

ISPS: Especificação Formal de Hardware

S. W. Song

MAC 412 - Organização de Computadores

ISPS

Veremos:

- Uma linguagem formal ISPS (de Bell e Newell) para especificação formal de hardware.
- **ISPS** significa **I**nstruction **S**et **P**rocessor **S**pecification.

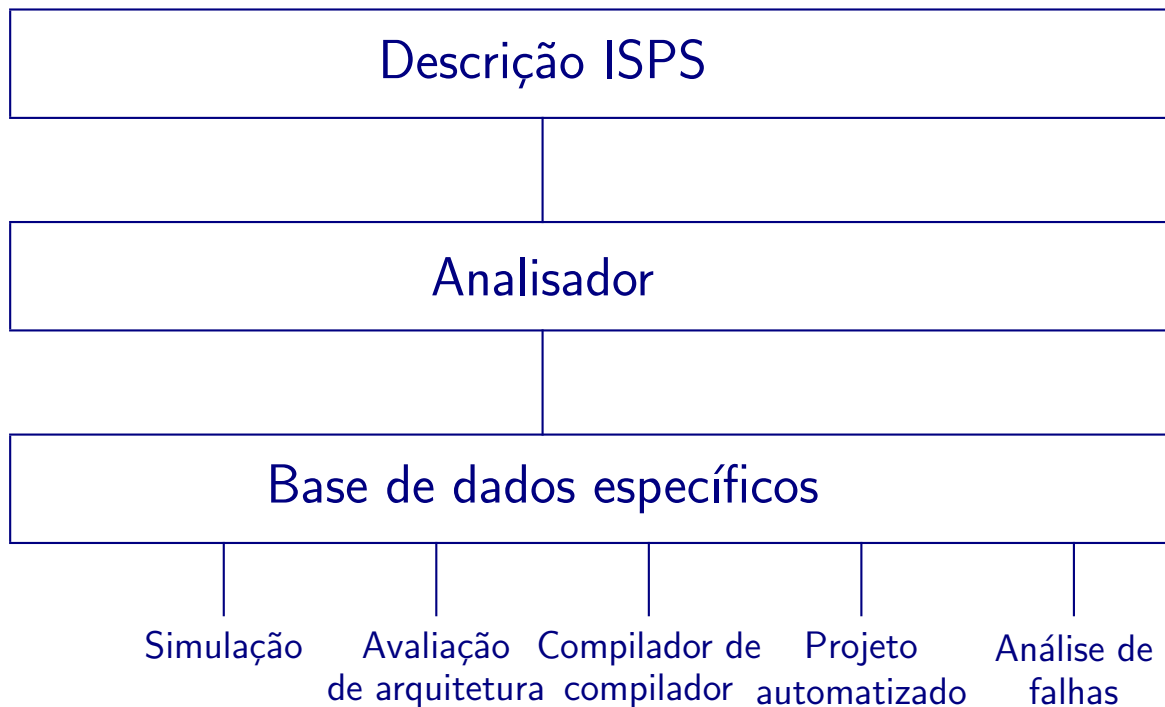
ISPS - Instruction Set Processor Specification

ISPS é uma linguagem de descrição formal de hardware do nível convencional de uma máquina, para descrever:

- Estado do processador Pc
- Estado da memória Mp
- Cálculo do endereço efetivo
- O que é executado em cada instrução
- Ciclo de interpretação de instruções

Utilidade de linguagem formal para especificar hardware

Uma especificação formal possibilita a simulação e síntese de hardware, avaliação de arquitetura, análise e diagnóstico de falhas, geração de compiladores, etc.

