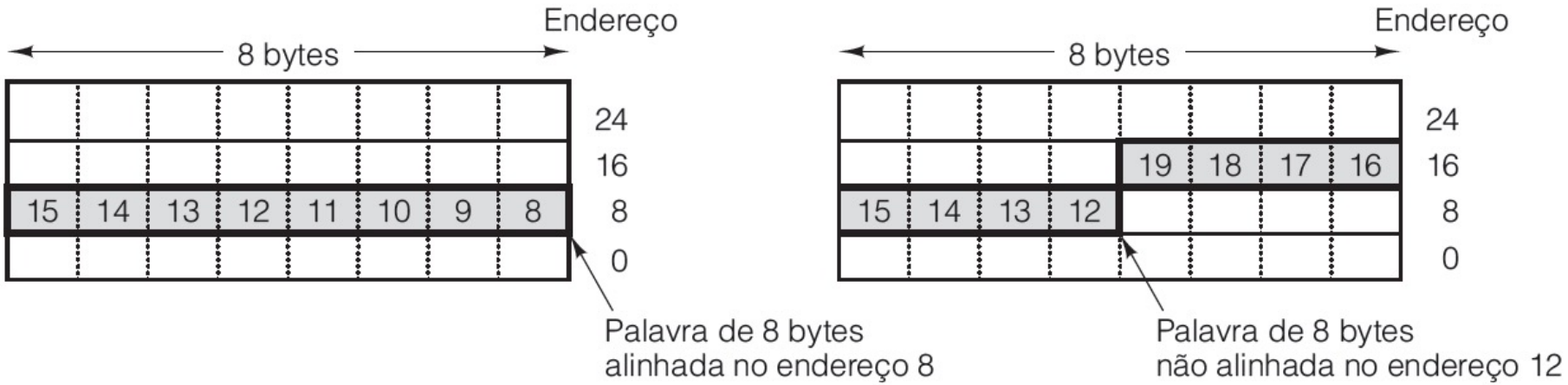
O nível de arquitetura do conjunto de instrução

Visão geral do nível ISA

- Código de nível ISA é o que um compilador produz.
- Para produzir código de nível ISA, o escritor de compilador tem de saber:
  - qual é o modelo de memória,
  - quais e quantos são os registradores,
  - quais tipos de dados e instruções estão disponíveis, e assim por diante.
- O conjunto de todas essas informações define o nível ISA.

Visão geral do nível ISA

- Em geral, os bytes são agrupados em palavras de 4 bytes (32 bits) ou 8 bytes (64 bits) com instruções disponíveis para manipular palavras inteiras.



Visão geral do nível ISA

- A maioria das máquinas tem um único espaço de endereço linear no nível ISA, que se estende do endereço 0 até algum máximo, geralmente 232 – 1 bytes ou 264 – 1 bytes.

- Ter um espaço de endereços separado para instruções e dados não é o mesmo que ter uma cache de nível 1 dividida.

- Também são possíveis modelos de memória intermediários, nos quais o hardware bloqueia automaticamente a emissão de certas referências à memória, mas não bloqueia outras.

Visão geral do nível ISA

- Todos os computadores têm alguns registradores visíveis no nível ISA.
- Eles estão lá para controlar a execução do programa, reter resultados temporários e para outras finalidades.
- Registradores de nível ISA podem ser divididos em duas categorias:
  - de uso especial e
  - de uso geral.

Visão geral do nível ISA

- A principal característica do nível ISA é o seu conjunto de instruções de máquina, que controlam o que a máquina pode fazer.

- Há sempre instruções LOAD e STORE (de uma forma ou de outra) para mover dados entre a memória e registradores e instruções MOVE para copiar dados entre os registradores.

- Instruções aritméticas estão sempre presentes, assim como instruções booleanas e aquelas para comparar itens de dados e desviar conforme os resultados.

Tipos de dados

- Os tipos de dados podem ser divididos em duas categorias:

  - numéricos e
  - não numéricos.

- O principal entre os tipos de dados numéricos são os inteiros.

- Para números que não podem ser expressos como um inteiro, como 3,5, são usados números de ponto flutuante.

Formatos de instrução

- Quatro formatos comuns de instrução:
  - Instrução sem endereço.

  - Instrução de um endereço.

  - Instrução de dois endereços.

