ISA:
suporte para aritmética inteira e sincronização de *threads*

Luiz C. V. dos Santos, INE/CTC/UFSC

Inteiros com sinal

- **Complemento de dois**

$$-2^{n-1} \times bit_{n-1} + 2^{n-2} \times bit_{n-2} \ldots 2^{2} \times bit_{2} + 2^{1} \times bit_{1} + 2^{0} \times bit_{0}$$

- **Faixa: [- 2^{n-1} ; + 2^{n-1})**
  - **Se número ≥ $+2^{n-1}$ ⇒ transbordo (*overflow*)**
  - **Se número < -2^{n-1} ⇒ transbordo (*overflow*)**
- **Uso**
  - **Variáveis inteiras**
  - **Constantes inteiras**

Revisão: troca de sinal

- **Complementar todos os bits e somar 1**
- **Exemplo: 0110 (+6) → ? (-6)**

  1001

  + 1

  1010 (-6)

sub $s0, $s1, $s3 induz troca do sinal de $s3 e soma com $s1

- **Consequência:**
  - **Subtração é convertida para adição**

  0111 (+7) **0111 (+7)**

  - 0110 (+6) **+1010 (-6)**

  0001

Não existe **subi** $s0, $s1, 1 pois pode ser realizada como **addi $s0, $s1, -1**

Revisão: *overflow*

- **Operandos representados em n bits**
  Resultado não representável em n bits

$$\begin{array}{r} 0111 \\ +\ 0001 \\ \hline 1000 \end{array}$$

- **Transbordo troca o sinal do resultado**

Ocorrência de overflow

- **Não ocorre**
  - **Soma de números com sinais opostos**
  - **Subtração de números com mesmo sinal**
- **Ocorre quando valor afeta sinal**

Operação	A	B	Resultado
A+B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A-B	≥ 0	< 0	< 0
A-B	< 0	≥ 0	≥ 0

Cin[MSB] $\oplus$ **Cout**[MSB] $\Rightarrow$ **overflow**

MIPS: suporte para overflow

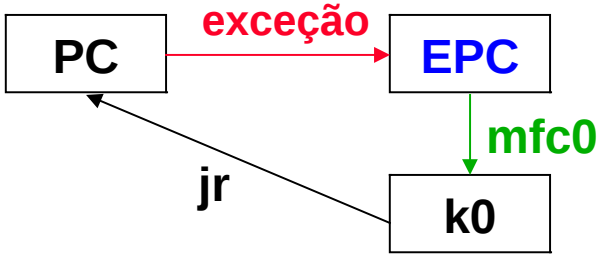
- **Mascaramento ou não**
  - **Situações em que se quer reconhecê-lo**
    - **add, addi, sub**
  - **Situações em que se quer ignorá-lo**
    - **addu, addiu, subu**

MIPS: suporte para overflow

- **Detecção**
  - **Mecanismo: overflow causa exceção**
- **Exceção**
  - **Situação excepcional que subverte execução do programa**
  - **HW chama subrotina não “programada”**
  - **Impropriamente chamada de interrupção**
    - » **Exceção com causa externa à CPU**

MIPS: dinâmica de uma exceção

- **Controle desvia p/ endereço pré-definido**
  - **Onde reside rotina de tratamento da exceção**
    - » **0x 8000 0180 (*kernel*)**
- **Endereço da instrução que causou exceção é salvo (para possível re-início)**
  - » **Registrador EPC ("exception program counter")**
  - » **Instrução mfc0 ("move from system control")**



0x 8000 0180

...

...

mfc0 $k0, $14 # $k0 ← EPC

jr $k0 # PC ← $k0

Multiplicação

- **Multiplicando e mutiplicador: n bits**

  Produto: 2 x n bits
- **Exemplo:**
  - **15 x 15 = 225**

```
    1111
x   1111
------
    1111
   1111
  1111
------
  101101
 1111
------
 1101001
1111
------
11100001
```

