"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 Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

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, o termo interrupção é utilizado para definir tanto exceções quanto interrupções.

O MIPS 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 MIPS.

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 $1, $2, $1 # onde o resultado não cabe nos 32 bits de $1
```

Passos Básicos

Salvar o endereço da instrução que causou a exceção.

Salvar o endereço de PC em um registrador especial (EPC → Exception PC).

Passos Básicos

Salvar o endereço da instrução que causou a exceção.

Salvar o endereço de PC em um registrador especial (EPC → Exception PC).

Salvar algum código informando o que causou a exceção.

Registrador cause no MIPS32.

Ex.: código 1 em cause indica overflow, 2 significa instrução inválida, ...

Passos Básicos

Salvar o endereço da instrução que causou a exceção.

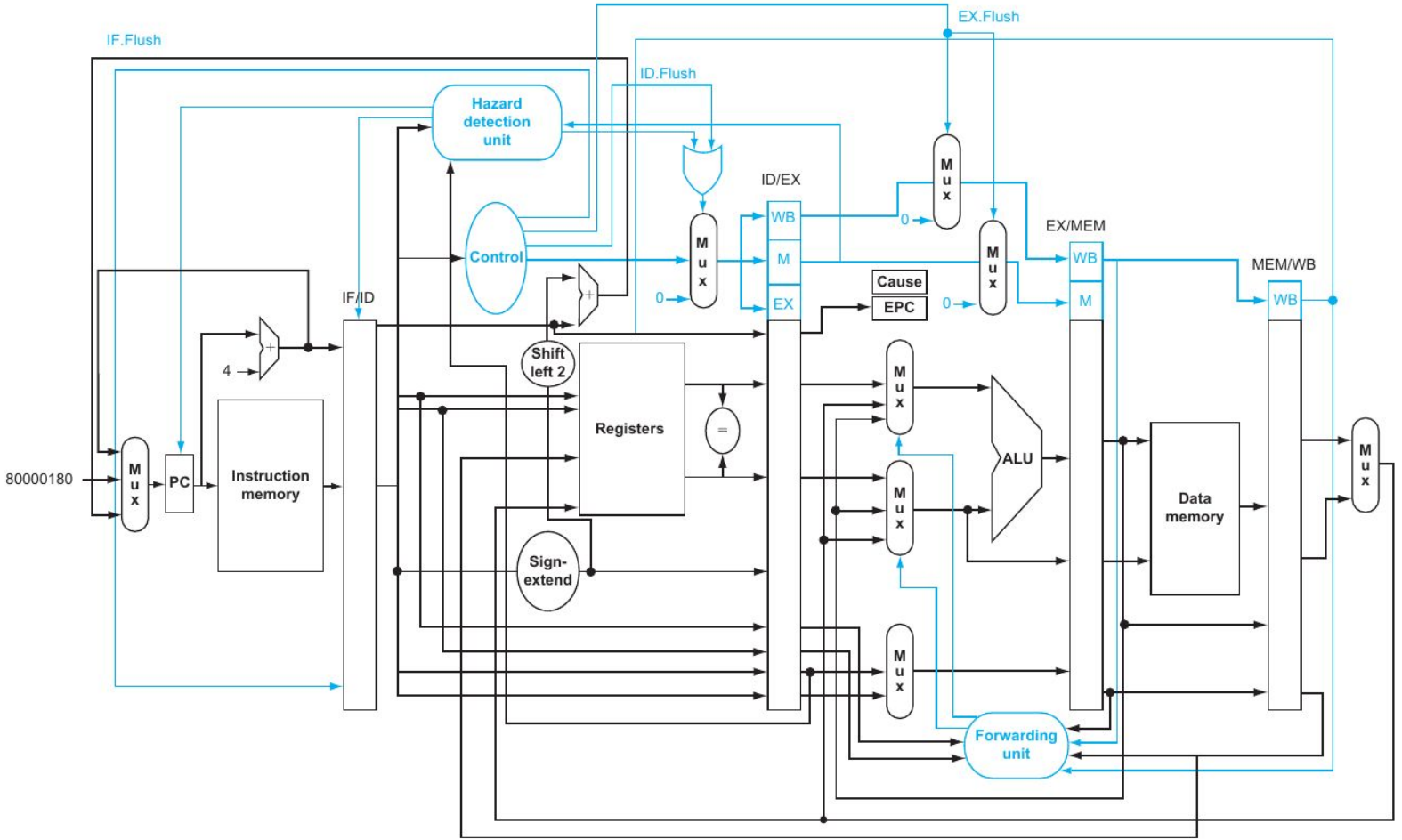
- Salvar o endereço de PC em um registrador especial (EPC → Exception PC).

Salvar algum código informando o que causou a exceção.

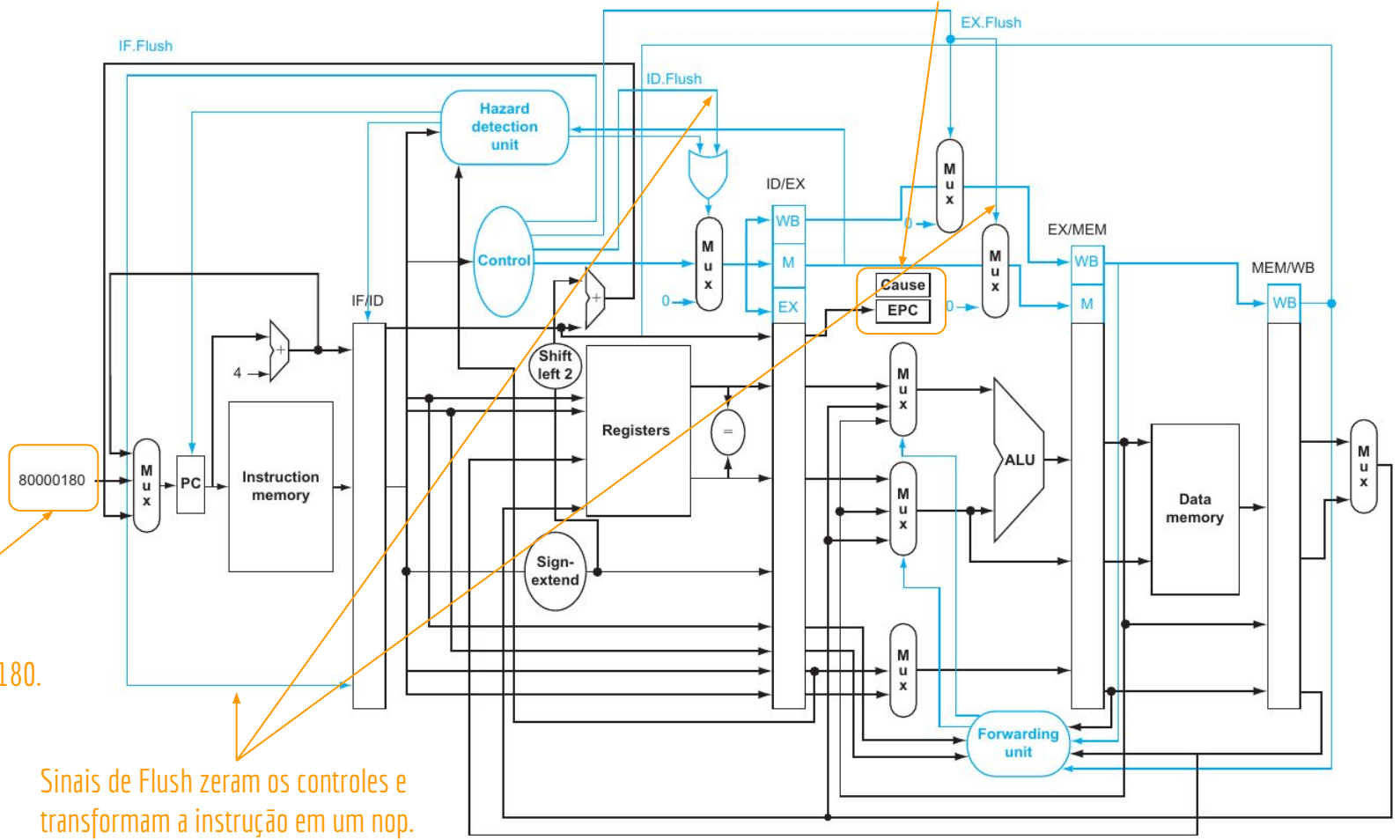
- Registrador cause no MIPS32.