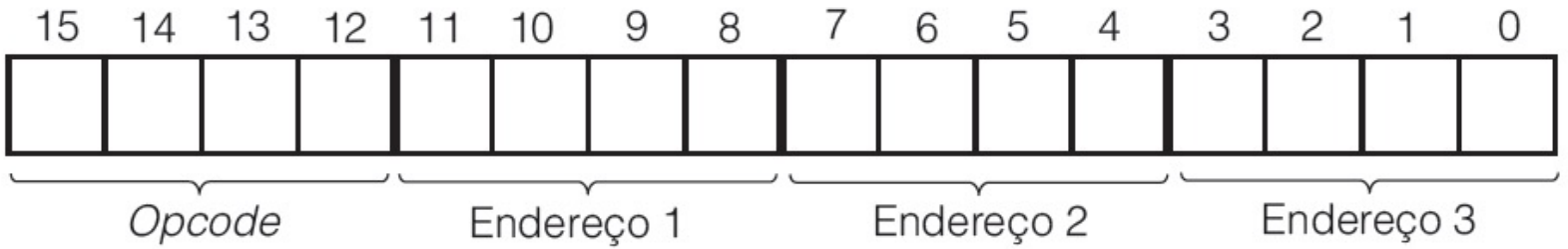
  - Instrução de três endereços.

Formatos de instrução

- Quando uma equipe de projeto de computador tem de escolher formatos de instruções para sua máquina, deve considerar vários fatores.
- Se todas as outras coisas forem iguais, instruções curtas são melhores do que as longas.
- Um segundo critério de projeto é espaço suficiente no formato da instrução para expressar todas as operações desejadas.
- Um terceiro critério se refere ao número de bits em um campo de endereço.

Expansão de *opcodes*

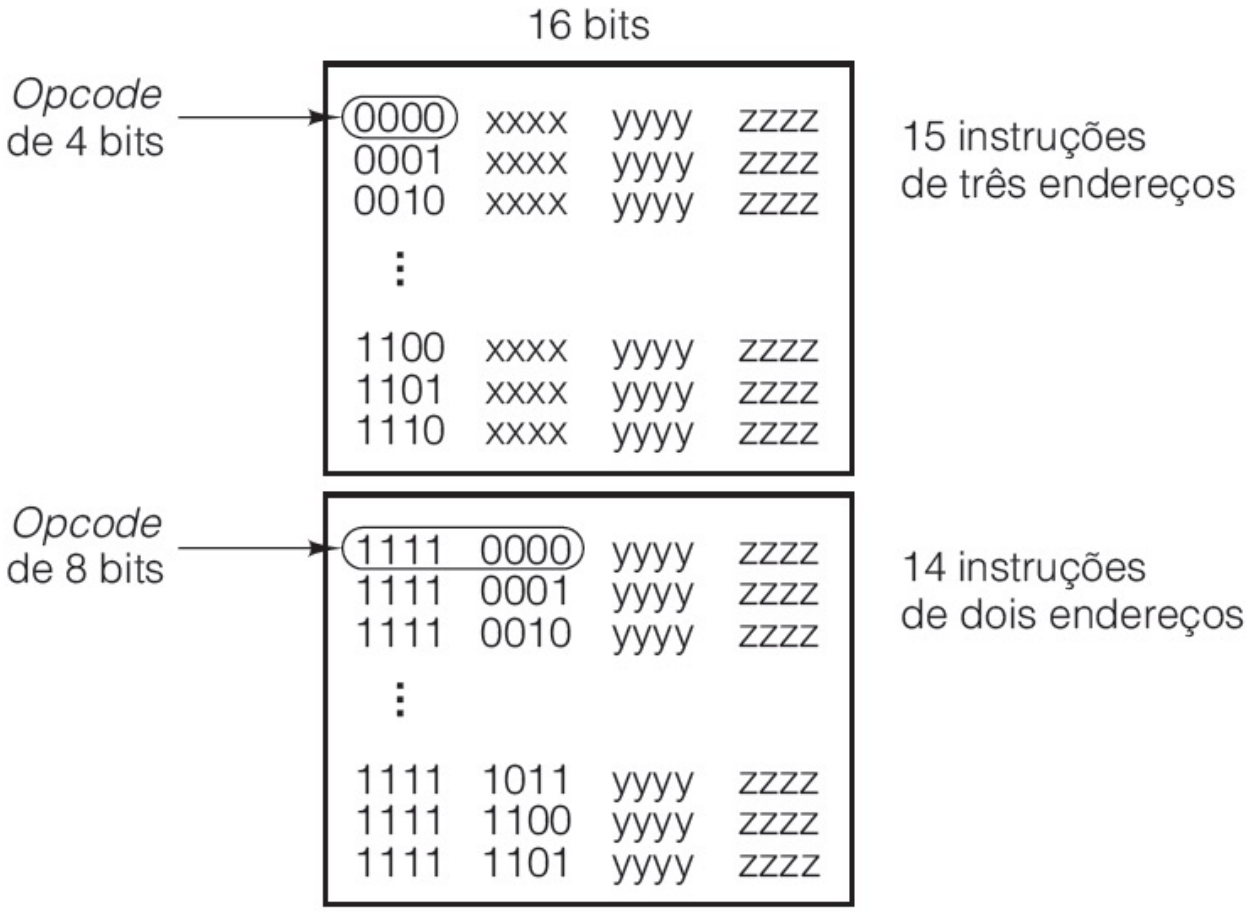
- Considere uma máquina na qual as instruções têm 16 bits de comprimento e os endereços têm 4 bits de comprimento:



