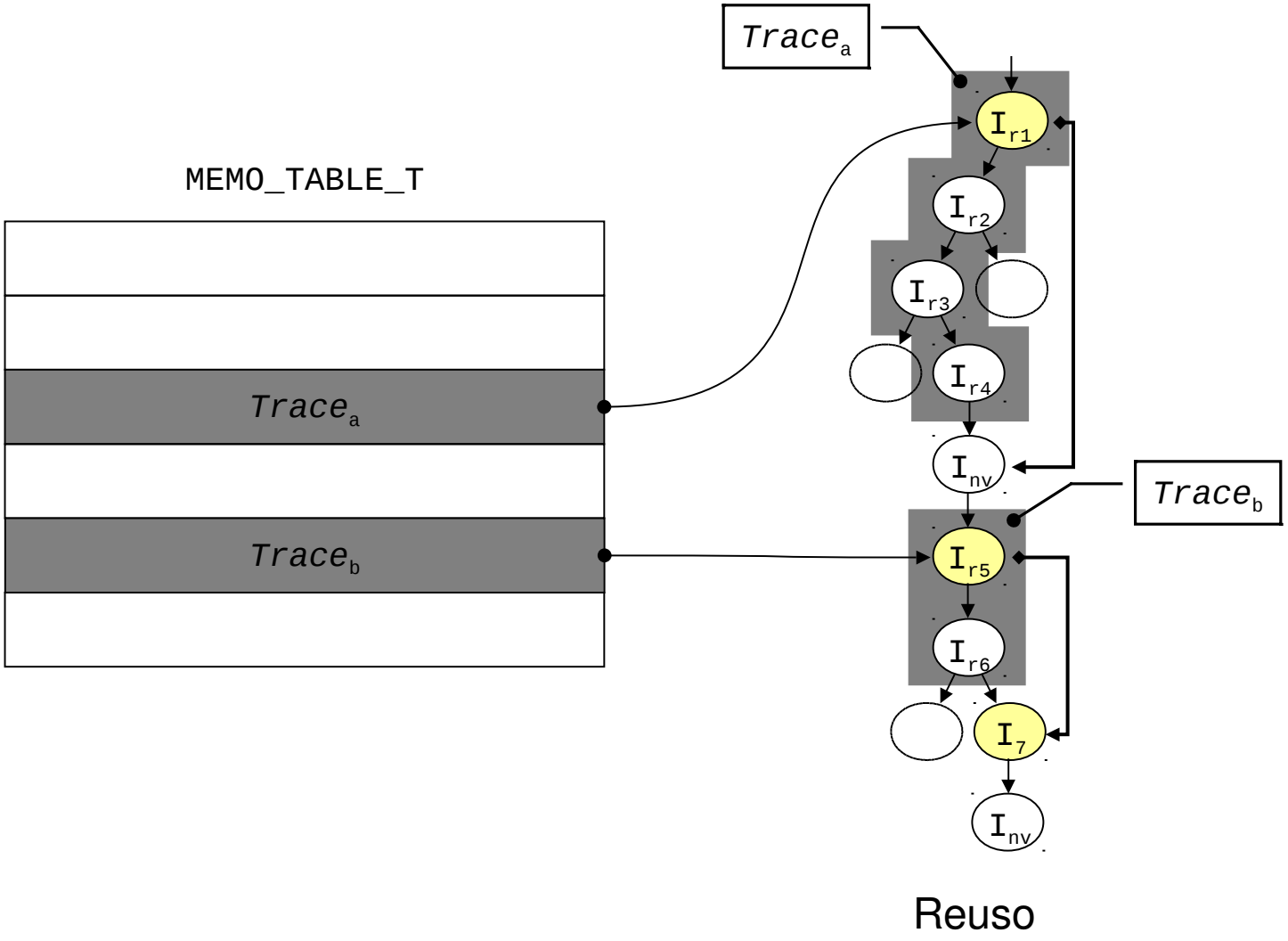
Memorização Dinâmica de *Traces* - DTM

Processo de Reuso de *Traces* (Identificação, Construção e Reuso)



Memorização Dinâmica de *Traces* - DTM

Processo de Reuso de *Traces*



Memorização Dinâmica de *Traces* - DTM

Formato das Entradas da Tabela de Memoriação Global – MEMO_TABLE_G

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

pc – endereço da instruçao

jmp – sinalizador de instrução de desvio incondicional

brc – sinalizador de instrução de desvio condicional

btaken – sinalizador de desvio **tomado** ou **nao_tomado**

sv1 – valor do operando fonte

sv2 – valor do operando fonte

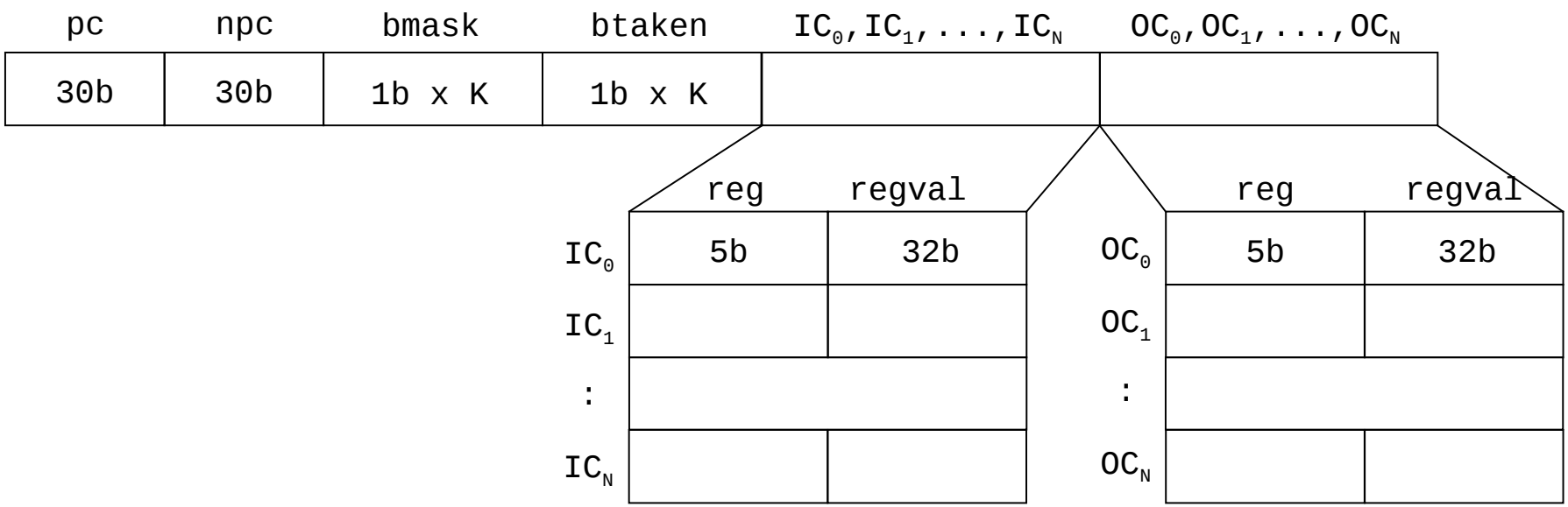
res/npc – resultado do oparacao/destino

```
        :
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

Memorização Dinâmica de *Traces* - DTM

Formato das Entradas da Tabela de Memoriação de *Traces* – MEMO_TABLE_T



pc – endereço da instruçao

npc – endereço da instrução seguinte ao *trace*

bmask – sinalizadores de instruções de desvios

btaken – sinalizadores de desvios **tomado**/**não tomado**

$IC_0, IC_1, \ldots, IC_N$ – contexto de entrada do *trace*

reg – endereço do registrador fonte

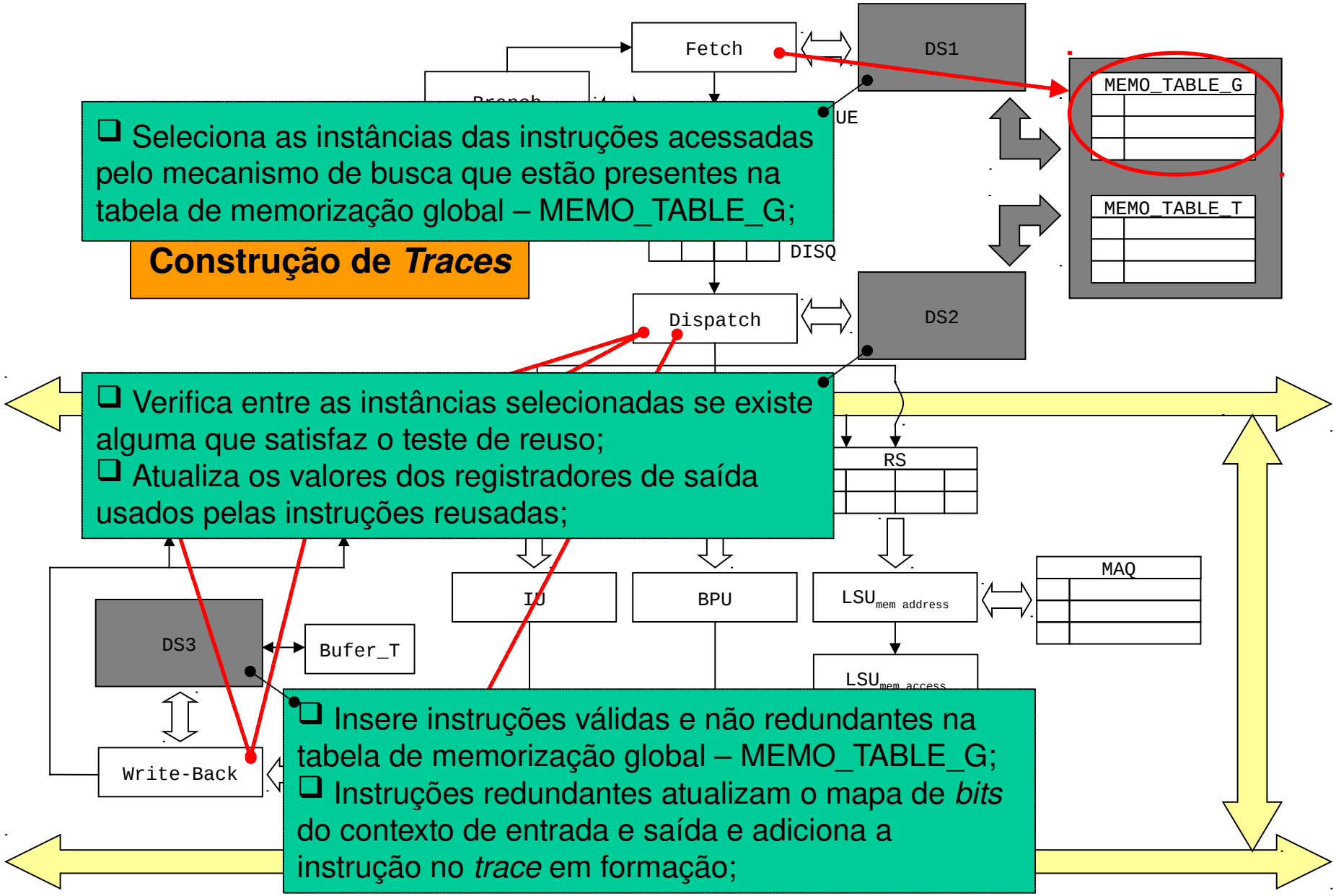
regval – valor do registrador fonte

$OC_0, OC_1, \ldots, OC_N$ – contexto de saida do *trace*

reg – endereço do registrador fonte

regval – valor do registrador fonte

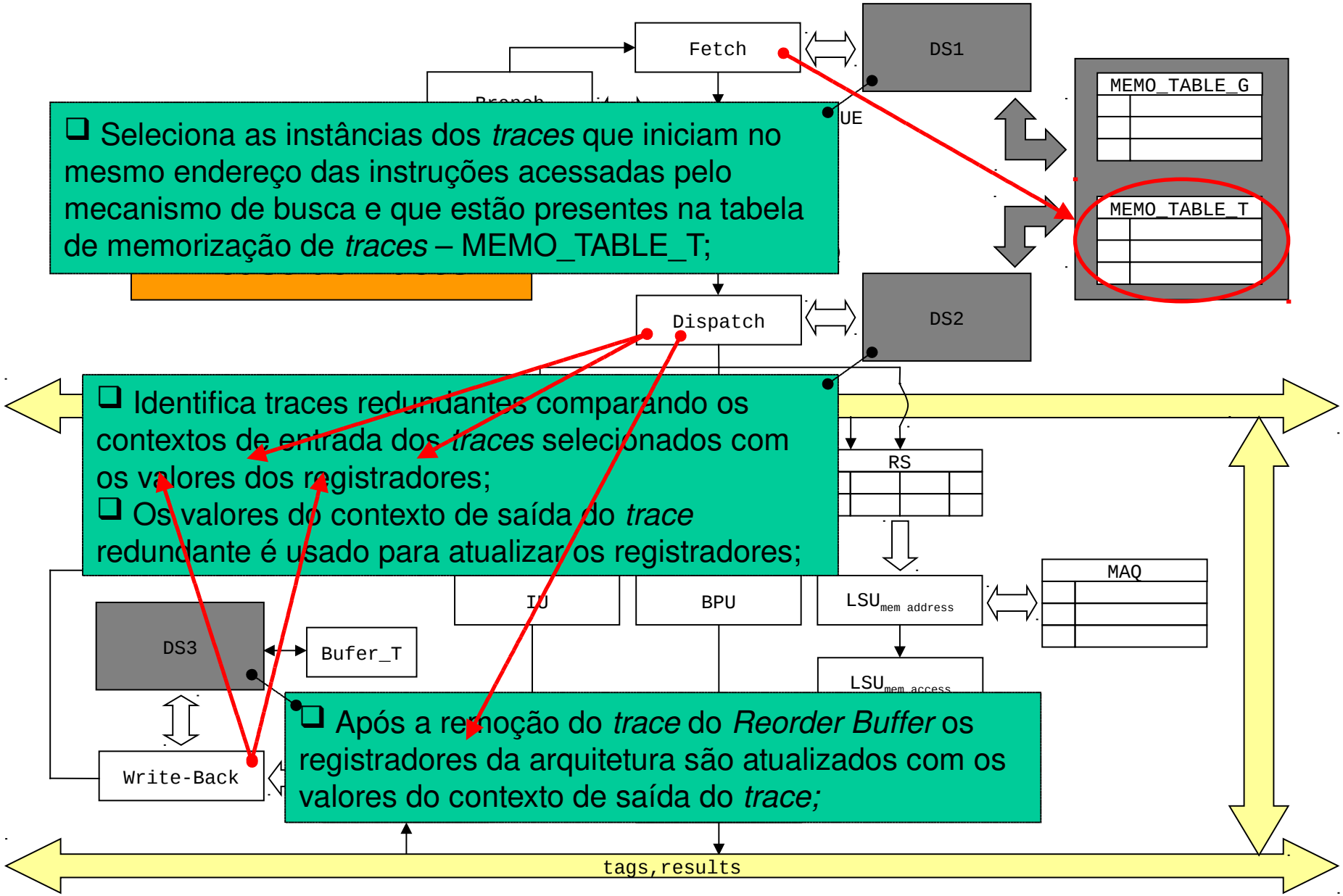
Memorização Dinâmica de *Traces* - DTM



Construção de *Traces*

- Seleciona as instâncias das instruções acessadas pelo mecanismo de busca que estão presentes na tabela de memorização global – MEMO_TABLE_G;
- Verifica entre as instâncias selecionadas se existe alguma que satisfaz o teste de reuso;
- Atualiza os valores dos registradores de saída usados pelas instruções reusadas;
- Insere instruções válidas e não redundantes na tabela de memorização global – MEMO_TABLE_G;