MIPS: suporte para multiplicação

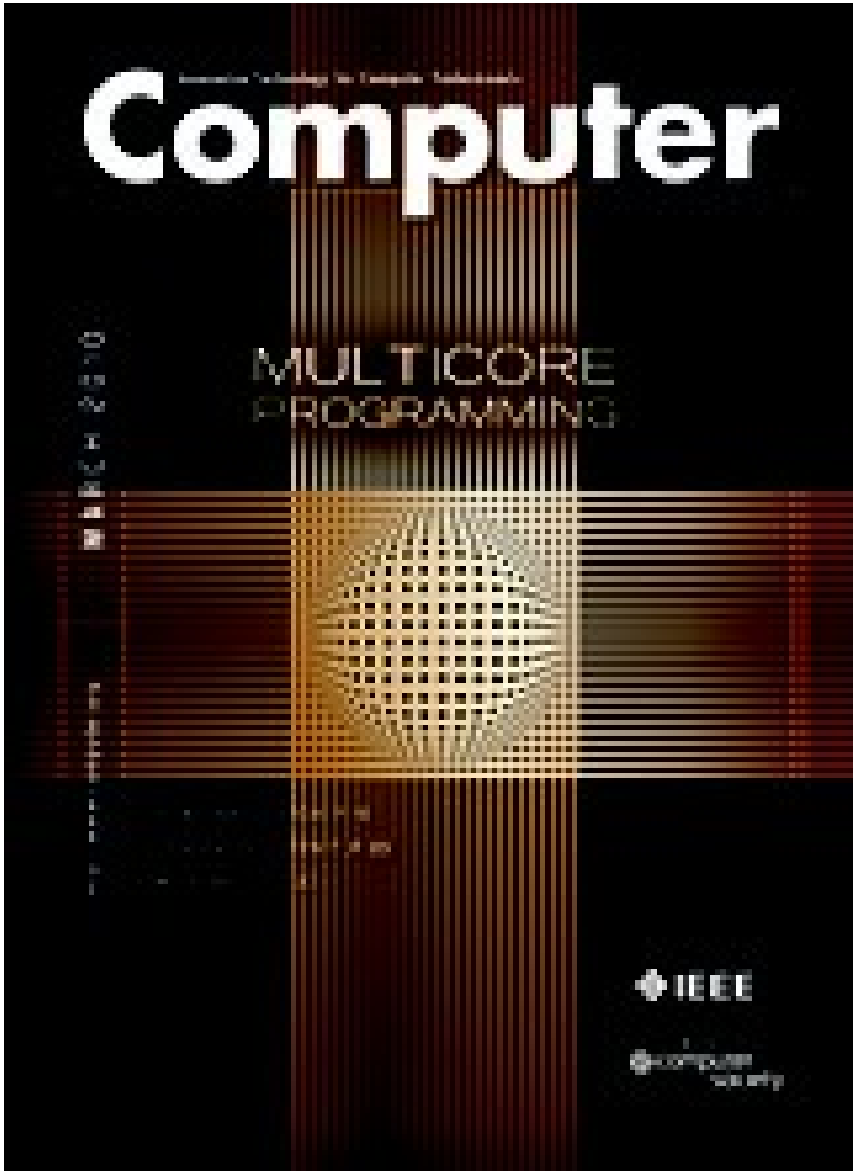
- **Armazenamento do produto**
  - **Par de registradores especiais**
    - » **Hi: armazena 32 MSBs do produto**
    - » **Lo: armazena 32 LSBs do produto**
  - **mult $s2, $s3**
    - » **Hi, Lo = $s2 x $s3**
- **Como extrair o produto de Hi e Lo ?**
  - **mfhi $s1**
    - » **$s1 = Hi**
  - **mflo $s0**
    - » **$s0 = Lo**

MIPS: suporte para divisão

- **Armazenamento de quociente e resto**
  - **Par de registradores especiais**
    - » **Lo: quociente representado em 32 bits**
    - » **Hi: resto representado em 32 bits**
  - **div $s2, $s3**
    - » **Lo = $s2 / $s3**
    - » **Hi = $s2 mod $s3**
- **Como extrair quociente e resto de Lo e Hi ?**
  - **Usando mfhi e mflo**

MIPS: multiplicação e divisão

- **Multiplicação**
  - **Produto representado em 64 bits em Hi e Lo**
    - » **mult (produto sinalizado)**
    - » **multu (produto não sinalizado)**
- **Divisão**
  - **Resto e quociente representados em Hi e Lo**
    - » **div (resto e quociente sinalizados)**
    - » **divu (resto e quociente não sinalizados)**
- **Extração de produto, quociente e resto**
    - » **mfhi e mflo**


Computer
MULTICORE
PROGRAMMING
IEEE

Paralelismo e instruções: sincronização

- **Motivação:**
  - ***Threads* cooperantes**
  - **Comunicam-se via uma variável compartilhada**

Paralelismo e instruções: sincronização

- **Motivação:**
  - ***Threads* cooperantes**
  - **Comunicam-se via uma variável compartilhada**

Valor inicial: data = = old;

P1:	**P2:**
…	**…**
data = new;	**data_copy = data;**
…	**…**

Paralelismo e instruções: sincronização

- **Motivação:**
  - ***Threads* cooperantes**
  - **Comunicam-se via uma variável compartilhada**