- Contudo, se os projetistas precisarem de 15 instruções de três endereços, 14 instruções de dois endereços, 31 instruções de um endereço e 16 instruções sem absolutamente endereço algum, podem usar *opcodes* de 0 a 14 como instruções de três endereços, mas interpretar o *opcode* 15 de modo diferente.

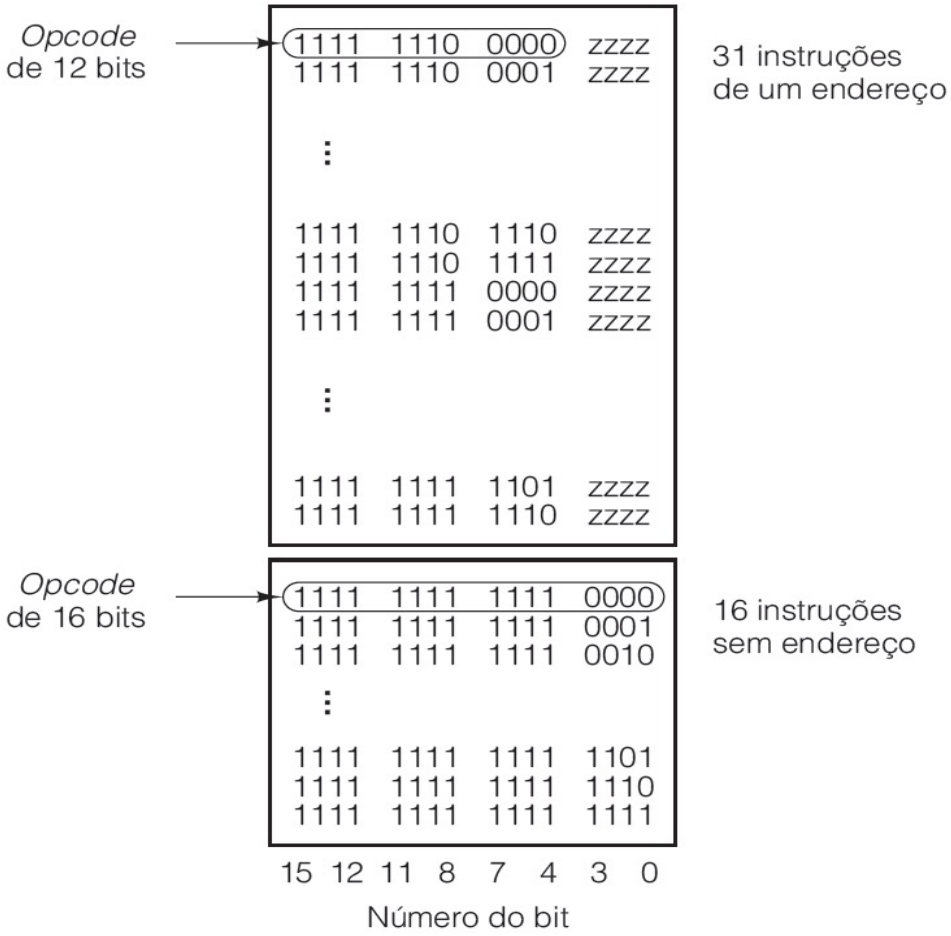
Expansão de *opcodes*

- Expansão de *opcode*.



Expansão de *opcodes*

- Expansão de *opcode*.



Endereçamento

- Grande parte das instruções tem operandos, portanto, é necessário algum modo de especificar onde eles estão.
- Esse assunto é denominado **endereçamento**.
- O modo mais simples de uma instrução especificar um operando é a parte da instrução referente ao endereço conter o operando de fato em vez de um endereço ou outra informação que descreva onde ele está.
- Instrução imediata para carregar 4 no registrador 1.

Endereçamento

- Um método para especificar um operando na memória é dar seu endereço completo.
- Esse modo é denominado **endereçamento direto**.
- **Endereçamento de registrador** é conceitualmente o mesmo que endereçamento direto, mas especifica um registrador em vez de uma localização de memória.
- Uma grande vantagem do **endereçamento indireto de registrador** é que ele pode referenciar a memória sem pagar o preço de ter um endereço de memória completo na instrução.

