"I don't know why we are here, but I'm pretty sure that it is not in order to enjoy ourselves" (Ludwig Wittgenstein).

Exceções, Interrupções e I/O

Paulo Lisboa de Almeida

DSBD

UFPR

Exceções e Interrupções

Eventos inesperados que da mesma forma que branches e jumps, podem mudar o fluxo normal do programa.

Exceção.

Algum evento interno inesperado que ocorreu **dentro do processador**.

Ex.: overflow.

Interrupção.

Algum evento inesperado **gerado externamente.**

Ex.: O usuário digitou algo no teclado.

Exceções e Interrupções

Em muitas arquiteturas, como no x86-64, o termo interrupção é utilizado para definir tanto exceções quanto interrupções.

O RISC-V faz o contrário.

> Tudo é uma exceção, e uma interrupção é um tipo especial de exceção, que foi gerada externamente.
>
> São só terminologias diferentes.
>
> Vamos utilizar a terminologia do RISC-V.

Exemplos

Evento	Fonte	Terminologia MIPS
Solicitação de E/S.	Externa	Interrupção
Chamada do Sistema Operacional (syscall).	Interna	Exceção
Overflow Aritmético.	Interna	Exceção
Instrução Inválida.	Interna	Exceção
Solicitação de reset.	Externa	Interrupção

Tratando uma Exceção

Considerando o caso de um overflow aritmético:

```
add x28, x29, x30 # onde o resultado não cabe nos 32 bits de x28
```

Passos Básicos

Salvar o endereço da instrução que causou a exceção.

Salvar o endereço de PC em um registrador especial.

No RISC-V: SEPC -› *Supervisor Exception Cause Register*.

Passos Básicos

Salvar o endereço da instrução que causou a exceção.

- Salvar o endereço de PC em um registrador especial.
  - No RISC-V: SEPC -› *Supervisor Exception Cause Register*.

Salvar algum código informando o que causou a exceção.

- No RISC-V: SCAUSE -› *Supervisor Exception Cause Register*
  - Ex.: código 1 em cause indica overflow, 2 significa instrução inválida, ...

Passos Básicos

Salvar o endereço da instrução que causou a exceção.

- Salvar o endereço de PC em um registrador especial.
  - No RISC-V: SEPC -› *Supervisor Exception Cause Register*.

Salvar algum código informando o que causou a exceção.

- No RISC-V: SCAUSE -› *Supervisor Exception Cause Register*
  - Ex.: código 1 em cause indica overflow, 2 significa instrução inválida, ...

Transferir a execução para uma rotina de tratamento (exemplo: Sistema Operacional).

- As rotinas do kernel para o tratamento básico de exceções no RISC-V iniciam no endereço $0000\ 0000\ 1C09\ 0000_{16}$.
- O S.O. verifica o registrador cause para definir o que houve.
  - Tratar a exceção e retornar a execução do programa a partir do ponto salvo em SEPC, ou terminar o programa.

Passos Básicos

Caso o S.O. opte por terminar o programa, o SEPC ainda pode servir para o debug do programa.

Salvar o endereço da instrução que causou a exceção.

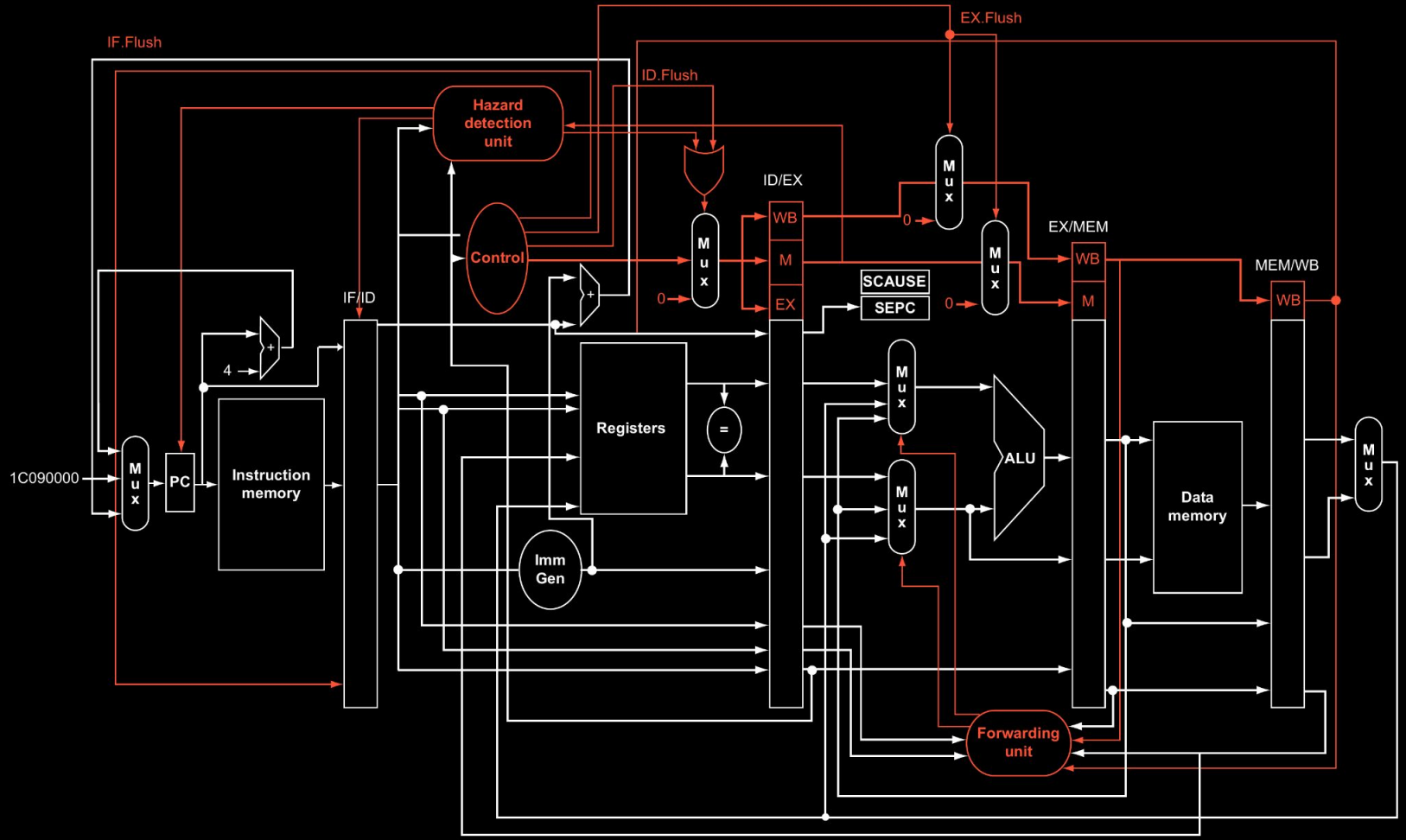
- Salvar o endereço de PC em um registrador especial.
  - No RISC-V: SEPC -› *Supervisor Exception Cause Register*.

