RISC - Reduced Instruction Set Computer

MAC0344 - Arquitetura de Computadores
Prof. Siang Wun Song
Slides usados: https://www.ime.usp.br/∼song/mac344/oc-risc-mac344.pdf

Baseado em Tanenbaum - Structured Computer Organization e W. Stallings - Computer Organization and Architecture

Críticas à arquitetura CISC

- Os computadores CISC estão ficando demasiadamente complexos, com várias centenas de instruções e dezenas de modos de endereçamento.
- Um grande questionamento é se muitas das instruções complexas são de fato necessárias nos programas “típicos”.
- Levantamentos feitos mostram os comandos mais usados pelos programas. Eles se concentram em alguns poucos comandos, em geral bastante simples.

Levantamento sobre os comandos mais típicos

- Knuth, Wortman, Tanenbaum e Patterson mediram programas escritos em várias linguagens.

Comando	Fortran	C	Pascal
atribuicao :=	51%	38%	45%
if	10%	43%	29%
call	5%	12%	15%
loop	9%	3%	5%
goto	9%	3%	0%
outros	16%	1%	6%

Número de termos	%
1	80%
2	15%
3	3%
4	2%
5 ou mais	0%

Uso de variáveis locais e parâmetros

Numero de variáveis locais	%
0	22%
1	17%
2	20%
3	14%
4	7%
5 ou mais	20%

Número parâmetros	%
0	41%
1	19%
2	15%
3	9%
4	7%
5 ou mais	9%

Obervações sobre a velocidade da memória

- Arquitetura CISC se justificava pela velocidade lenta da memória no passado. Tendo buscado uma instrução na memória, a sua realização por dezenas ou centenas de microinstruções acontece na maior parte do tempo no processador, exceto quando operandos precisam ser buscados na memória. Isso é uma vantagem como uma forma de contornar o gargalo de von Neumann.
- No entanto, as memórias de hoje são bem mais rápidas, e dispensam talvez o uso de uma ROM dentro do processador contendo as microinstruções.

Obervações sobre chamada de procedimentos

- Chamada de procedimentos consume bastante tempo.
- Depende do número de parâmetros do procedimento.
- Depende do nível de encadeamento (*nested loops*).
- Maioria das variáveis são locais.
- Como podemos agilizar chamada de procedimentos? Veremos daqui a pouco.

Obervações sobre interrupções

- Interrupções são eventos que requerem a atenção do Sistema Operacional (S.O.)
- Interrupções podem se originar de dispositivos de entrada e saída, ou de uma instrução de software que requer atenção do S.O.
- Uma interrupção pode ser interrompida por outra de maior prioridade, dando origem a um encadeamento de interrupções.
- Cada interrupção tem seu contexto próprio que precisa ser salvo quando a interrupção é interrompida, para poder continuar quando retornar.
- Como podemos agilizar o armazenamento e restauração de contextos? Veremos daqui a pouco.

Surgimento da arquitetura RISC

Com as constatações sobre a arquitetura CISC:

- Um microprograma complexo significa maior tempo para decodificar e executar uma instrução complexa, muitas das quais raramente são usadas.
- A vantagem da CISC por causa da memória lenta já não vale diante de memórias modernas mais rápidas.

Surgiu a arquetitura RISC: Reduced Instruction Set Computer.

Característica da arquitetura RISC

- Não há microprograma para interpretar as instruções.
- Existe um conjunto reduzido de instruções simples RISC parecidas com as microinstruções da arquitetura CISC.
- O código gerado pelos compiladores é constituído de instruções simples desse conjunto reduzido. Essas instruções são armazenadas na memória RAM, buscadas e executadas diretamente em hardware na CPU, sem nenhuma interpretação.
- As instruções são executadas na sua maioria em apenas um ciclo da máquina.

Os primeiros computadores RISC

- IBM 801 (1980): é o antecessor do IBM PC/RT (Risc Technology).
- Berkeley RISC I e RISC II (1980 e 1981): projetado por David Patterson e Carlo Séquin; inspirou o projeto do processador SPARC, da SUN Microsystems.
- Stanford MIPS (1981): projetado por John Hennessy; deu origem a MIPS Computer Systems.

John Hennesy e David Patterson

Source: A.C.M.

- John Hennesy é professor da Universidade de Stanford, tendo sido o 10.o Reitor daquela universidade. É um dos fundadores da MIPS Computer Systems Inc. (hoje MIPS Technologies Inc.).
- David Patterson é professor da Universidade de California, Berkeley. Ele é o vice-chair do Board of Directors da RISC-V Foundation. Contribuiu no desenvolvimento dos discos RAID.
- Hennesy e Patterson ganharam o 2017 Turing Award, pelas suas contribuições no desenvolvimento da arquitetura RISC, hoje usada em 99% de novos *chips* (segundo Wikipédia).
- O vídeo do discurso dos dois na premiação da Turing Award.
- Curiosidade: o inventor de Microprogramação (CISC) - Sir Maurice Wilkes - também recebeu o Turing Award em 1967 (50 anos atrás :-).

