

**Instituto de
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



Organização Básica de computadores e linguagem de montagem

Interrupções

Prof. Edson Borin

<https://www.ic.unicamp.br/~edson>

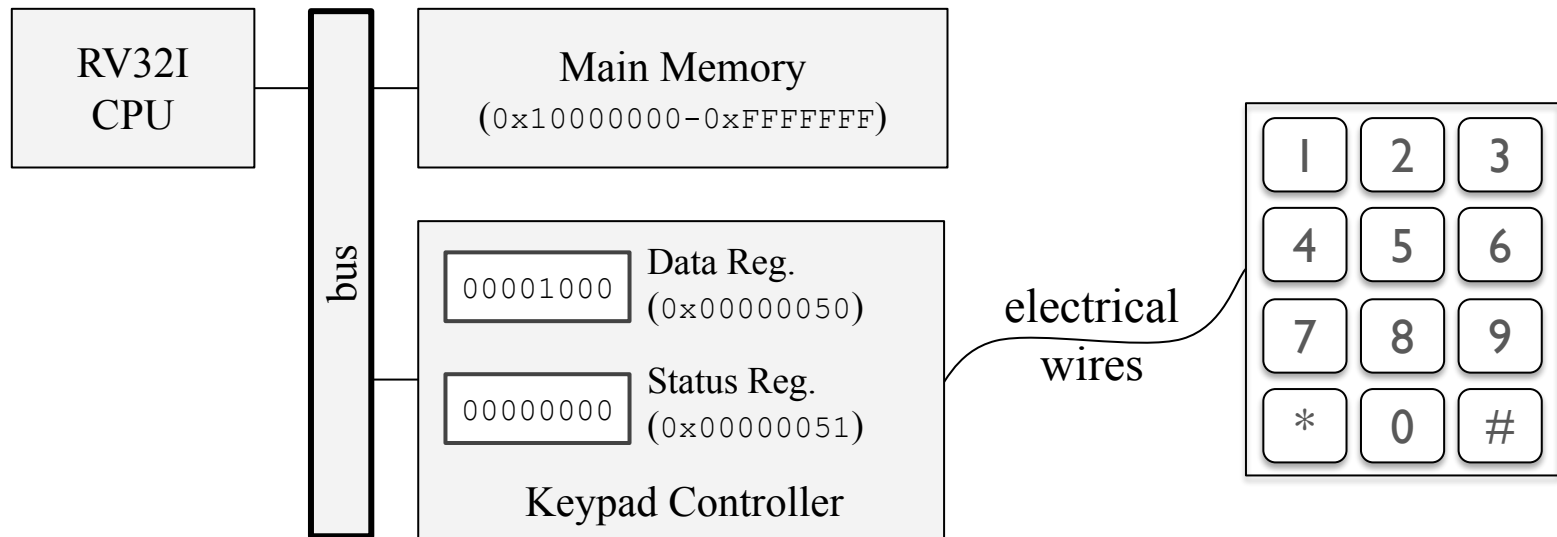
Institute of Computing - UNICAMP

Agenda

- Motivação
- Polling
- Interrupções Externas
- Interrupções Externas no RISC-V

Motivação

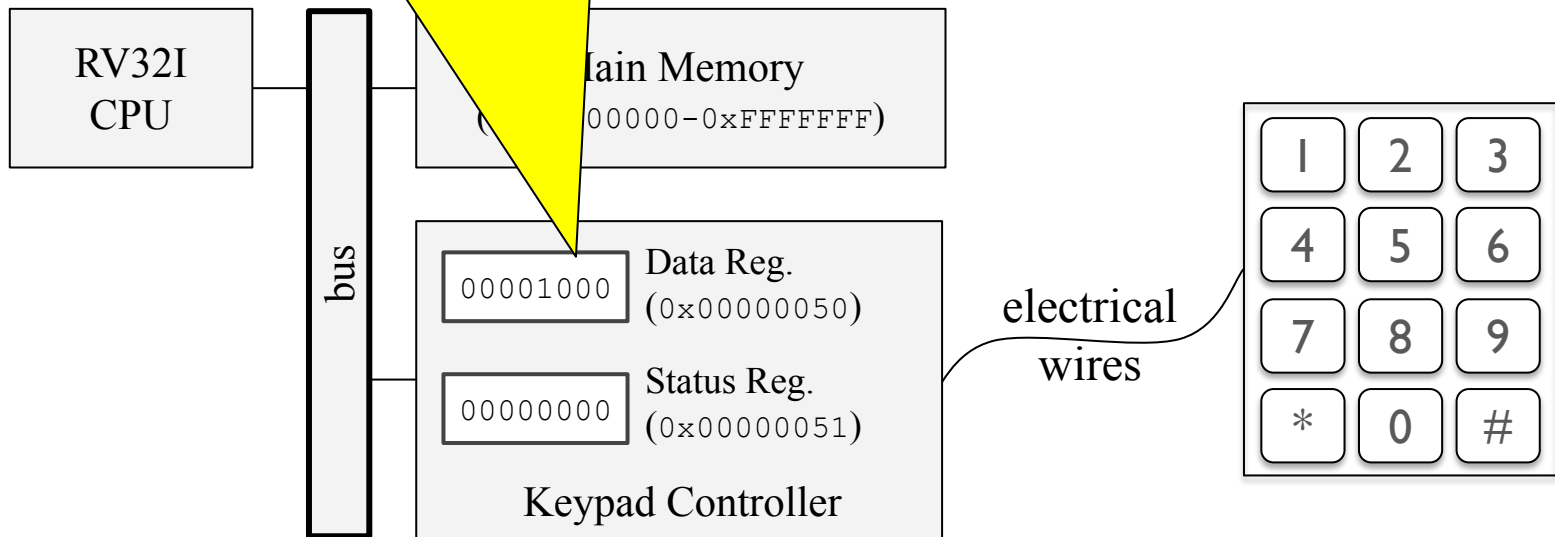
- Alguns eventos que ocorrem em periféricos exigem a atenção imediata da CPU.
- Para ilustrar este conceito, considere o seguinte sistema computacional:



Motivação

Quando uma tecla é pressionada, o Keypad Controller registra o código da tecla no Data Reg.

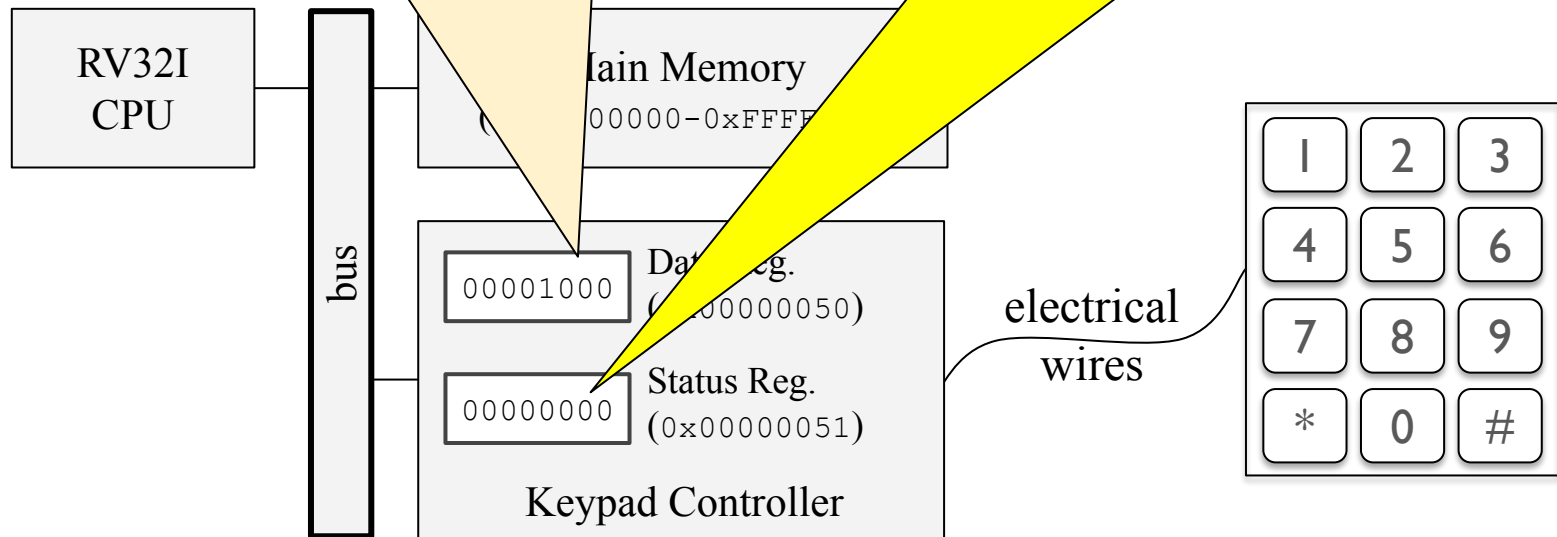
ocorrem em periféricos
mediata da CPU.
eito, considere o seguinte
:



Motivação

Quando uma tecla é pressionada, o Keypad Controller registra o código da tecla no Data Reg.