Endereçamento

- **Endereçamento indexado** é o nome que se dá ao endereçamento de memória que fornece um registrador (explícito ou implícito) mais um deslocamento constante.
- Algumas máquinas têm um modo de endereçamento no qual o endereço de memória é calculado somando dois registradores mais um deslocamento.
- É denominado **endereçamento de base indexado**.
- O limite final na redução de comprimentos de endereços é não ter endereços.

Endereçamento

- Instruções de desvio também precisam de modos de endereçamento para especificar o endereço de destino.
- O endereçamento direto é, sem dúvida, uma possibilidade, bastando incluir o endereço de destino completo na instrução.
- O endereçamento indireto de registrador permite que o programa calcule o endereço de destino, coloque-o em um registrador e então vá até lá.
- Outro modo razoável é o indexado, cujo deslocamento em relação a um registrador é uma distância conhecida.

Endereçamento

- Outra opção é o **endereçamento em relação ao PC**.
- Nesse modo, o deslocamento (com sinal) na própria instrução é adicionado ao contador de programa para obter o endereço de destino.
- Todos os *opcodes* devem permitir todos os modos de endereçamento onde quer que faça sentido.
- Todos os registradores devem estar disponíveis para todos os modos de registrador, incluindo o ponteiro de quadro (FP), o ponteiro de pilha (SP) e o contador de programa (PC).

Tipos de instrução

Instruções para movimento de dados

- Um nome melhor para instruções de movimento de dados seria instruções de “duplicação de dados”, mas o termo “movimento de dados” já está consagrado.
- Instruções de movimento de dados devem indicar, de alguma forma, a quantidade de dados a ser movida.
- Em algumas ISAs, existem instruções para mover quantidades variáveis de dados que vão de 1 bit até a memória inteira.

Tipos de instrução

Operações diádicas

- Operações diádicas são as que combinam dois operandos para produzir um resultado.
- Outro grupo de operações diádicas inclui as instruções booleanas.
- Em geral, estão presentes as operações AND, OR e NOT.
- Às vezes, também aparecem as operações EXCLUSIVE OR, NOR e NAND.

Tipos de instrução

Operações monádicas

- Operações monádicas têm um só operando e produzem um só resultado.
- Certas operações diádicas ocorrem com tanta frequência com determinados operandos que, às vezes, as ISAs têm operações monádicas para efetuá-las rapidamente.
- Instruções diádicas e monádicas costumam ser agrupadas conforme sua utilização, em vez de pelo número de operandos que requerem.

Tipos de instrução

Instruções de chamada de procedimento

- Uma pequena melhoria é fazer a instrução de chamada de procedimento armazenar o endereço de retorno na primeira palavra do procedimento.
- Uma melhoria maior é a instrução de chamada de procedimento colocar o endereço de retorno em um registrador.
- A melhor coisa para a instrução de chamada de procedimento fazer com o endereço de retorno é passá-lo para uma pilha.

Tipos de instrução

Controle de laço

- O laço do tipo teste no final tem a seguinte propriedade: o laço sempre será executado ao menos uma vez, mesmo que n seja menor ou igual a 0.

```
        i = 1;
L1:     primeira declaração;
        .
        .
        .
        última declaração;
        i = i + 1;
        if (i < n) goto L1;
```

Tipos de instrução

Controle de laço

- Abaixo outro modo de executar o teste, que funciona bem mesmo para *n* menor ou igual a 0.

```
            i = 1;
L1:         if (i > n) goto L2;
            primeira declaração;
            .
            .
            .
            última declaração;
            i = i + 1;
            goto L1;
L2:
```

Tipos de instrução

Entrada/Saída

- Há três esquemas diferentes de E/S em uso corrente em computadores pessoais. São eles:

1. E/S programada com espera ocupada.
2. E/S por interrupção.
3. E/S por DMA.

- Veja a seguir um exemplo de E/S programada.

Tipos de instrução

Entrada/Saída

```
public static void output_buffer(char buf[ ], int count) {
    // Produza um bloco de dados para o dispositivo
    int status, i, ready;
    for (i = 0; i < count; i++) {
        do {
            status = in(display_status_reg);      // obtenha estado
            ready = (status >> 7) & 0x01;         // isole o bit de pronto
        } while (ready != 1);
        out(display_buffer_reg, buf[i]);
    }
}
```

Tipos de instrução

Entrada/Saída

- Embora a E/S por interrupção seja um grande passo à frente em comparação com a E/S programada, está longe de ser perfeita.