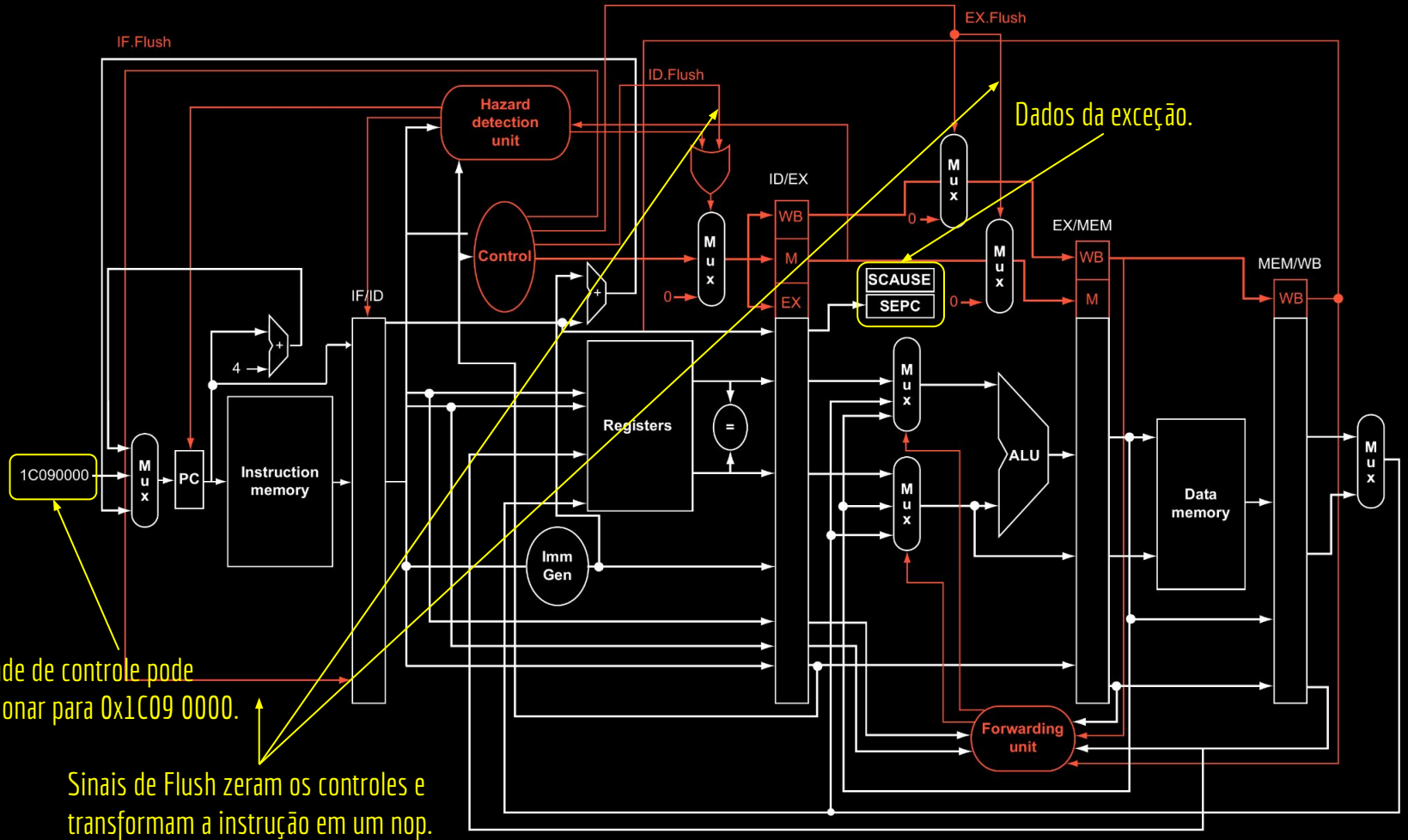
Salvar algum código informando o que causou a exceção.

- No RISC-V: SCAUSE -› *Supervisor Exception Cause Register*
  - Ex.: código 1 em cause indica overflow, 2 significa instrução inválida, ...

Transferir a execução para uma rotina de tratamento (exemplo: Sistema Operacional).

- As rotinas do kernel para o tratamento básico de exceções no RISC-V iniciam no endereço $0000\ 0000\ 1C09\ 0000_{16}$.
- O S.O. verifica o registrador cause para definir o que houve.
  - Tratar a exceção e retornar a execução do programa a partir do ponto salvo em SEPC, ou terminar o programa.


IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
SCAUSE
SEPC
IF/ID
Mux
0
4
1C090000
PC
Instruction memory
Registers
=
Imm Gen
ALU
Data memory
Forwarding unit

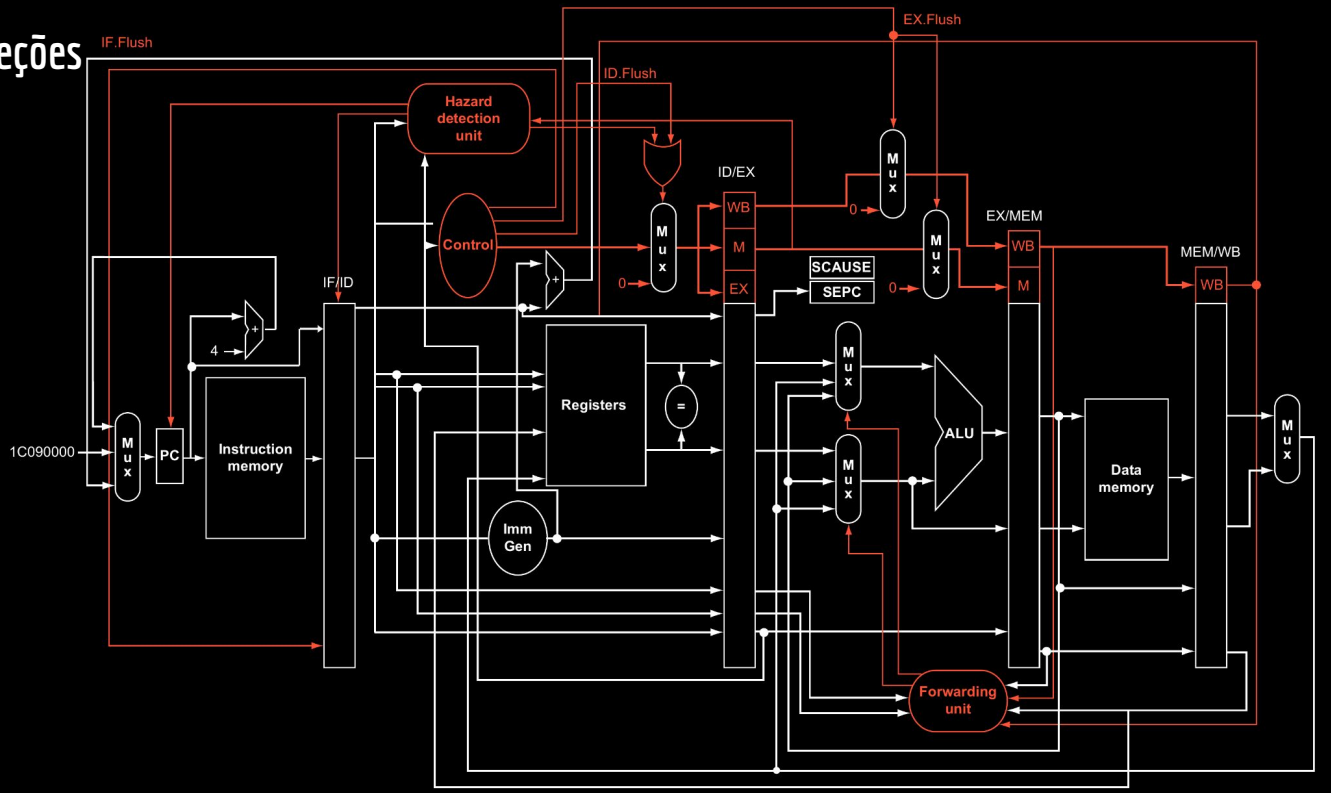

IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Dados da exceção.
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
IF/ID
SCAUSE
SEPC
Mux
0
4
1C090000
PC
Instruction memory
Registers
=
ALU
Imm Gen
Data memory
Forwarding unit
A unidade de controle pode redirecionar para 0x1C09 0000.
Sinais de Flush zeram os controles e transformam a instrução em um nop.