Comparação entre CISC e RISC

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

Source: W. Stallings

Como é definido o conjunto de instruções RISC

Para definir o conjunto de instruções de um computador RISC, os projetistas usam a seguinte regra de ouro:

- Analisar aplicações para identificar operações-chave.
- Projetar o processador que seja eficiente para essas operações.
- Projetar instruções que realizam as operações-chave.
- Acrescentar mais instruções necessárias, cuidando para não afetar a velocidade da máquina.

Poucos modos de endereçamento

- A maioria das instruções RISC envolvem endereçamento por registrador, sem acesso à memória.
- Portanto as instruções são executadas em um ciclo.
- Pergunta: mas como os valores entram ou saem dos registradores?
  Resposta: há duas instruções LOAD e STORE que acessam a memória.
- Assim, retornamos à origem, como eram os computadores antigos.
- LOAD e STORE em geral levam mais de um ciclo para completar (por envolver acesso a memória).
- Para adiantar as coisas, usa-se o conceito de *pipelining*

Uso de pipelining

- A regra de ouro exige a execução de uma instrução por ciclo.
- No caso de LOAD e STORE, como esses levam mais de um ciclo, a regra de ouro será relaxada.
- Ao invés de exigir a execução de uma instrução por ciclo, será exigido o início da execução de uma instrução em cada ciclo.
- Suponha que a execução de uma instrução envolve 2 etapas (ou 3 etapas no caso de LOAD e STORE):
  1. busca a instrução da memória
  2. executa a instrução
  3. etapa adicional, no caso de LOAD e STORE
- Cada uma dessas etapas são executadas por circuitos próprios, numa espécie de linha de montagem (pipelining).

Uso de pipelining

Ciclo	1	2	3	4	5	6	7	8	9	10
Busca instrução	1	2	L	4	5	6	S	8	9	10
Execuçaõ instrução		1	2	L	**4**	5	6	S	**8**	9
Ref. memória					L				S	

- No ciclo 1, a instrução 1 é buscada. No ciclo 2, a instrução 2 é buscada e a instrução 1 executada.
- No ciclo 3, a instrução 3 (marcada L para significar LOAD) é buscada e a instrução 2 executada.
- No ciclo 4, a instrução L é iniciada e, como envolve acesso à memória, deve levar mais um ciclo para ser completada.
- No ciclo 5, algo interessante acontece (marcado pelo círculo). A instrução **4** é executada, embora a instrução L não tenha sido completada ainda. Isso é possível desde que a instrução **4** não utilize o registrador que está sendo carregado pela instrução L.

Problema de instruções de desvio

Valor de PC	10	11	12	13	14	29	30
Busca instrução	inst A	inst B	inst C	goto 29	inst E	inst X	inst Y
Executa instrução		inst A	inst B	inst C	goto 29	inst E !!!	inst X

- A execução de instruções por pipelining tem o velho problema do desvio.
- Quando a instrução é um desvio, a linha de montagem já preechida precisa ser descartada. Perde-se assim tempo com a pré-busca.
- No exemplo acima, o endereço 13 contém goto 29 e o endereço 14 contém inst E.
- A execução de goto 29 modifica o PC para 29, que contém a próxima instrução a executar, ao invés da próxima na sequência. Portanto inst E deve ser descartada.
- Como podemos aproveitar a vantagem de pipelining mesmo com desvio?

Problema de instruções de desvio

Pipeline original:

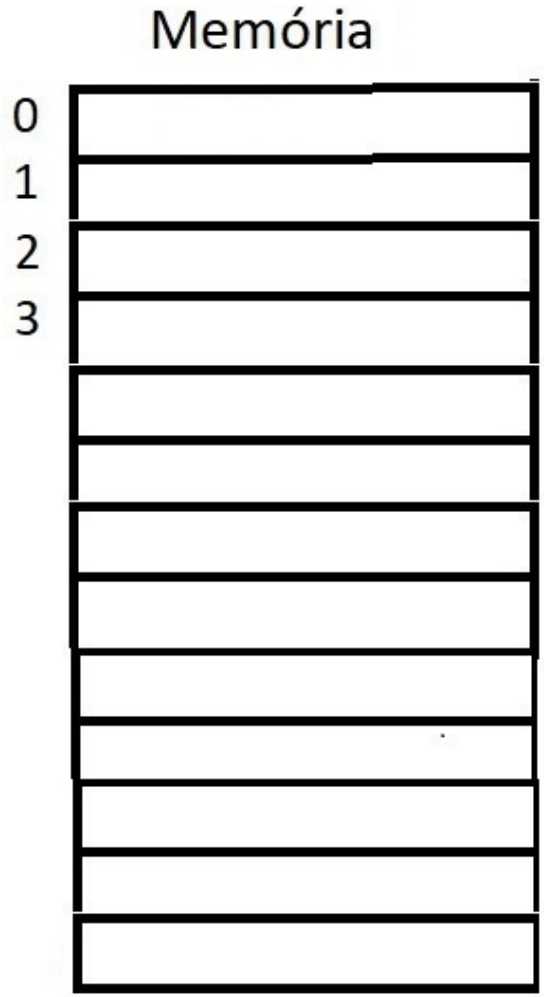
Valor de PC	10	11	12	13	14	29	30
Busca instrução	inst A	inst B	inst C	goto 29	inst E	inst X	inst Y
Executa instrução		inst A	inst B	inst C	goto 29	inst E !!!	inst X