- Instruções redundantes atualizam o mapa de *bits* do contexto de entrada e saída e adiciona a instrução no *trace* em formação;

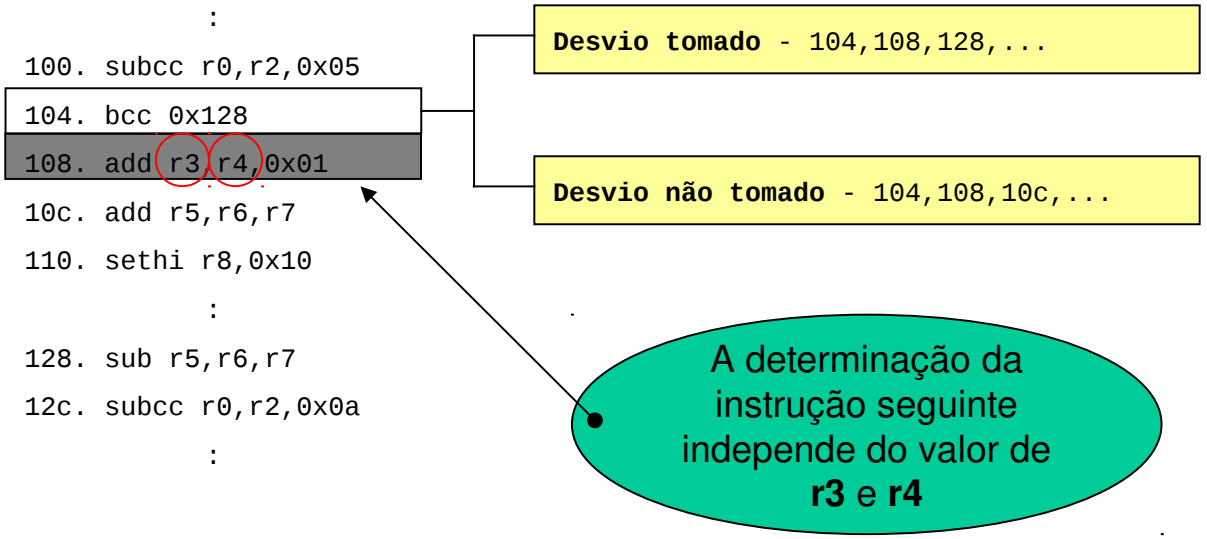
Memorização Dinâmica de *Traces* - DTM



- Seleciona as instâncias dos *traces* que iniciam no mesmo endereço das instruções acessadas pelo mecanismo de busca e que estão presentes na tabela de memorização de *traces* – MEMO_TABLE_T;
- Identifica traces redundantes comparando os contextos de entrada dos *traces* selecionados com os valores dos registradores;
- Os valores do contexto de saída do *trace* redundante é usado para atualizar os registradores;
- Após a remoção do *trace* do *Reorder Buffer* os registradores da arquitetura são atualizados com os valores do contexto de saída do *trace;*

Memorização Dinâmica de *Traces* - DTM

Detalhes de Implementação - *Delay Slot*



- Instruções no *delay slot* não podem ser reusadas pois introduzem ambiguidades no caminho seguido pelo fluxo do programa;
- *Traces* não podem conter instruções no *delay slot* porque podem apresentar ambiguidade no fluxo do programa se a instrução de desvio que sofre o retardo não estiver contida no *trace*;
- Instruções de *delay slot* são necessariamente incluídas no *trace* em formação finalizando ele, porém tais instruções não são reusadas;
  - Desvios com *annul bit* habilitado introduzem problemas com o reuso de *traces* que contém instruções de desvios que estão sendo executados especulativamente;

Memorização Dinâmica de *Traces* - DTM

Detalhes de Implementação – Janela de Registradores

incluir instruções de save e restore
x
não incluir instruções de save e restore

incluindo save e restore	**não incluindo save e restore**
Nesta abordagem atribuimos a cada instrução ou *trace*, um identificador único para cada registrador da máquina utilizado pelos contextos de entrada e saída.	Nesta abordagem armazenamos para os operandos de entrada e saída da instrução ou *trace* o endereço relativo de cada registrador dentro da janela corrente.