Mais problemas...

Tudo parece muito simples, ledo engano...

Mais problemas...

Podem ocorrer **duas ou mais exceções ao mesmo tempo**. Como?

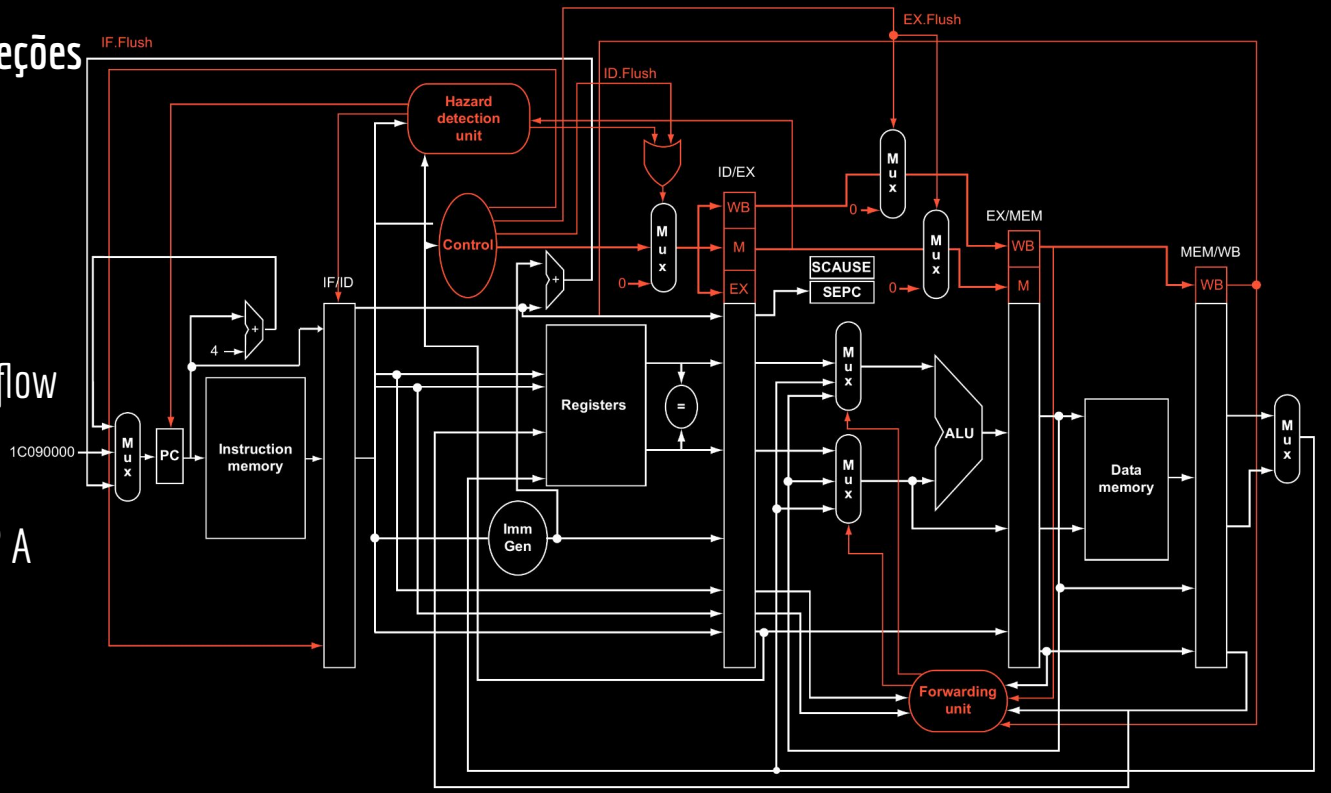


Mais problemas...

Podem ocorrer **duas ou mais exceções ao mesmo tempo**. Como?

Exemplo: instrução inválida sendo decodificada no estágio ID, e overflow no estágio EX.

E agora? Qual instrução registrar? A primeira? A última? As duas?...

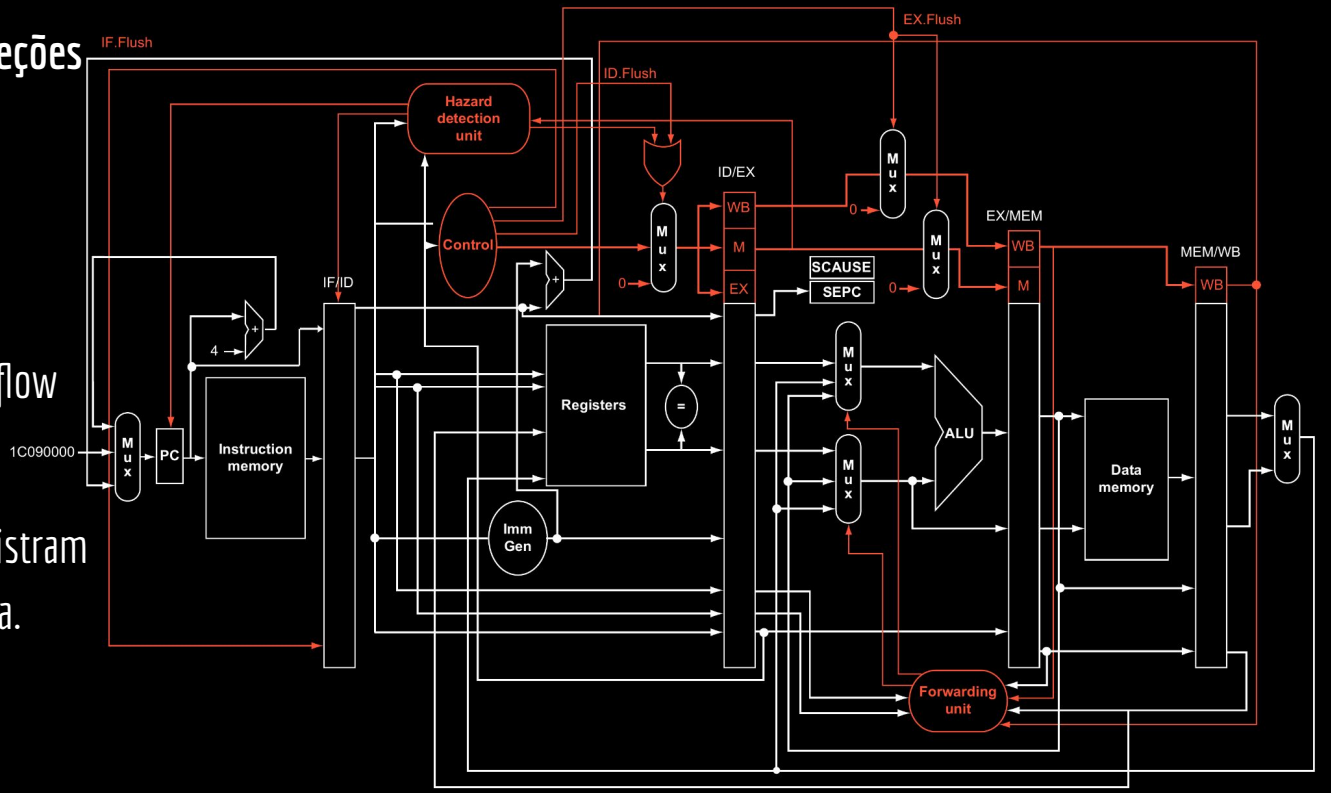


Mais problemas...

Podem ocorrer **duas ou mais exceções ao mesmo tempo**. Como?

Exemplo: instrução inválida sendo decodificada no estágio ID, e overflow no estágio EX.