Valor inicial: data = = old;

P1:	**P2:**
…	**…**
data = new;	**data_copy = data;**
…	**…**

- **Resultado em “data_copy” é não-determinístico**
  - **Depende da ordem em que forem disparadas as threads**

Data race e sincronização

- ***Data race*:**
  - **2 acessos à memória vindos de *threads* distintas**
  - **Acessos referem-se ao mesmo endereço**
  - **Pelo menos um é uma escrita**
  - **Os acessos são sucessivos**
- ***Date race* ⇒programa com ≠s resultados**
  - **Dependendo da ordem entre eventos (acessos)**
  - **Para o mesmo arquivo de entrada**
- **Para eliminar *data races*:**
  - **Operações de sincronização**

Semáforo

sem { **1: ocupado**
0: livre

lock (sem); Espera até que **sem = 0**, **sem ←1** e retorna

/* seção crítica */

unlock (sem); **sem ←0** e retorna

Exemplo de concorrência

	$a0	**$t1**	**MEM[saldo]**	**$t1**		**$a0**
deposito:	lw $t1, saldo				saque:	lw $t1, saldo
	add $t1, $t1, $a0					**sub** $t1, $t1 $a0
	sw $t1, saldo					sw $t1, saldo
	jr $ra					jr $ra

Execução simultânea

	$a0	$t1	MEM[saldo]	$t1		$a0
	20		**100**			**50**
deposito:	lw $t1, saldo	100		100	saque:	lw $t1, saldo
	add $t1, $t1, $a0	120		50		**sub** $t1, $t1 $a0
	sw $t1, saldo		**?**			sw $t1, saldo
	jr $ra					jr $ra

Uma execução possível

	$a0	**$t1**	**MEM[saldo]**	**$t1**		**$a0**
	20		**100**			**50**
deposito:	lw $t1, saldo	100		100	saque:	lw $t1, saldo
	add $t1, $t1, $a0	120		50		**sub** $t1, $t1 $a0
	sw $t1, saldo		120			
	jr $ra		**50**			sw $t1, saldo
						jr $ra

Outra execução possível

	$a0	$t1	MEM[saldo]	$t1		$a0
	20		**100**			**50**
deposito:	lw $t1, saldo	100		100	saque:	lw $t1, saldo
	add $t1, $t1, $a0	120		50		**sub** $t1, $t1 $a0
			50			sw $t1, saldo
	sw $t1, saldo		**120**			jr $ra
	jr $ra					

Sincronizando as threads

	$a0	$t1	MEM[saldo]	$t2		$a0
	20		**100**			**50**
deposito:	jal lock				saque:	jal lock
	lw $t1, saldo					lw $t2, saldo
	add $t1, $t1, $a0					**sub** $t2, $t2 $a0
	sw $t1, saldo					sw $t2, saldo
	jal unlock					jal unlock
	jr $ra					jr $ra

Uma execução possível

	$a0	$t1	MEM[saldo]	$t2		$a0
	20		**100**			**50**
deposito:	jal lock				saque:	jal lock
	lw $t1, saldo					
	add $t1, $t1, $a0	120				
	sw $t1, saldo		120			
	jal unlock					
	jr $ra			120		lw $t2, saldo
				70		**sub** $t2, $t2 $a0
			70			sw $t2, saldo
						jal unlock
						jr $ra

Outra execução possível

	$a0	$t1	MEM[saldo]	$t2		$a0
	20		**100**			**50**
deposito:	jal lock				saque:	jal lock
				100		lw $t2, saldo
				50		**sub** $t2, $t2 $a0
			50			sw $t2, saldo
						jal unlock
	lw $t1, saldo	50				jr $ra
	add $t1, $t1, $a0	70				
	sw $t1, saldo		**70**			
	jal unlock					
	jr $ra					

Sincronização: níveis de uso

- **Programador de aplicativo (programa concorrente)**
  - **Usa operações de sincronização abstratas**
    - » **Para eliminar *data races***
    - » **Exemplo: *lock/unlock***
  - **Disponíveis como rotinas de biblioteca**
    - » **Exemplo: pthreads.h + biblioteca de *threads***
      - **pthread_mutex_lock()**
      - **pthread_mutex_unlock()**