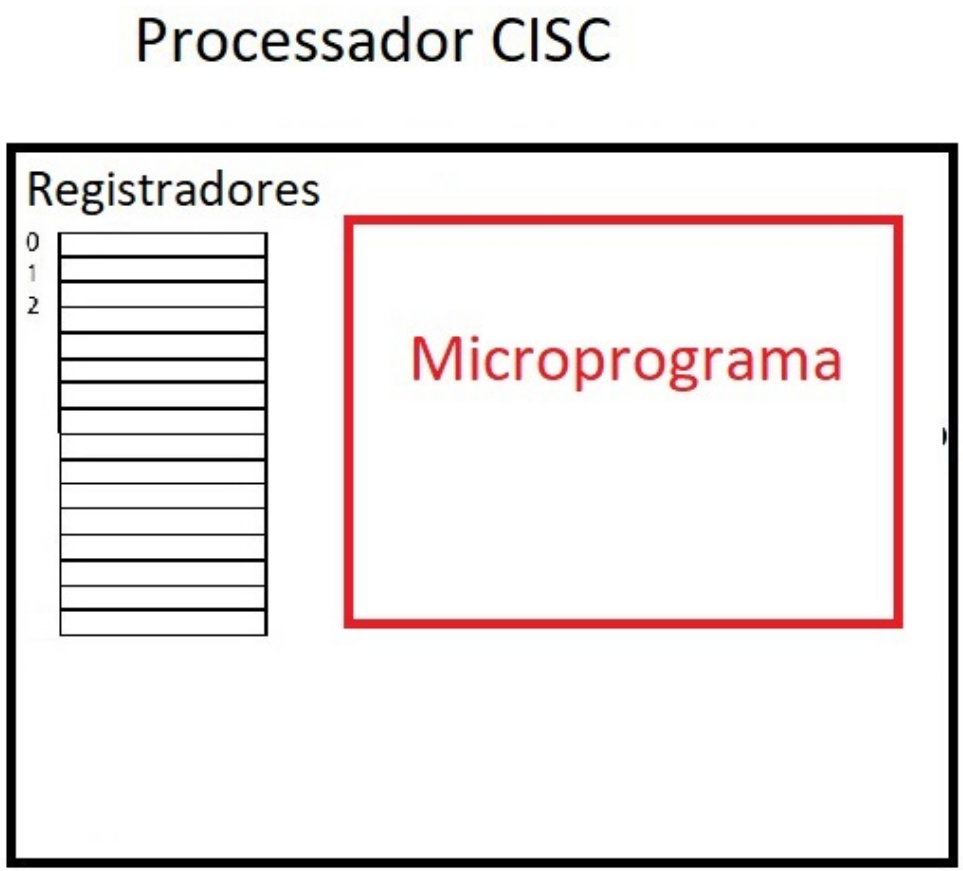
Pipeline modificado:

Valor de PC	10	11	12	13	29	30	31
Busca instrução	inst A	inst B	goto 29	inst C	inst X	inst Y	inst Z
Executa instrução		inst A	inst B	goto 29	inst C	inst X	inst Y