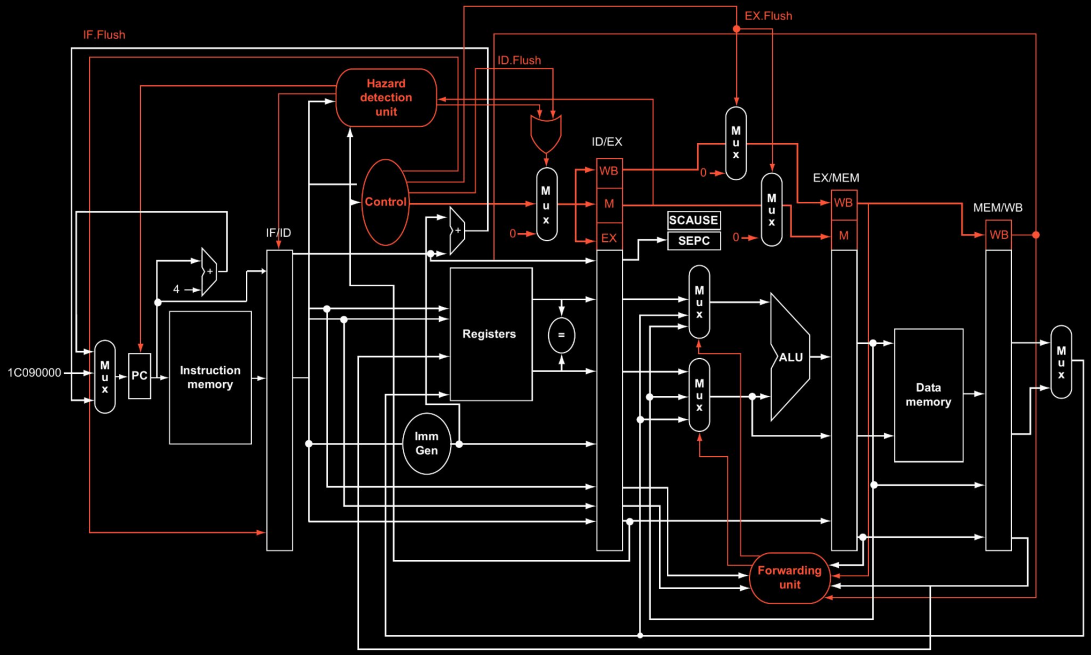
Muitos processadores RISC-V registram a exceção da instrução mais antiga.



E agora?

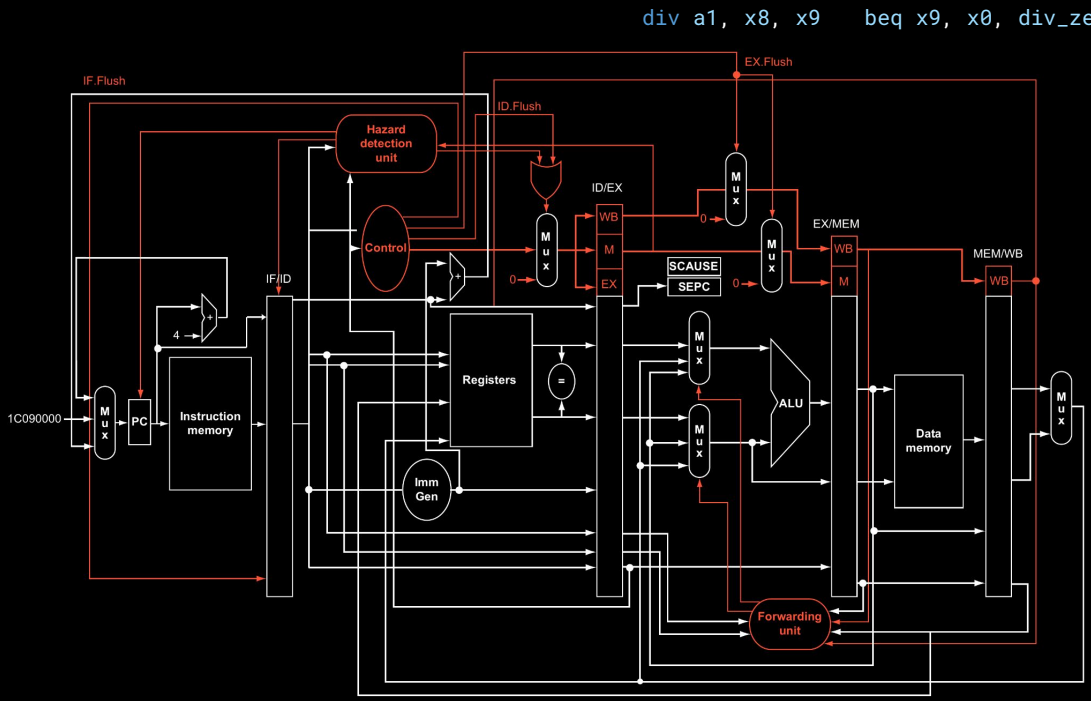
Considere que uma divisão por zero gera uma exceção.

Entenda o trecho assembly a seguir. Qual o problema com o previsor de desvios se o pipeline for profundo?




```
    .text
main:
    addi x8, x0, 20 #um valor qualquer em s0
    addi x9, x0, 2  #um valor qualquer em s1

    beq x9, x0, div_zero

    div a1, x8, x9
    beq x0, x0, saida_beq

div_zero:
    li a1, -1 # -1 para indicar erro

saida_beq:
    li a0, 1  # syscall 1: print integer
    ecall
end:
    ori x10, x0, 17
    ori x11, x0, 0
    ecall
```

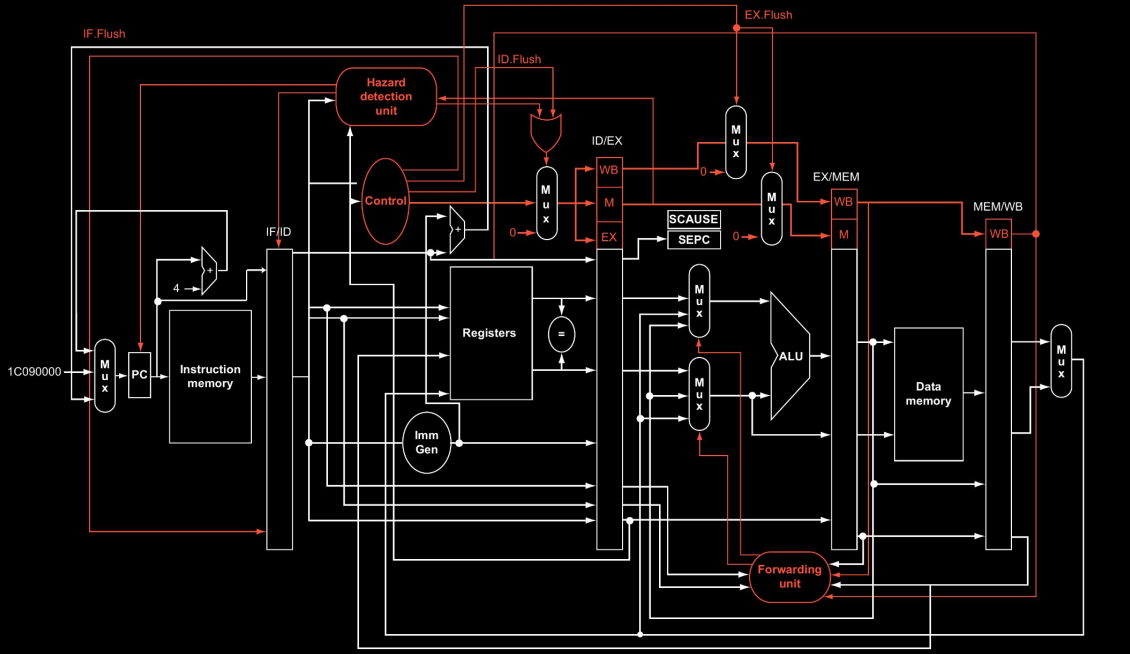
E agora?




```
    .text
main:
    addi x8, x0, 20 #um valor qualquer em s0
    addi x9, x0, 2  #um valor qualquer em s1

    beq x9, x0, div_zero

    div a1, x8, x9
    beq x0, x0, saida_beq

div_zero:
    li a1, -1 # -1 para indicar erro

saida_beq:
    li a0, 1  # syscall 1: print integer
    ecall
end:
    ori x10, x0, 17
    ori x11, x0, 0
    ecall
```

E agora?

```
div a1, x8, x9    beq x9, x0, div_zero
```



Uma instrução **especulativa** que gera uma exceção precisa esperar o resultado do branch para então definir se a exceção realmente deveria existir.