- Ex.: código 1 em cause indica overflow, 2 significa instrução inválida, ...

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

- As rotinas do kernel para o tratamento básico de exceções no MIPS32 iniciam no endereço 0x80000180.
- O S.O. verifica o registrador cause para definir o que houve.
- O S.O. pode tratar a exceção e retornar a execução do programa a partir do ponto salvo em EPC, ou terminar o programa.
  - Caso o S.O. opte por terminar o programa, o EPC ainda pode servir para o debug do programa.


IF.Flush
EX.Flush
ID.Flush
Hazard detection unit
Control
ID/EX
WB
M
EX
EX/MEM
MEM/WB
Cause
EPC
IF/ID
Shift left 2
Registers
ALU
Data memory
Instruction memory
PC
80000180
4
Sign-extend
Forwarding unit
Mux
0

Salvando PC +4 em EPC. O S.O. deve levar isso em consideração ao tratar uma exceção.



A unidade de controle pode redirecionar para 0x80000180.

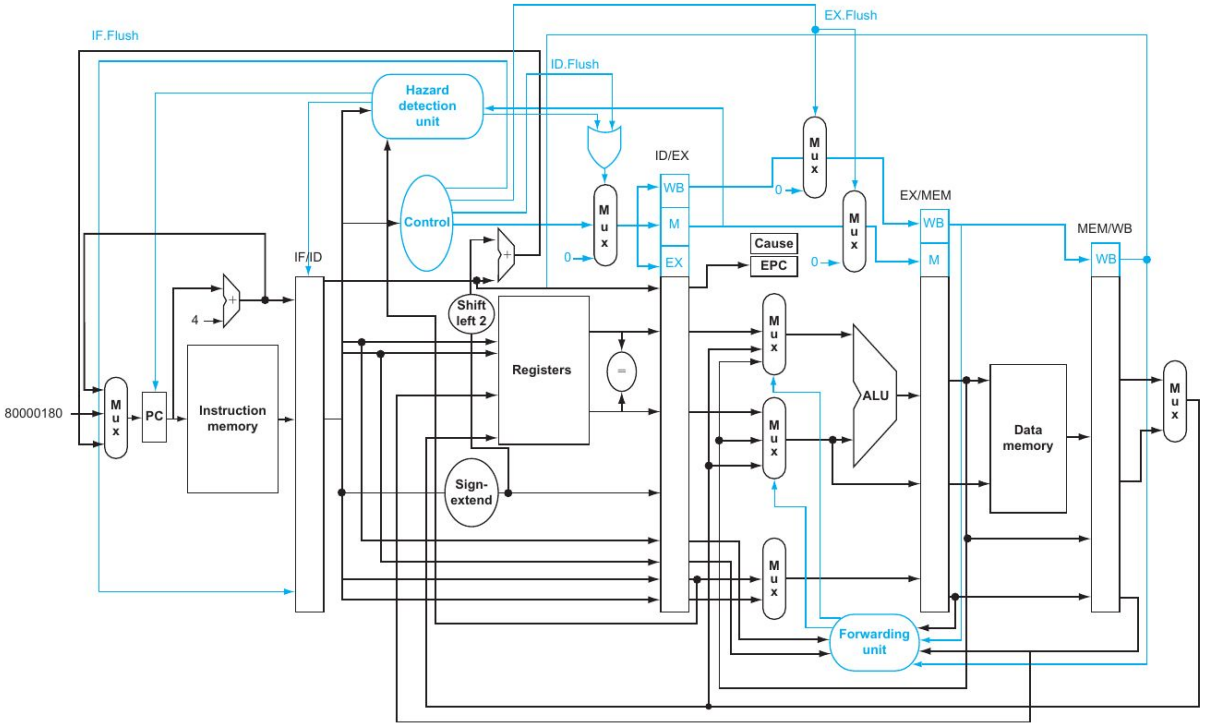
Sinais de Flush zeram os controles e transformam a instrução em um nop.

Mais problemas...

Tudo parece muito simples, ledo engano...

Mais problemas...