❑ Simplicidade de projeto; ❑ Permite a inclusão de instruções de save e restore nos *traces*; ❑ Pode aumentar significativamente o tamanho do contexto de entrada e saída dos *traces*; ❑ Não explora a redundância encontrada em procedimentos recursivos;	❑ Favorece o reuso de *tr* instruções redundantes d funções recursivas; ❑ Não permite a inclusão de instruções de save e restore nos *traces;* ❑ Maior complexidade de projeto;

Conteúdo

- Objetivo
- Memorização Dinamica de *Traces* – DTM
- **Adicionando Instruções de Acesso à Memória ao DTM**
- O Mecanismo DTM*m*
- Ambiente de Simulação
- Análise dos Resultados
- Conclusão e Trabalhos Futuros

Adicionando Instruções de Acesso à Memória

Motivações

- Aumentar o tamanho médio dos *traces*;
  - Experimentos realizados mostram que 13% dos *traces* são finalizados por instruções de acesso à memória e 36% deles por instruções não redundantes;
  - Instruções de acesso à memória apresentam grande localidade de valores;
- Aumentar a disponibilidades dos operandos fontes;
  - Uso frequente de instruções de leitura no inicio de uma computação;
- Instruções de acesso à memória podem sofrer latência significativa nos acessos que não são capturados pela *cache*;

DTM*inv*

DTM*upd*

Adicionando Instruções de Acesso à Memória

Modificações Realizadas nas Entradas da Tabela de Memoriação Global – MEMO_TABLE_G

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

pc – endereço da instruçao

jmp – sinalizador de instrução de desvio incondicional

brc – sinalizador de instrução de desvio condicional

btaken – sinalizador de desvio **tomado** ou **nao_tomado**

sv1 – valor do operando fonte

sv2 – valor do operando fonte

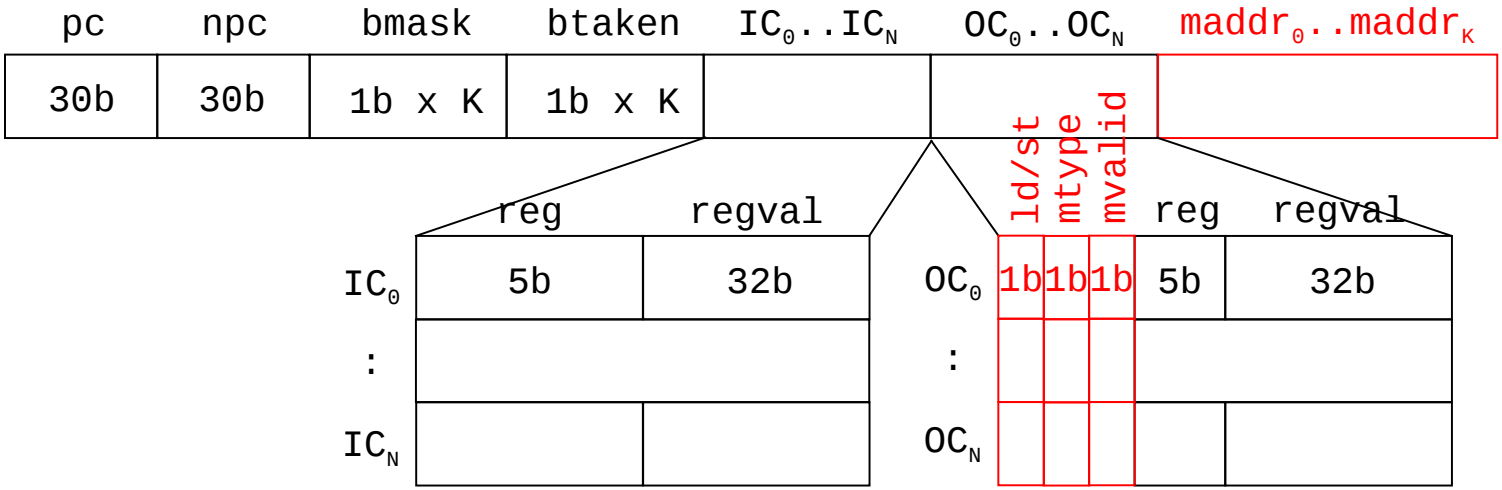
maddr – armazena o endereço de acesso à memória

mem valid – sinalizador de valor de leitura de memória válido

res/npc – resultado da oparacao/endereço alvo de desvio/valor à ser lido e armazenado em memória

Adicionando Instruções de Acesso à Memória

Modificações Realizadas nas Entradas da Tabela de Memoriação de *Traces* – MEMO_TABLE_T



pc – endereço da instruçao

npc – endereço da instrução seguinte ao *trace*

bmask – sinalizadores de instruções de desvios

btaken – sinalizadores de desvios **tomado**/**não tomado**

$IC_0, \ldots, IC_N$ – contexto de entrada do *trace*

reg – endereço do registrador fonte

regval – valor do registrador fonte

$OC_0, \ldots, OC_N$ – contexto de saida do *trace*

reg – endereço do registrador fonte

regval – valor do registrador fonte

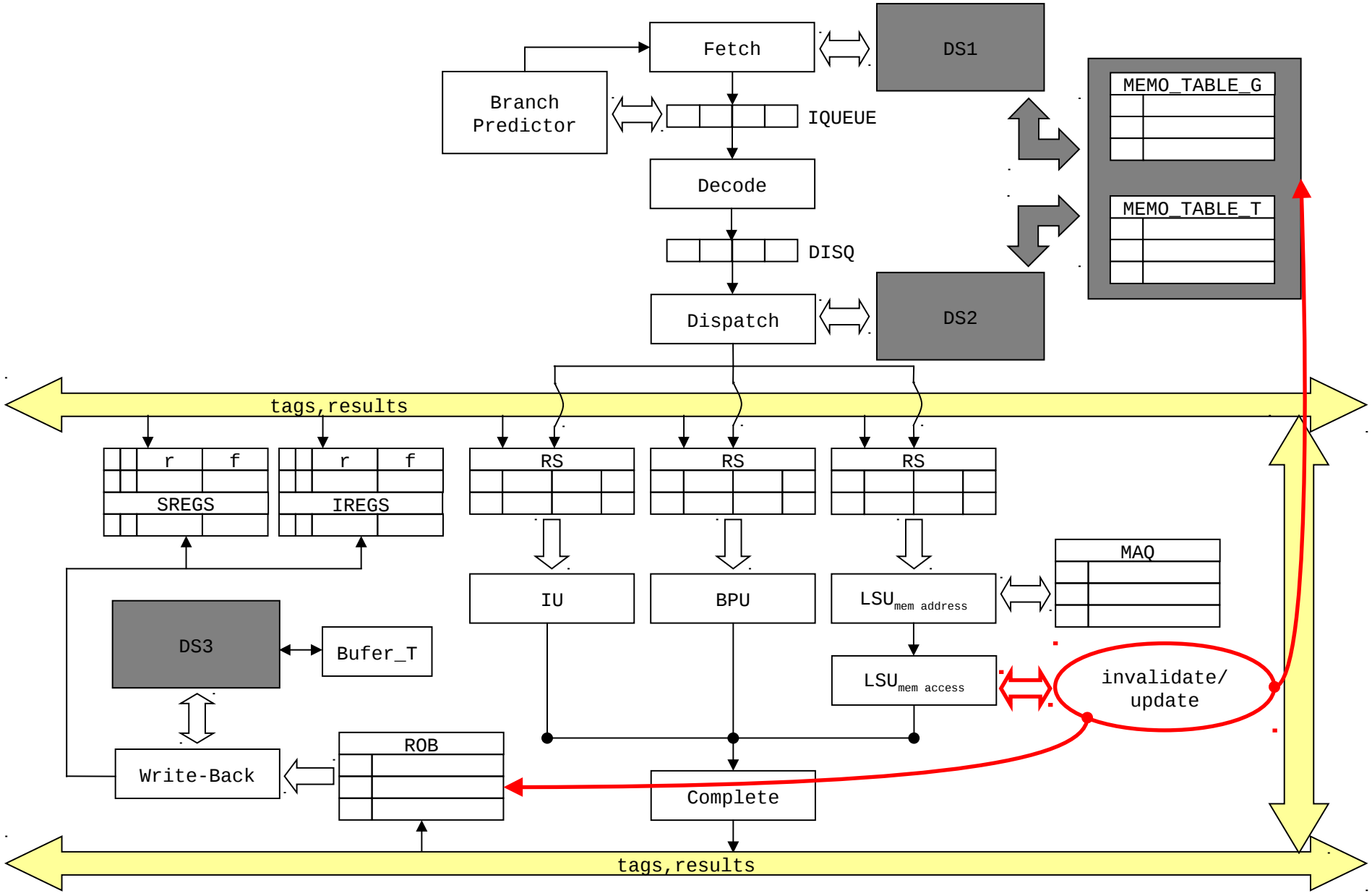
ld/st – sinalizador de instrução de acesso à memória

mtype – sinalizador de tipo de acesso à memória (load/store)

mvalid – sinalizador de valor de memória válido para reuso

$maddr_0, \ldots, maddr_K$ – endereço de acesso à memória

