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

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

Representação numérica

- **Como representar**
  - **Números inteiros “unsigned”?**
    - » **Dígitos binários representam a magnitude**
  - **Números inteiros positivos ou negativos ?**
    - » **Um dígito binário tem acepção de sinal**
    - » **Representação em complemento de dois**
  - **Número fracionários ou reais ?**
    - » **Ponto flutuante (mantissa e expoente)**
    - » **Fora do âmbito desta disciplina**

Representação numérica

- **Representação finita**
  - **Qual o maior número representável ?**
    - » **Número positivo de maior magnitude**
    - » **Número negativo de maior magnitude**
  - **Quando valor fora da faixa de representação ?**
    - » **Transbordo ou “overflow”**
  - **Como converter um valor de n para n+1 bits ?**
    - » **Extensão de sinal**

Inteiros sem sinal

- **Representação binária simples**

$$2^{n-1} \times \text{bit}_{n-1} + 2^{n-2} \times \text{bit}_{n-2} \ldots 2^{2} \times \text{bit}_{2} + 2^{1} \times \text{bit}_{1} + 2^{0} \times \text{bit}_{0}$$

- **Faixa: [0, 2^n-1]**
  - **Se número ≥ 2^n ⇒ transbordo ou "overflow"**
- **Uso**
  - **Endereços de memória**
  - **Inteiros sem sinal (unsigned int)**

Inteiros com sinal

- **Complemento de dois**

$$-2^{n-1} \times \text{bit}_{n-1} + 2^{n-2} \times \text{bit}_{n-2} \ldots 2^2 \times \text{bit}_2 + 2^1 \times \text{bit}_1 + 2^0 \times \text{bit}_0$$

- **Faixa: [- 2^{n-1} ; + 2^{n-1})**
  - **Se número $\geq$ $+2^{n-1}$ $\Rightarrow$ transbordo (“overflow”)**
  - **Se número $<$ -2^{n-1} $\Rightarrow$ transbordo (“overflow”)**
- **Uso**
  - **Variáveis inteiras**
  - **Constantes inteiras**

MIPS: faixa de representação

- **Números com sinal (32 bits):**

```
0000 0000 0000 0000 0000 0000 0000 0000 =   0dec
0000 0000 0000 0000 0000 0000 0000 0001 = + 1dec
0000 0000 0000 0000 0000 0000 0000 0010 = + 2dec
   ...
0111 1111 1111 1111 1111 1111 1111 1110 = + 2.147.483.646dec
0111 1111 1111 1111 1111 1111 1111 1111 = + 2.147.483.647dec   Max int
1000 0000 0000 0000 0000 0000 0000 0000 = – 2.147.483.648dec   Min int
1000 0000 0000 0000 0000 0000 0000 0001 = – 2.147.483.647dec
1000 0000 0000 0000 0000 0000 0000 0010 = – 2.147.483.646dec
   ...
1111 1111 1111 1111 1111 1111 1111 1101 = – 3dec
1111 1111 1111 1111 1111 1111 1111 1110 = – 2dec
1111 1111 1111 1111 1111 1111 1111 1111 = – 1dec
```

Instruções de suporte no MIPS

- **Comparação com sinal × sem sinal**
  - **(slt, slti) × (sltu, sltiu)**

  - **$s0: 1111 1111 1111 1111 1111 1111 1111 1111**
  - **$s1: 0000 0000 0000 0000 0000 0000 0000 0001**

  - **slt $t0, $s0, $s1 # $t0=1, pois -1 < 1**

  - **sltu $t1, $s0, $s1 # $t1=0, pois 4.294.967.295 > 1**

Atalho para checar limites de arranjo

- **Tratar números sinalizados como não-sinalizados**
  - **Alternativa de baixo custo para testar $0 \leq x < y$**
  - **Limites de indexação de um arranjo com y elementos**
- **Idéia-chave:**
  - **Inteiros negativos**
    - » **Em notação complemento de dois**
    - » **MSB é bit de sinal**
  - **Parecem números grandes**
    - » **Em notação não-sinalizada**
    - » **MSB contribui com um grande valor para a magnitude**
- **Consequência**
  - **Comparação não-sinalizada: sltu $t0, x, y**
  - **Equivale a duas comparações: $0 \leq x < y$**