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

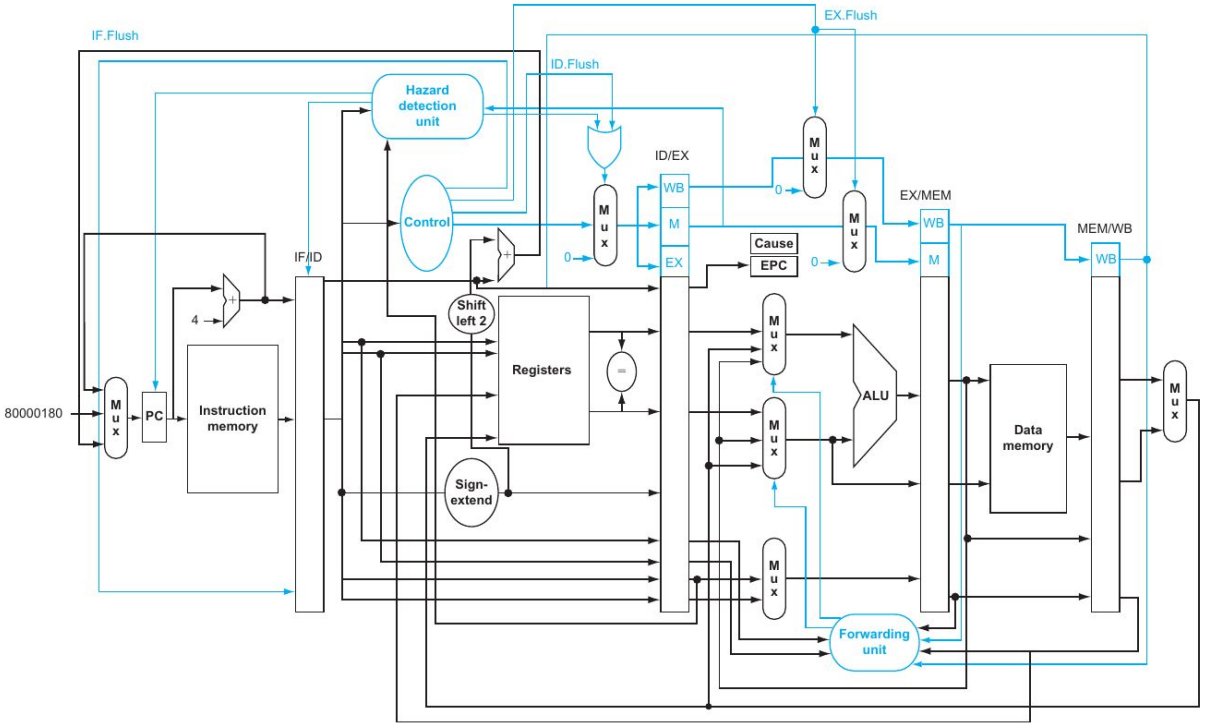


Mais problemas...

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

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

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

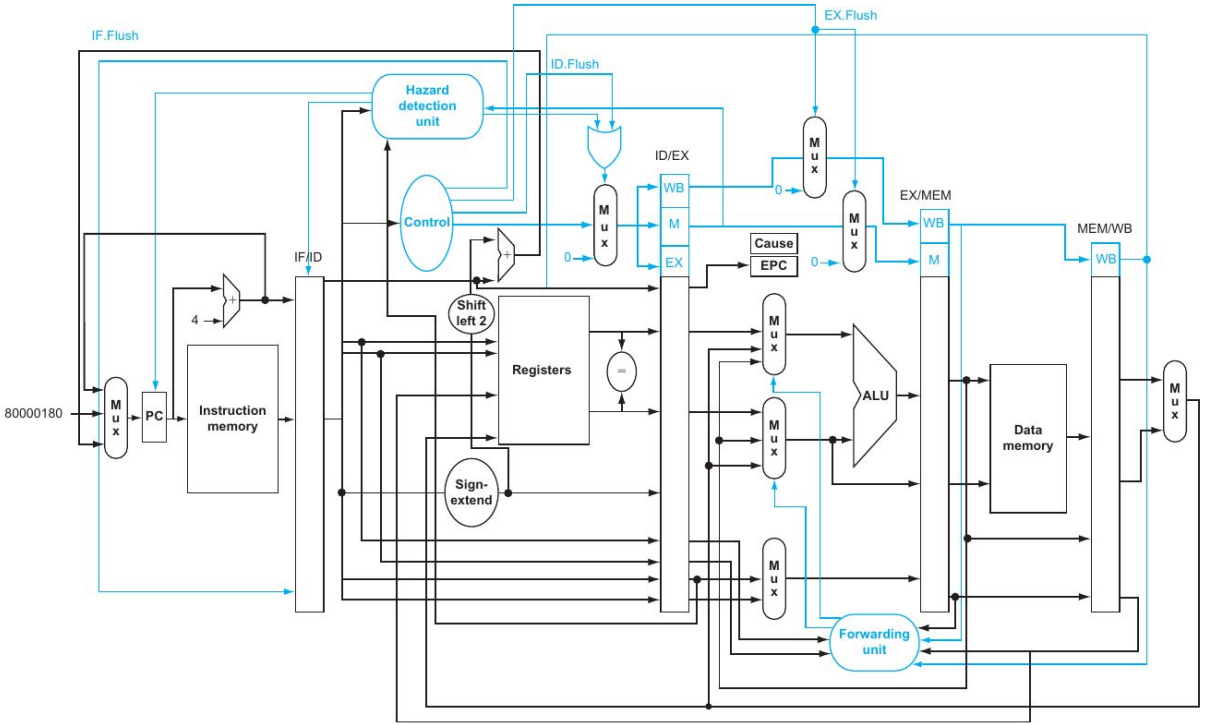


Mais problemas...

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

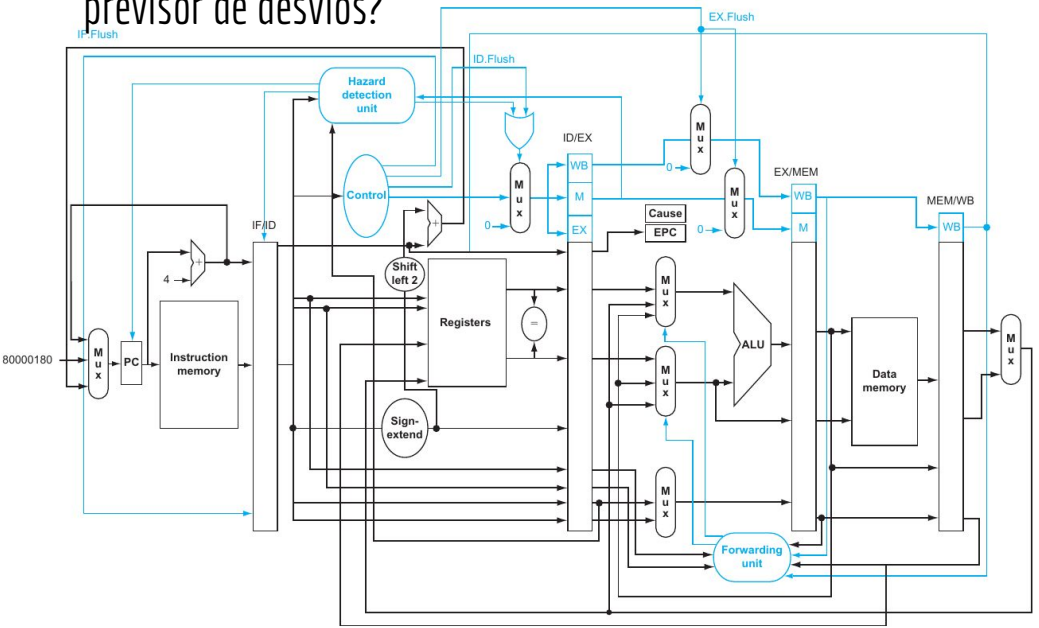
Exemplo: instrução inválida sendo decodificada no estágio ID, e overflow no estágio EX.