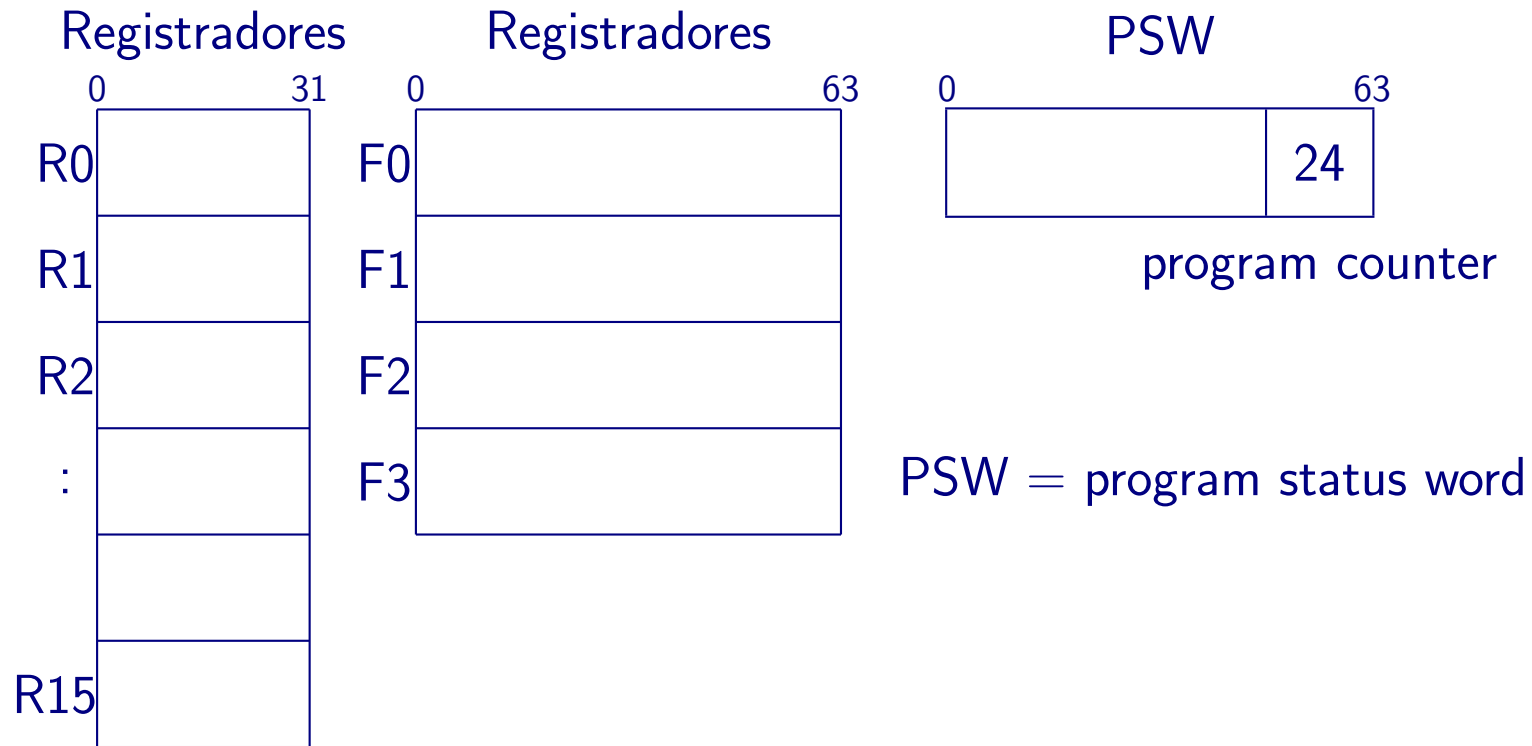


Estado do processador - Pc

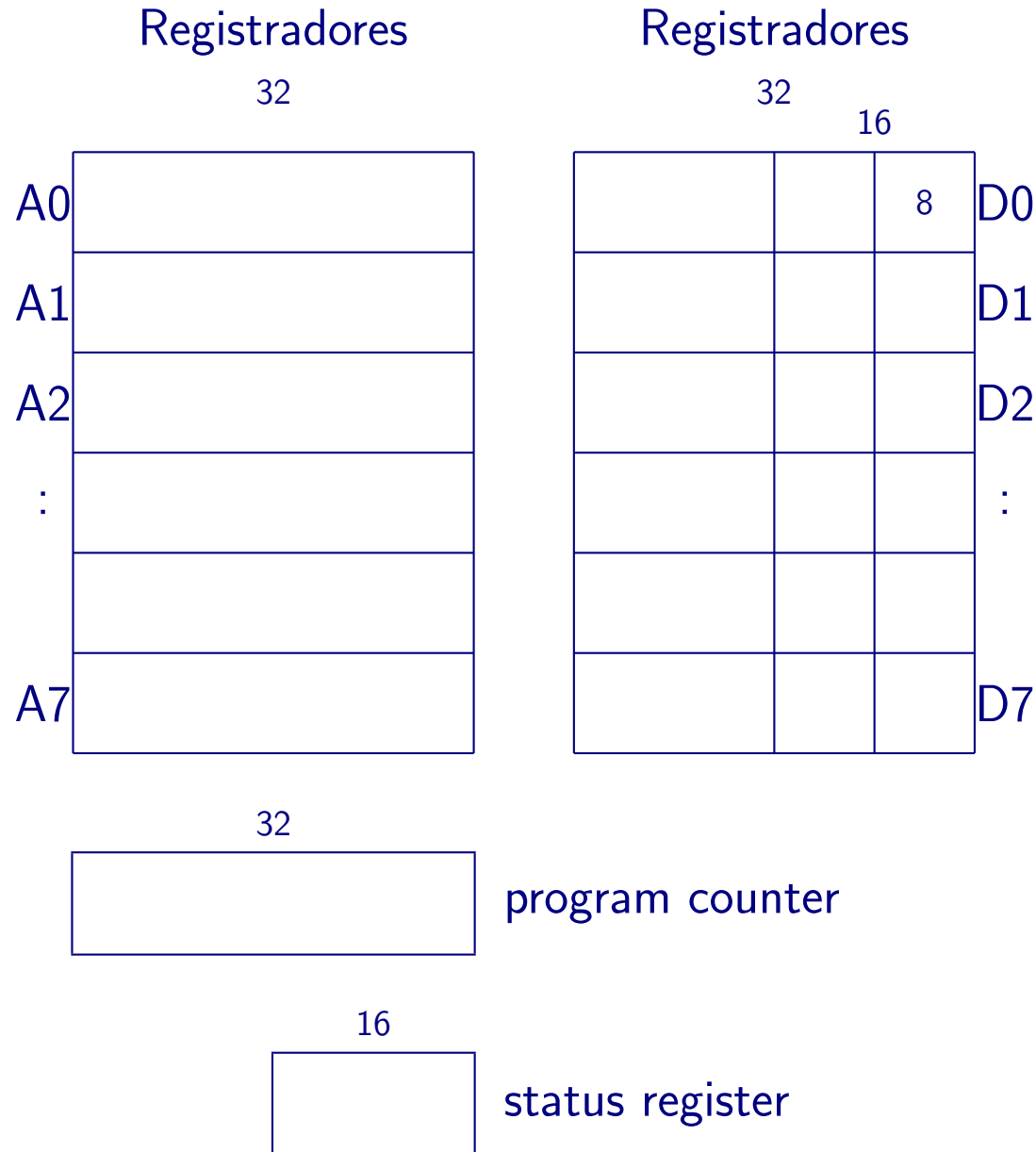
Registradores e flags (registradores de 1 bit) que guardam informações após a execução de uma instrução, informações essas disponíveis para a execução da próxima instrução.

- PC ou “program counter”
- apontador a pilha
- códigos de condição (resultados de comparação)
- prioridade de processo
- valores intermediários de computações

Exemplo: Estado do processador do IBM 370



Exemplo: Estado de processador do Motorola 68030



Estado da memória - Mp

É o conjunto das palavras da memória principal.

Exemplo: o estado de memória do IBM 370.

$$Mp[0 : 2^{24} - 1] < 0 : 7 >$$

ou seja 16 M palavras cada uma de 8 bits.