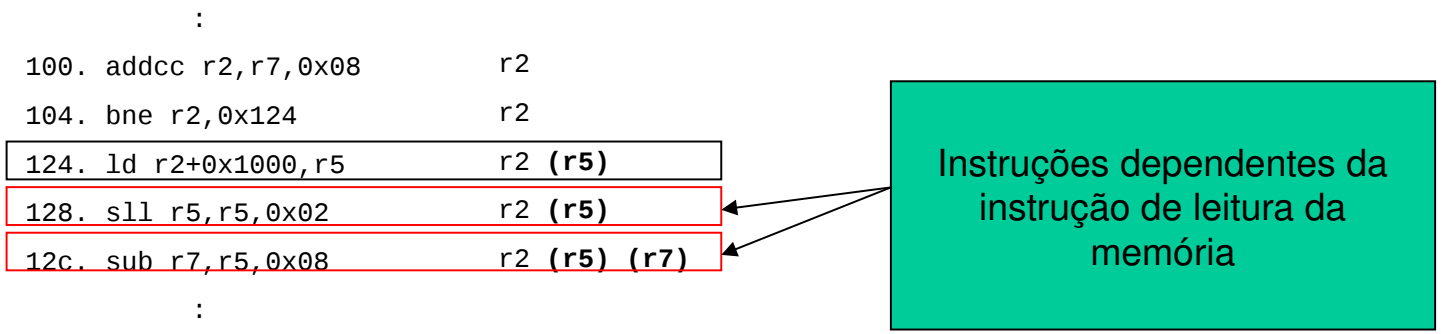
Adicionando Instruções de Acesso à Memória



Adicionando Instruções de Acesso à Memória

Detalhes de Implementação

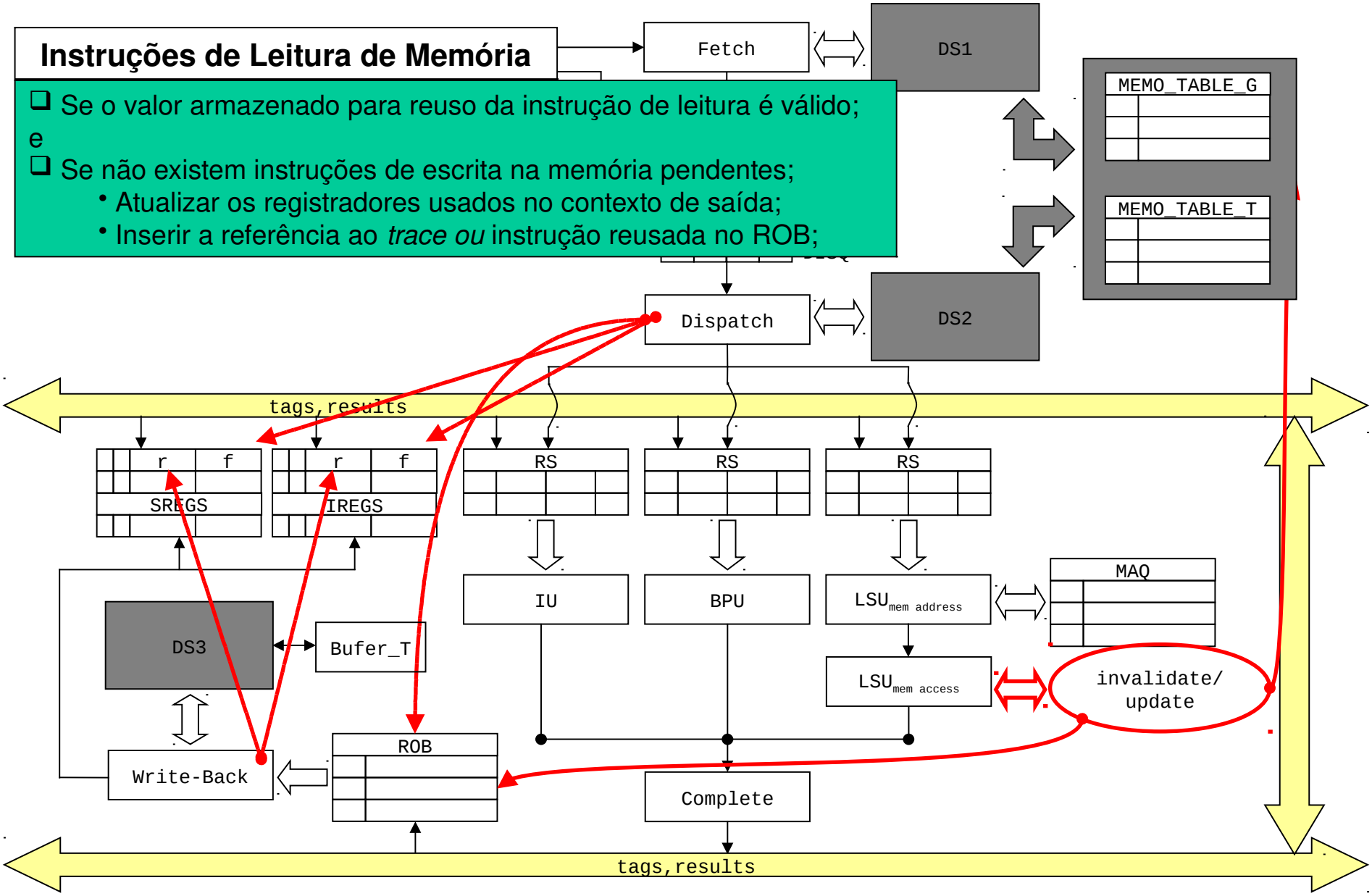
- Instruções de leitura são necessariamente as últimas instruções do *trace*;
  - Instruções de leitura podem ter seus valores modificados por instruções externas ao *trace*;
  - A inclusão de instruções dependentes de uma instrução de leitura da memória em um mesmo *trace* pode levar a invalidação total do *trace* se o conteúdo da memória for modificado;
- Apenas uma instrução de leitura na memória é permitida por *trace*;
  - Evita a necessidade de atualizar e invalidar o *Buffer_T* na ocorrencia de um acesso de escrita na memória em um endereço usado por uma instrução de leitura presente no *trace* em formação;
  - Reduz o custo de implementação de um mecanismo de invalidação e antecipação de valores para um número grandes de campos com valores de acesso à memória;



Adicionando Instruções de Acesso à Memória

Instruções de Leitura de Memória

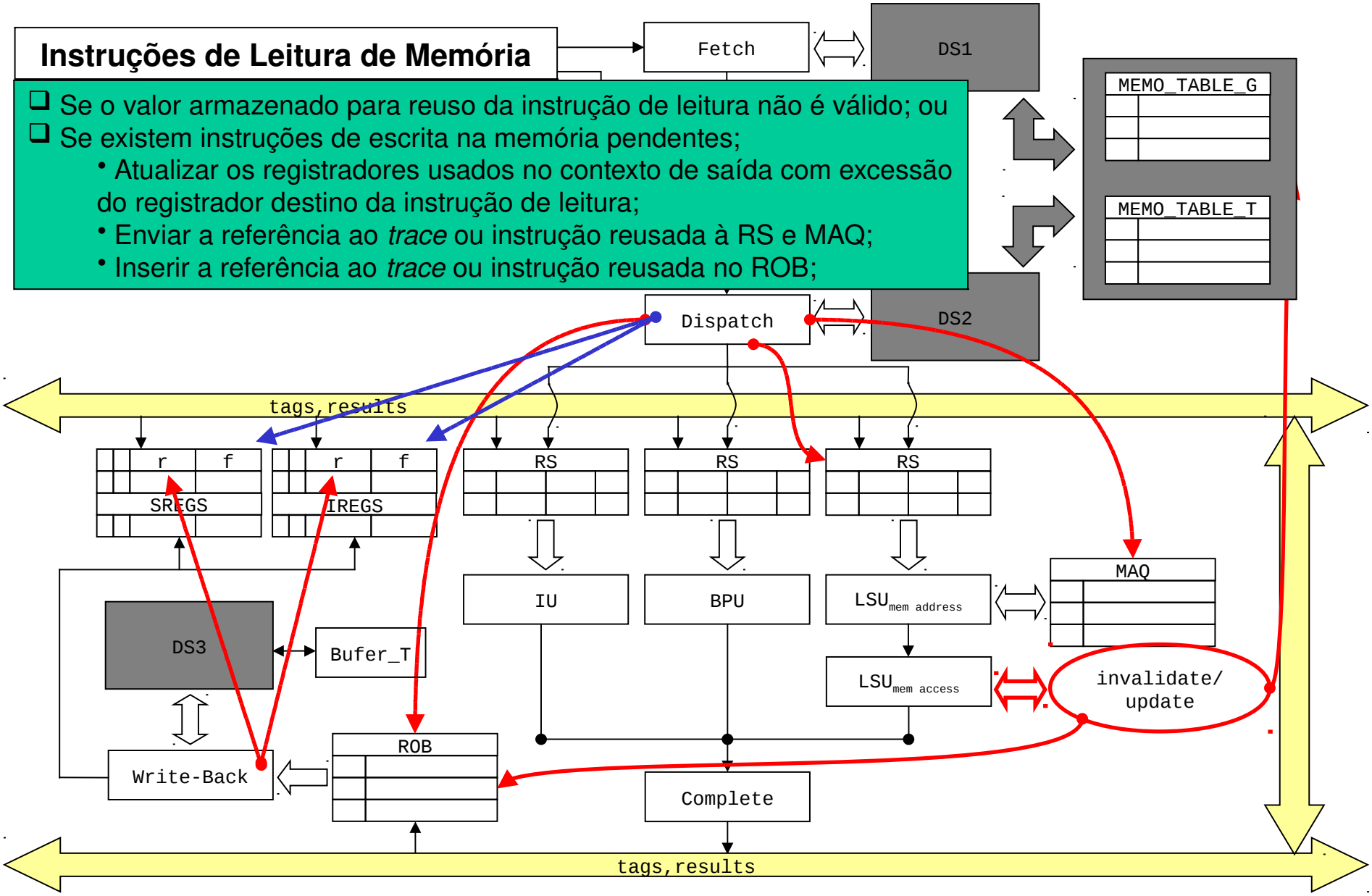
- Se o valor armazenado para reuso da instrução de leitura é válido; e
- Se não existem instruções de escrita na memória pendentes;
  - Atualizar os registradores usados no contexto de saída;
  - Inserir a referência ao *trace ou* instrução reusada no ROB;



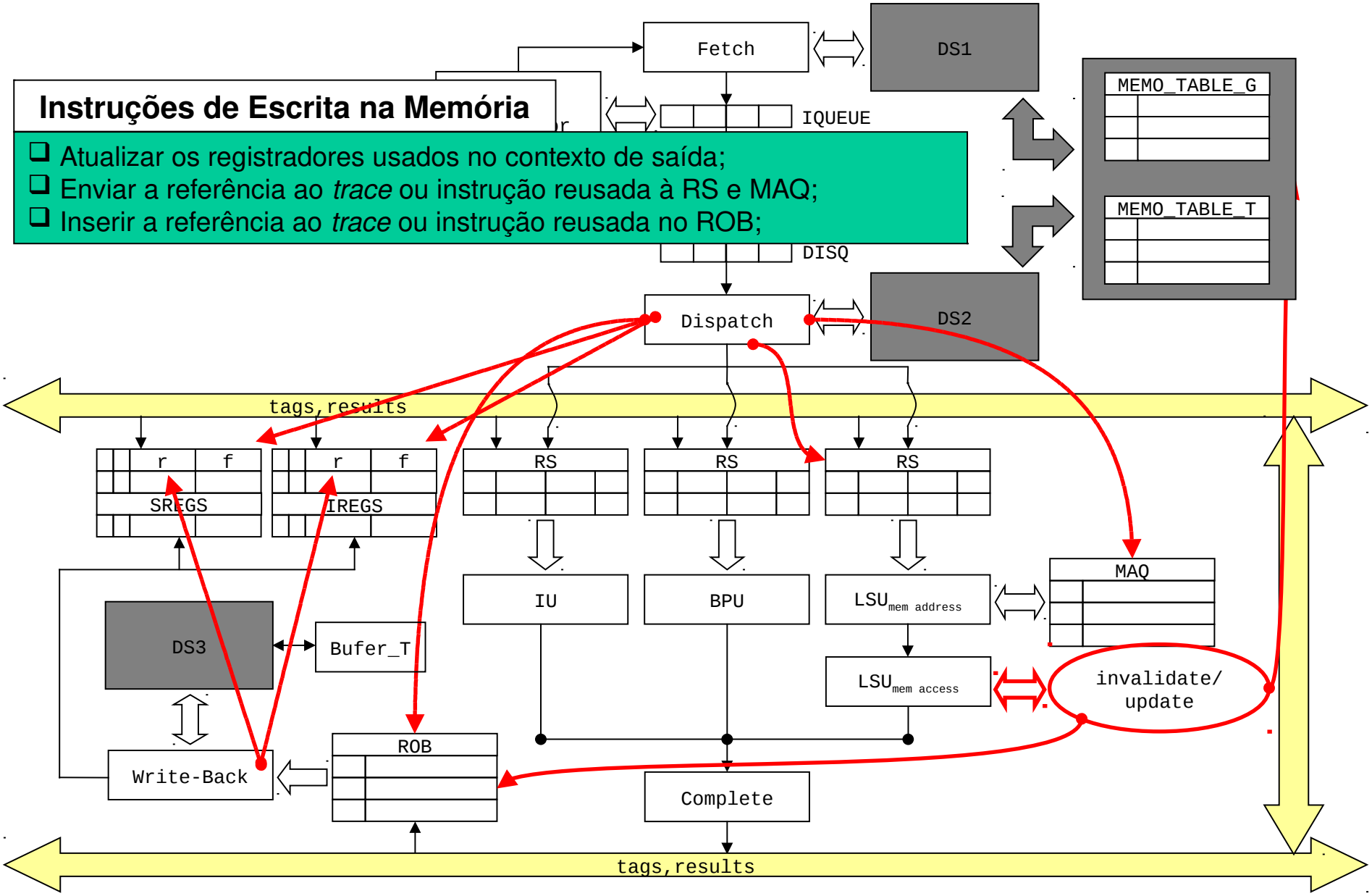
Adicionando Instruções de Acesso à Memória

Instruções de Leitura de Memória

- Se o valor armazenado para reuso da instrução de leitura não é válido; ou
- Se existem instruções de escrita na memória pendentes;
  - Atualizar os registradores usados no contexto de saída com excessão do registrador destino da instrução de leitura;
  - Enviar a referência ao *trace* ou instrução reusada à RS e MAQ;
  - Inserir a referência ao *trace* ou instrução reusada no ROB;



Adicionando Instruções de Acesso à Memória



Conteúdo

- Objetivo
- Memorização Dinamica de *Traces* – DTM
- Adicionando Instruções de Acesso à Memória ao DTM
- **O Mecanismo DTM*m***
- Ambiente de Simulação
- Análise dos Resultados
- Conclusão e Trabalhos Futuros

O Mecanismo DTM*m*

Objetivos Iniciais

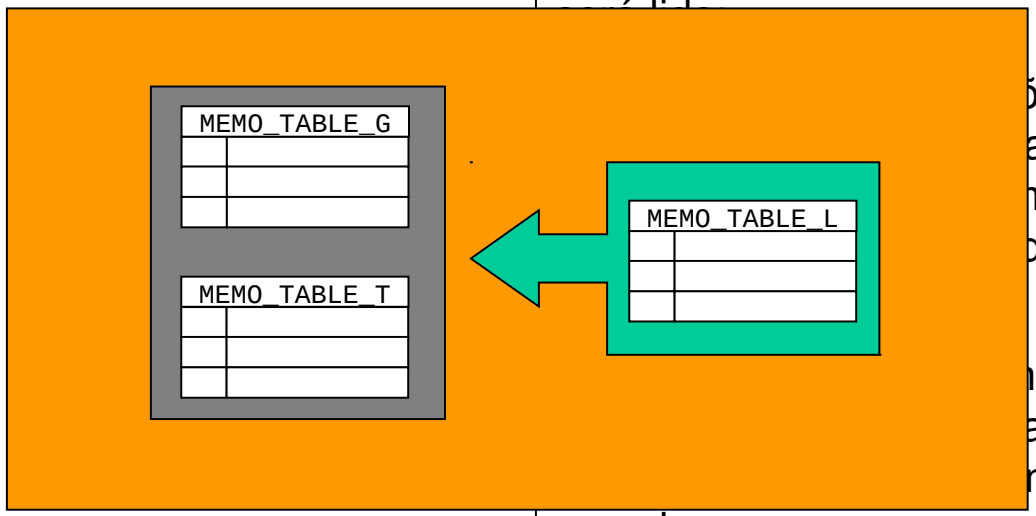
- Reduzir o custo com a inclusão de mecanismos de invalidação/antecipação de valores;
- Evitar o aumento do tamanho das entradas da tabela de memorização de *traces*;

Instruções de Escrita na Memória

- Quando uma instrução de escrita na memória possui os mesmos valores de entrada podemos assegurar que o endereço de acesso á memória e o valor a ser armazenado são os mesmo;

Instruções de Leitura da Memória

- Quando uma instrução de leitura da memória possuí os mesmos valores de entrada podemos assegurar que o endereço de acesso à memória armazenado é o mesmo, porém não podemos afirmar nada quanto ao valor que