Muitos processadores MIPS 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?

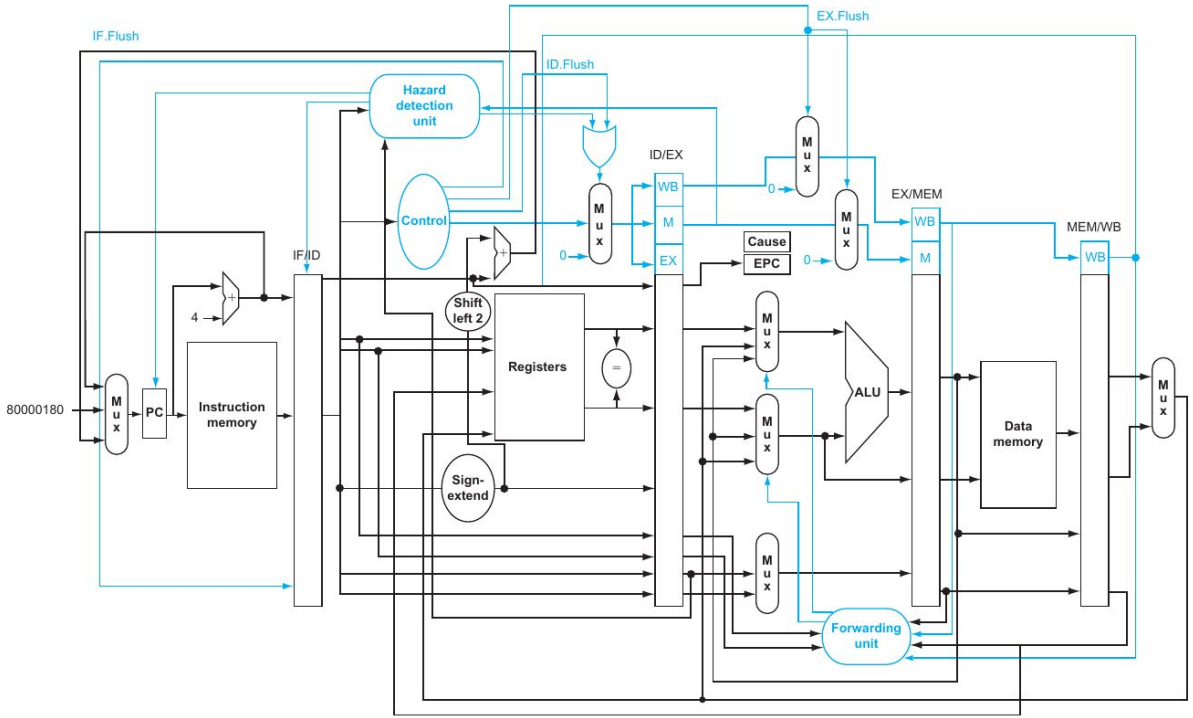



```
        .text
        .globl main
main:
        li $v0, 5           #5 em $v0 para S.O ler inteiro
        syscall             #chama o S.O. e resultado em $v0
        ori $s0, $v0, 0     #salva em $s0

        li $v0, 5           #5 em $v0 para S.O ler inteiro
        syscall             #chama o S.O. e resultado em $v0
        ori $s1, $v0, 0     #salva em $s1

        beq $s1, $zero, div_zero
        div $s0, $s1        #dividir $s0 por $s1
        mflo $a0            #mover resultado para $a0
        j saida_beq
div_zero:
        ori $a0,$zero,0     #0 para erro
saida_beq:

        li $v0, 1           #S.O escreve $a0 na tela
        syscall
end:
        li $v0, 10
        syscall
```

E agora?

Se o previsor de desvio errar a predição, a instrução vai gerar uma exceção aqui que não deveria existir.

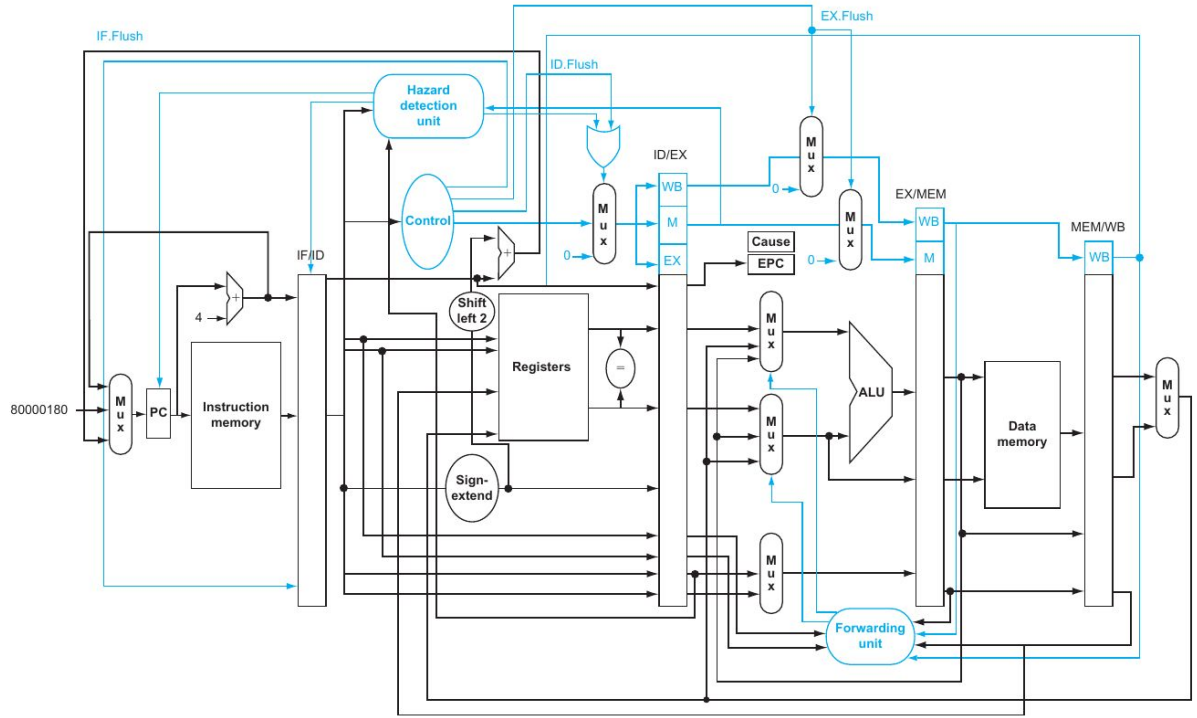
`beq $s1,$zero,div_zero` `div $s0, $s1`




```
        .text
        .globl main
main:
        li $v0, 5
        syscall
        ori $s0, $v0, 0

        li $v0, 5
        syscall
        ori $s1, $v0, 0

        beq $s1, $zero, div_zero
        div $s0, $s1
        mflo $a0
        j saida_beq
div_zero:
        ori $a0,$zero,0
saida_beq:

        li $v0, 1
        syscall
end:
        li $v0, 10
        syscall
```

E agora?