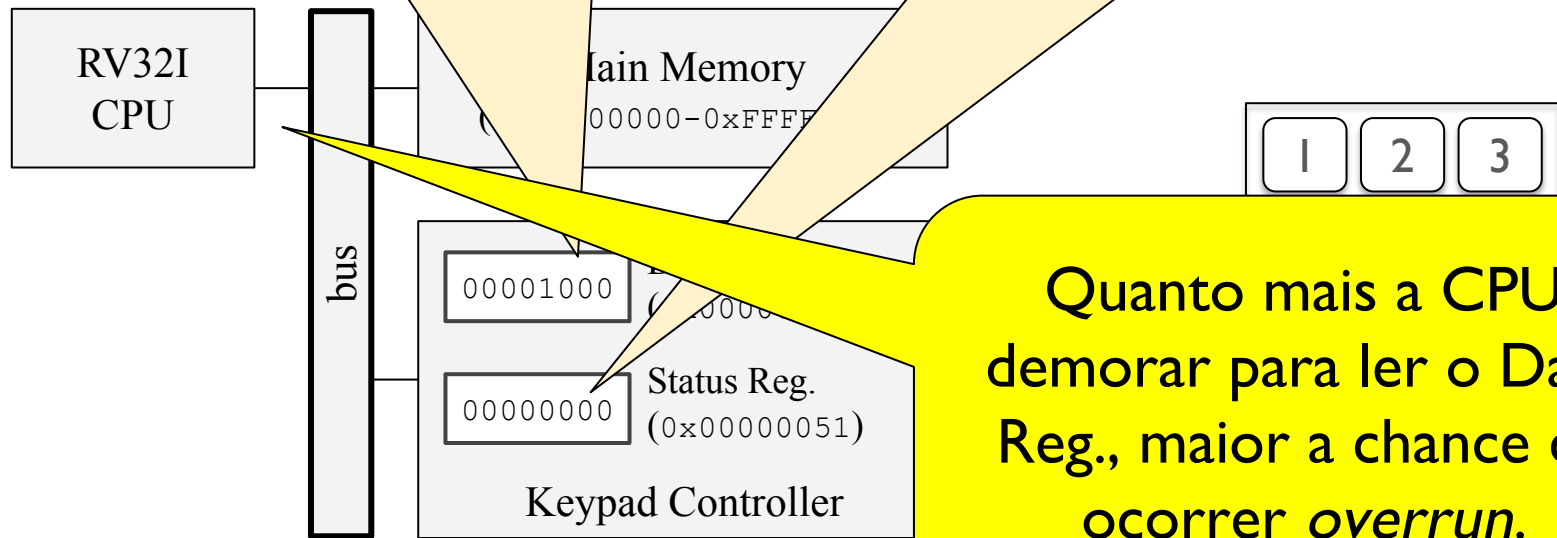
Se uma tecla for pressionada antes da CPU ter a chance de ler o último valor, então o Keypad Controller seta o *bit* OVRN (overrun) no Status Reg.



Motivação

Quando uma tecla é pressionada, o Keypad Controller registra o código da tecla no Data Reg.

Se uma tecla for pressionada antes da CPU ter a chance de ler o último valor, então o Keypad Controller seta o *bit* OVRN (*overflow*) no Status Reg.

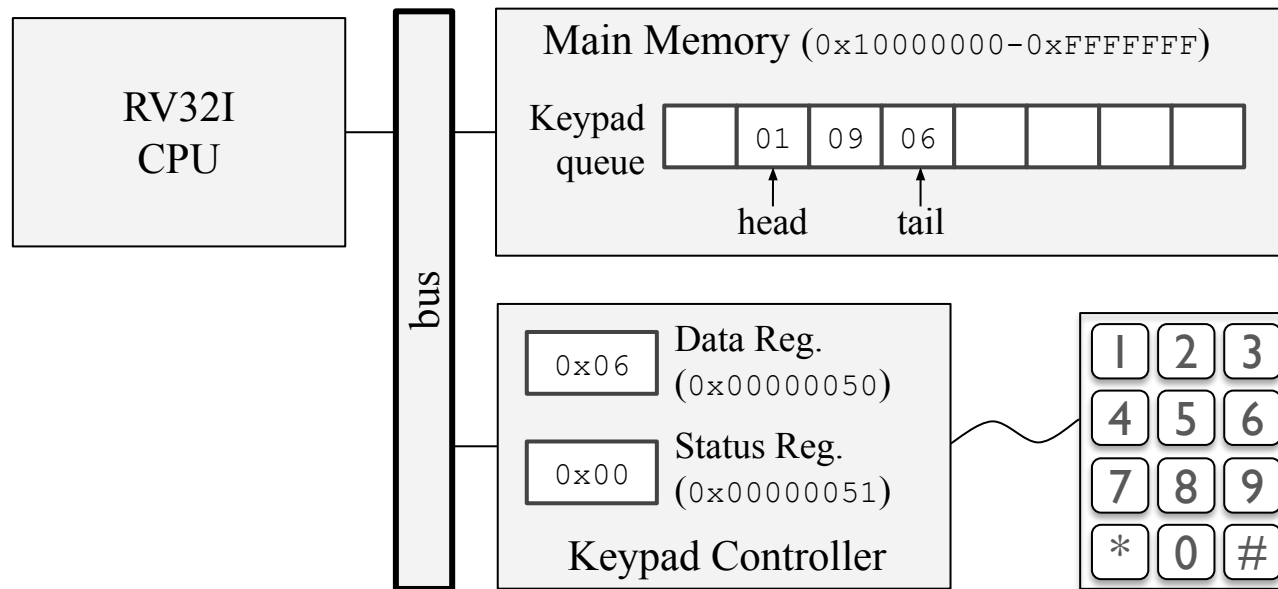


Quanto mais a CPU demorar para ler o Data Reg., maior a chance de ocorrer *overflow*.

Motivação

Estratégia: Assim que uma tecla for pressionada, fazer a CPU copiar o valor do Keypad Controller para um buffer na memória principal.

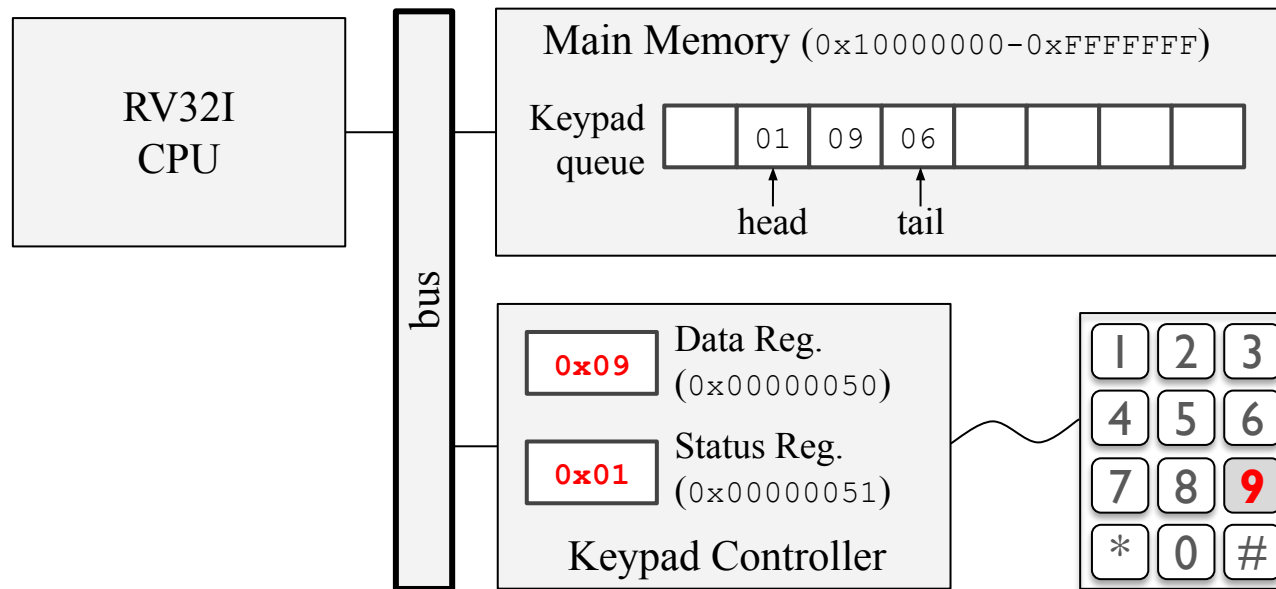
- Os programas do sistema passam a ler teclas do *buffer* em vez do Keypad Controller.