Atalho para checar limites de arranjo

- **Exemplo:**
  - **Desvie para IndexOutofBounds se x < 0 ou x ≥ y**
  - **Suponha que (x, y) → ($s1, $t2)**

```
sltu $t0, $s1, $t2          # $t0 = 1, se 0 ≤ x < y
                            # $t0 = 0, se x < 0 ou x ≥ y
beq $t0, $zero, IndexOutofBounds
```

Troca de sinal

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

  1001

  + 1

  1010 (-6)
- **Conseqüência:**
  - **Subtração é convertida para adição**

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

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

  0001

Extensão de sinal

- **Copiar o bit de sinal nos MSBs**
- **Exemplo:**

  101 = $-2^2 + 2^0 = -3$

  1101 = $-2^3 + 2^2 + 2^0 = -3$

  11101 = $-2^4 + 2^3 + 2^2 + 2^0 = -3$

  – **Aplicação**

    » **Conversão de 8 para 32 bits**

    » **Conversão de 16 para 32 bits**

Instruções de suporte no MIPS

- **Load sinalizado**
  - **Carrega o número nos LSBs**
  - **E copia o sinal nos MSB**
    - » **lb**
    - » **lh**
- **Load não sinalizado**
  - **Carrega o número nos LSBs**
  - **E preenche com zeros os MSB**
    - » **lbu**
    - » **lhu**

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

MIPS: suporte para overflow

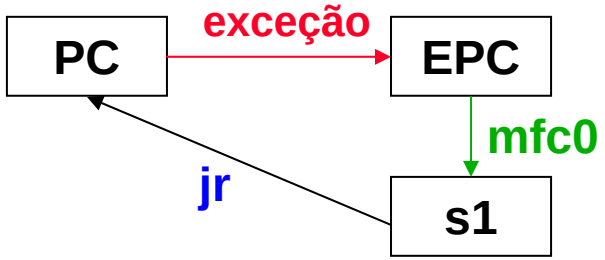
- **Mascaramento**
  - **Situações em que se quer reconhecê-lo**
    - **add, addi, sub**
  - **Situações em que se quer ignorá-lo**
    - **addu, addiu, subu**

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: suporte para overflow

- **Dinâmica da exceção**
  - **Controle desvia para endereço pré-definido onde reside rotina de tratamento da exceção**
  - **Endereço da instrução que causou overflow é salvo para possível re-início**
    - » **Registrador EPC (“exception program counter”)**
    - » **Instrução mfc0 (“move from system control”)**



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**

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 os eventos venham a ocorrer**

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* $\Rightarrow$programa com $\neq$s resultados**
  - **Dependendo da ordem entre eventos (acessos)**
  - **Para o mesmo arquivo de entrada**
- **Para eliminar *data races*:**
  - **Operações de sincronização**

Esboço de um mecanismo de sincronização

Valores iniciais: data = = old; lock == 0;

P1:

```
lock = 1;
...
data = new;
lock = 0;
...
```

P2:

```
...
while (lock == 1) { };
data_copy = data;
...
```

Esboço de um mecanismo de sincronização

Valores iniciais: data = = old; lock == 0;

```
P1:                          P2:
lock = 1;
...
data = new;
lock = 0;
...                          ...
                             while (lock == 1) { };
                             data_copy = data;
                             ...
```

- **Resultado em "data_copy" é determinístico**
  - **Qualquer que seja a ordem em que os eventos venham a ocorrer**