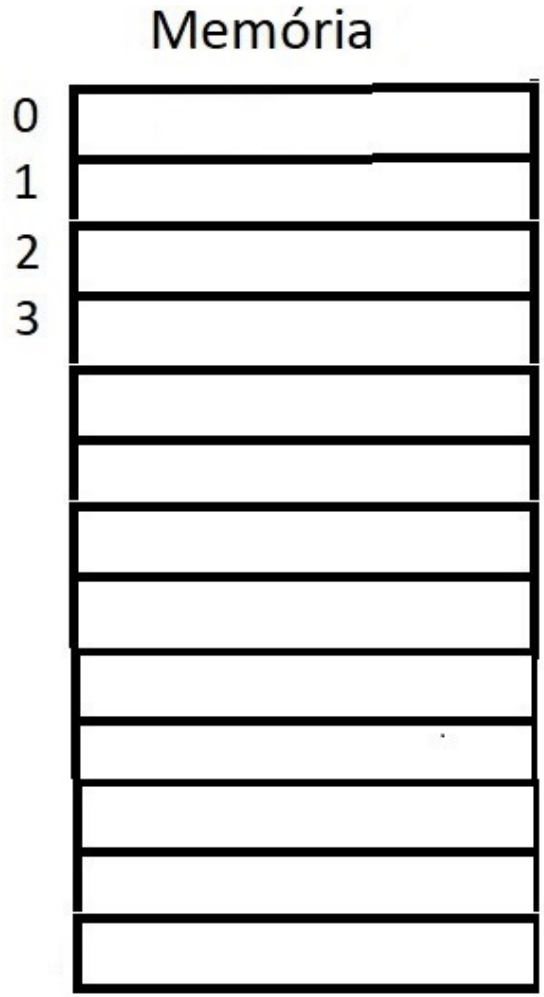
- Para resolver esse problema de desvio, o compilador pode colocar após a instrução de desvio uma instrução que logicamente deveria ser executada antes do desvio, e executar sempre a instrução seguinte a uma instrução de desvio.
- No pipeline modificado acima, a inst C foi colocada depois da instrução de desvio goto 29. Em outras palavras, as instruções inst C e goto 29 trocaram de lugar.
- Note o importante papel do compilador em máquinas RISC para melhorar o desempenho.

CISC contém um microprograma





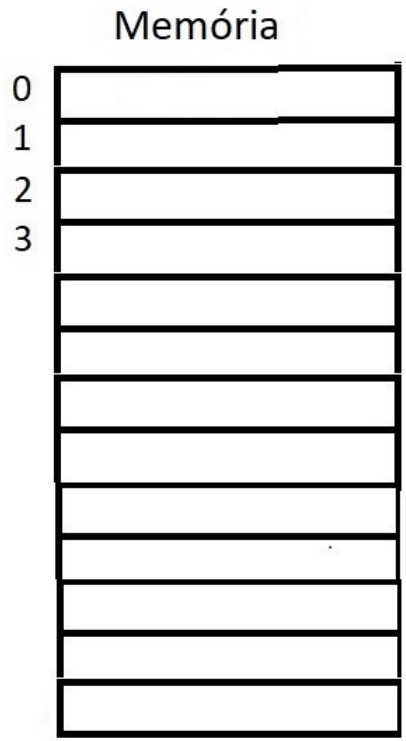
RISC não contém e no lugar tem muitos registradores



Há muitos registradores na máquina RISC

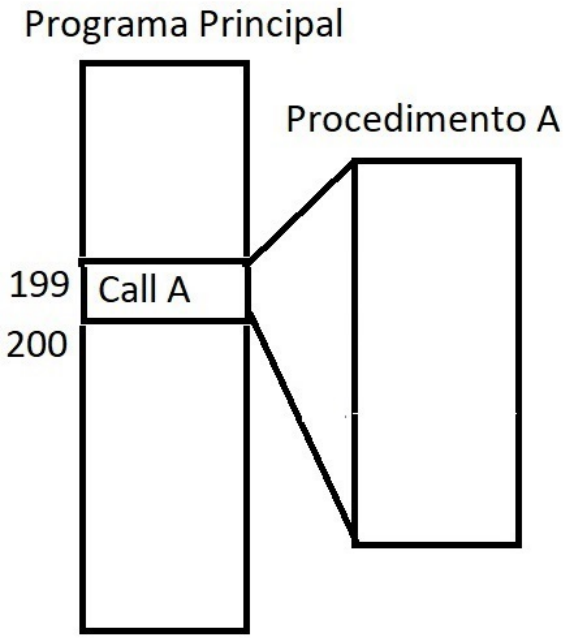
- O processador RISC não tem microprograma.
- Uma maneira de aproveitar esse espaço é colocar mais registradores.
- Não é incomum ter mais de 500 registradores numa CPU do tipo RISC.
- Mais registradores significa mais variáveis podem ser alocadas em registradores.
- Parte desses registradores adicionais pode ser usada para acelerar as chamadas de procedimentos.
- Outro uso dos registradores é para agilizar a mudança de contexto em interrupções.