ões de leitura
az a necessidade
mos de
de valores nas

nos de reuso de
amente nas
mecanismo mais
complexo;

O Mecanismo DTM*m*

Tabela de Memoriação de Instruções de Leitura da Memória – MEMO_TABLE_L

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

pc – endereço da instruçao

sv1 – valor do operando fonte

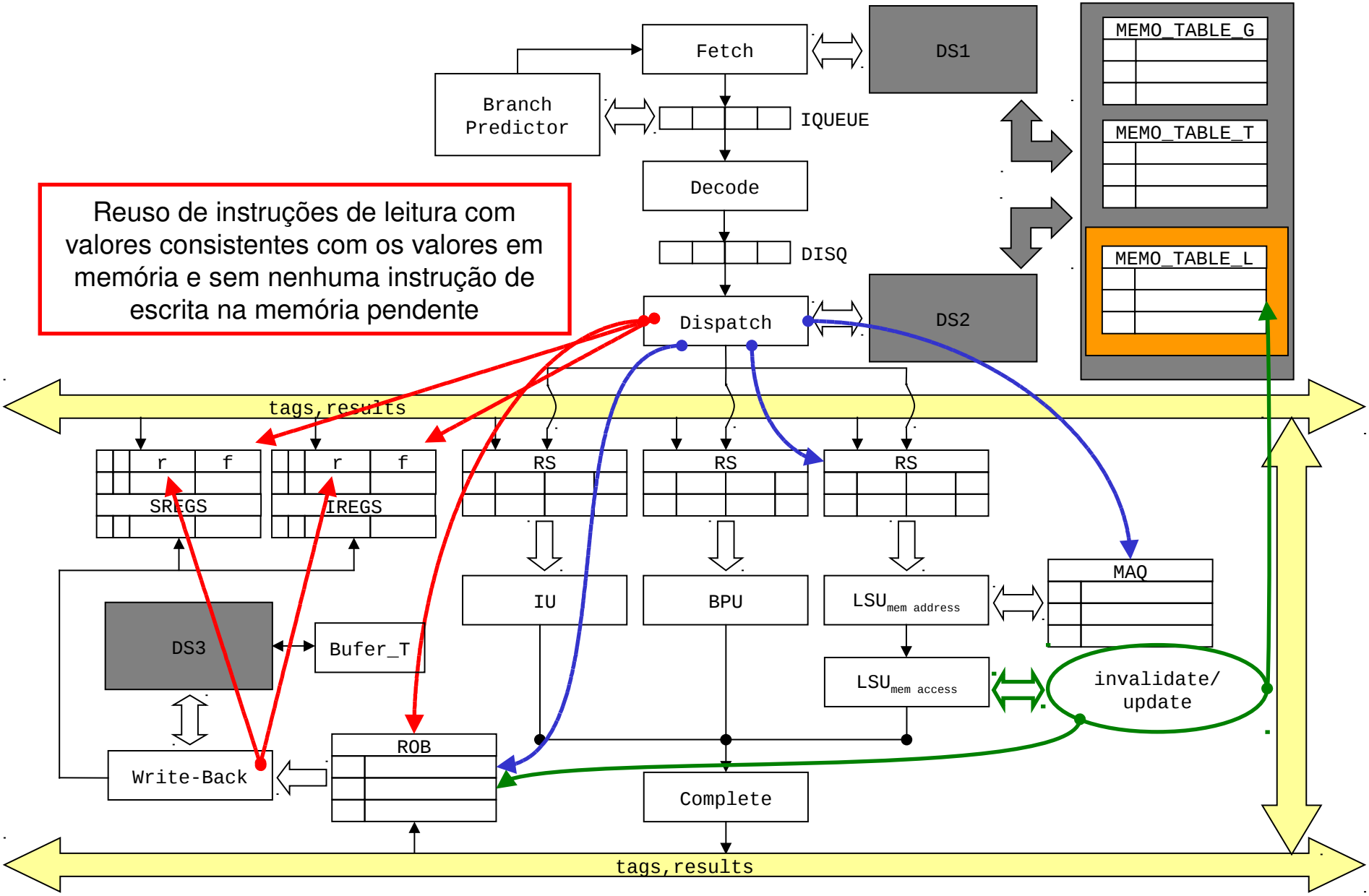
sv2 – valor do operando fonte

maddr – armazena o endereço de acesso à memória

mem valid – sinalizador de valor de leitura de memória válido

res – resultado da oparacao

Adicionando Instruções de Acesso à Memória



Conteúdo

- Objetivo
- Memorização Dinamica de *Traces* – DTM
- Adicionando Instruções de Acesso à Memória ao DTM
- O Mecanismo DTM*m*
- **Ambiente de Simulação**
- Análise dos Resultados
- Conclusão e Trabalhos Futuros

Ambiente de Simulação - SuperSIM

Recursos Implementados

- Executa o programas compilados para o processador **Sparc v7**;
- Previsão de desvios, execução especulativa, registradores futuros, estações de reserva, *buffer* de reordenação;
- Execução passo à passo, execução completa do programa e execução de um número determinado de instruções;
- Relatórios para acompanhamento do programa em execução, memória principal, BTB, fila de instruções, fila de despacho, registradores, estações de reserva, unidades de execução, fila de acesso à memória, *buffer* de reordenação, tabelas de memorização e estastísticas;
- Executa em plataformas SUN Solaris 7 e INTEL Linux;

Limitações do Simulador

- Não implementa registradores de ponto-flutuante e não suporta operações de leitura e escrita na memória usando registradores de ponto-flutuante;
- Não implementa unidade de ponto-flutuante;
- Não implementa controle de *overflow* e *underflow* da janela de registradores;
- Implementa parcialmente as cadeias de duplos desvios – *delayed control-transfer couples* definidas na arquitetura **Sparc v7**;

Conteúdo

- Objetivo
- Memorização Dinamica de *Traces* – DTM
- Adicionando Instruções de Acesso à Memória ao DTM
- O Mecanismo DTM*m*
- Ambiente de Simulação
- **Análise dos Resultados**
- Conclusão e Trabalhos Futuros

Análise dos Resultados

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)
vortex	vortex.in (ref)	100.000.000 (não finalizado)

- Todos os programas foram compilados usando gcc-2.5.2 com otimização -O, opção de biblioteca estática -static e uso de janela de registradores desabilitado -mflat;

Análise dos Resultados

	Simplescale Tool Set	**SuperSIM**
busca de instruções	4 instruções por ciclo. Apenas um desvio tomado por ciclo. Pode ultrapassar a fronteira da linha de *cache*.	4 instruções por ciclo. Apenas um desvio tomado por ciclo. Não há limite imposto pela *cache*.