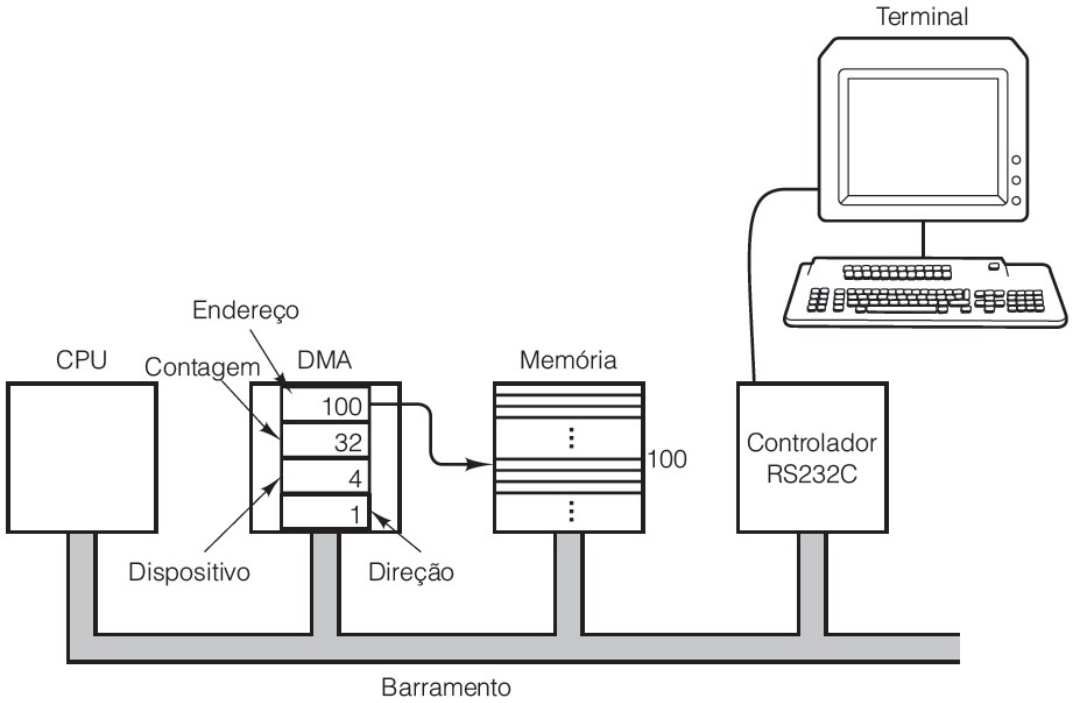
- O problema é que é requerida uma interrupção para todo caractere transmitido.

- Como processar uma interrupção é caro, precisamos de um meio de nos livrar da maioria das interrupções.

Tipos de instrução

Entrada/Saída

- A solução está em acrescentar ao sistema um novo chip, um controlador **DMA**, com acesso direto ao barramento.