Motivação

Estratégia: Assim que uma tecla for pressionada, fazer a CPU copiar o valor do Keypad Controller para um buffer na memória principal.

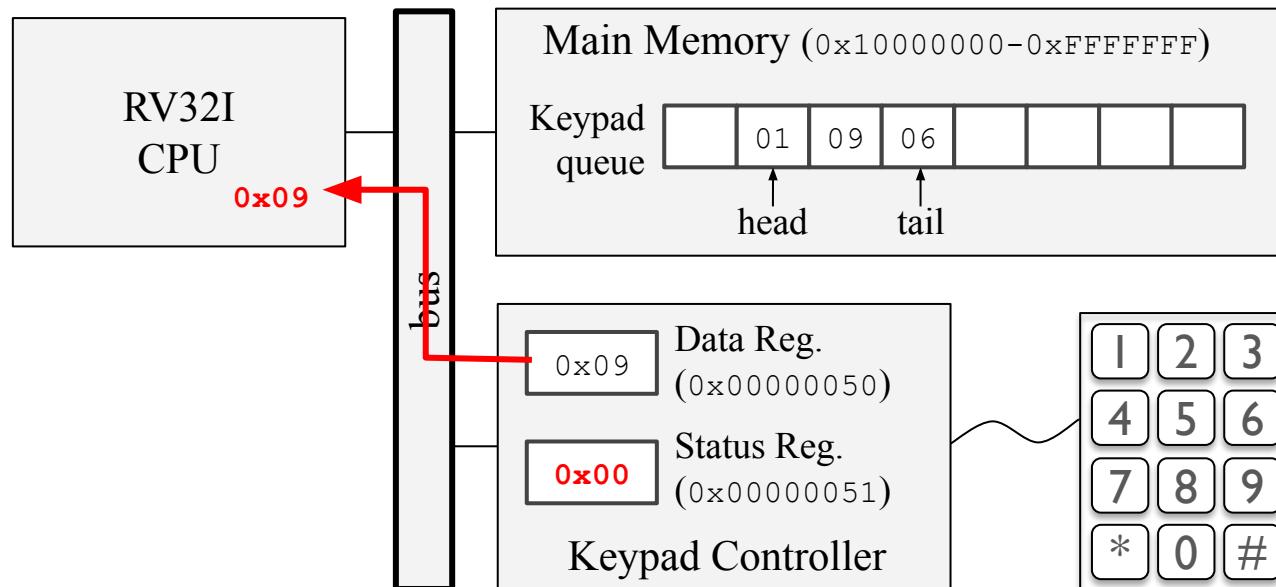
- Os programas do sistema passam a ler teclas do *buffer* em vez do Keypad Controller.



Motivação

Estratégia: Assim que uma tecla for pressionada, fazer a CPU copiar o valor do Keypad Controller para um buffer na memória principal.

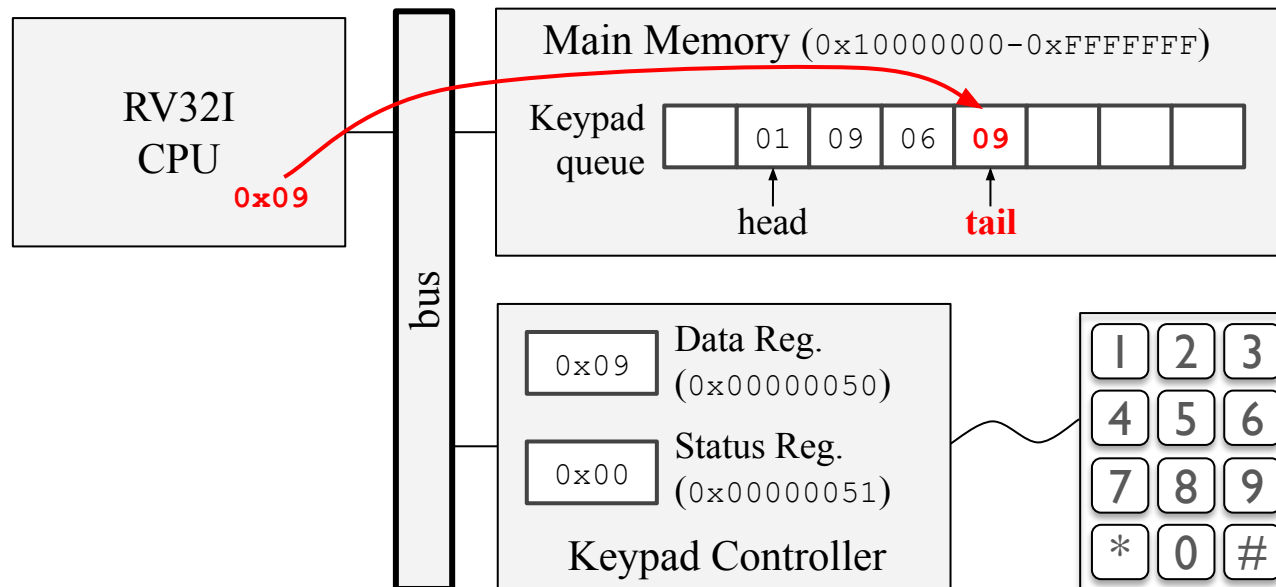
- Os programas do sistema passam a ler teclas do *buffer* em vez do Keypad Controller.



Motivação

Estratégia: Assim que uma tecla for pressionada, fazer a CPU copiar o valor do Keypad Controller para um buffer na memória principal.

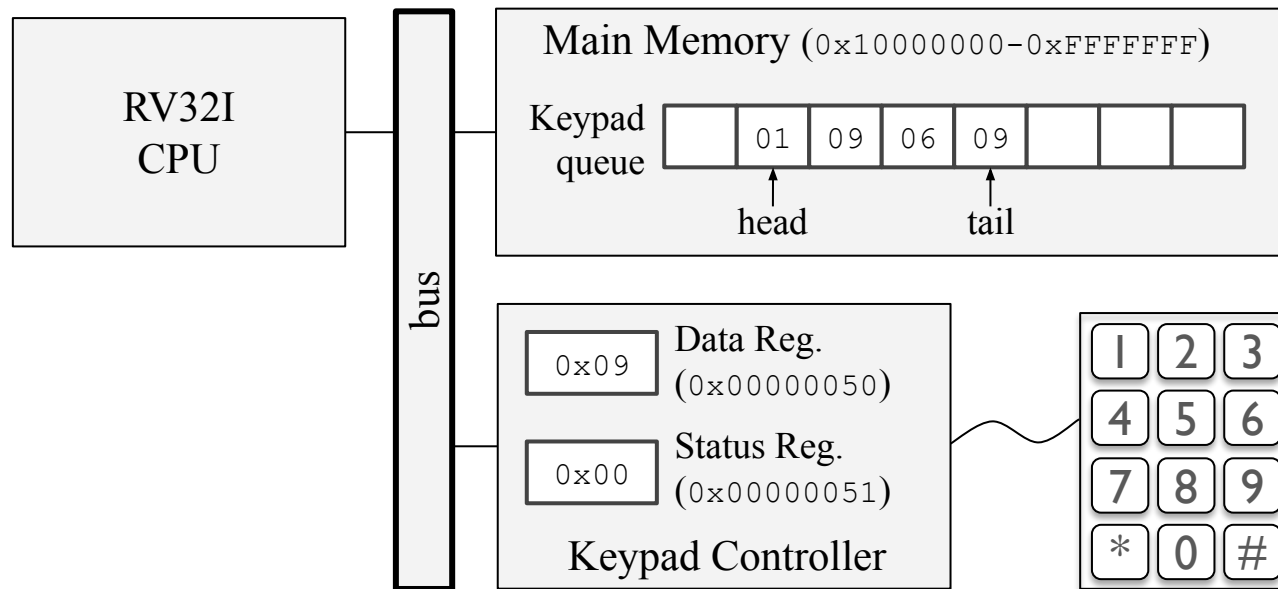
- Os programas do sistema passam a ler teclas do *buffer* em vez do Keypad Controller.



Motivação

Estratégia: Assim que uma tecla for pressionada, fazer a CPU copiar o valor do Keypad Controller para um buffer na memória principal.

- Os programas do sistema passam a ler teclas do *buffer* em vez do Keypad Controller.



Motivação

Importante: A atenção da CPU deve ser direcionada para tratar o evento do periférico.