cache de instruções	16 Kb, associativa - 2 por conjunto, 32 bytes por linha, latência de 6 ciclos para *miss* no *cache* L1 e 20 ciclos para *miss* no *cache* L2.	Não implementado, considera acerto na *cache* de 100%.
preditor de desvios	Bimodal, 2k entradas, pode predizer vários desvios simultaneamente.	Bimodal, 1k entradas, pode predizer vários desvios simultaneamente.
mecanismo de execução especulativa	Execução de até quatro instruções por ciclo fora de ordem, *buffer* de reordenação com 16 entradas e fila de acesso à memória com 8. Loads são executados após serem conhecidos todos os endereços de *stores* precedentes. *Loads* são servidos por *stores* que acessam o mesmo endereço se ambos estiverem na fila de acesso à memória.	Execução de até quatro instruções por ciclo fora de ordem, *buffer* de reordenação com 16 entradas e fila de acesso à memória com 16. Loads são executados após todos os *stores* precedentes terem sido executados. Instruções de *loads* não são servidas por *stores*.
registradores arquiteturais	32 registradores de inteiros, 32 registradores de ponto flutuante, registradores hi, lo e fcc.	Janela de registradores com 520 registradores de inteiros ativos por janela, sendo 8 registradores fixos e 32 x 24 registradores sobrepostos, 2 registradores especiais PSR e Y.
unidades funcionais	4 ULAs de inteiros, 2 unidades *load/store*, 4 *adders* de ponto flutuante, 1 mult/div inteiro, 1 mult/div ponto flutuante.	3 ULAS que resolvem operações com inteiros incluindo multiplicação. 2 unidades de desvios e 2 unidades de *loads/stores*.
latência das unidades funcionais	ULA-inteiros/1, *load/store*/1, mult int/3, int div/20, fp *adders*/2, fp mult/4, fp div/12, fp sqrt/24.	Todas as instruções de inteiro possuem latência 1. Instruções de *load/store* possuem latência 2.
cache de dados	16Kb, associativo-2 por conjunto, 32 bytes por linha, latência de 6 ciclos para *miss* no *cache* L1 e 20 ciclos para *miss* no *cache* L2.	Não implementado, considerado acesso 100% na *cache*.

Análise dos Resultados

	DTMmips	DTMsparc
contexto de entrada	6 entradas	7 entradas
contexto de saída	6 entradas	7 entradas
tamanho max dos traces	16 instruções	Ilimitado
Número max de desvios	10 desvios	10 desvios
heurística	Repetição de instruções simples.	Repetição de instruções simples.
conjunto de seleção	Instruções aritméticas, lógicas, desvios, chamadas e retorno de sub-rotina, e calculo do endereço de acesso à memória.	Instruções aritméticas, lógicas, desvios, chamada e retorno de sub-rotina, e calculo do endereço e valor de acesso à memória.
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.

Análise dos Resultados

Métricas Utilizadas

- Percentual de reuso:

  % reuso = ir / itot ; ir - número de instruções reusadas
  ; itot - número de instruções executadas

- Percentual de aceleração:

  aceleração = IPC_t / IPC_{base} ; IPC_t - instruções executadas por ciclo
  ; com o mecanismo
  ; IPC_{base} - instruções executadas por
  ; ciclo na arquitetura base

- Média aritimética:

  $AM = (\sum_{n=0}^{i} S_i) / n$; n - total de valores computados
  ; S_i - valores computados

- Média harmônica:

  $HM = n (\sum_{n=0}^{i} 1/S_i)^{-1}$; n - total de valores computados
  ; S_i - valores computados

Análise dos Resultados

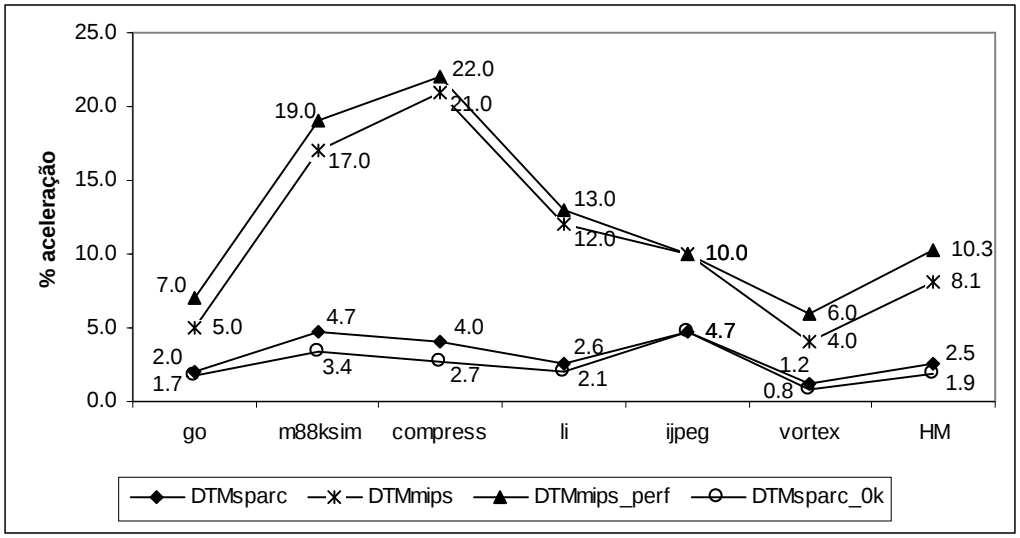
Distribuição das Instruções nos Programas