Exceções Precisas e Imprecisas

Exceções precisas armazenam as informações exatas sobre a exceção.

- Endereço da instrução.
- Conteúdo dos registradores quando a instrução foi executada.

Exceções imprecisas podem relaxar isso.

- Exemplo: Armazenar o endereço aproximado do contador de programa.
- Dá mais trabalho identificar o que houve de errado.

Exceções Precisas e Imprecisas

Exceções precisas armazenam as informações exatas sobre a exceção.

- Endereço da instrução.
- Conteúdo dos registradores quando a instrução foi executada.

Exceções imprecisas podem relaxar isso.

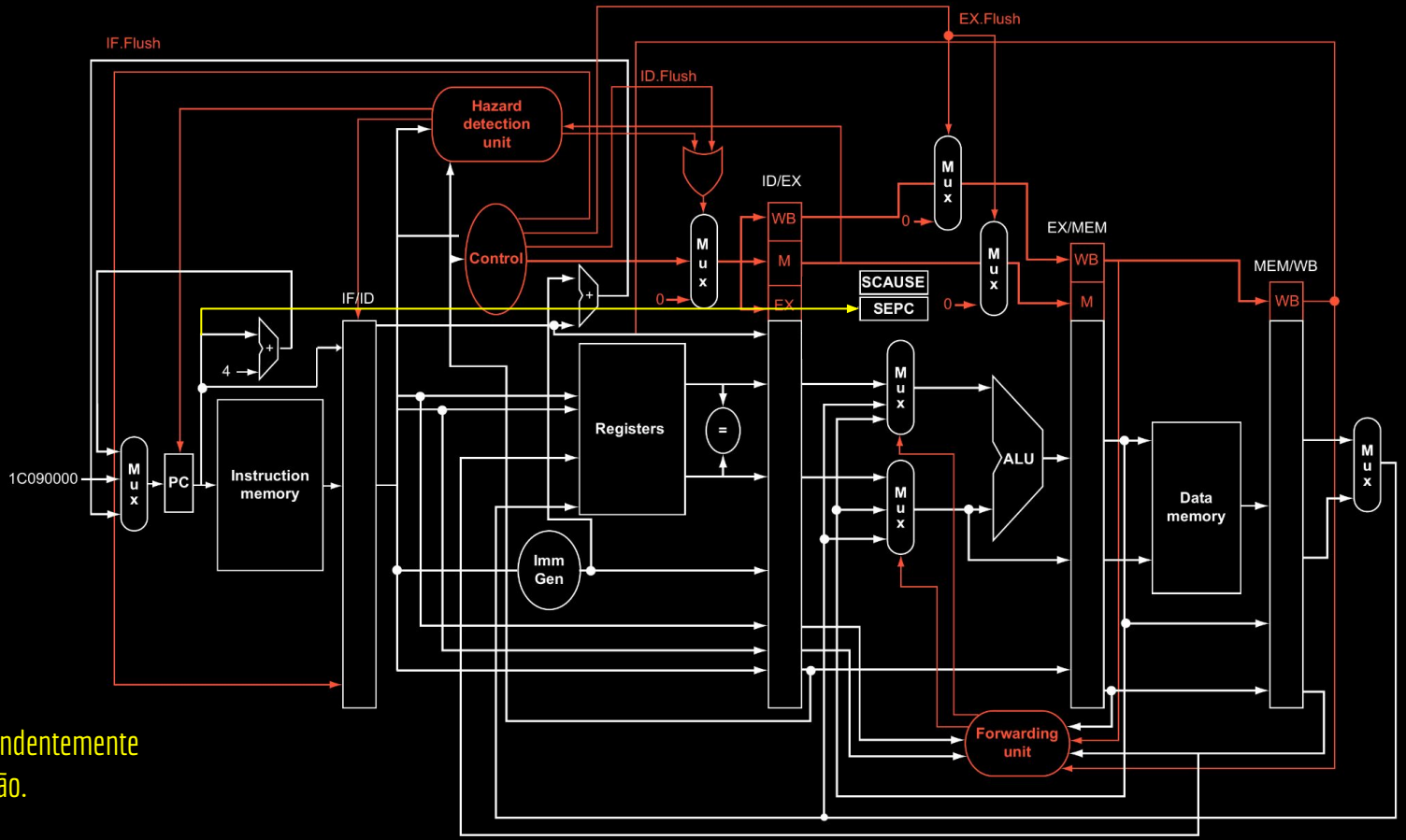
- Exemplo: Armazenar o endereço aproximado do contador de programa.
- Dá mais trabalho identificar o que houve de errado.

É mais fácil gerar exceções imprecisas, e muitos processadores podem gerar esse tipo de exceção.

Especialmente útil para processadores com execução fora de ordem e especulativas (proc. superescalares).

- Estudaremos nas próximas aulas.

Exemplo



Armazenando o PC atual, independentemente do estágio onde ocorreu a exceção.

Exceções Vetorizadas

Podemos eliminar o registrador SCAUSE, e transferir o controle para um endereço específico, dependendo da fonte da exceção.

Exemplo:

Tipo da exceção	Endereço de desvio (hexa)
Instrução indefinida	0000 0000 1C09 0000
Overflow aritmético	0000 0000 1C09 0180
Requisição de E/S do usuário	0000 0000 1C09 0300
...	...

Quais as vantagens e desvantagens de interrupções vetorizadas?

Exceções Vetorizadas

+ Tratamento mais rápido.

    O S.O. não precisa ler SCAUSE para definir o que fazer.

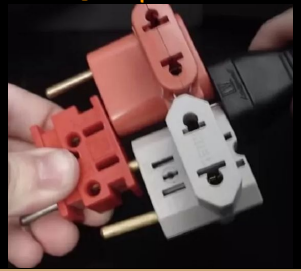
        O fluxo do programa já é redirecionado para o endereço de tratamento específico.

- Necessário mais hardware.

    O sistema de desvios para cada exceção precisa ser implementado de alguma forma.

- Menor flexibilidade.

    Se um novo tipo de exceção precisa ser tratado no hardware, precisamos criar uma nova entrada na tabela, ou fazer alguma gambiarra.



Exceções Vetorizadas

x86-64

Utiliza interrupções vetorizadas.

RISC-V

Utiliza registrador cause (SCAUSE).

Na verdade, falta de páginas é tratado como uma interrupção vetorizada.

Tão comum que vale a pena otimizar esse tratamento através de vetorização. Veremos no futuro.

Interrupções

O mecanismo para tratamento de **interrupções** é similar ao tratamento de exceções.

Pergunta:

Quando uma exceção é detectada, o processamento deve parar imediatamente.
Por quê? Isso também precisa acontecer para interrupções?

Interrupções

O mecanismo para tratamento de **interrupções** é similar ao tratamento de exceções.