```
beq $s1,$zero,div_zero    div $s0, $s1
```



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.

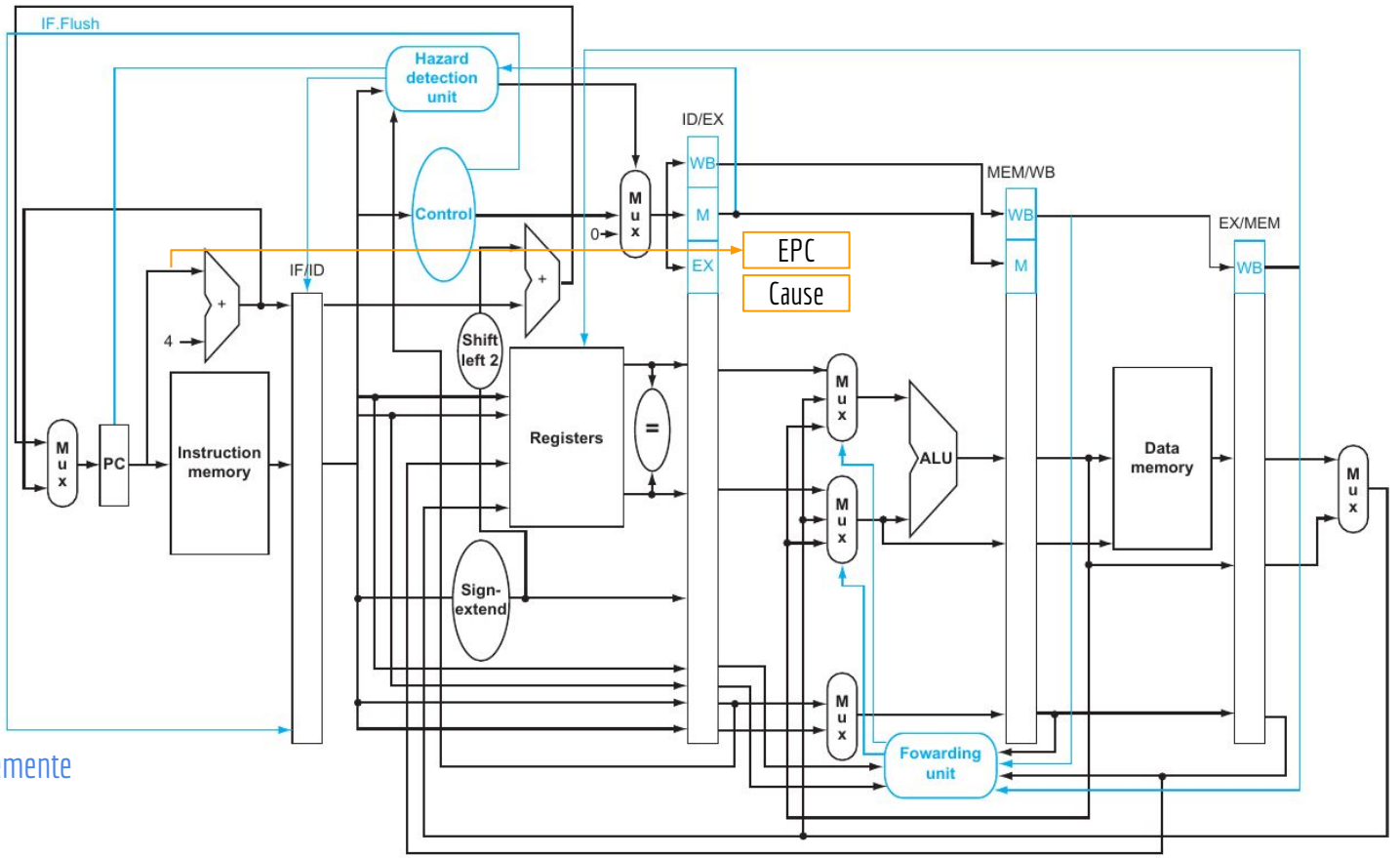
Exceções imprecisas podem relaxar isso.

- 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).
  - Podemos ver no futuro.

Exemplo



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

Exceções Vetorizadas

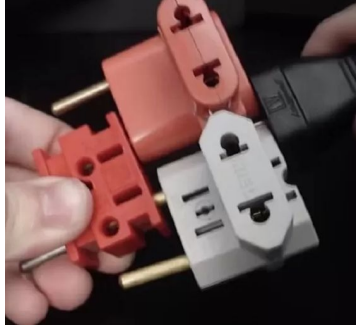
Podemos eliminar o registrador cause, e transferir o controle para um endereço específico, dependendo da fonte da exceção.

Exemplo:

Tipo da exceção	Endereço de desvio
Instrução indefinida	0x80000000
Overflow aritmético	0x80000180
Requisição de E/S do usuário	0x80000360
...	...

Quais as vantagens e desvantagens de interrupções vetorizadas?

Obs.: Sinônimos de enjambre → gambiarra, jeitinho brasileiro, emenda constitucional, engenharia caseira.



Exceções Vetorizadas

+ Tratamento mais rápido.

    Agora o S.O. não precisa ler *cause* 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 algum enjambre.

Exceções Vetorizadas

x86-64

Utiliza interrupções vetorizadas.

MIPS

Utiliza registrador cause.

Na verdade, um único cenário é tratado como uma interrupção vetorizada.

Falta de páginas.

Tão comum que vale a pena otimizar esse tratamento através de vetorização.

Você vai aprender o que é uma falta de páginas em Sistemas Operacionais.

Interrupções

O mecanismo para tratamento de interrupções pode ser 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 pode ser similar ao tratamento de exceções.

Uma interrupção é um evento externo.

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

Comunicando com dispositivos

Atenção: 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.

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.

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 as instruções 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

MIPS 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.

- Um dos temas dos trabalhos.