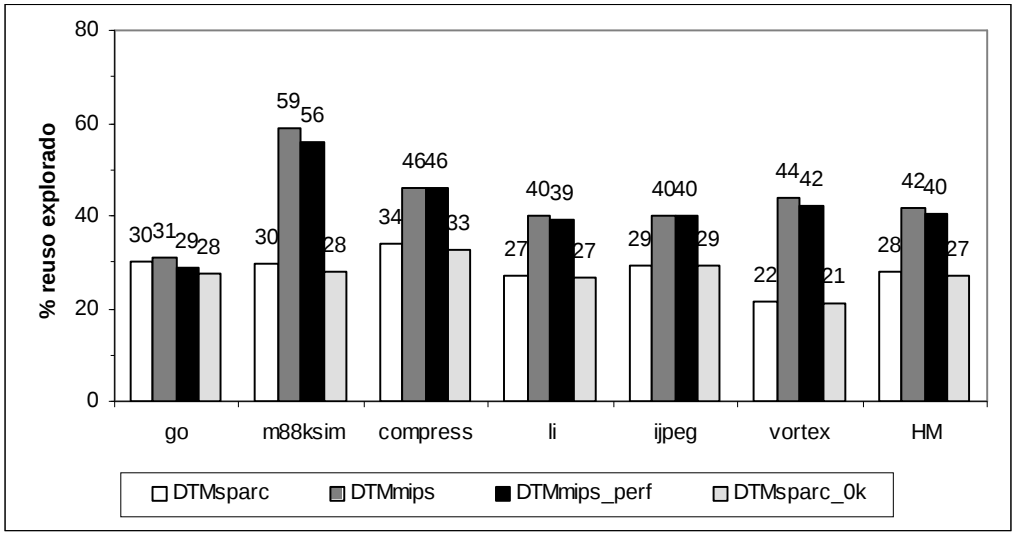
	go	m88ksim	compress	li	ijpeg	vortex	AM
call	431747	733152	939892	1647111	2205754	1119034	1179448
bicc	8509903	12592476	8031639	12185567	11028625	11947490	10715950
jmpl	451801	814256	940050	2002746	2206465	1134519	1258306
ticc	12	65	12	203	546	4224	844
load	22677213	17947340	12084545	25117843	24989669	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	19463133	16766290	20129041	13230038	12370162	16850267
others	96434	2414	4	26	3078	49904	25310

- Instruções de acesso à memória representam em média 34% das instruções executadas;
- A frequencia de instruções de *save* e *restore* é muito baixa e não teve efeito para a compilação aplicada;
- Instruções de leitura da memória representam 22% das instruções executadas;
- SETHI representam cerca de 19% das instruções executadas;

Análise dos Resultados

Resultados da Implementação do DTM no Sparc





- DTM*mips* obteve aceleração média 5.6% acima do resultado apresentado pelo DTM*sparc;*
- DTM*mips* obteve um resultado em média 14% superior ao reuso explorado apresentado pelo DTM*sparc*;
- DTM*sparc* obteve um resultado médio 0.6% acima do resultado obtido pelo DTM*sparc_0k* (sem MEMO_TABLE_T);
- Reuso explorado por DTM*sparc* e DTM*sparc_0k* obtiveram resultados similares;

Análise dos Resultados

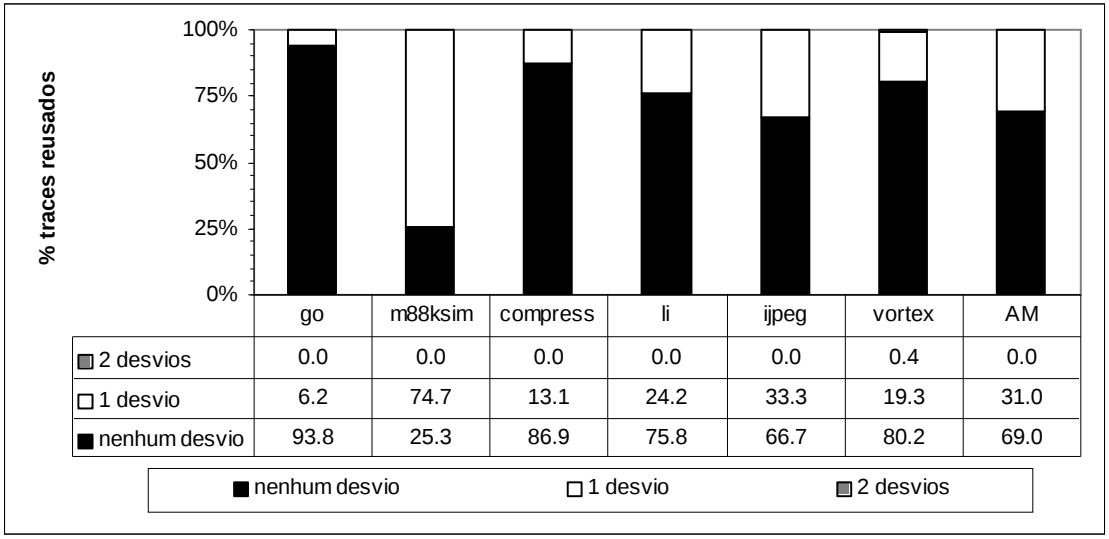
Resultados da Implementação do DTM no Sparc

	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
vortex	0.00	19.68	0.00	75.12	0.04	5.16
AM	0.00	30.86	0.00	56.02	0.10	13.01

- *Traces* finalizados por instruções no *delay-slot* representou até 74% para m88ksim e obteve média harmônica de 31%*;*
- No compress 27% dos *traces* foram finalizados por instruções de acesso à memória;

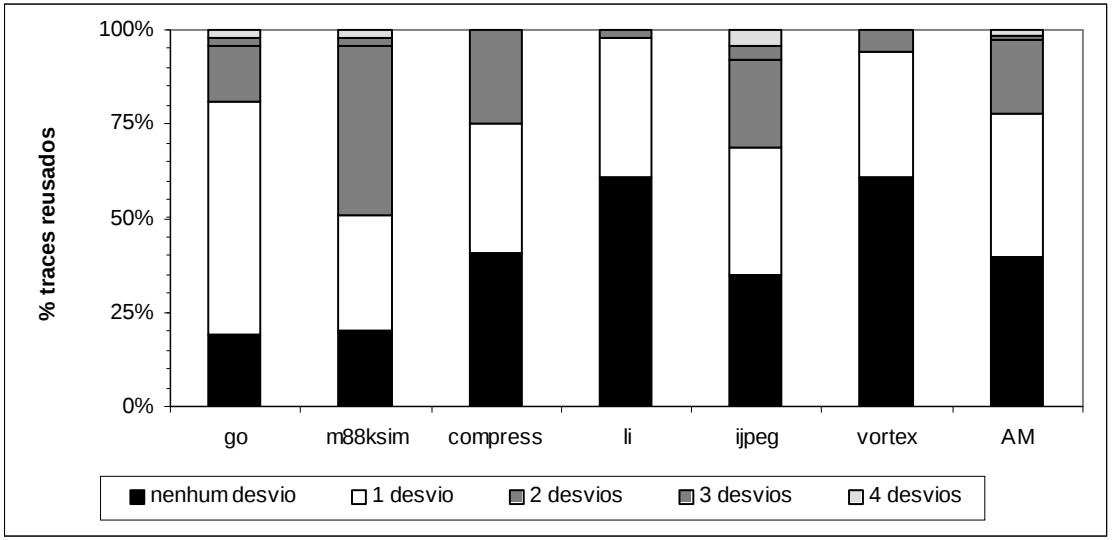
Análise dos Resultados

Resultados da Implementação do DTM no Sparc



DTM*sparc*

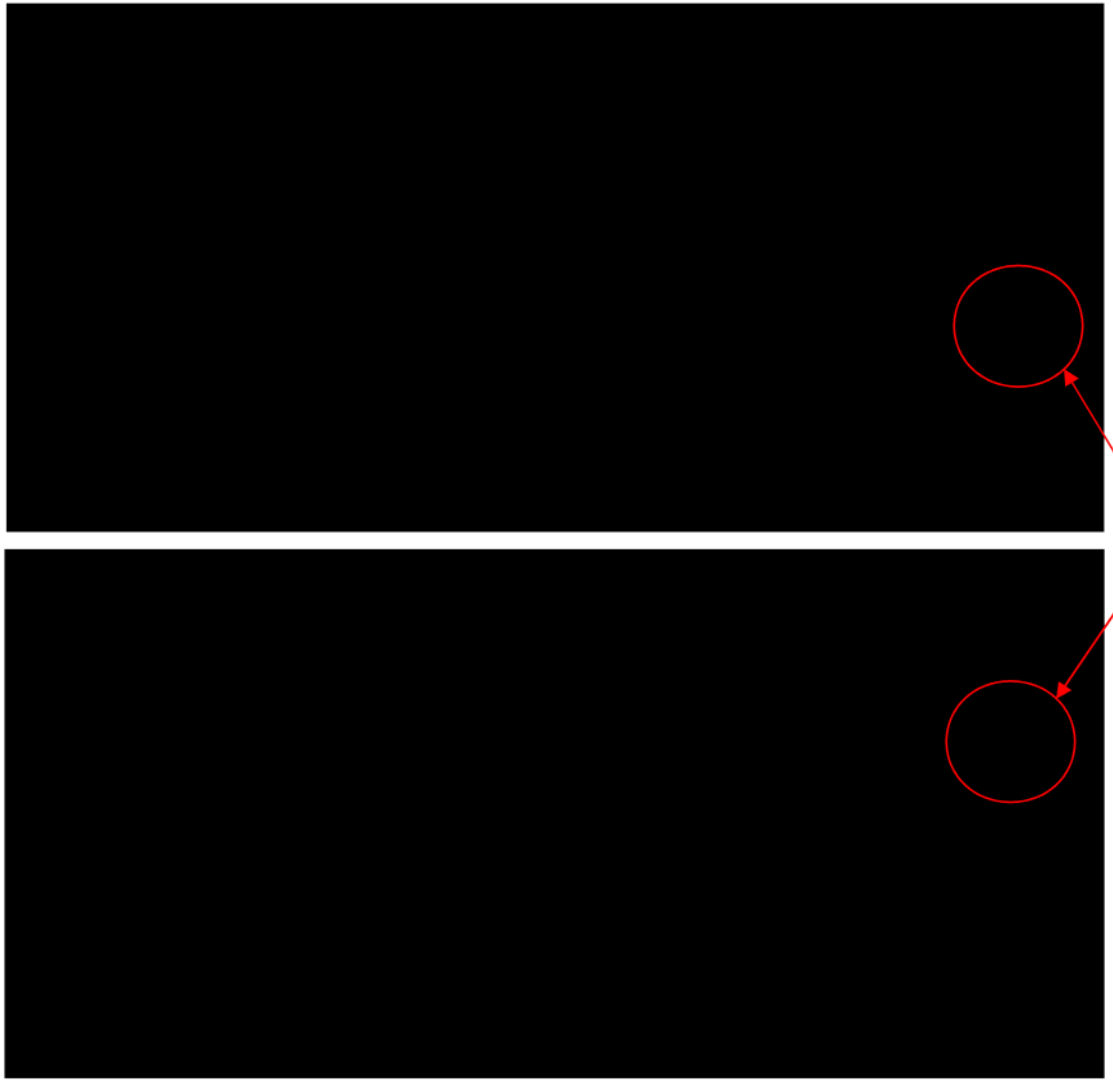
- No DTM*sparc* 69% dos *traces* reusados não possuem desvios e 31% deles possui apenas um desvio;
- No DTM*mips* 39% dos *traces* não possuem desvios e 38.5% dos *traces* possuem apenas um desvio, e cerca de 22% dos *traces* possuem mais de um desvio;



DTM*mips*

Análise dos Resultados

Resultados das Diferentes Implementações do DTM com Reuso de Instruções de Memória



- A aceleração média dos mecanismo com antecipação de valores de memória foram 4.1% e 2.9% acima do valor obtido com o DTM;
- O reuso explorado apresentou uma variação média de 1% entre os mecanismos;
- A aceleração média obtida pelo mecanismo com invalidação de valores de memória foi igual a obtida com o DTM;

Análise dos Resultados

Analisando os Resultados do 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
AM	1.45	25.67	0.00	49.35	0.11	24.87

- DTM*inv* explora pouco o reuso de instruções de leitura da memória devido as frequentes invalidações das tabelas de reuso;
- Houve um acréscimo no número de *traces* reusados finalizados por *overflow* do contexto de saída;

Análise dos Resultados

Analisando os Resultados do 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

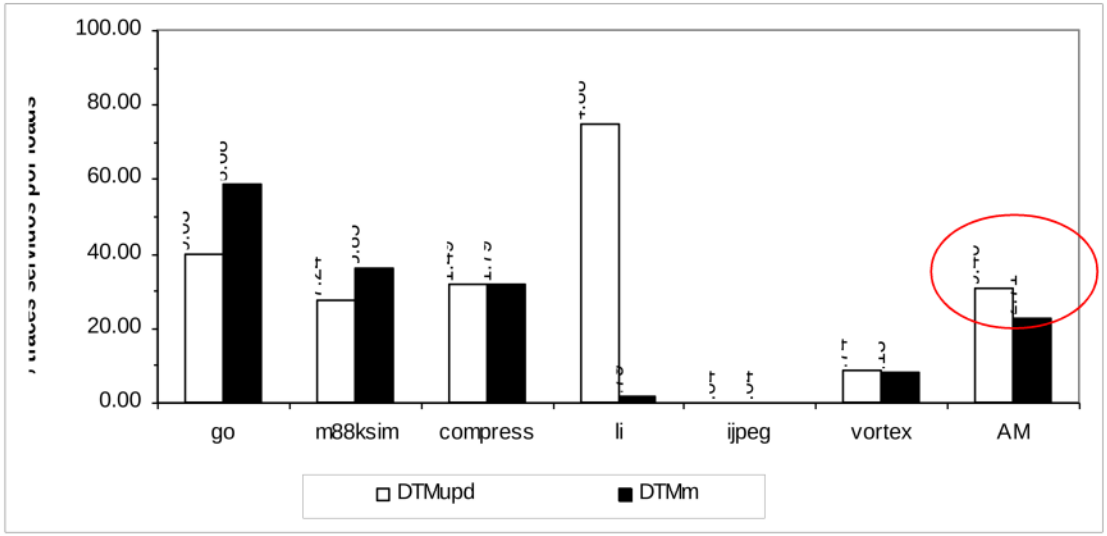
- Houve um aumento no número de *traces* com duas instruções compostos por uma instrução de escrita na memória;

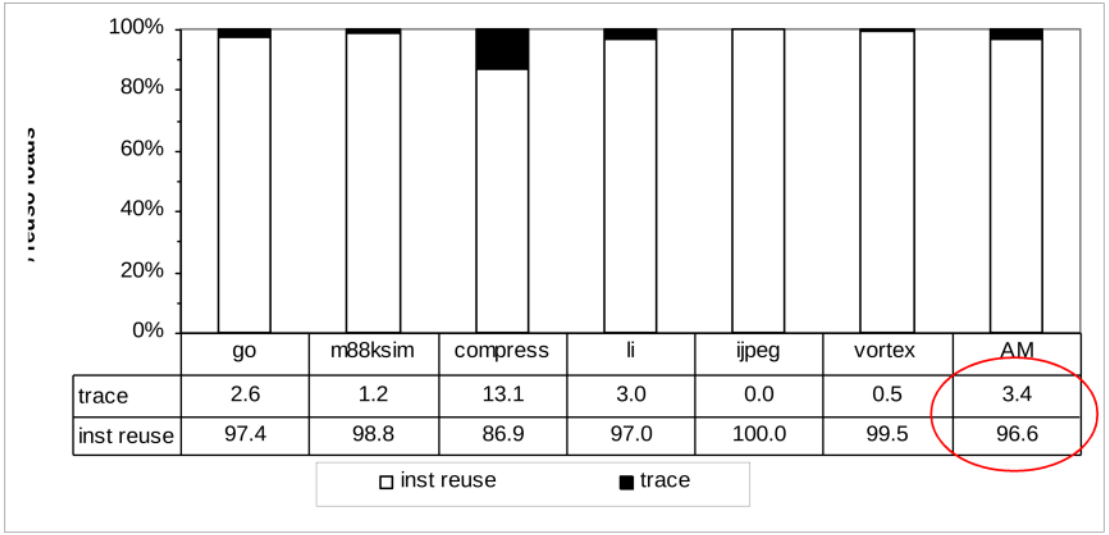
- *Traces* com duas instruções e que possuem instruções de acesso à memória são constituídos por instruções de *stores*;