Duas abordagens:

- Polling
- Interrupções Externas

Agenda

- Motivação
- **Polling**
- Interrupções Externas
- Interrupções Externas no RISC-V

Polling

Polling é um método no qual o programa é projetado para checar os periféricos periodicamente.

- Tratamento de periféricos com polling

Algorithm 3: Handling peripherals with polling.

```
1 while True do
2   | // Handle peripherals
3   | for p in Peripherals do
4   |   | if needsAttention(p) then
5   |   |   | handlePeripheral(p) ;
6   |   |   end
7   |   end
8   | PerformSomeComputation();
9 end
```

Polling

Polling é um método no qual o programa é projetado para checar os periféricos periodicamente.

- Exemplo com o teclado numérico

Algorithm 4: Handling the keypad with polling.

```
1 while True do
2   | if keypadPressed() then
3   |   |  $k \leftarrow \text{getKey}()$  ;
4   |   | pushKeyOnQueue(k) ;
5   | end
6   | Compute() ;
7 end
```

Polling

Polling é um método no qual o programa é projetado para checar os periféricos periodicamente.

- Exemplo com o teclado numérico

Algorithm 4: Handling the keypad with polling.

```
1 while True do
2   | if keypadPressed() then
3   |   |  $k \leftarrow \text{getKey}()$  ;
4   |   | pushKeyOnQueue(k) ;
5   | end
6   | Compute() ;
7 end
```

Quantidade de trabalho realizado pela rotina *Compute()* afeta a frequência com que a CPU verifica o teclado!

Agenda

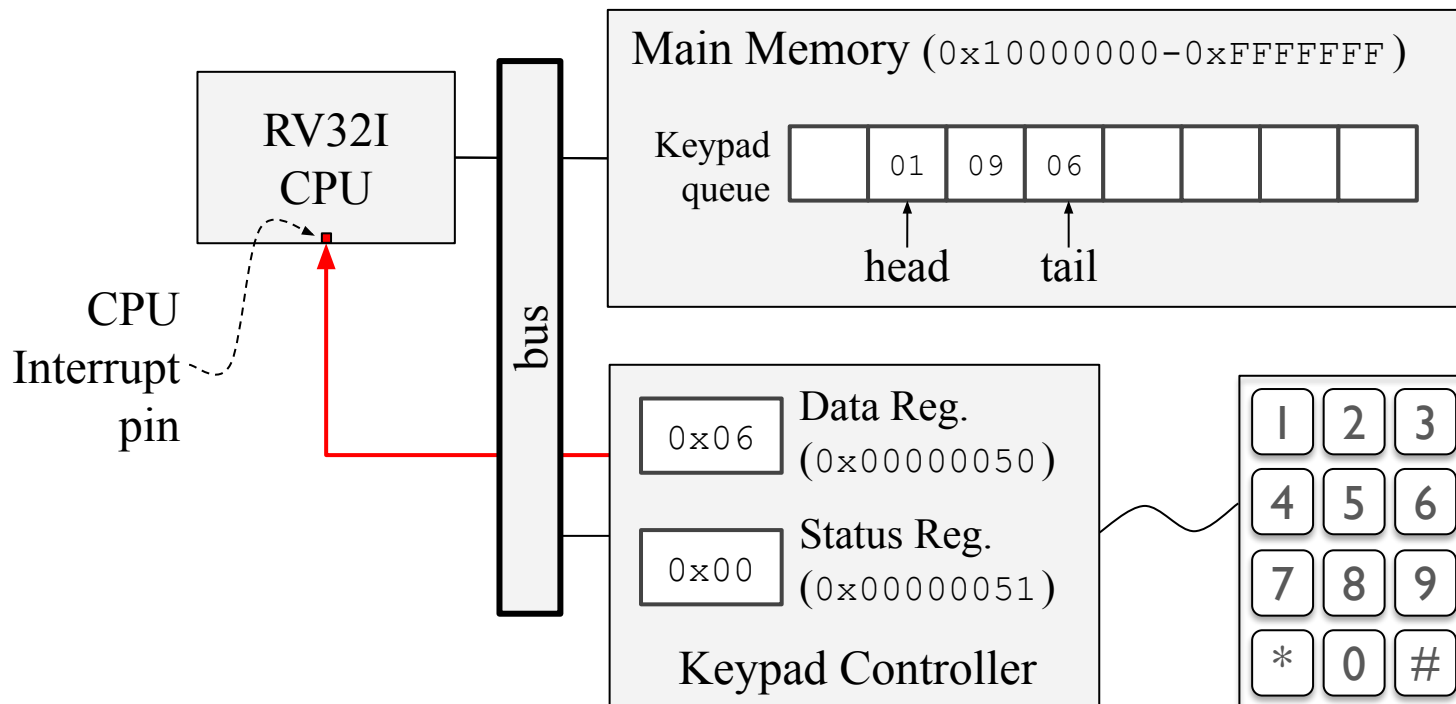
- Motivação
- Polling
- **Interrupções Externas**
- Interrupções Externas no RISC-V

Interrupções Externas

- **Interrupções Externas** são sinais enviados por dispositivos externos à CPU (periféricos) para informar que eles precisam de atenção!
- Exemplo:
 - Quando o teclado é pressionado, o controlador do teclado envia um sinal de interrupção para a CPU;
 - A CPU pára o que está fazendo para atender o teclado;
 - Após tratar a interrupção (p.ex: ler o dado do periférico), a CPU continua com o que estava fazendo.

Interrupções Externas

- **Interrupções Externas** são sinais enviados por dispositivos externos à CPU (periféricos) para informar que eles precisam de atenção!



Interrupções Externas

A CPU pára o que está fazendo para atender o teclado;

- O que acontece com o programa que estava sendo executado pela CPU?

Interrupções Externas

A CPU pára o que está fazendo para atender o teclado;

- O que acontece com o programa que estava sendo executado pela CPU?

```
# programa faça algo útil 1000 vezes
main:
    li a4, 1000
loop:
    jal  algo_util
    addi a4, a4, -1
    bnez a4, loop
    ...
```

Interrupções Externas

A CPU pára o que está fazendo para atender o teclado;

- O que acontece com o programa que estava sendo executado pela CPU?

```
# programa faça algo útil 1000 vezes
main:
    li a4, 1000
loop:
    jal algo_util
    addi a4, a4, -1
    bnez a4, loop
    ...
```

Interrupção

Interrupções Externas

A CPU pára o que está fazendo para atender o teclado;

- O que acontece com o programa que estava sendo executado pela CPU?
 - Antes de tratar a interrupção, é importante salvar todo o “contexto” do programa que está executando
 - Ex: Valores em registradores, preservar a pilha do programa, ...

Interrupções Externas: Fluxo de Tratamento

```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

```
loop:
```

```
    jal algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
    # salva contexto
```

```
    # trata a interrupção
```

```
    # restaura o contexto
```


Interrupções Externas: Fluxo de Tratamento


```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

1 Interrupção acontece

```
loop:
```

```
 1 jal  algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
    # salva contexto
```

```
    # trata a interrupção
```

```
    # restaura o contexto
```

Interrupções Externas: Fluxo de Tratamento

```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

```
loop:
```

```
    jal algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
    # salva contexto
```

```
    # trata a interrupção
```

```
    # restaura o contexto
```

1 Interrupção acontece

2 Fluxo de controle é desviado para a ISR

Interrupções Externas: Fluxo de Tratamento

```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

```
loop:
```

```
    jal algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
    # salva contexto
```

```
    # trata a interrupção
```

```
    # restaura o contexto
```

1 Interrupção acontece

2 Fluxo de controle é desviado para a ISR

ISR, ou *interrupt service routine*, é a rotina que trata a interrupção!