Sincronização: níveis de uso

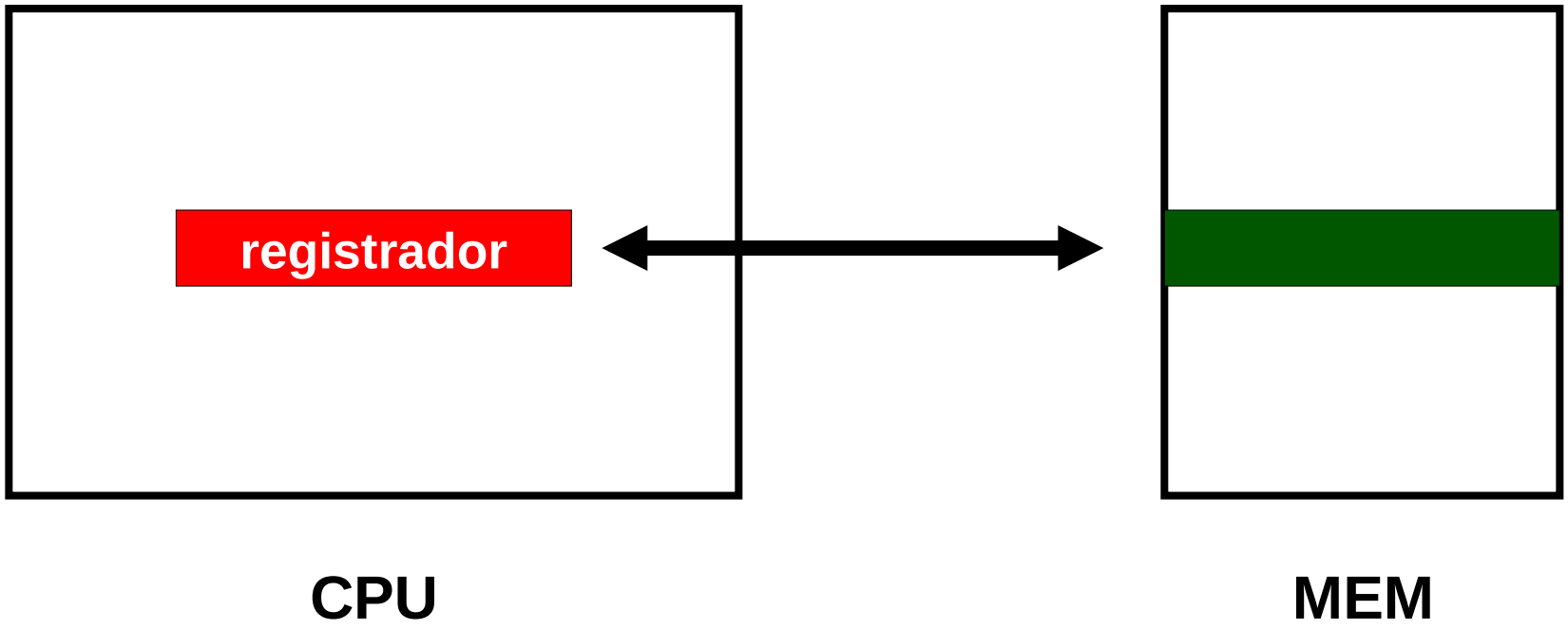
- **Programador de sistema**
  - **Usa primitivas de baixo nível**
    - » **Para construir biblioteca de sincronização**
    - » **Exemplo: *atomic swap* (*exchange*)**
  - ***Atomic swap* construído com:**
    - » **Uma única instrução (e.g. ARMv7)**
    - » **Múltiplas instruções (e.g. MIPS e ARMv8)**

Ideia-chave para sincronização

- **Operação atômica de leitura-e-modificação**
  - **Leitura (R) e modificação (W)**
  - **De uma mesma posição de memória**
  - **Nenhum acesso intermediário entre R e W**
- **Diferentes primitivas em HW para suportar:**
  - **Leitura-e-modificação atômica**
    - » **ARMv7 (SWAP)**
  - **Indicação se operação foi atômica**
    - » **MIPS (ll: *load linked*, sc: *store conditional*)**
    - » **ARMv8 (LDX: *Load-exclusive*; STX: *store-exclusive*)**