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

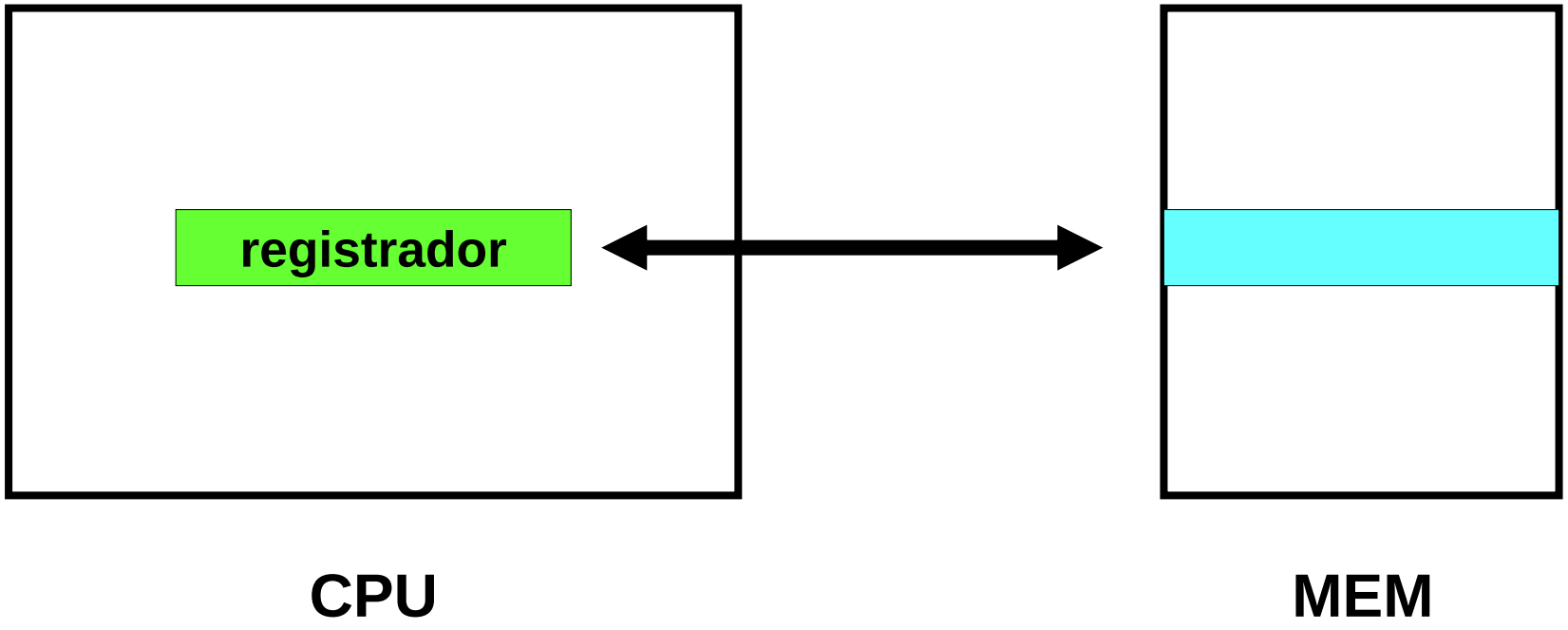
- **Programador de sistema**
  - **Usa primitivas de baixo nível**
    - » **Para construir biblioteca de sincronização**
    - » **Exemplo: *atomic swap* (*exchange*)**
  - **Primitivas construídas com instruções**
    - » **Exemplo: MIPS (ll: *load linked*, sc: *store conditional*)**

Idéia-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**
  - **Indicação se operação foi atômica**