- *Traces* com mais de duas instruções possuem maior frequencia de *loads*;

Análise dos Resultados

Analisando os Resultados do DTM*upd* e DTM*m*





- A quantidade de *traces* servidos por *instruções de loads* reusadas apresentou um resultado médio de 30% e 23% para o DTM*upd* e DTM*m* respectivamente;
- O reuso de instruções de *load* representa em média 97% dos reusos deste tipo de instrução;

Análise dos Resultados

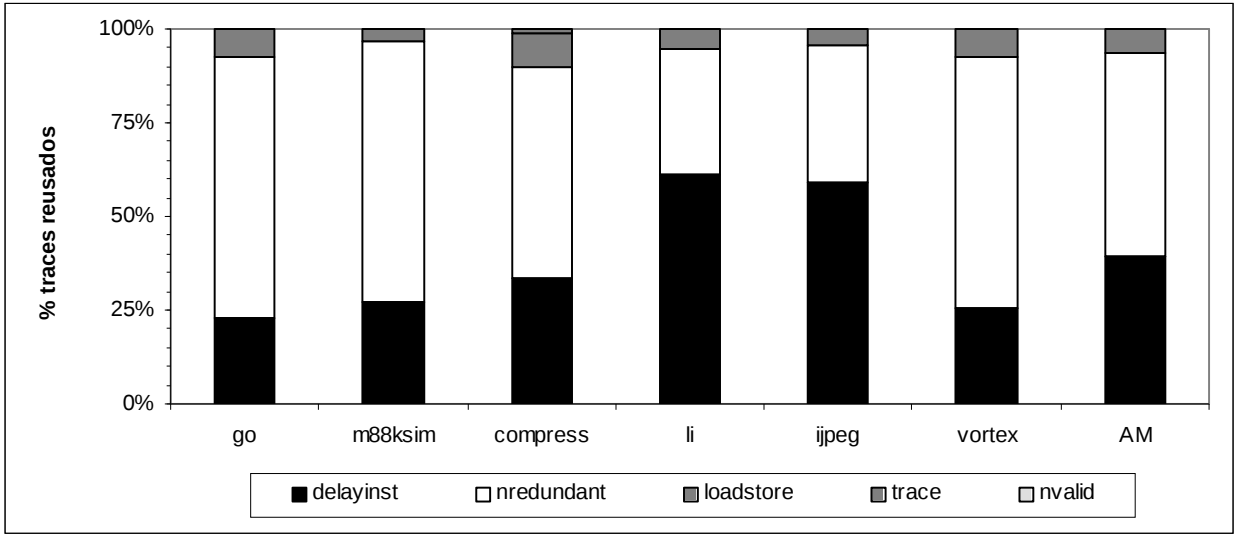
Resultados Obtidos com o Reuso de Cadeias de Instruções e *Traces* Dependentes em um Mesmo Ciclo

- DTM*upd* e DTM*m* apresentaram resultados 7.2% e 6.5% respectivamente de média harmônica sobre a implementação original destes mecanismos;
  - Número excessivo de instruções ocupando *delay-slot* pode ter fragmentado de forma excessiva os *traces*;
  - *Traces* que possuem um subconjunto do contexto de entrada fornecido por *traces* ou instruções redundantes despachadas no mesmo ciclo;

Análise dos Resultados

Motivo de Finalização dos *Traces* não Formados

	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% dos *traces* não formados foram finalizados por instruções presentes em *delay-slot*;
- O compress apresentou uma quantidade consideravel de *traces* não formados que são finalizados por *traces* redundantes;

Conteúdo

- Objetivo
- Memorização Dinamica de *Traces* – DTM
- Adicionando Instruções de Acesso à Memória ao DTM
- O Mecanismo DTM*m*
- Ambiente de Simulação
- Análise dos Resultados
- **Conclusão e Trabalhos Futuros**

Conclusões

- A inclusão de instruções de acesso à memória no DTM fornece um acréscimo de aceleração de 4.1% em média harmônica;
- O compartilhamento dos campos destinados ao contexto de saída com os endereços e valores de instruções de memória aumentam a frequencia de traces finalizados por *overflow* do contexto de saída, além disso a inclusão de instruções de *store* não trazem ganhos significativos ao mecanismo;
- Os experimentos demonstraram que a separação das instruções de *loads* traz simplificações ao mecanismo, reduz o custo de implementação e mantém desempenho comparável ao obtido com o DTM*upd*;
- A implementação de mecanismos capazes de despachar *traces* e instruções redundantes com dependências de dados em um mesmo ciclo fornece ao mecanismo um acréscimo de 7.2% na média hamônica das acelerações;

Trabalhos Futuros

- Implementar o mecanismo DTM*upd* e DTM*m* em uma arquitetura que não possui instruções de *delay-slot* – MIPS;
- Estudar diferentes políticas de escalonamento e construção de *traces* para melhorar a qualidade dos *traces* reusados;.
- Avaliar o desempenho do DTM*upd* e DTM*m* removendo do conjunto de instruções válidas as instruções de escrita na memória;
- Redimensionar as tabelas de memorização com base nos mecanismos DTM*upd* e DTM*m*;
- Implementar mecanismos capazes de concatenar *traces* em construção com *traces* redundantes;
- Implementar múltiplas instruções de leitura dentro de um mesmo *trace* identificando as instruções dependentes e finalizando os *traces* na ocorrência delas;
- A implementação de mecanismos capazes de antecipar valores dos operandos de entrada dos *traces* tais como mecanismos de predição de valores para o contexto de entrada dos *traces*;
- Concatenar cadeias de *traces* e instruções redundantes dinamicamente;