Enviando sinais ao processador

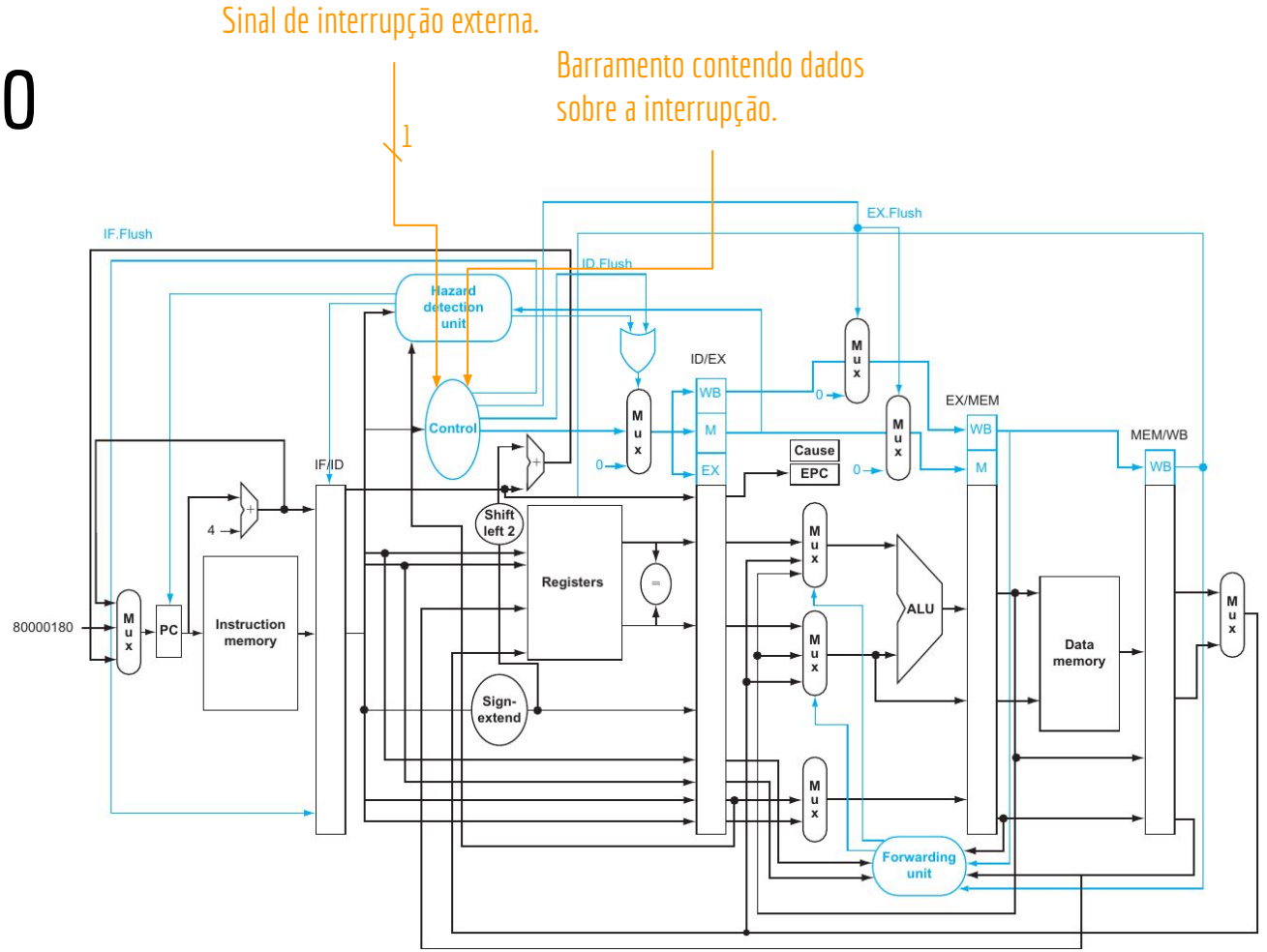
Um dispositivo de E/S pode enviar um sinal ao processador.

- 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
EX/MEM
MEM/WB
WB
M
EX
Cause
EPC
0
Mux
IF/ID
4
Shift left 2
Registers
ALU
Sign-extend
80000180
PC
Instruction memory
Data memory
Forwarding unit

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.

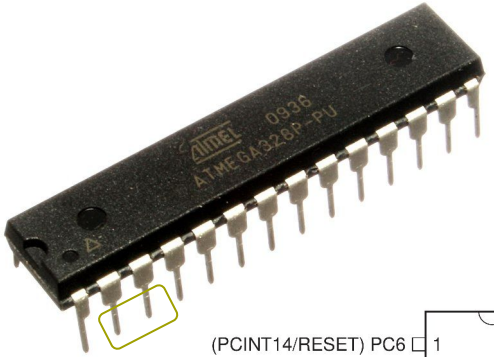
A interrupção pode ser assíncrona com relação às instruções que estão dentro da CPU.

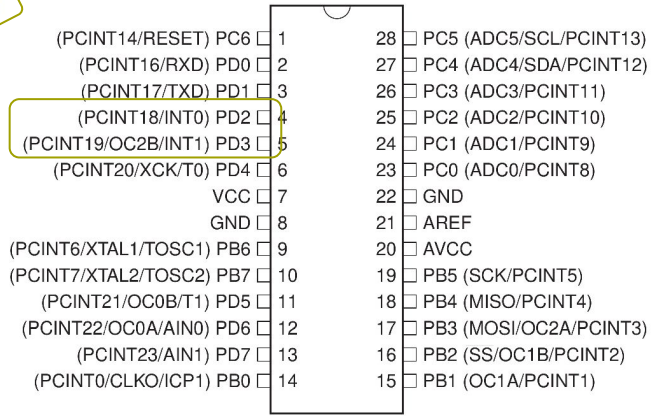
O processador armazena as informações sobre a interrupção, redireciona para o S.O., e o restante do tratamento é o mesmo de uma exceção qualquer.

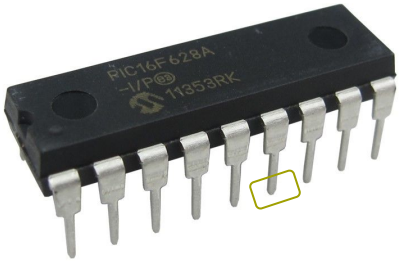
O problema agora é do S.O. ou da rotina de tratamento.

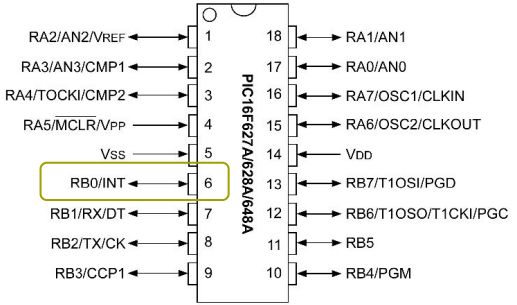
Mundo real

Pinos de interrupção do ATMega328p (mesmo microcontrolador do Arduino) na esquerda, e do PIC16F168a na direita.