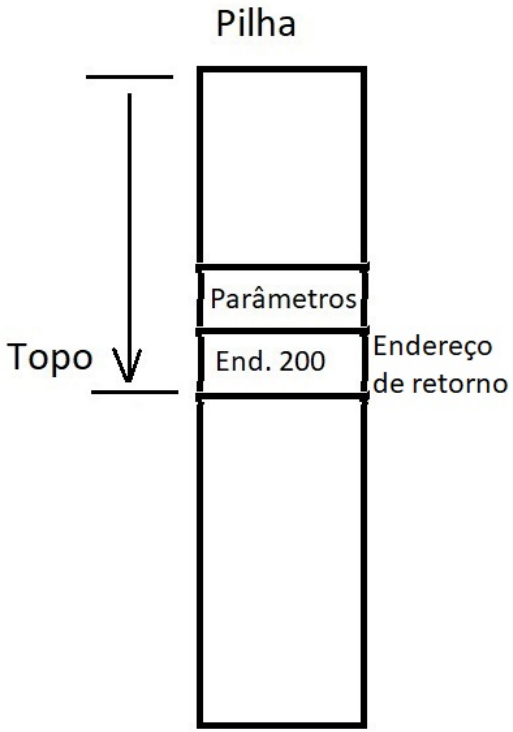
Mais variáveis alocadas em registradores



- Mais registradores permitem alocar mais variáveis locais escalares em registradores.
- Um conjunto de registradores pode ser alocado para armazenar variáveis globais.
- Isso reduz acesso à memória.