Interrupções Externas: Fluxo de Tratamento

```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

```
loop:
```

```
    jal algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
    3 salva contexto
```

```
    # trata a interrupção
```

```
    # restaura o contexto
```

1 Interrupção acontece

2 Fluxo de controle é desviado para a ISR

3 Contexto é salvo

Interrupções Externas: Fluxo de Tratamento

```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

```
loop:
```

```
1 jal algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
3 salva contexto
```

```
4 trata a interrupção
```

```
# restaura o contexto
```

- 1 Interrupção acontece
- 2 Fluxo de controle é desviado para a ISR
- 3 Contexto é salvo
- 4 A interrupção é tratada

Interrupções Externas: Fluxo de Tratamento

```
# programa faça algo útil 1000 vezes
```

```
main:
```

```
    li a4, 1000
```

```
loop:
```

```
  ① jal  algo_util
```

```
    addi a4, a4, -1
```

```
    bnez a4, loop
```

```
    ...
```

```
int_service_routine:
```

```
  ③ salva contexto
```

```
  ④ trata a interrupção
```

```
  ⑤ restaura o contexto
```

① Interrupção acontece

② Fluxo de controle é desviado para a ISR

③ Contexto é salvo

④ A interrupção é tratada

⑤ Contexto é recuperado

Detectando interrupções externas

- A CPU contém um ou mais pinos de entrada para receber sinais de interrupção.
- A unidade de controle da CPU monitora estes pinos. Exemplo:

```
1 while True do  
2   | // Fetch instruction and update PC ;  
3   | IR ← MainMemory[PC] ;  
4   | PC ← PC+4;  
5   | ExecuteInstruction(IR);  
6 end
```

Detectando interrupções externas

- A CPU contém um ou mais pinos de entrada para receber sinais de interrupção.
- A unidade de controle da CPU monitora estes pinos. Exemplo:

```
1 while True do
2     // Check for interrupts
3     if (interrupt_pin = '1') and (interrupts_enabled = '1') then
4         // Invoke the ISR
5         SAVED_PC ← PC ;
6         PC ← ISR_ADDRESS;
7         interrupts_enabled ← '0';
8     end
9     // Fetch instruction and update PC
10    IR ← MainMemory[PC] ;
11    PC ← PC+4;
12    ExecuteInstruction(IR);
13 end
```

Chamando a ISR adequada

- O sistema pode conter diversos periféricos que geram interrupções.
 - Cada um tem sua própria ISR
- Como (i) determinar qual periférico interrompeu a CPU e (ii) invocar a rotina de tratamento adequada?

Chamando a ISR adequada

- O sistema pode conter diversos periféricos que geram interrupções.
 - Cada um tem sua própria ISR
- Como (i) determinar qual periférico interrompeu a CPU e (ii) invocar a rotina de tratamento adequada?
 - Existem diversas abordagens.
 - Discutiremos 3 abordagens:
 - SW-only
 - HW/SW
 - HW-only

Chamando a ISR adequada

Abordagem **SW-only**

- A CPU invoca uma ISR genérica, que é responsável por (i) determinar qual periférico interrompeu a CPU e (ii) invocar a rotina de tratamento adequada.
- Como não tem suporte de HW, a ISR genérica interage com os periféricos para determinar qual periférico interrompeu a CPU
- Vantagem:
 - Simplifica o projeto do HW da CPU
- Desvantagem:
 - Desempenho do mecanismo de interrupções pode ser muito ruim (principalmente se houver muitos periféricos e se alguns deles forem lentos!)

Chamando a ISR adequada

Abordagem **SW-only**

Exemplo de implementação da lógica de controle da CPU

```
1 while True do
2     // Check for interrupts
3     if (interrupt_pin = '1') and (interrupts_enabled = '1') then
4         // Invoke the ISR
5         SAVED_PC ← PC ;
6         PC ← ISR_ADDRESS;
7         interrupts_enabled ← '0';
8     end
9     // Fetch instruction and update PC
10    IR ← MainMemory[PC] ;
11    PC ← PC+4;
12    ExecuteInstruction(IR);
13 end
```

Chamando a ISR adequada

Abordagem **HW/SW**

- A CPU invoca uma ISR genérica, que é responsável por (i) determinar qual periférico interrompeu a CPU e (ii) invocar a rotina de tratamento adequada.
- O HW provê um suporte para a ISR genérica descobrir qual periférico interrompeu a CPU
 - P.ex: Escreve um número que identifica o periférico em um registrador
- Vantagem:
 - Desempenho do mecanismo de interrupções não depende do número de periféricos nem da velocidade deles.
- Desvantagem:
 - O HW da CPU precisa incluir mais funcionalidades

Chamando a ISR adequada

Abordagem **HW/SW**

Exemplo de implementação da lógica de controle da CPU

```
1 while True do
2     // Check for interrupts
3     if (interrupt_pin = '1') and (interrupts_enabled = '1') then
4         // Invoke the ISR
5         SAVED_PC ← PC ;
6         INT_CAUSE ← Peripheral ID ;
7         PC ← ISR_ADDRESS;
8         interrupts_enabled ← '0';
9     end
10    // Fetch instruction and update PC
11    IR ← MainMemory[PC] ;
12    PC ← PC+4;
13    ExecuteInstruction(IR);
14 end
```

Chamando a ISR adequada

Abordagem **HW-only**

- A CPU determina a causa da interrupção e invoca a ISR adequada automaticamente.
 - A ISR é invocada com a ajuda de uma tabela que é armazenada na memória principal (*interrupt vector table*)
- Vantagem:
 - Melhor desempenho entre as abordagens.
- Desvantagem:
 - O HW da CPU se torna mais complexo.

Chamando a ISR adequada

Abordagem **HW-only**

Exemplo de implementação da lógica de controle da CPU

```
1 while True do
2     // Check for interrupts
3     if (interrupt_pin = '1') and (interrupts_enabled = '1') then
4         // Save the previous PC
5         SAVED_PC ← PC ;
6         // Retrieve the ISR address from the interrupt vector table and set PC
7         PC ← MainMemory[INT_TABLE_BASE + INTERRUPT_ID × 4];
8         interrupts_enabled ← '0' ;
9     end
10    // Fetch instruction and update PC
11    IR ← MainMemory[PC] ;
12    PC ← PC+4;
13    ExecuteInstruction(IR);
14 end
```

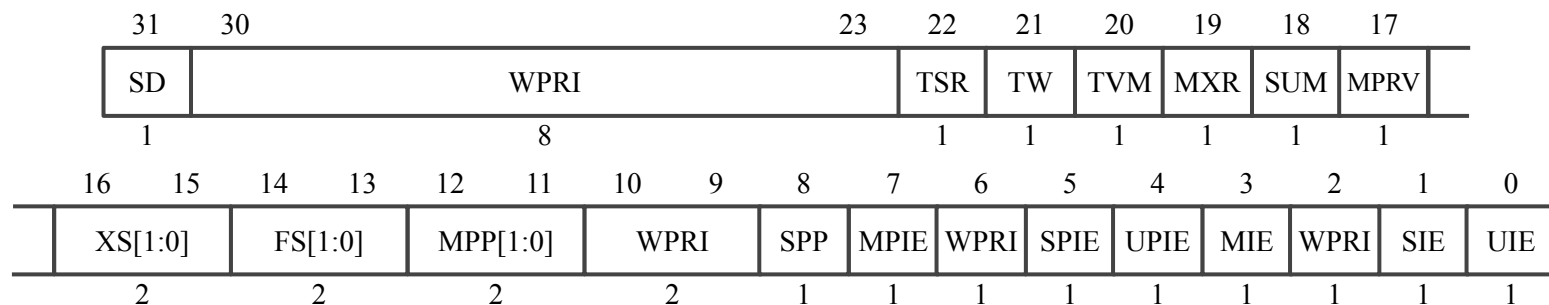

Agenda

- Motivação
- Polling
- Interrupções Externas
- **Interrupções Externas no RISC-V**
 - **Control and Status Registers (CSRs)**
 - Fluxo de tratamento de interrupções na CPU
 - Implementação de ISRs
 - Configuração do mecanismo de interrupções

Control and Status Registers (CSRs)

Além dos registradores de propósito geral ($\times 0 - \times 31$), a ISA do RISC-V contém um conjunto de registradores especiais chamados *Control and Status Registers* (CSR).

- Permite que o *software* configure/controle a CPU e expõem estados internos da CPU para o *software*.
- No RV32 os CSRs possuem 32 *bits*.
- Exemplo: `mstatus`



Control and Status Registers (CSRs)

Leitura e escritas nos CSRs são realizadas com instruções especiais: `csrr`, `csrw`, `csrrw`.