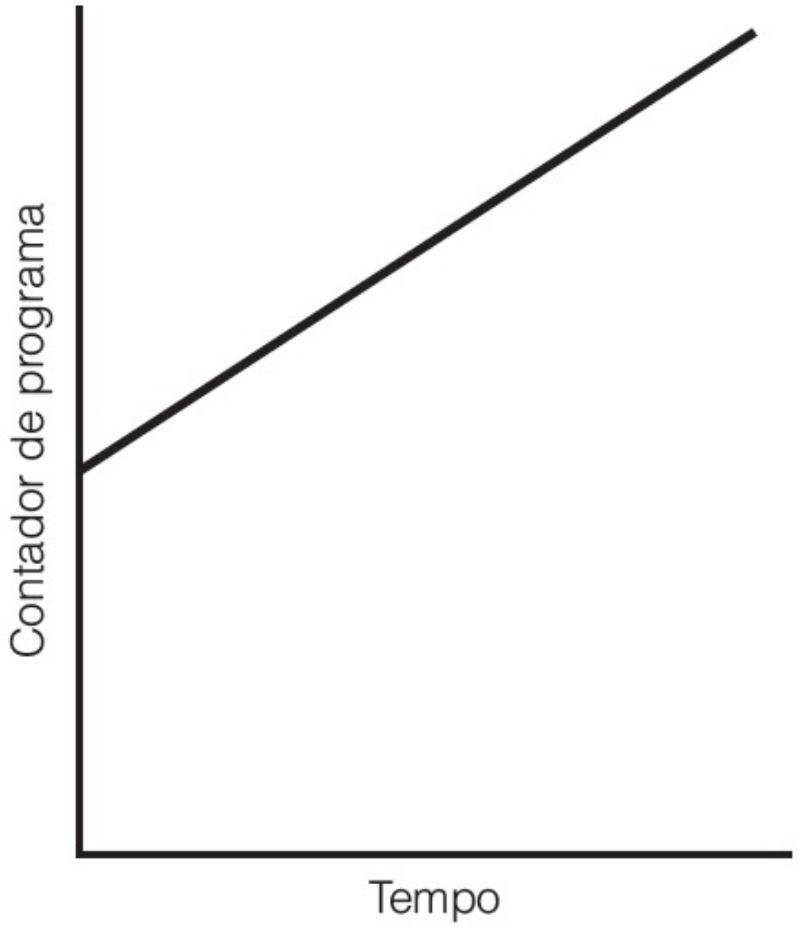


Fluxo de controle

- Fluxo de controle se refere à sequência em que as instruções são executadas dinamicamente, isto é, durante a execução do programa.
- A maioria das instruções não altera o fluxo de controle.
- A ordem dinâmica na qual o processador de fato executa as instruções é a mesma em que elas aparecem na listagem do programa, como mostra a figura a seguir.
- Quando há desvios presentes, o contador de programa deixa de ser uma função monotônica crescente do tempo, como mostra a figura seguinte.

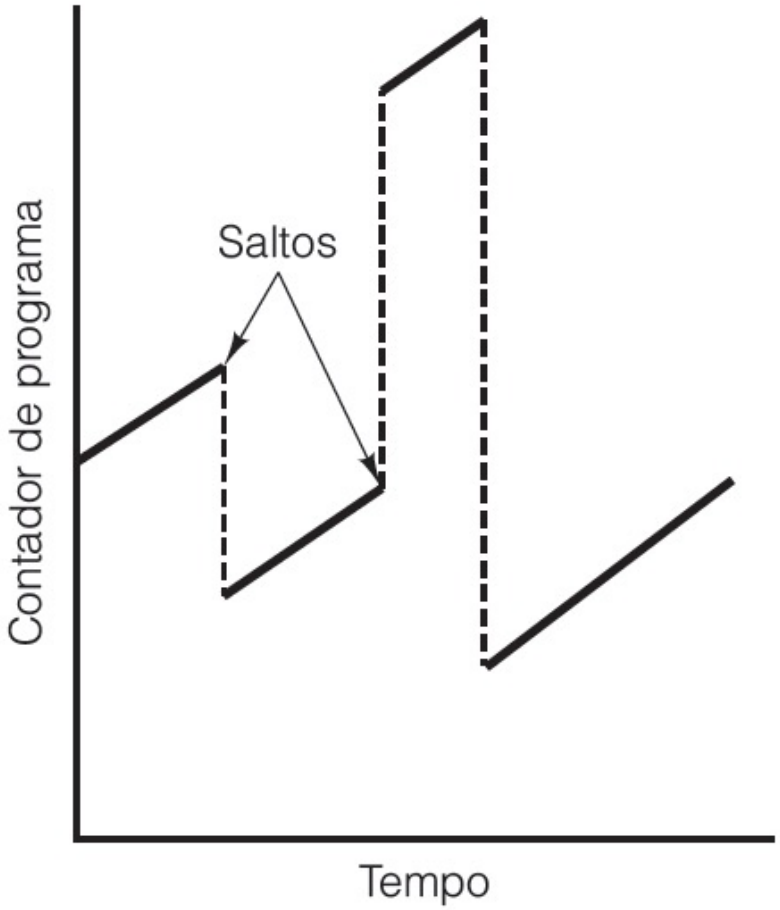
Fluxo de controle

- Contador de programa como função de tempo (ajustada) – sem desvios.



Fluxo de controle

- Contador de programa como função de tempo (ajustada) – com desvios.



Fluxo de controle

- A técnica mais importante para estruturar programas é o procedimento.
- Para entender um fragmento de código que contém uma chamada de procedimento, basta saber *o que* ele faz, e não *como* o faz.
- **Procedimento recursivo** é um procedimento que chama a si mesmo, direta ou indiretamente, por meio de uma cadeia de outros procedimentos.
- Para ter procedimentos recursivos, precisamos de uma pilha para armazenar os parâmetros e variáveis locais para cada chamada, o mesmo que tínhamos na IJVM.