Veremos agora a sintaxe do ISPS.

ISPS - Sintaxe e Semântica

- Apresentamos uma introdução a ISPS, sem incluir todos os detalhes.
- Daremos a sintaxe e semântica da linguagem.
- Para fins didáticos, mostramos o ISPS de um minicomputador simples (PDP-8).

Estado da memória

No PDP-8, a memória é um array de palavras, de 0 a 255, especificados entre colchetes quadrados. Cada palavra é de 12 bits, numerados de 0 a 11, da esquerda para a direita, especificada pelos colchetes angulares.

`Mp\Primary.Memory [0:255] <0:11>`

- Mp é o nome oficial da memória
- Primary.Memory é um “apelido” ou comentário

Convenção no ISPS: nomes que começam com letra maiúscula para designar componentes concretas (físicas).

Estado do processador

```
Acc\Accumulator<0:11>,  
PC\Program.Counter<0:7>,  
IR\Instruction.Register<0:11>,  
    Op\Operation.Code<0:2>:=IR<0:2>,  
    Ibit\Indirect.Bit<>:=IR<3>,  
    Adr\Address<0:7>:=IR<4:11>,  
Interrupt.Enable<>,  
Interrupt.Request<>
```

Explicações a seguir.

Estado do processador

Acc\Accumulator<0:11> ,

PC\Program.Counter<0:7> ,

IR\Instruction.Register<0:11> ,

- Acc é um registrador (acumulador) de 12 bits.
- PC tem 8 bits, para endereçar uma memória de 256 palavras.
- IR contém a atual instrução (de 12 bits) em execução.

Estado do processador - cont.

```
IR\Instruction.Register<0:11>,  
  Op\Operation.Code<0:2>:=IR<0:2>,  
  Ibit\Indirect.Bit<>:=IR<3>,  
  Adr\Address<0:7>:=IR<4:11>,
```

- IR pode ser subdivido em Op, Ibit e Adr, respectivamente código de operação, bit indireto, e endereço.
- $\text{Adr}\langle 0 : 7 \rangle := \text{IR}\langle 4 : 11 \rangle$ significa que os bits 0 a 7 de Adr correspondem aos bits 4 a 11 do IR.
- Op, Ibit e Adr não são registradores novos; apenas representam nomes alternativos dos vários campos do IR, dando o formato da instrução.

Estado do processador - cont.

`Interrupt.Enable<>`,

`Interrupt.Request<>`

- Dois registradores de 1 bit cada.
- `Interrupt.Enable` vale 1 se o processador permite ser interrompido e 0 caso contrário.
- `Interrupt.Request` vale 1 se há pedido de interrupção e 0 caso contrário.

Cálculo do endereço efetivo

- A memória só tem 256 palavras; portanto os endereços são de 8 bits.
- O endereço efetivo (de um operando ou uma instrução) é obtido por uma regra de interpretação, especificado na forma de um algoritmo.

```
Z\Effective.Address<0:7>:=
```

```
  Begin
```

```
    If Ibit Eq1 0 => Z <- Adr;
```

```
    If Ibit Eq1 1 => Z <- Mp[Adr]<4:11>
```

```
  End
```

Cálculo do endereço efetivo - cont.

`Z\Effective.Address<0:7>`

- Z tem 8 bits e guarda o resultado do cálculo do endereço efetivo.

```
Begin
```

```
If Ibit Eq1 0 => Z <- Adr;
```

```
If Ibit Eq1 1 => Z <- Mp[Adr]<4:11>
```

```
End
```

- O corpo do procedimento entre Begin e End.
- São dois comandos separados por ponto-e-vírgula (;).

Cálculo do endereço efetivo - cont.

```
Begin  
If Ibit Eq1 0 => Z <- Adr;  
If Ibit Eq1 1 => Z <- Mp[Adr]<4:11>  
End
```

- O separador (;) separa comandos que podem ser executados **em paralelo**.
- No ISPS, o paralelismo ou concorrência é o padrão.
- Comandos executados em sequência são separados pela palavra **Next**, a ser visto a seguir.