- Exemplo 1: copiar o conteúdo do CSR `mstatus` para `t0`.

```
csrr t0, mstatus
```

- Exemplo 2: copiar o conteúdo em `t0` para o CSR `mstatus`.

```
csrw mstatus, t0
```

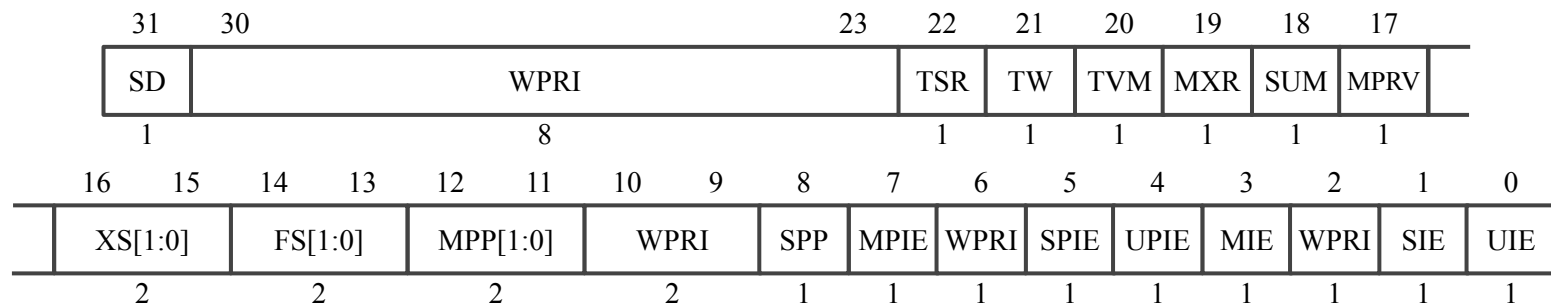
- Exemplo 3: trocar o conteúdo de `t0` com o conteúdo do CSR `mscratch`.

```
csrrw t0, mscratch, t0
```

Control and Status Registers (CSRs)

Alguns CSRs são organizados em campos.

- **Exemplo:** `mstatus`



- **Notação:** `mstatus.MIE` = Campo MIE (*bit 3*) do registrador `mstatus`.
- **Escrever '1' em `mstatus.MIE`:**

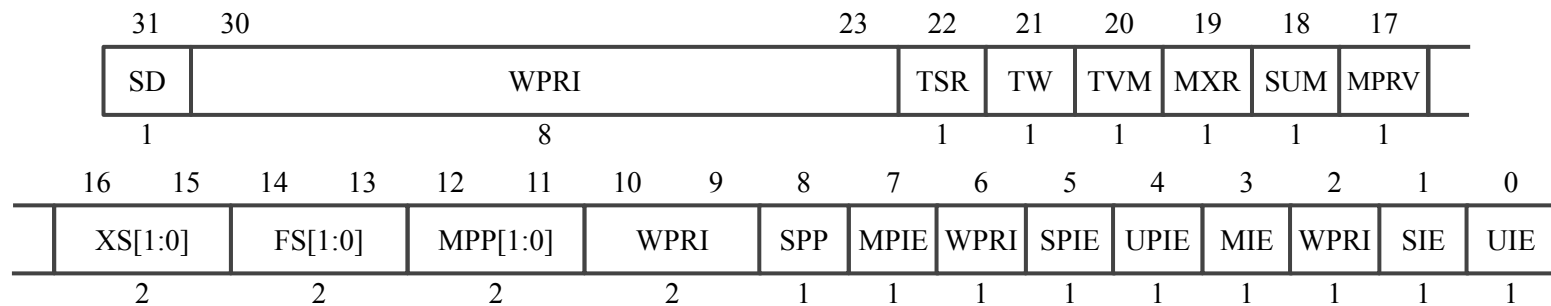
```
csrr t0, mstatus
ori t0, t0, 0x8
csrw mstatus, t0
```

```
li t0, 0x8
csrs mstatus, t0
```

Control and Status Registers (CSRs)

Alguns CSRs são organizados em campos.

- **Exemplo:** `mstatus`



- **Notação:** `mstatus.MIE` = Campo MIE (*bit 3*) do registrador `mstatus`.
- **Escrever '0'** em `mstatus.MIE`:

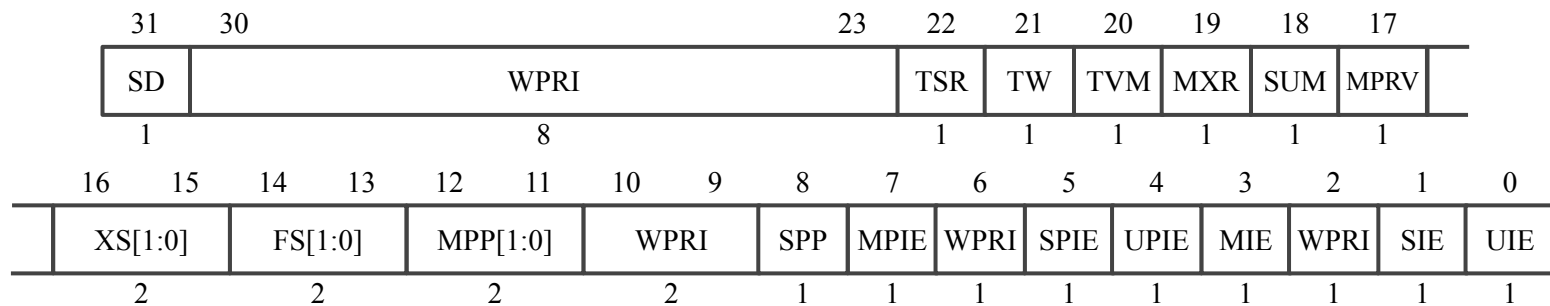
```
csrr t0, mstatus
li t1, ~0x8
and t0, t0, t1
csrw mstatus, t0
```

```
li t0, 0x8
csrc mstatus, t0
```

Control and Status Registers (CSRs)

Alguns CSRs são organizados em campos.

- Exemplo: `mstatus`



- Notação: `mstatus.MIE` = Campo MIE (*bit 3*) do registrador `mstatus`
- Escrever '0' em `mst`

`~0x8 = 0xFFFFFFFF7`

```
csrr t0, mstatus
li t1, ~0x8
and t0, t0, t1
csrw mstatus, t0
```

```
li t0, 0x8
csrc mstatus, t0
```

Control and Status Registers (CSRs)

CSRs ligados ao mecanismo de interrupções externas no modo Machine.

- `mstatus`: campos **MIE**, **MPIE**, e **MPP**
 - `mstatus.MIE` (*bit 3*): Habilita (1) ou desabilita (0) o tratamento de interrupções pela CPU. Ele é automaticamente setado com 0 quando a CPU trata uma interrupção.
 - `mstatus.MPIE` (*bit 7*): Salva o valor anterior de `mstatus.MIE` quando uma interrupção é tratada pela CPU.
 - `mstatus.MPP` (*bits 11-12*): Quando uma interrupção ocorre, o modo de operação pode ser modificado. Neste caso, o modo anterior é salvo neste campo.

Control and Status Registers (CSRs)

CSRs ligados ao mecanismo de interrupções externas no modo Machine.

- `mcause`: campos **INTERRUPT** e **EXCCODE**
 - `mcause . INTERRUPT` (*bit 31*): indica se a interrupção foi causada por uma interrupção (1) or por uma exceção (0).
 - `mcause . EXCCODE` (*bits 0-30*): indica a causa da interrupção. Interrupções externas são indicadas com o valor 0xB neste campo.

Control and Status Registers (CSRs)

CSRs ligados ao mecanismo de interrupções externas no modo Machine.

- `mtvec`: campos **MODE** e **BASE**
 - `mtvec.BASE` (*bits 2-31*): contém um endereço que será utilizado para invocar a ISR.
 - `mtvec.MODE` (*bits 0-1*): indica a forma de se identificar o endereço da ISR.
 - *Direct mode* (MODE = 00): O endereço da ISR é determinado diretamente pelo valor em `mtvec.BASE`.
 - *Vectored mode* (MODE = 01): O endereço da ISR é determinado pela expressão `mtvec.BASE + (4 x mcause.EXCCODE)`

Control and Status Registers (CSRs)

CSRs ligados ao mecanismo de interrupções externas no modo Machine.

- `mie`: campo MEIE
 - `mie.MEIE` (*bit 11*): habilita (1) ou desabilita (0) o tratamento de interrupções externas pela CPU.
- `mip`: campo MEIP
 - `mip.MEIP` (*bit 11*): indica se há uma interrupção externa pendente (solicitada mas que não foi atendida ainda).

Control and Status Registers (CSRs)

CSRs ligados ao mecanismo de interrupções externas no modo Machine.

- `mepc`: quando uma interrupção ocorre, a CPU salva o valor do registrador PC em `mepc` antes de setar PC com o endereço da ISR.

Control and Status Registers (CSRs)

CSRs ligados ao mecanismo de interrupções externas no modo Machine.

- `mscratch`: é um registrador auxiliar que pode ser usado para facilitar a implementação da ISR.
 - Discutiremos como utilizá-lo para salvar e recuperar o contexto.