Uma interrupção é um **evento externo**.

- Não aconteceu “algo errado” no processador.
- Podemos esperar as instruções que já estão no pipeline terminarem.

Comunicando com dispositivos

Todo dispositivo de E/S é composto por pelo menos dois componentes:

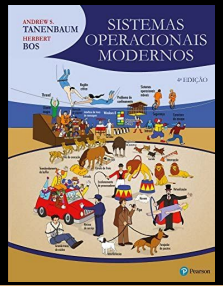
Mecânicos.

Ex.: Os discos do seu HD, partes mecânicas do seu teclado, do seu mouse, ...

Eletrônicos.

Possui pelo menos alguns registradores para indicar o que o componente está fazendo, ou precisa fazer. Exemplo: um registrador de 8 bits em uma impressora simples, onde escrevemos o próximo caractere que a impressora deve imprimir.

Obs.: Esse trecho foi baseado no livro de Tanembaum, Bos (2016), que tem um foco em S.O., e utiliza instruções similares às do x86-64.



Espaço de E/S

Controladores especializados (que podem ser encontrados na placa mãe ou dentro da própria CPU) podem dar um número único para cada registrador de cada dispositivo de E/S conectado - **Número de Porta de E/S**.

Podemos então desenvolver instruções para nossa CPU, que **recebe um número de Porta de E/S e o valor a ser escrito.**

Exemplos (instruções estilo x86-64):

```
in REG,PORTA #lê o conteúdo da porta especificada e armazena em REG

out PORTA,REG #escreve o conteúdo de REG para a porta especificada
```

Em ambos os casos, `REG` é um registrador da CPU, e `PORTA` é o identificador de um registrador de um dispositivo de E/S qualquer.

E/S Mapeada em Memória

Outra opção é o controlador dar um endereço de memória para cada registrador de E/S.

Ao Ler/escrever nesse endereço de memória, o controlador detecta, e redireciona os dados para o registrador do dispositivo.
Nada é realmente lido/escrito na memória principal da máquina.

Chamamos de **E/S mapeada em memória.**

Agora instruções como `lw` e `sw` podem ser usadas para escrever na memória, e também para nos comunicar com os demais dispositivos de E/S.

Necessário apenas de um controlador que intercepta o sinal da CPU, e redireciona para o local correto.

E/S

RISC-V utiliza E/S mapeada em memória.

- E/S mapeada em memória é também comum em microcontroladores.
  - Exemplo: família de microcontroladores PIC e ATMega.

X86-64 utiliza ambas técnicas.

- Possui instruções especializadas para lidar com portas de E/S.
- Memória entre [0-(64K-1)] reservada para portas de E/S.

Temos ainda o conceito de DMA.

- **Faça você mesmo! Estude sobre como funciona o conceito de DMA.**

Enviando sinais ao processador

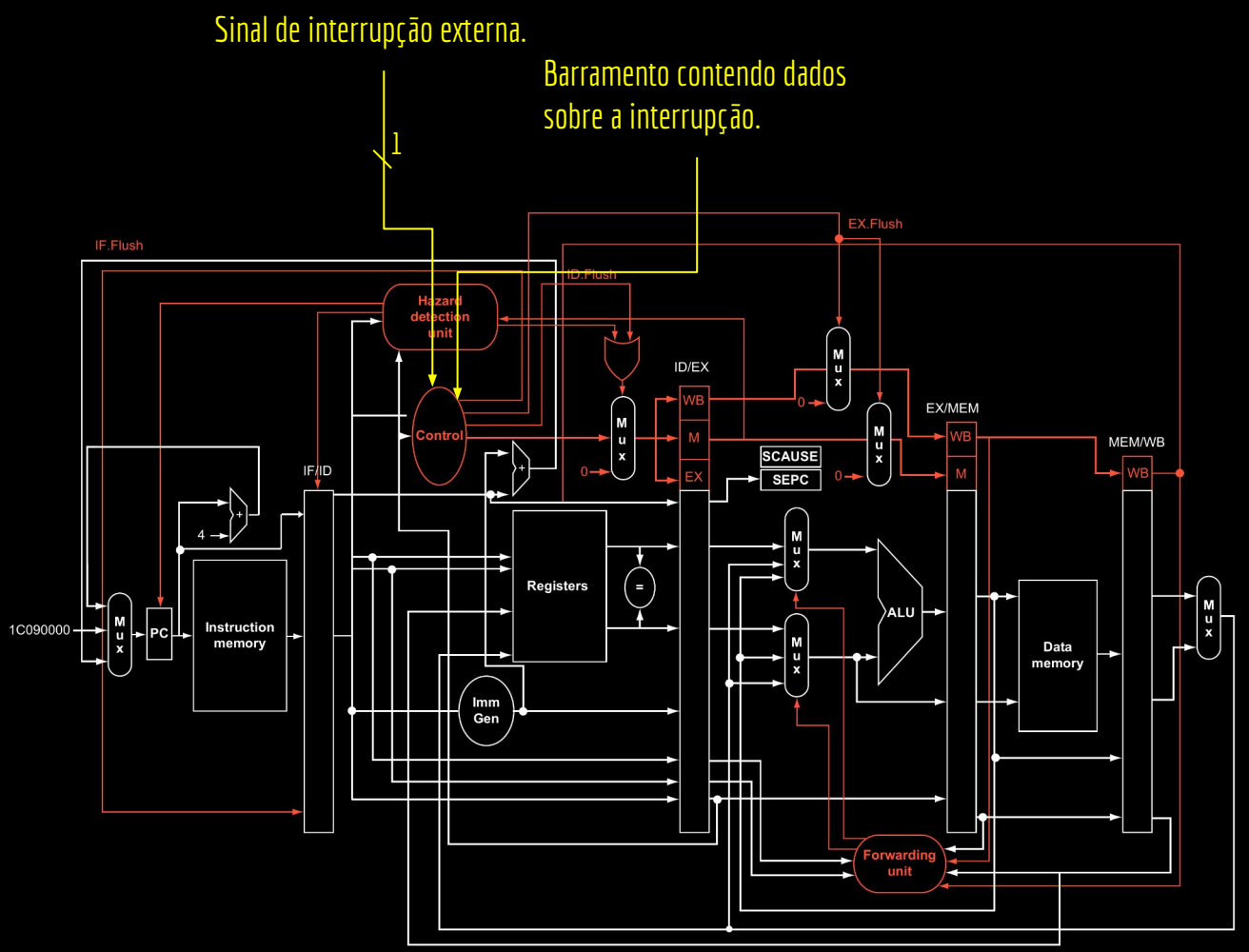
Um dispositivo de E/S pode enviar um sinal ao processador.

- E/S baseada em interrupção.

Precisamos de **sinais de controle externos** no processador.

- Quando o processador recebe um sinal externo, ele gera uma interrupção.
- O processador pode também ler do barramento externo informações sobre a interrupção.
  - Exemplo: no barramento podemos incluir informações sobre o hardware que solicitou a interrupção.

Exemplo


Sinal de interrupção externa.
Barramento contendo dados sobre a interrupção.
1
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
WB
M
EX
EX/MEM
MEM/WB
SCAUSE
SEPC
IF/ID
1C090000
PC
Instruction memory
4
Registers
Imm Gen
ALU
Data memory
Forwarding unit
Mux
0

Enviando sinais ao processador

Quando uma interrupção é gerada, o processador pode terminar de executar as instruções que já estão no pipeline, antes de redirecionar para o S.O.

Evitar stalls.