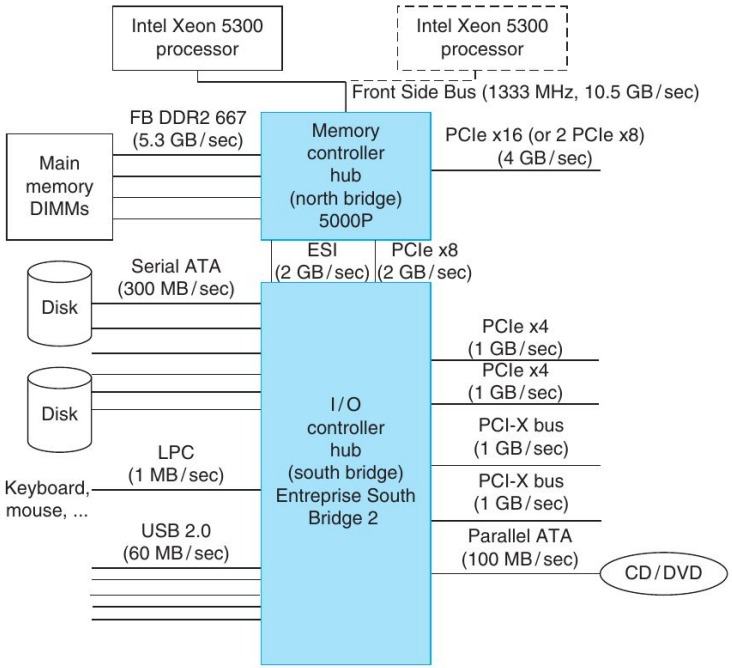




Mundo + ou - Real

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.



Mundo + ou - Real

Ponte Norte.

Mais próxima da CPU.

Conecta os dispositivos que exigem maior largura de banda.

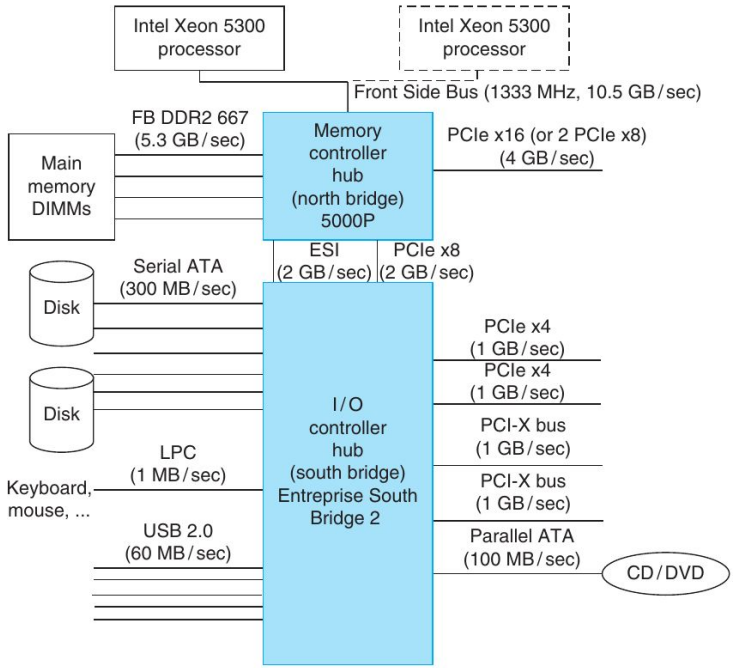
Memoria, PCIe x16, ...

Caminho para a ponte sul.

Ponte Sul.

Conecta os dispositivos mais lentos.

Mouse, teclado, placa de rede, discos, ...



Processadores atuais

A maioria dos 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.

A ideia dos projetistas é reduzir os atrasos no circuito com isso, criando um controlador otimizado que está muito próximo aos processadores.

Problemas:

Agora muitas coisas devem se comunicar “diretamente com a CPU”;

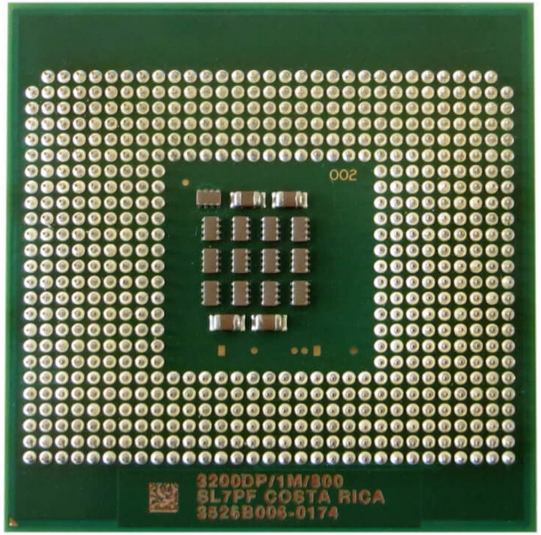
Gastamos área do chip da CPU com esses controladores;

Podemos precisar de pinos extras para conectar as coisas à CPU.

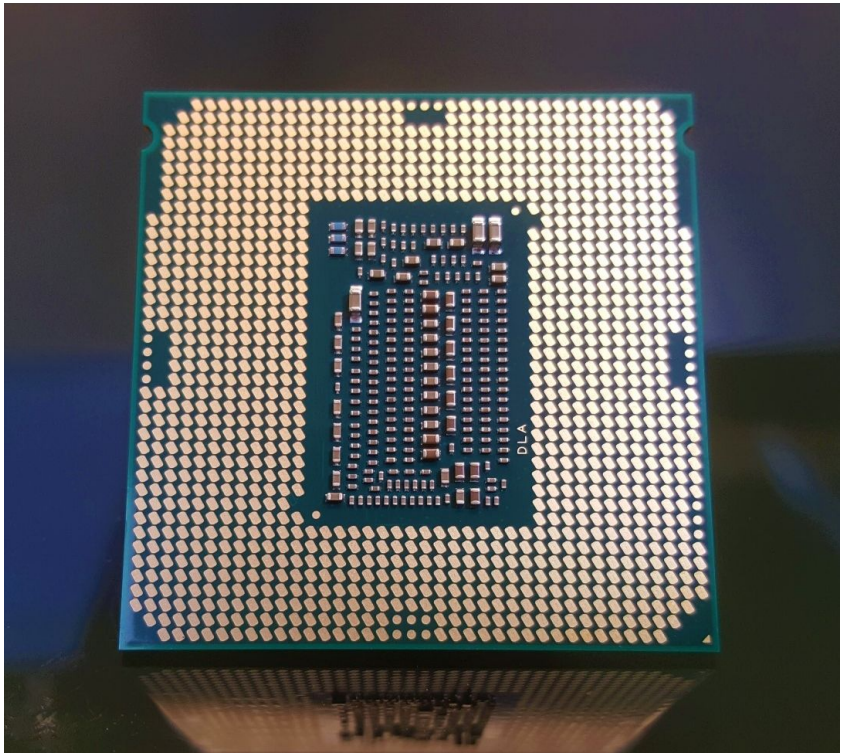
Obs.: Ainda temos pequenos “hubs” externos, mas não fazem todo o trabalho que as pontes fazem.