Cálculo do endereço efetivo - cont.

```
Begin  
If Ibit Eql 0 => Z <- Adr;  
If Ibit Eql 1 => Z <- Mp[Adr]<4:11>  
End
```

- Se $Ibit = 0$, então o comando depois de $=>$ é executado.
- $=>$ é uma espécie de “then”.
- $<-$ representa o operador de atribuição: o valor da expressão do lado direito é atribuído ao lado esquerdo.

Cálculo do endereço efetivo - cont.

O corpo de Z pode também ser expresso de um outro modo:

```
Begin
```

```
Decode Ibit =>
```

```
    Begin
```

```
    0 := Z ← Adr,
```

```
    1 := Z ← Mp[Adr]
```

```
    End
```

```
End
```

- Decode Ibit => indica um grupo de comandos, um dos quais é executado dependendo do valor de Ibit.
- Decode é parecido com o comando “case” ou “switch”.

Interpretação da instrução

$IR \setminus \text{Instruction.Register} \langle 0:11 \rangle,$
 $Op \setminus \text{Operation.Code} \langle 0:2 \rangle := IR \langle 0:2 \rangle,$
 $Ibit \setminus \text{Indirect.Bit} \langle \rangle := IR \langle 3 \rangle,$
 $Adr \setminus \text{Address} \langle 0:7 \rangle := IR \langle 4:11 \rangle,$

- No PDP-8 o código de operação Op tem 3 bits.
- Temos assim 8 instruções.

Interpretação da instrução - cont.

```
IExec\Instruction.Execution:=  
  Begin  
    If Op Eq1 0 => Acc <- Acc And Mp[Z()];  
    If Op Eq1 1 => Acc <- Acc + Mp[Z()];  
    If Op Eq1 2 =>  
      Begin  
        Mp[Z] <- Mp[Z()] + 1 Next  
        If Mp[Z] Eq1 0 => PC <- PC+1  
      End;  
    If Op Eq1 3 => Mp[Z()]@Acc <- Acc@#0000;  
    If Op Eq1 4 => Stop();  
    If Op Eq1 5 => PC <- Z();  
    If Op Eq1 6 => ... (a ser visto)  
    If Op Eq1 7 => ... (a ser visto)  
  End
```

Veremos cada uma dessas 8 instruções.

Interpretação da instrução - cont.

If Op Eq1 0 => Acc <- Acc And Mp[Z()];

- Z() é usado para indicar uma chamada ao algoritmo cálculo de endereço efetivo Z.

If Op Eq1 1 => Acc <- Acc + Mp[Z()];

- Nada de novo para explicar.

Interpretação da instrução - cont.

```
If Op Eq1 2 =>  
  Begin  
    Mp[Z] <- Mp[Z()] + 1 Next  
    If Mp[Z] Eq1 0 => PC <- PC+1  
  End;
```

- **Next** separa comandos executados **sequencialmente**: o comando que precede Next deve ser completado antes da execução do comando seguinte.
- O cálculo de endereço é feito por **Z()**, apenas uma vez. **Z sem parêntesis** significa usar o endereço calculado.

Interpretação da instrução - cont.

A distinção entre $Z()$ e Z é importante quando o algoritmo Z introduz efeitos colaterais, como por exemplo:

```
Z\Effective.Address<0:7>:=  
  Begin  
    Decode Ibit =>  
      Begin  
        0:= Z <- Adr,  
        1:= Z <- Mp[Adr] <- Mp[Adr]+1  
      End  
    End  
  End
```


Interpretação da instrução - cont.

ISPS adota as seguintes notações

- # indica dígito octal

e.g. #7777 indica uma cadeia de 4 dígitos octais.

- ' indica dígito binário

e.g. '111111111111 (12 dígitos binários)

- " indica dígito hexadecimal

e.g. "AAA (3 dígitos hexadecimais).