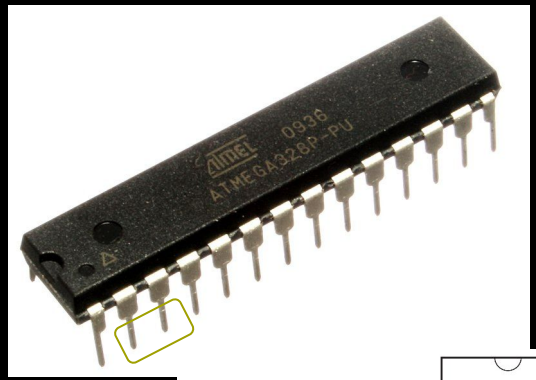
A interrupção pode ser **assíncrona** com relação às instruções que estão dentro da CPU.

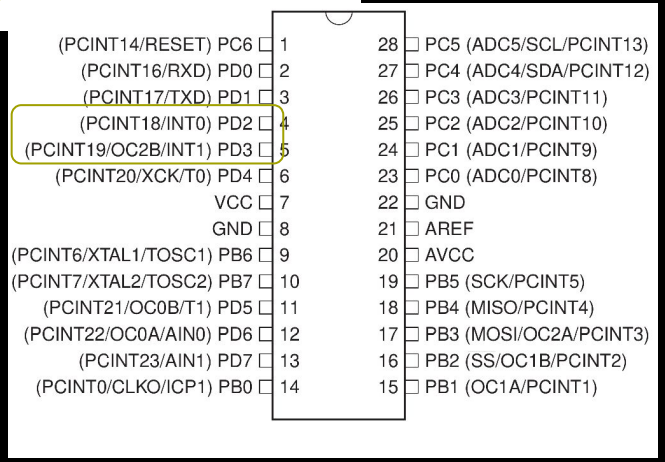
O processador armazena as informações sobre a interrupção, redireciona para o S.O., e o restante do tratamento é o mesmo de uma exceção qualquer.

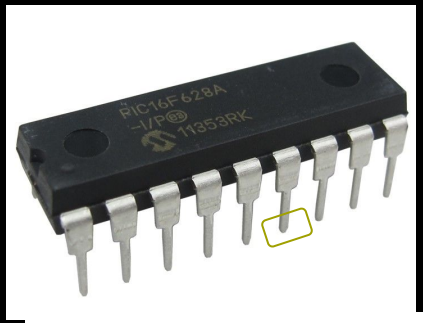
O problema agora é do S.O. ou da rotina de tratamento.

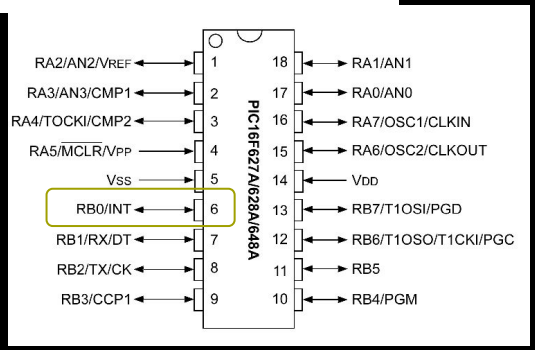
Mundo real

Pinos de interrupção do ATMega328p na esquerda, e do PIC16F168a na direita.





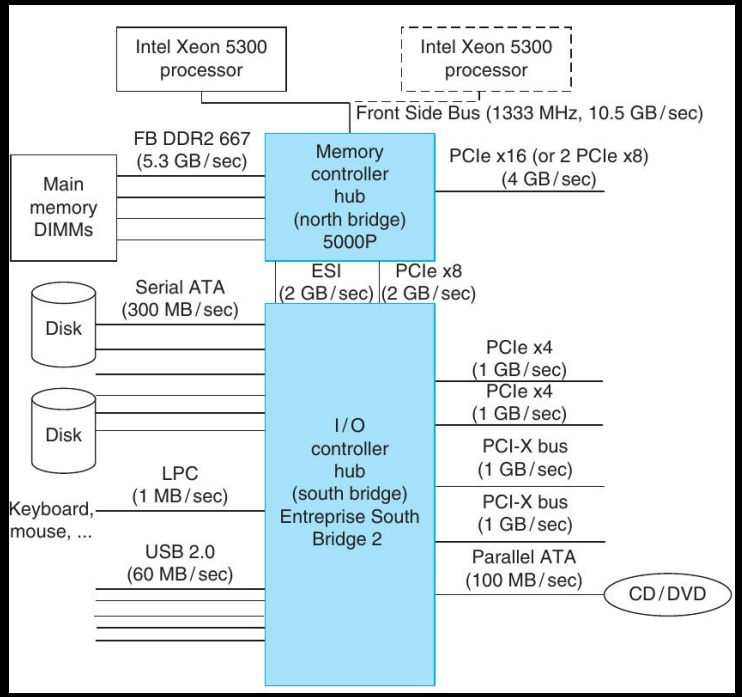




Mundo Antigo

Os principais controladores que fazem as traduções e enviam as interrupções dos dispositivos de E/S para a CPU são a ponte norte e a ponte sul.

Comum até meados dos anos 2010.



Processadores atuais

Processadores x86-64 atuais incorporaram muitas das tarefas antes feitas pelas pontes norte e sul.

Esses controladores ainda existem, mas se encontram dentro da CPU.

Reduzir os atrasos no circuito, criando um controlador otimizado que está muito próximo aos processadores.

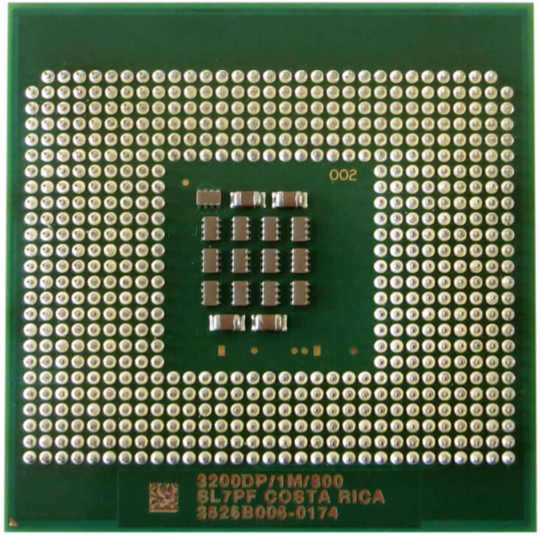
Problemas:

- Agora muitas coisas devem se comunicar diretamente com a CPU;
- Gastamos área do chip da CPU com esses controladores;
- Podemos precisar de pinos extras para conectar as coisas à CPU.

Obs.: Ainda temos pequenos *hubs* externos, mas não fazem todo o trabalho que as pontes fazem.

Exemplos

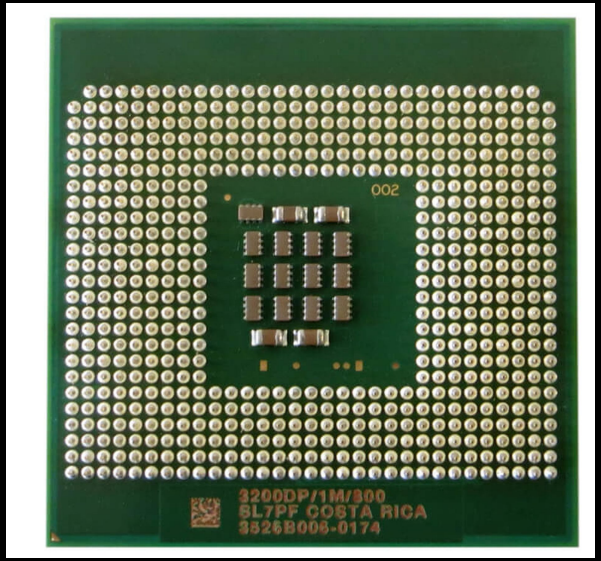
As imagens **não estão em escala**!
Ambos processadores são dual channel.