Exemplos

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

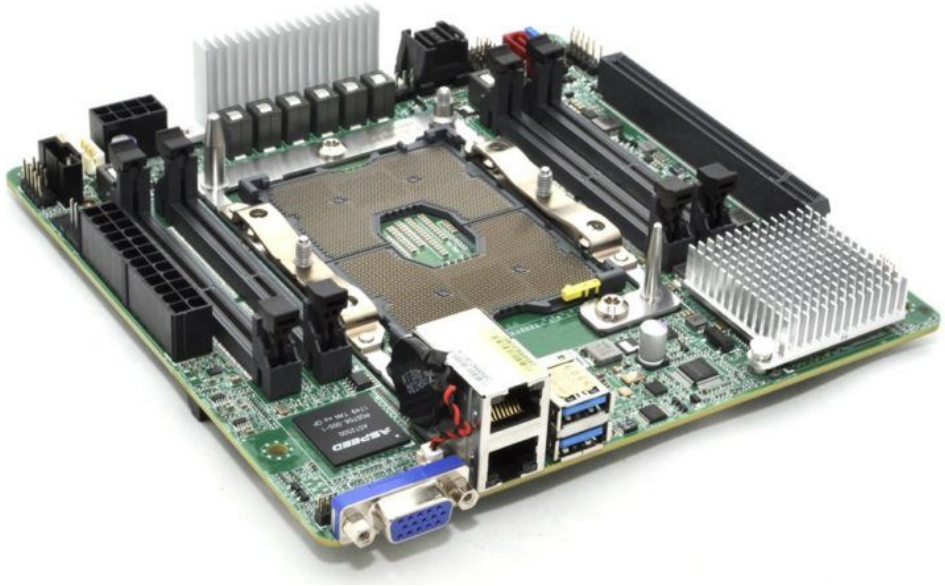


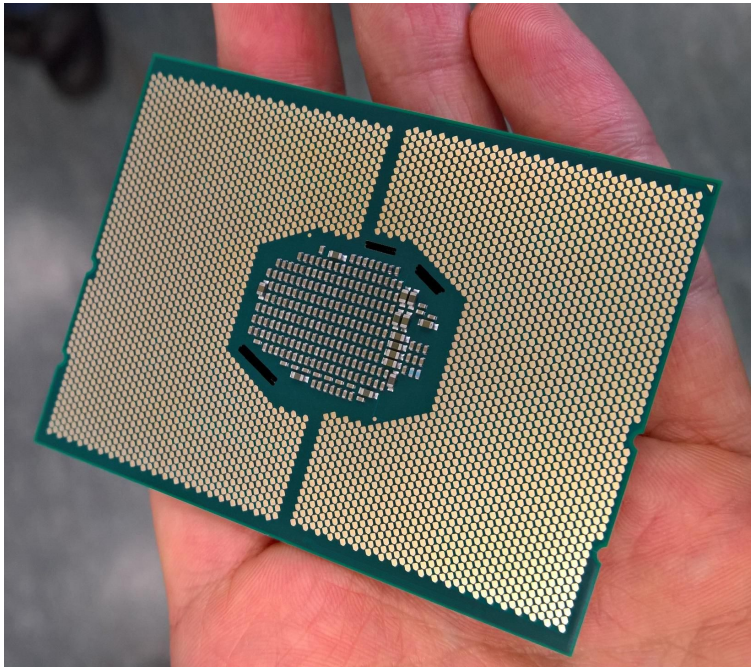
Intel Xeon para Socket PPGA604 (mesmo socket do Xeon 5300).
604 Pinos.
Processador de 2006.



Intel i9-9900k para Socket FCLGA1151.
1151 Pinos.
Processador de 2019.

Um monstro





Processador Xeon para socket LGA 3647 (3647 pinos)

Curiosidade

Uma **syscall** é uma interrupção.

Redireciona para o sistema operacional

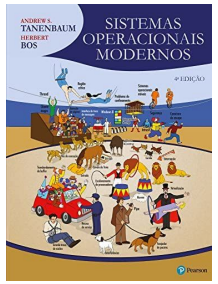
Se você abrir o assembly de um programa em C que faz um *printf*, vai notar que em algum momento, uma syscall ou similar é realizada para chamar o S.O., que vai realizar a impressão.

Exercícios

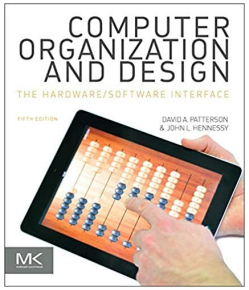
1. Abra o simulador Mars e verifique que processadores MIPS realmente possuem os registradores *cause* e *EPC*. Eles ficam na aba "Coproc 0" - Coproc 0 é comumente o coprocessador de exceções do MIPS, enquanto Coproc 1 é o coprocessador de ponto flutuante.
2. O S.O. é o responsável por tratar as exceções. 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).
3. O que é DMA (Direct Memory Access)?

Referências

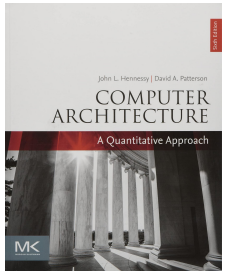
Tanenbaum, Bos. Sistemas operacionais modernos. 4a ed. 2016.



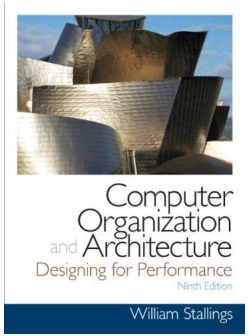
Patterson, Hennessy . Arquitetura e Organização de Computadores: A interface hardware/software. 2014.



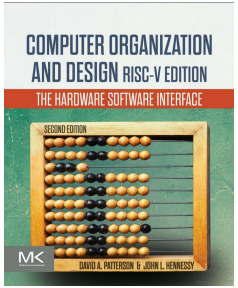
Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Stallings, W. Organização de Arquitetura de Computadores. 10a Ed. 2016.



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



Licença