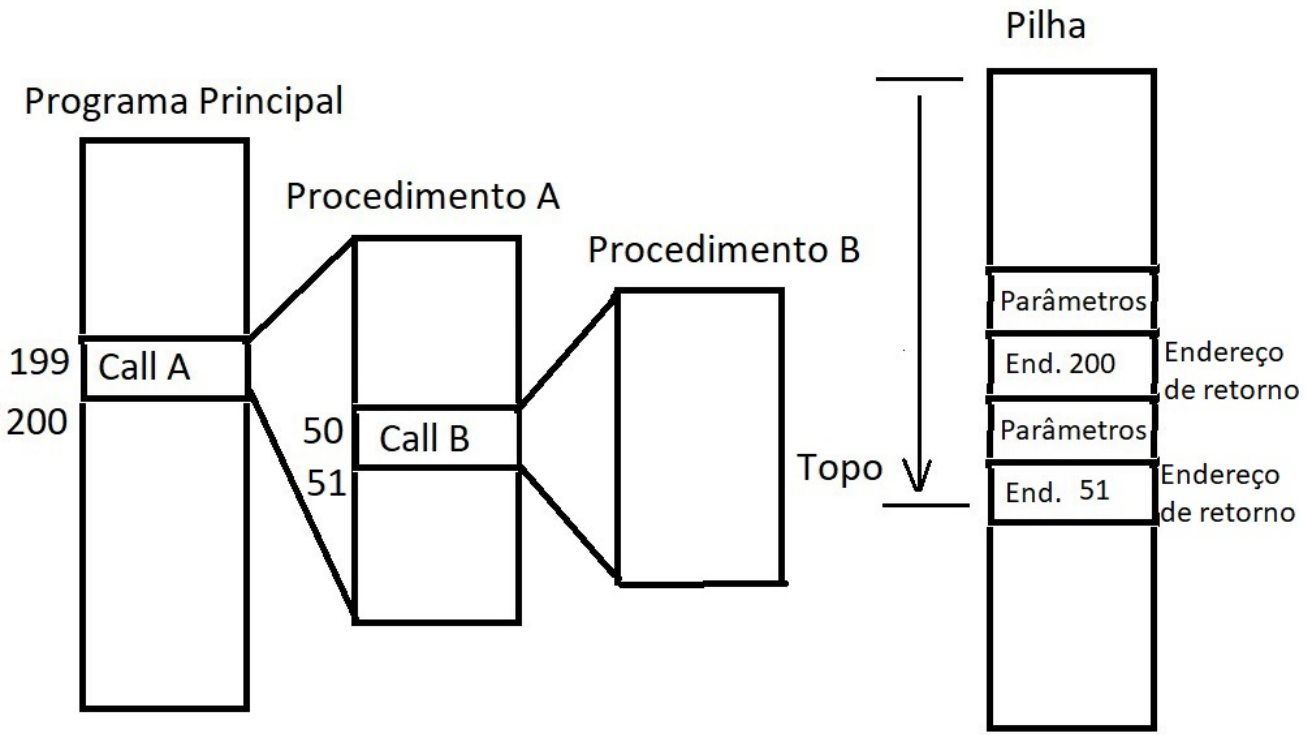
Chamada de procedimentos





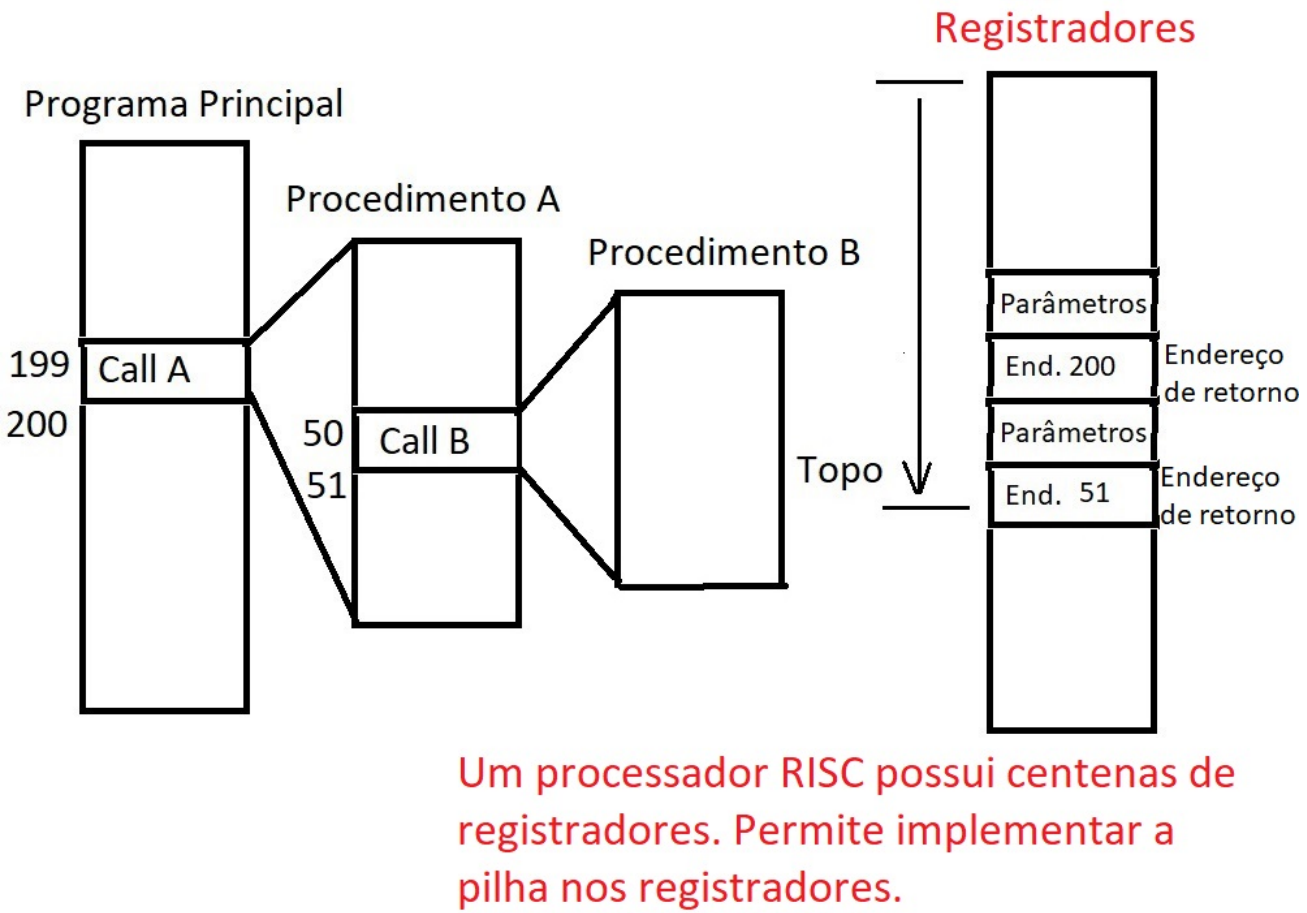
- Quando um procedimento é chamado, uma pilha é usada para os parâmetros de chamada, variáveis locais e o endereço de retorno.
- A pilha é implementada na memória.
- Na figura, o programa principal chama o procedimento A.
- Armazena na pilha os parâmetros, variáveis locais e o endereço de retorno (200).

Chamada de procedimentos



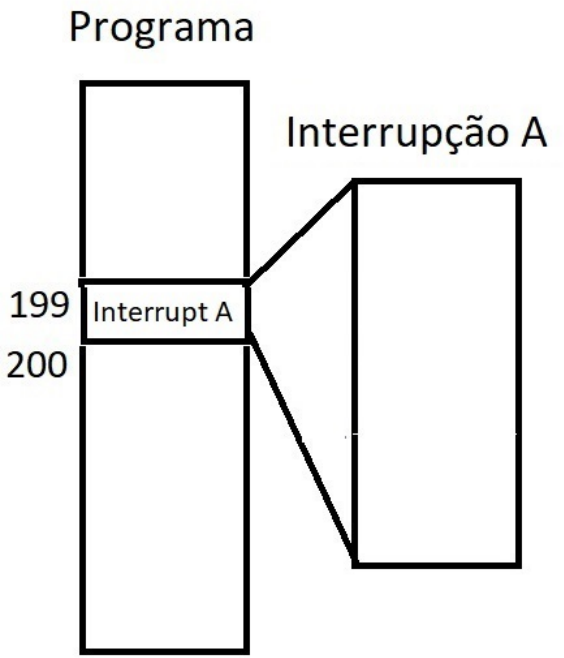
- O Procedimento A chama o Procedimento B.
- Empilha os dados necessários e o endereço de retorno (51).
- Terminado o Procedimento B, desempilha-se e usa o endereço de retorno (51) para retomar a A.
- Terminado o Procedimento A, desempilha-se e usa o endereço de retorno (200) para retomar o Programa Principal.