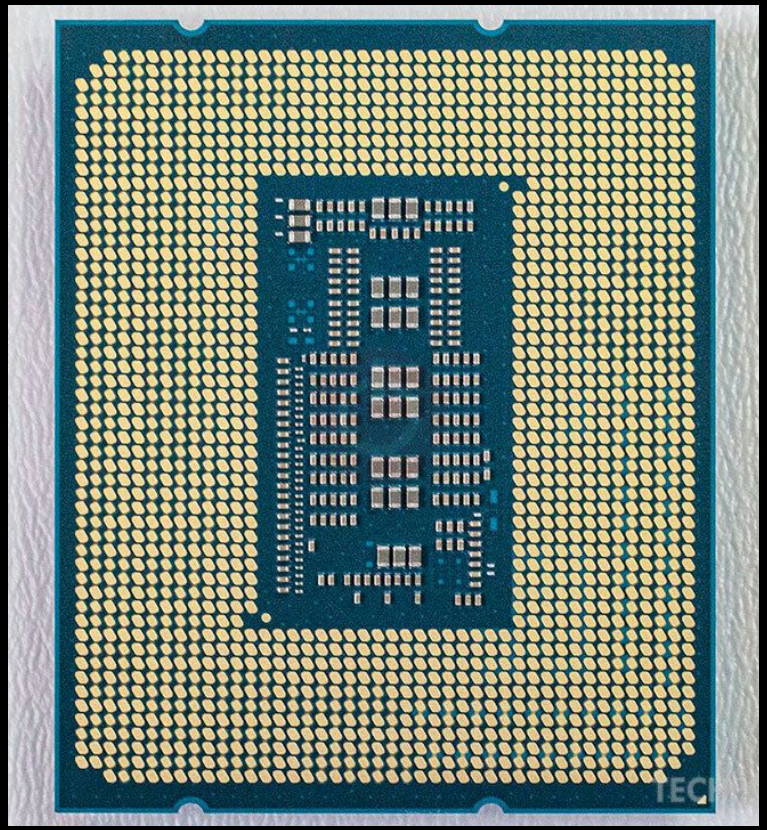
Intel Xeon para Socket PPGA604 (mesmo socket do Xeon 5300 do exemplo anterior). 604 Pinos. Processador de 2006.

Exemplos

As imagens **não estão em escala**!
Ambos processadores são dual channel.

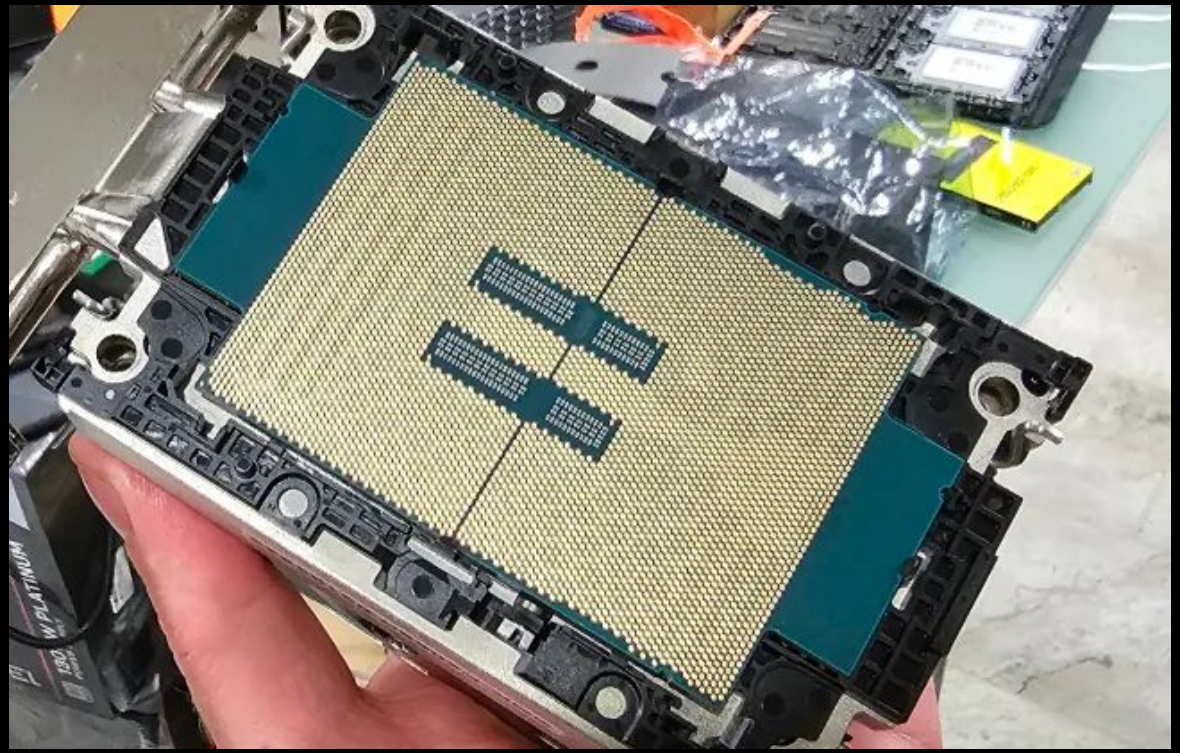


Intel Xeon para Socket PPGA604 (mesmo socket do Xeon 5300 do exemplo anterior).
604 Pinos. Processador de 2006.



Intel i7 14700K para Socket 1700.
1700 Pinos.
Lançado em 2023.

Um monstro



Intel Xeon Max 9468 - Lançado em 2023.
Socket FCLGA4677 de 4677 pinos.

Curiosidade

Uma **syscall** é uma exceção.

- Instrução `ecall` no RISC-V.
- Redireciona para o sistema operacional

Se você abrir o assembly de um programa em C que faz um printf, vai notar que em algum momento, uma syscall ou similar é realizada para chamar o S.O., que vai realizar a impressão.

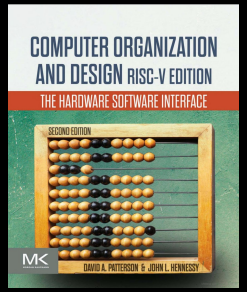
Exercícios

1. O S.O. é o responsável por tratar as exceções. O S.O. também é um programa, que utiliza os mesmos registradores do seu programa na CPU. Escreva o trecho de código assembly que o S.O. sempre deve executar antes, e depois do tratamento da exceção, para que as informações salvas nos registradores não se percam, e seu programa possa continuar executando normalmente após uma exceção (se o sistema operacional decidir que o seu programa pode continuar executando).

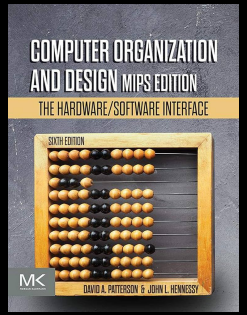
2. O que é DMA (Direct Memory Access)?

Referências

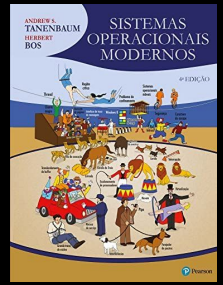
Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



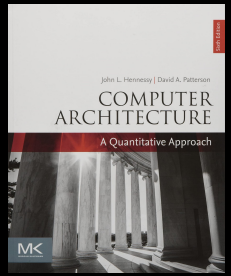
Patterson, Hennessy. Computer Organization and Design MIPS Edition: The Hardware/Software Interface. 2020.



Tanenbaum, Bos. Sistemas operacionais modernos. 4a ed. 2016.



Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Licença