Fluxo de controle

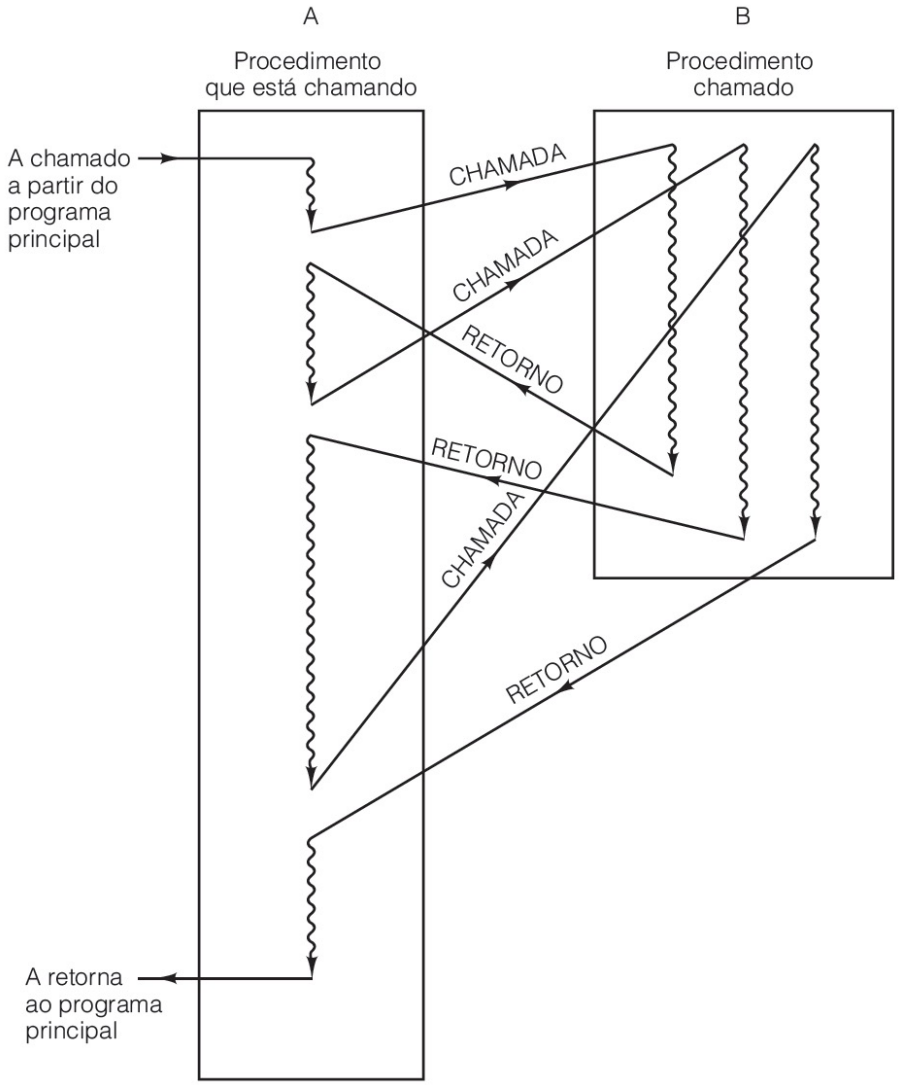
- Além do ponteiro de pilha, que aponta para o topo da pilha, muitas vezes é conveniente ter um ponteiro de quadro (FP – *frame pointer*), que aponta para um local fixo dentro do quadro.

- O código que salva o ponteiro de quadro antigo, ajusta o novo e adianta o ponteiro de pilha para reservar espaço para variáveis locais é denominado **prólogo de procedimento**.

- À saída do procedimento, a pilha tem de ser limpa novamente, o que é chamado **epílogo de procedimento**.