Interpretação da instrução - cont.

If Op Eq1 3 => Mp[Z()]@Acc <- Acc@#0000;

- O operador @ indica a concatenação.

O resultado dessa instrução é armazenar 24 bits (12 do acumulador mais 12 bits 0), metade na memória e metade no acumulador. Isso é equivalente aos dois comandos abaixo:

```
Begin  
Mp[Z()] <- Acc Next  
Acc <- 0  
End
```

Interpretação da instrução - cont.

```
If Op Eq1 4 => Stop();
```

- Stop é uma instrução primitiva do ISPS para congelar o estado da máquina permanentemente.

```
If Op Eq1 5 => PC <- Z();
```

- É a instrução de desvio: altera a sequência normal de execução de instruções.

Explicaremos as **instruções 6 e 7** mais tarde.

Interpretação da instrução - cont.

Repetimos abaixo IExec:

```
IExec\Instruction.Execution:=  
  Begin  
    If Op Eq1 0 => Acc <- Acc And Mp[Z()];  
    If Op Eq1 1 => Acc <- Acc + Mp[Z()];  
    If Op Eq1 2 =>  
      Begin  
        Mp[Z] <- Mp[Z()] + 1 Next  
        If Mp[Z] Eq1 0 => PC <- PC+1  
      End;  
    If Op Eq1 3 => Mp[Z()]@Acc <- Acc@#0000;  
    If Op Eq1 4 => Stop();  
    If Op Eq1 5 => PC <- Z();  
    If Op Eq1 6 => ... (a ser visto)  
    If Op Eq1 7 => ... (a ser visto)  
  End
```

Ciclo de interpretação de instrução

```
ICycle\Interpretation.Cycle:=  
  Begin  
  Repeat  
    Begin  
    IR <- Mp[PC] Next  
    PC <- PC + 1 Next  
    IExec() Next  
    If Interrupt.Enable And Interrupt.Request =>  
      Begin  
        Mp[0] <- PC Next  
        PC <- 1  
      End  
    End  
  End  
End
```

O ciclo é busca instrução, execução instrução, verifica interrupção.

Ciclo de interpretação de instrução - cont.

```
IR ← Mp[PC] Next
```

```
PC ← PC + 1 Next
```

```
IExec() Next
```

Cada ciclo começa com:

- Buscar uma instrução na posição PC da memória.
- Carregar a instrução lida em IR.
- Incrementar PC de 1 (preparando para a próxima instrução).
- Executar a instrução chamando IExec().

Ciclo de interpretação de instrução - cont.

```
If Interrupt.Enable And Interrupt.Request =>
  Begin
    Mp[0] <- PC Next
    PC <- 1
  End
```

- Se Interrupt.Enable e Interrupt.Request ambos valem 1, então o pedido de interrupção será atendido.
- Ao atender ao pedido de interrupção, o valor de PC é salvo e guardado em Mp[0] e depois o PC passa a valer 1.
- A instrução da palavra 1 da memória será então executada, para iniciar o tratamento da interrupção.

Ciclo de interpretação de instrução - cont.

```
If Interrupt.Enable And Interrupt.Request =>  
  Begin  
    Mp[0] <- PC Next  
    PC <- 1  
  End
```

- O que deve conter a palavra 1 da memória?



- Na palavra 1 começa a rotina de tratamento de interrupção.
- É necessário salvar o contexto do processo a ser suspenso.

Ciclo de interpretação de instrução - cont.

```
If Interrupt.Enable And Interrupt.Request =>  
  Begin  
    Mp[0] <- PC Next  
    PC <- 1  
  End
```

- Depois de salvar o contexto do proceso pode-se iniciar o tratamento da interrupção propriamente dita.
 - Agora vem a parte delicada e sutil: antes de salvar o contexto, não podemos permitir nova interrupção.
-
- A primeira instrução deve **inibir** novas interrupções.