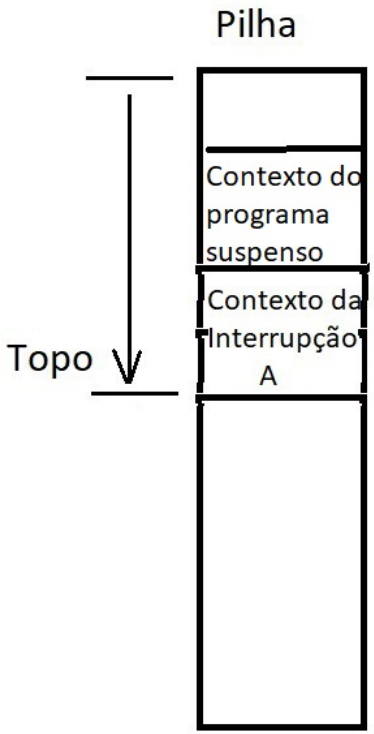
Chamada de procedimentos



- O processador CISC possui um grande número de registradores.
- Ao invés de usar uma pilha na memória, podemos usar registradores para fazer o papel da pilha.
- Fica muito mais rápido pois os registradores estão no processador.

Interrupções



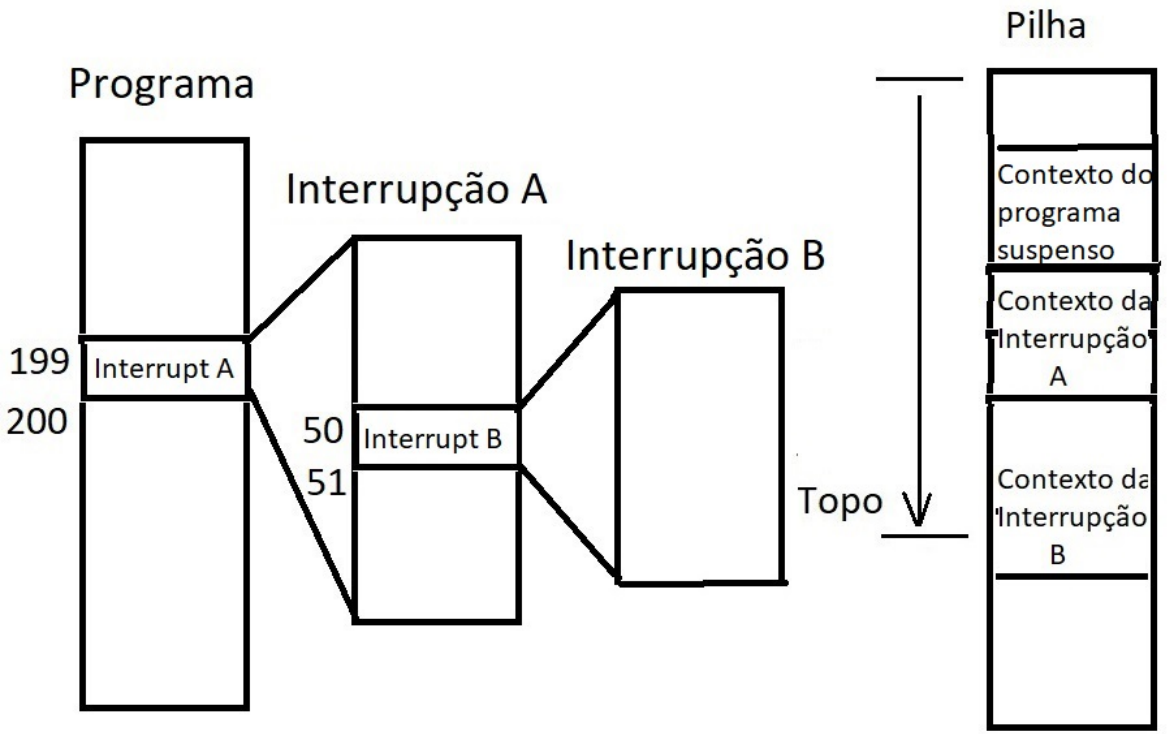


- Tratar uma interrupção é parecido a chamar um procedimento. Ambos precisam salvar dados do processo que será suspenso.
- Na figura, um processo precisa ser interrompido por causa da interrupção A.
- O S.O. entra em ação para tratar da interrupção A.
- A interrupção A tem o seu contexto próprio. Então o contexto do processo suspenso precisa ser salvo.