Agenda

- Motivação
- Polling
- Interrupções Externas
- **Interrupções Externas no RISC-V**
 - Control and Status Registers (CSRs)
 - **Fluxo de tratamento de interrupções na CPU**
 - Implementação de ISRs
 - Configuração do mecanismo de interrupções

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita  
| IR <= MainMemory[PC]  
| PC <= PC + 4  
| ExecuteInstruction(IR)
```

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
| se mstatus.MIE = `1' então
| | se mip.MEIP = `1' e mie.MEIE = `1' então
| | | mstatus.MPIE <= mstatus.MIE
| | | mstatus.MIE <= `0'
| | | mepc <= pc
| | | mcause.INTERRUPT <= `1'
| | | mcause.EXCCODE <= `0xB'
| | | se mtvec.MODE = '0' então
| | | | PC <= mtvec.BASE
| | | senão
| | | | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
| IR <= MainMemory[PC]
| PC <= PC + 4
| ExecuteInstruction(IR)
```

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
  se mstatus.MIE = `1' então
    se mip.MEIP = `1' e mpie.MEIE = `1' então
      mstatus.MPIE <= mstatus.MIE
      mstatus.MIE <= `0'
      mepc <= pc
      mcause.interrupt <= `1'
      mcause.EXCCODE <= `0xB'
      se mtvec.MODE = '0' então
        | PC <= mtvec.BASE
      senão
        | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
      IR <= MainMemory[PC]
      PC <= PC + 4
      ExecuteInstruction(IR)
```


Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
  se mstatus.MIE = `1' então
    se mip.MEIP = `1' e mpie.MEIE = `1' então
      mstatus.MPIE <= mstatus.MIE
      mstatus.MIE <= `0'
      mepc <= pc
      mcause.interrupt <= `1'
      mcause.EXCCODE <= `0xB'
      se mtvec.MODE = '0' então
        | PC <= mtvec.BASE
      senão
        | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
  IR <= MainMemory[PC]
  PC <= PC + 4
  ExecuteInstruction(IR)
```

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
| se mstatus.MIE = `1' então
| | se mip.MEIP = `1' e mpie.MEIE = `1' então
| | | mstatus.MPIE <= mstatus.MIE
| | | mstatus.MIE <= `0'
| | | mepc <= pc
| | | mcause.interrupt <= `1'
| | | mcause.EXCCODE <= `0xB'
| | | se mtvec.MODE = '0' então
| | | | PC <= mtvec.BASE
| | | senão
| | | | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
| IR <= MainMemory[PC]
| PC <= PC + 4
| ExecuteInstruction(IR)
```

Salva parte do contexto!

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
| se mstatus.MIE = `1' então
| | se mip.MEIP = `1' e mpie.MEIE = `1' então
| | | mstatus.MPIE <= mstatus.MIE
| | | mstatus.MIE <= `0'
| | | mepc <= pc
| | | mcause.interrupt <= `1'
| | | mcause.EXCCODE <= `0xB'
| | | se mtvec.MODE = '0' então
| | | | PC <= mtvec.BASE
| | | senão
| | | | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
| IR <= MainMemory[PC]
| PC <= PC + 4
| ExecuteInstruction(IR)
```

Registra a causa da interrupção

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
| se mstatus.MIE = `1' então
| | se mip.MEIP = `1' e mpie.MEIE = `1' então
| | | mstatus.MPIE <= mstatus.MIE
| | | mstatus.MIE <= `0'
| | | mepc <= pc
| | | mcause.interrupt <= `1'
| | | mcause.EXCCODE <= `0xB'
| | | se mtvec.MODE = '0' então
| | | | PC <= mtvec.BASE
| | | | senão
| | | | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
| | IR <= MainMemory[PC]
| | PC <= PC + 4
| | ExecuteInstruction(IR)
```

Escreve o endereço da ISR em PC

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

```
repita
  se mstatus.MIE = `1' então
    se mip.MEIP = `1' e mpie.MEIE = `1' então
      mstatus.MPIE <= mstatus.MIE
      mstatus.MIE <= `0'
      mepc <= pc
      mcause.interrupt <= `1'
      mcause.EXCCODE <= `0xB'
      se mtvec.MODE = '0' então
        | PC <= mtvec.BASE
      senão
        | PC <= mtvec.BASE + (4 x mcause.EXCCODE)
      IR <= MainMemory[PC]
      PC <= PC + 4
      ExecuteInstruction(IR)
```

Prossigue com o ciclo normal de execução de instruções.

Fluxo de tratamento de interrupções na CPU

Ciclo de execução de instruções

- Parte do contexto já foi salvo automaticamente pela CPU (Ex: PC e `mstatus.MIE`)
- O restante deve ser salvo pela ISR!

Agenda

- Motivação
- Polling
- Interrupções Externas
- **Interrupções Externas no RISC-V**
 - Control and Status Registers (CSRs)
 - Fluxo de tratamento de interrupções na CPU
 - **Implementação de ISRs**
 - Configuração do mecanismo de interrupções

Implementação de ISRs

Exemplo de código com uma ISR (`main_isr`):

```
# Main Interrupt Service Routine
main_isr:
    # Salva contexto
    # Trata interrupção
    # Recupera contexto

# Configuração do sistema durante a operação de reset
_start:
    # Registrar a ISR
    la    t0, main_isr    # Grava o endereço da ISR principal
    csrw mtvec, t0       # no registrador mtvec
    ...
```


Implementação de ISRs

No início da rotina de tratamento de interrupção:

- O valor do PC anterior foi salvo em MEPC;
- No entanto, os registradores $\times 1$ a $\times 31$ possuem valores do programa que estava executando e também devem ser preservados!
- **Onde salvar estes valores?**

Implementação de ISRs

No início da rotina de tratamento de interrupção:

- O valor do PC anterior foi salvo em MEPC;
- No entanto, os registradores $\times 1$ a $\times 31$ possuem valores do programa que estava executando e também devem ser preservados!
- **Onde salvar estes valores? Na memória!**

Implementação de ISRs

No início da rotina de tratamento de interrupção:

- O valor do PC anterior foi salvo em MEPC;
- No entanto, os registradores $\times 1$ a $\times 31$ possuem valores do programa que estava executando e também devem ser preservados!
- Onde salvar estes valores? Na memória!
 - **No RISC-V, instruções de store precisam de um registrador com um endereço de memória.**
 - Como iniciar um registrador com um endereço de memória sem perder dados?

Implementação de ISRs

Como iniciar um registrador com um endereço de memória sem perder dados?

- Na inicialização do sistema: Fazer `mscratch` apontar para uma posição de memória que possa ser usada pela ISR.
- Na ISR: Trocar o conteúdo de `mscratch` por um registrador de propósito geral no início e no final da ISR.
 - Em nosso exemplo suporemos que `mscratch` aponta para uma pilha dedicada às ISRs.

Implementação de ISRs

Usando o `mscratch` para auxiliar a ISR a salvar e recuperar o contexto

```
main_isr:
  # Salvar o contexto
  csrrw sp, mscratch, sp # Troca sp com mscratch
  addi sp, sp, -64      # Aloca espaço na pilha da ISR
  sw a0, 0(sp)         # Salva a0
  sw a1, 4(sp)        # Salva a1
  ...
  # Trata a interrupção
  ...
  # Recupera o contexto
  ...
  lw a1, 4(sp)          # Recupera a1
  lw a0, 0(sp)          # Recupera a0
  addi sp, sp, 64      # Desaloca espaço da pilha da ISR
  csrrw sp, mscratch, sp # Troca sp com mscratch novamente
  mret                 # Retorna da interrupção
```

Fazer SP apontar para a pilha da ISR e salvar o contexto na pilha da ISR!

Implementação de ISRs

Usando o `mscratch` para auxiliar a ISR a salvar e recuperar o contexto

```
main_isr:
  # Salvar o contexto
  csrrw sp, mscratch, sp # Troca sp com mscratch
  addi  sp, sp, -64      # Aloca espaço na pilha da ISR
  sw   a0, 0(sp)        # Salva a0
  sw   a1, 4(sp)        # Salva a1
  ...
  # Trata a interrupção
  ...
  # Recupera o contexto
  ...
  lw   a1, 4(sp)        # Recupera a1
  lw   a0, 0(sp)        # Recupera a0
  addi  sp, sp, 64      # Desaloca espaço da pilha da ISR
  csrrw sp, mscratch, sp # Troca sp com mscratch novamente
  mret                    # Retorna da interrupção
```

Tratar a interrupção.

Implementação de ISRs

Usando o `mscratch` para auxiliar a ISR a salvar e recuperar o contexto

```
main_isr:
  # Salvar o contexto
  csrrw sp, mscratch, sp # Troca sp com mscratch
  addi sp, sp, -64      # Aloca espaço da pilha da ISR
  sw a0, 0(sp)         # Salva a0
  sw a1, 4(sp)         # Salva a1
  ...
  # Trata a interrupção
  ...
  # Recupera o contexto
  ...
  lw a1, 4(sp)        # Recupera a1
  lw a0, 0(sp)        # Recupera a0
  addi sp, sp, 64     # Desaloca espaço da pilha da ISR
  csrrw sp, mscratch, sp # Troca sp com mscratch novamente
  mret                 # Retorna da interrupção
```

Recuperar o contexto do programa anterior:

I - Recuperar os registradores de propósito geral e desalocar o espaço da pilha da ISR.

Implementação de ISRs

Usando o `mscratch` para auxiliar a ISR a salvar e recuperar o contexto