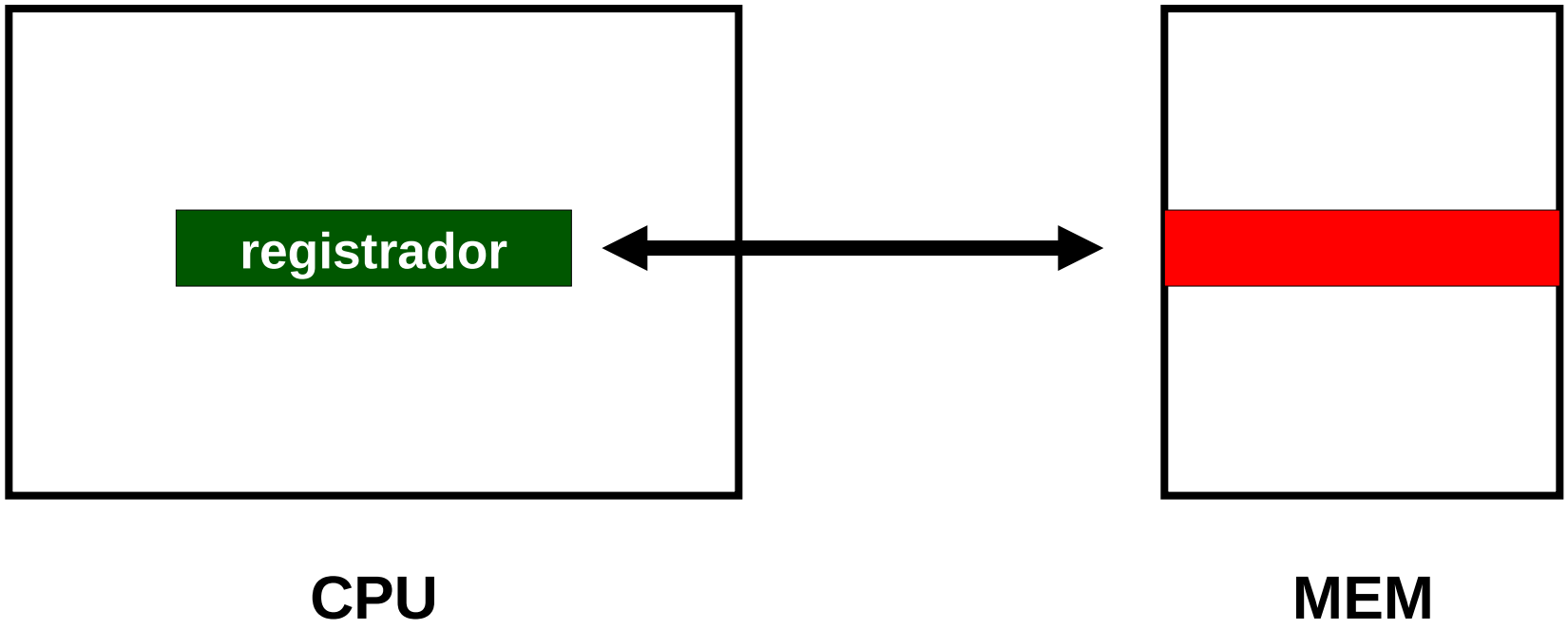
Primitiva de baixo nível: *atomic swap*

- **Troca valores entre registrador e memória**



Primitiva de baixo nível: *atomic swap*

- **Troca valores entre registrador e memória**



Instruções de sincronização: ll e sc

- ***Linked load***
  - **Exemplo: ll $t1, 0($s1)**
  - **Busca o valor residente num dado endereço**
  - **E o carrega em um registrador**
  - **Vinculando o endereço ao início de uma operação atômica**
    - » **Ativa um flag invisível para ISA (link status bit)**
- ***Store conditional***
  - **Exemplo: sc $t0, 0($s1)**
  - **Tenta escrever o valor de registrador no endereço vinculado**
    - » **Só escreve se o flag estiver ativado**
  - **Se escrita se completar, retorna "1" no registrador**
    - » **E desativa o flag**
  - **Se escrita falhar, retorna "0" no registrador**
    - » **Outro processador escreveu no endereço vinculado**

lock: implementação no MIPS

```
lock: la $t0, sem
try:  ll $t1, 0($t0)        Tenta iniciar operação atômica
      bne $t1, $zero, try   Se $t1 = 1, semáforo ocupado: tenta de novo
      addi $t1, $zero, 1    Semáforo livre, prepara para ocupá-lo
      sc $t1, 0($t0)        Tenta concluir operação atômica
      beq $t1, $zero, try   Se $t1 = 0, não foi atômica: tenta de novo
      /* capturou o semáforo */
      jr $ra
```

unlock: implementação no MIPS

```
unlock: la $t0, sem
        sw $zero, 0($t0)
        jr $ra
```

Conclusões e perspectivas

- **Instruções de sincronização**
  - **Base para implementação de primitivas**
    - » **Base para biblioteca de rotinas de sincronização**
- **Uso na sincronização de processos cooperantes de um programa concorrente**
  - **Executados sequencialmente em *single core***
    - » **S.O multi-tarefa**
- **Uso na sincronização de *threads* cooperantes de um programa concorrente**
  - **Executadas em paralelo em *multicores***