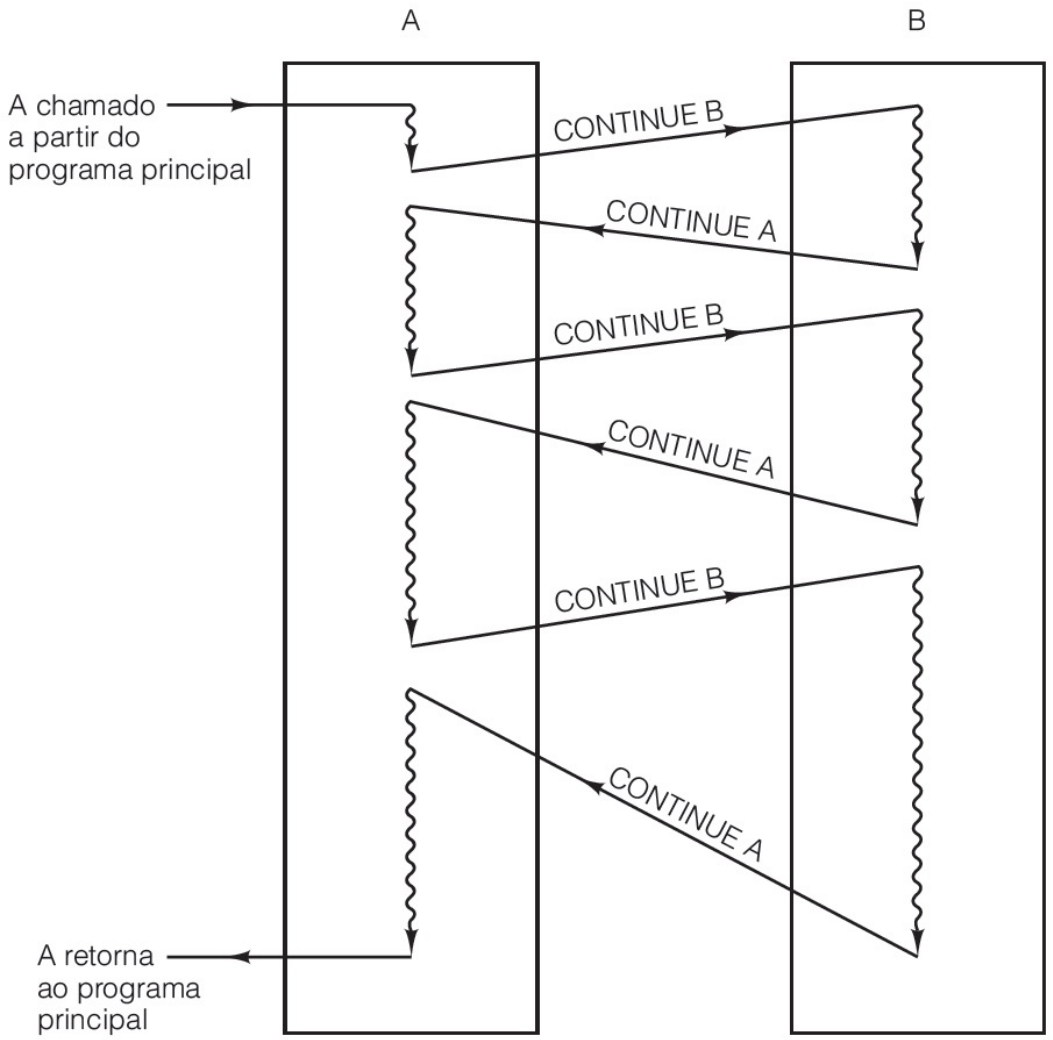
Fluxo de controle

Quando um procedimento é chamado, a execução do procedimento sempre começa na primeira declaração do procedimento.



Fluxo de controle

Quando uma corrotina é reiniciada, a execução começa na declaração de onde ela partiu da vez anterior, e não no início.



Fluxo de controle

- Uma exceção (*trap*) é um tipo de chamada de procedimento automática iniciada por alguma condição causada pelo programa, em geral uma condição importante, mas que ocorre raramente.
- Um bom exemplo é o transbordo (*overflow*).
- O ponto essencial de uma exceção é que ela é iniciada por alguma condição excepcional causada pelo próprio programa e detectada pelo hardware ou microprograma.
- A exceção pode ser efetuada por um teste explícito realizado pelo microprograma (ou hardware).

Fluxo de controle

- **Interrupções** são mudanças no fluxo de controle que não são causadas pelo programa em execução, mas por alguma outra coisa, em geral relacionada à E/S.

- A diferença essencial entre exceções e interrupções é a seguinte:

- *exceções* são síncronas com o programa e *interrupções* são assíncronas.

- De forma simplificada, as etapas são as seguintes:

Fluxo de controle

- **AÇÕES DO HARDWARE**

1. O controlador de dispositivo ativa uma linha de interrupção no barramento de sistema para iniciar a sequência de interrupção.

2. A CPU ativa um sinal de reconhecimento de interrupção no barramento.

3. Quando o controlador de dispositivo vê que seu sinal de interrupção foi reconhecido, coloca um inteiro pequeno nas linhas de dados para se identificar. Esse número é denominado **vetor de interrupção**.

Fluxo de controle

- **AÇÕES DO HARDWARE**

4. A CPU retira o vetor de interrupção do barramento e o salva temporariamente.

5. Então, a CPU passa o contador de programa e a PSW para a pilha.

6. Em seguida, a CPU localiza um novo contador de programa usando o vetor de interrupção como um índice para uma tabela na parte inferior da memória.

Fluxo de controle

- **AÇÕES DO SOFTWARE**

7. A primeira coisa que a rotina de serviço de interrupção faz é salvar todos os registradores que ela usa para poderem ser restaurados mais tarde.

8. O número do terminal pode ser encontrado pela leitura de algum registrador de dispositivo.

9. Agora pode ser lida qualquer outra informação sobre a interrupção, tal como códigos de estado.

10. Se ocorrer um erro de E/S, ele pode ser tratado nesse caso.

Fluxo de controle

- **AÇÕES DO SOFTWARE**

11. As variáveis globais, *ptr* e *count*, são atualizadas.

12. Se requerido, é produzido um código especial para informar ao dispositivo ou ao controlador de interrupção que a interrupção foi processada.

13. Restaura todos os registradores salvos.

14. Executa a instrução RETURN FROM INTERRUPT, devolvendo a CPU ao modo e estado em que ela estava exatamente antes de acontecer a interrupção.

Fluxo de controle

- Exemplo de sequência temporal de interrupções múltiplas.