Encadeamento de interrupções

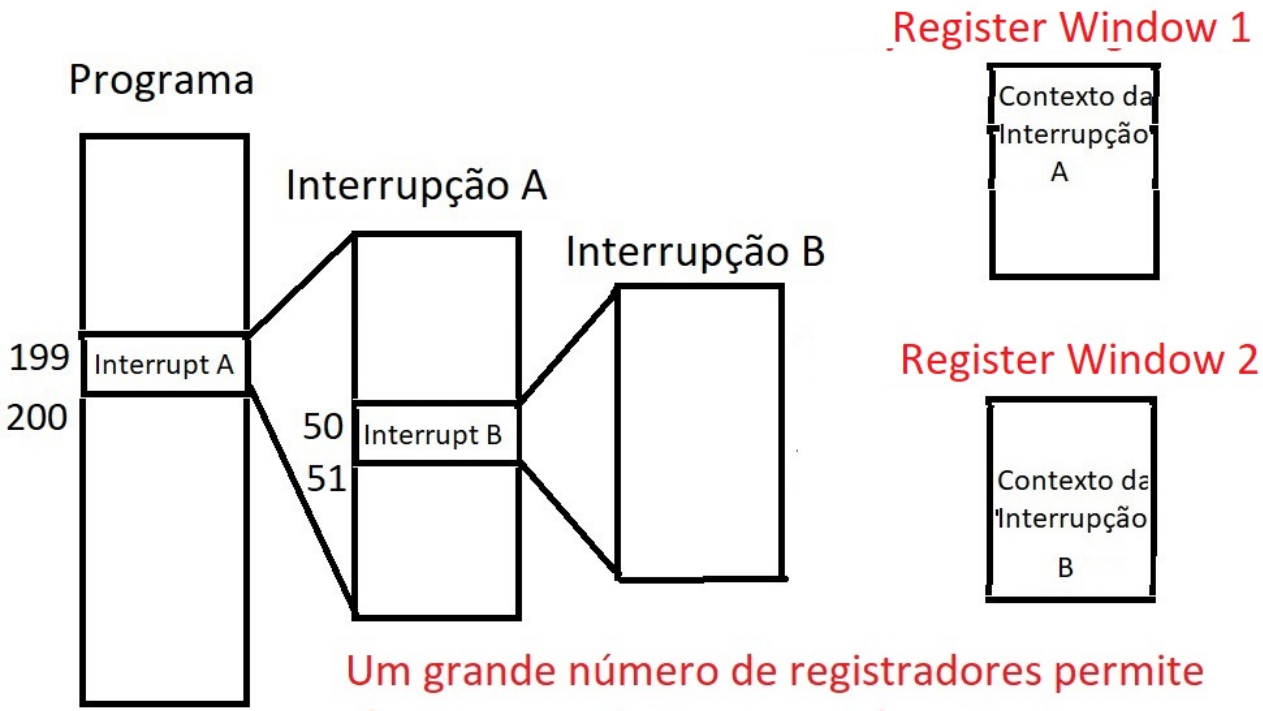
- Uma interrupção pode ser interrompida por outra, provocando um encadeamento de interrupções.
- Uma interrupção de prioridade maior pode interromper uma de prioridade inferior. Exemplo:
  - Um processo provoca uma interrupção (*page fault*) ao necessiar de uma página que não está na memória. O S.O. entra para tratar da interrupção.
  - Um dispositivo que controla a temperatura de um forno causa uma interrupção de alta prioridade para ter a atenção do S.O. pois a temperatura está alta demais. O S.O. passa a tratar essa nova interrupção.
  - Tendo tratada a nova interrupção, o S.O. retorna ao tratamento da interrupção anterior.
- O encadeamento de interrupções se assemelha a chamada de procedimentos quando um procedimento chamado por chamar outros.

Interrupções



- Quando o S.O. está tratando a interrupção A, ocorre uma interrupção B de prioridade maior.
- O S.O. passa a tratar da interrupção B, guardando o contexto da interrupção A para poder retomar depois.
- A interrupção B tem seu contexto próprio.

Interrupções



Um grande número de registradores permite alocar um conjunto para cada contexto.
A passagem de um contexto para outro é rápido, basta mudar um ponteiro a outro Register Window.

- Um computador RISC possui um grande número de registradores.
- Conjuntos diferentes de registradores, chamados *Register Windows*, podem ser alocados para representar os contextos de cada interrupção.
- Estando os registradores dentro do processador, o acesso ao contexto de cada interrupção fica ágil e basta mudar um ponteiro ao Register Window desejado.

Resumo - características da arquitetura RISC

- Grande número de registradores e uso de técnicas de otimização do compilador para otimizar uso de registradores.
- Um conjunto limitado de instruções de máquina simples.
- Uma instrução por ciclo.
- Operações registrador-para-registrador.
- Poucos modos de endereçamento.
- Projeto por hardware, não tem microprograma.
- Ênfase na otimização de *pipelining* de instruções.
- Requer mais esforço e tempo do compilador.