Ciclo de interpretação de instrução - cont.

```
If Op Eq 6 => Interrupt.Enable <- 0;
```

- Para inibir novas interrupções, o PDP-8 define a operação com código 6 acima.

Ciclo de interpretação de instrução - cont.

- Como vai ser o final do trecho de tratamento de interrupção? Quais as últimas instruções?



- Quando terminamos o tratamento da interrupção, devemos restaurar o valor do PC guardado, para retornar ao ponto suspenso antes de acontecer a interrupção.

- Antes disso, porém, devemos também habilitar novas interrupções.

- A última instrução deve ser **habilitar** interrupção ou **restaurar PC** ?

Ciclo de interpretação de instrução - cont.

- A última instrução deve ser **habilitar** interrupção ou **restaurar PC** ?



- Devemos habilitar interrupções e **imediatamente depois** restaurar o PC antigo para retomar o processo suspenso.
- Após habilitar a interrupção, gostaríamos que seu efeito seja retardado por um ciclo, para dar tempo à restauração do PC.
- Isso é conseguido através da instrução 7.

Ciclo de interpretação de instrução - cont.

```
If Op Eq1 7 =>  
  Begin  
    Interrupt.Enable <- 1 Next  
  Restart ICycle  
  End
```

- Restart significa ir diretamente para o começo do ciclo de interpretação de instrução.
- Isso é fundamental para que após a execução da instrução 7 (habilitando a interrupção), não se testem os bits Interrupt.Enable (que acabou de ficar 1) e Interrupt.Request (que pode ser 1, significando uma interrupção pendente).

Ciclo de interpretação de instrução - cont.

Discussão em classe:

- No PDP-8 suponha que é permitido o encadeamento de interrupções (i.e. uma interrupção também pode ser interrompida por outra).
- Que tipo de estrutura de dado deve ser usada e como ela é usada.
- Especifique sucintamente como deve ser as etapas principais da rotina de tratamento de interrupção.

Ciclo de interpretação de instrução - cont.

- Etapa 1: salvar contexto do processo sendo interrompido

(Primeira instrução) Inibir novas interrupções

Salvar em uma pilha o contexto atual: incluindo o valor do PC, os registradores.

Habilitar interrupções

- Etapa 2 tratamento da interrupção propriamente dita

Aqui começam as instruções para tratar a interrupção.

- Etapa 3 finalização para restaurar contexto anterior retornar

Inibir novas interrupções

Restaurar o contexto (desempilhando o que havia sido empilhado)

Habilitar interrupções e imediatamente depois Redefine PC para prosseguir à execução do processo interrompido.

Reescrevendo IExec completo

```
IExec\Instruction.Execution:=
  Begin
  Decode Op =>
    Begin
    0\AND:= Acc <- Acc And Mp[Z()],
    1\TAD:= Acc <- Acc + Mp[Z()], !Two's complement add
    2\ISZ:=          !Increment and skip if zero
      Begin
      Mp[Z] <- Mp[Z()] + 1 Next
      If Mp[Z] Eq 0 => PC <- PC+1
      End,
    3\DCA:= Mp[Z()]@Acc <- Acc@#0000, !Deposit and clear Acc
    4\HLT:= Stop(), !Halt
    5\JMP:= PC <- Z(), !Jump
    6\IOF:= Interrupt.Enable <- 0, !Interrupt OFF
    7\ION:=          !Interrupt ON
      Begin
      Interrupt.Enable <- 1 Next
      Restart ICycle
      End
    End
  End
End
```

Complementamos as instruções 6 e 7 e acrescentamos mnemônicos para as instruções.

Linguagens de especificação HDL

- IPSP é simples e didaticamente interessante.
- Há linguagens formais de especificação de hardware mais modernas. Os interessados podem consultar

Abel Hardware Description Language HDL:

<http://mzsola.iit.uni-miskolc.hu/cae/docs/xabel.html>

Verilog Hardware Description Language HDL:

<http://www.sutherland-hdl.com/>