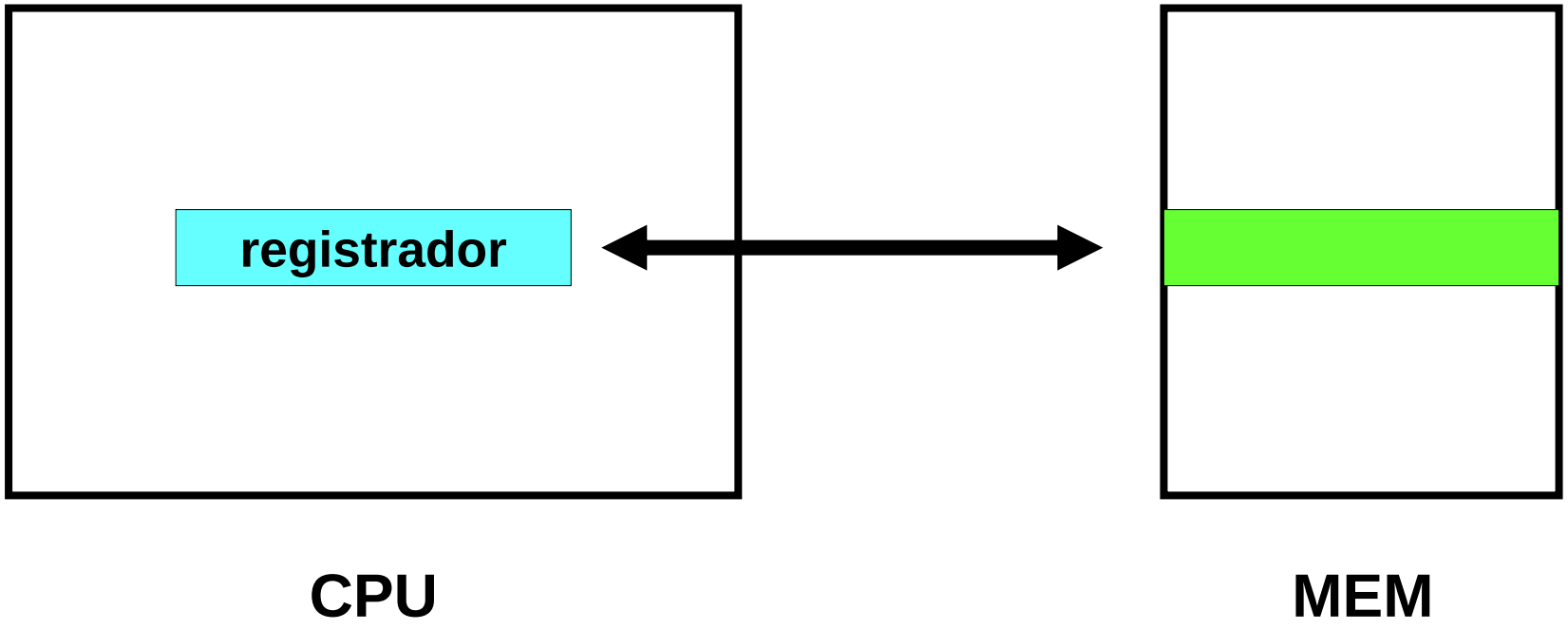
Primitiva de baixo nível: *atomic swap*

- **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**
- **Store conditional**
  - **Exemplo: sc $t0, 0($s1)**
  - **Tenta escrever o valor de registrador no endereço vinculado**
  - **Se escrita se completar, retorna "1" no registrador**
  - **Se escrita falhar, retorna "0" no registrador**
    - » **Outro processador escreveu no endereço vinculado**

Atomic swap: implementação no MIPS

- **swap: $s4 ↔ MEM[$s1]**

```
try:  add $t0, $zero, $s4
      ll $t1, 0($s1)
      sc $t0, 0($s1)
      beq $t0, $zero, try
      add $s4, $zero, $t1
```

Tenta-se operação atômica

Se $t0 = 0, não foi atômica: tenta de novo

Atomic swap: implementação no MIPS

- **swap: $s4 ↔ MEM[$s1]**

```
try:  add $t0, $zero, $s4
      ll $t1, 0($s1)
      sc $t0, 0($s1)
      beq $t0, $zero, try
      add $s4, $zero, $t1
```

Repita até que seja atômica

Atomic swap: implementação no MIPS

- **SWAP: $s4 ↔ MEM[$s1]**

```
try:  add $t0, $zero, $s4
      ll $t1, 0($s1)
      sc $t0, 0($s1)
      beq $t0, $zero, try
      add $s4, $zero, $t1
```

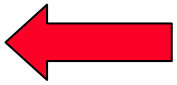
$s4 → MEM[$s1]

MEM[$s1] → $s4

Esboço de como implementar lock

- **Variável de sincronização: 0($s1)**
- **Código para efetuar *locking***

```
        addi $s4, $zero, 1
lockit: SWAP $s4, 0($s1)
        bne $s4, $zero, lockit
```

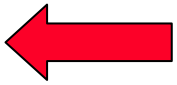


Operação atômica garante que um único processador tomará posse de variável compartilhada

Esboço de como implementar lock

- **Variável de sincronização: 0($s1)**
- **Código para efetuar *locking***

```
            addi $s4, $zero, 1
lockit:     SWAP $s4, 0($s1)
            bne $s4, $zero, lockit
```



Operação atômica garante que um único processador tomará posse de variável compartilhada

- **Se $s4 é ‘0’,**
  - **Outra thread já tomou posse (aguarde-se)**
- **Se $s4 é ‘1’**
  - **Esta thread tomará posse**

Conclusões e perspectivas

- **Instruções de sincronização**
  - **Base para implementação de primitivas**
    - » **Base para biblioteca de rotinas de sincronização**
- **Uso na sincronização de *threads* cooperantes de um programa concorrente**
  - ***Multicores***
- **Uso na sincronização de processos cooperantes de um programa sequencial**
  - **Uniprocessadores**