```
main_isr:
    # Salvar o contexto
    csrrw sp, mscratch, sp # Troca sp com mscratch
    addi sp, sp, -64      # Aloca espaço da pilha da ISR
    sw a0, 0(sp)         # Salva a0
    sw a1, 4(sp)         # Salva a1
    ...
    # Trata a interrupção
    ...
    # Recupera o contexto
    ...
    lw a1, 4(sp)         # Recupera a1
    lw a0, 0(sp)         # Recupera a0
    addi sp, sp, 64      # Desaloca espaço da pilha da ISR
    csrrw sp, mscratch, sp # Troca sp com mscratch novamente
    mret                 # Retorna da interrupção
```

**Recuperar o contexto do programa anterior:
2 - Recuperar o SP do programa**

Implementação de ISRs

Usando o `mscratch` para auxiliar a ISR a salvar e recuperar o contexto

```
main_isr:
    # Salvar o contexto
    csrrw sp, mscratch, sp # Troca sp com mscratch
    addi sp, sp, -64      # Aloca espaço na pilha
    sw a0, 0(sp)         # Salva a0
    sw a1, 4(sp)         # Salva a1
    ...
    # Trata a interrupção
    ...
    # Recupera o contexto
    ...
    lw a1, 4(sp)         # Recupera a1
    lw a0, 0(sp)         # Recupera a0
    addi sp, sp, 64      # Desaloca espaço da pilha da ISR
    csrrw sp, mscratch, sp # Troca sp com mscratch novamente
    mret                # Retorna da interrupção
```

Recuperar o contexto do programa anterior:

3 - Recuperar o contexto que foi salvo automaticamente pela CPU (`mstatus.MIE`, `pc`, ...)

Implementação de ISRs

A instrução `mret`:

- Instrução especial utilizada para retornar de ISRs.
- Recupera o estado que foi salvo automaticamente pela CPU quando a interrupção foi tratada.
 - `pc <= mepc`
 - `mstatus.MIE <= mstatus.MPIE`
 - ...

Implementação de ISRs

Tratando a interrupção

Uma vez que o contexto está salvo, é necessário identificar a causa da interrupção: valor no CSR `mcause`

mcause		Descrição
INTERRUPT	EXCCODE	
1	1	Interrupção de software de supervisor
1	3	Interrupção de software da máquina
...
1	11	Interrupção externa da máquina
0	0	Endereço de instrução desalinhado
0	1	Falha de acesso à instrução
0	2	Instrução inválida
...
0	5	Falha de leitura (load)
...
0	8	Chamada de sistema (ecall)
...

Implementação de ISRs

Tratando a interrupção

Uma vez que o contexto está salvo, é necessário identificar a causa da interrupção: valor no CSR `mcause`

```
main_isr:
    ...
    # Trata a interrupção
    csrr a1, mcause          # lê a causa da interrupção
    bgez a1, handle_exc     # Verifica se é exceção ou int.
    andi a1, a1, 0x3f       # Isola a causa de interrupção
    li    a2, 11            # a2 = interrupção externa
    bne a1, a2, otherInt    # desvia se não for interrupção
                            # externa
    # Trata interrupção externa
    jal external_isr
    ...
```

Implementação de ISRs

Tratando a interrupção

Uma vez que o contexto está salvo, é necessário identificar a causa da interrupção: valor no CSR `mcause`

- Podemos colocar os endereços da ISRs em uma tabela indexada pelo EXCCODE.

```
interrup_isr_table:
    .word user_sw_int          # EXCCODE = 0
    .word supervisor_sw_isr  # EXCCODE = 1
    .word unimp_isr          # EXCCODE = 2
    .word unimp_isr          # EXCCODE = 3
    ...
    .word external_isr       # EXCCODE = 11
```

Implementação de ISRs

Tratando a interrupção

Uma vez que o contexto está salvo, é necessário identificar a causa da interrupção: valor no CSR `mcause`

```
main_isr:
    ...
    csrr a1, mcause          # lê a causa da interrupção
    bgez a1, handle_exc     # Verifica se é exceção ou int.
    andi a1, a1, 0x3f       # Isola EXCCODE
    slli a1, a1, 2          # a1 = EXCCODE x 4
    la   a0, interrup_isr_table
    add  a0, a0, a1
    lw   a0, (a0)           # a0 = interrupt_isr_table[EXCCODE]
    jalr a0                 # Invoca o tratador correto
    ...
```

Fluxo de tratamento de interrupções na CPU

Em suma, quando uma interrupção é tratada:

- O *hardware* (CPU) automaticamente
 - salva parte do contexto em registradores especiais (`mepc`, `mstatus.MPIE`, ...);
 - desvia o fluxo de execução de acordo com o modo de operação em `mtvec.MODE` e o endereço em `mtvec.BASE`
- O *software* (código da ISR) deve
 - salvar o restante do contexto;
 - tratar a interrupção;
 - recuperar o contexto.
 - Contexto salvo pela CPU é recuperado com a instrução `mret`!

Agenda

- Motivação
- Polling
- Interrupções Externas
- **Interrupções Externas no RISC-V**
 - Control and Status Registers (CSRs)
 - Fluxo de tratamento de interrupções na CPU
 - Implementação de ISRs
 - **Configuração do mecanismo de interrupções**

Configuração do mecanismo de interrupções

Durante a inicialização do sistema (*boot*), o mecanismo de interrupções deve ser configurado. Para isso, o código da rotina de inicialização precisa:

1. Registrar a ISR;
2. Configurar o registrador `mscratch` para apontar para a pilha da ISR;
3. Configurar os periféricos; e
4. Habilitar as interrupções.

Configuração do mecanismo de interrupções

Registrando a ISR (*direct mode*)

- Para registrar a ISR no modo *direct*, basta escrever o endereço da ISR no registrador `mtvec`.
 - Como o código da ISR começa em um endereço múltiplo de 4, os dois últimos *bits* do endereço são zero (00). Logo, `mtvec.MODE = 00`

```
la    t0, main_isr    # Carrega o endereço da main_isr
csrw  mtvec, t0       # em mtvec
```

Configuração do mecanismo de interrupções

Registrando a ISR (*vectored mode*)

- Para registrar a ISR no modo *vectored*, carregamos o endereço da tabela em um registrador, setamos o *bit* menos significativo com 1 e gravamos o valor no registrador `mtvec`.
 - `mtvec.MODE = 01`

```
la    t0, ivt          # Carrega o endereço da tabela em t0
ori   t0, t0, 0x1     # t0[1:0] <= 01 (vectored mode)
csrw  mtvec, t0       # mtvec <= t0
```

Configuração do mecanismo de interrupções

Configurando o mscratch

- Em nosso exemplo faremos o `mscratch` apontar para uma pilha alocada especialmente para as ISRs.

```
.section .bss
.align 4
isr_stack:      # Final da pilha das ISRs
.skip 1024      # Aloca 1024 bytes para a pilha
isr_stack_end:  # Base da pilha das ISRs

.section .text
.align 2
_start:
# Configura mscratch com o topo da pilha das ISRs.
la    t0, isr_stack_end # t0 <= base da pilha
csrw mscratch, t0      # mscratch <= t0
```

Configuração do mecanismo de interrupções

Configurando os periféricos

- Antes de habilitar as interrupções, o código de inicialização do sistema deve configurar os periféricos
 - **OBS:** Em alguns sistemas, periféricos não configurados podem gerar interrupções não desejadas.
- Cada periférico deve ser configurado de acordo com sua especificação. A configuração é geralmente realizada através da escrita em registradores do periférico.
 - No caso do RISC-V, isso é feito com método MMIO.

Configuração do mecanismo de interrupções

Habilitando as interrupções

- O registrador `mie` deve ser configurado para habilitar as interrupções específicas.
 - P. Ex: *bit 11* habilita interrupções externas.
- O campo `mstatus.MIE` deve ser setado com 1 para habilitar interrupções de forma global

```
# Habilita Interrupções Externas
csrr t1, mie          # Seta o bit 11 (MEIE)
li t2, 0x800         # do registrador mie
or t1, t1, t2
csrw mie, t1

# Habilita Interrupções Global
csrr t1, mstatus     # Seta o bit 3 (MIE)
ori t1, t1, 0x8      # do registrador mstatus
csrw mstatus, t1
```