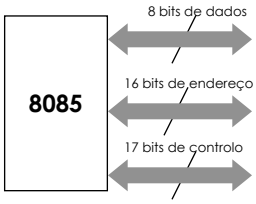
Microprocessador 8085

V.Lobo, Escola Naval
v1.6 2007

INTRODUÇÃO

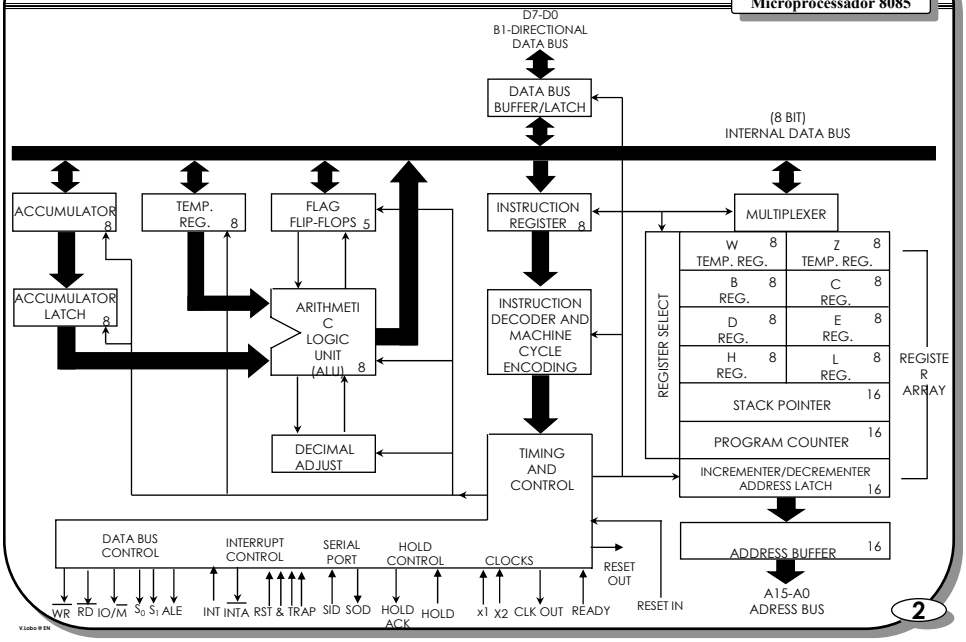
- **Intel 8080**
  - **Primeiro microprocessador de 8 bits da Intel**
  - **Sucessor do primeiro microprocessador do mundo (de 4 bits)**
- **Intel 8085**
  - **Versão melhorada do 8080**
  - **Primeiro microprocessador com grande sucesso comercial**
  - **Muito usado em instrução por ter uma arquitectura simples e "limpa"**
- **Arquitectura externa básica**
  - **Bus de dados de 8 bits**
  - **Bus de endereçamento de 16 bits (espaço de endereçamento de 64K)**



1

V.Lobo @ EN

Arquitectura interna



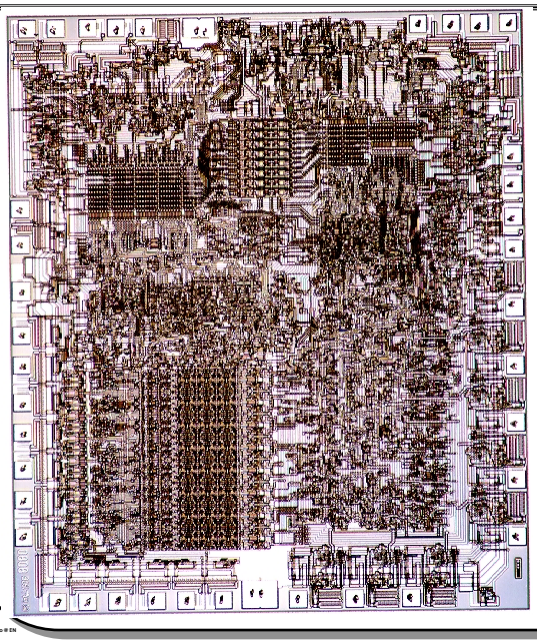
2

V.Lobo @ EN

Microprocessador 8085

V.Lobo, Escola Naval
v1.6 2007

Implementação em Silício



- Nº de transistors: 6000
- Início de fabrico: Abril 1974 (8080)
- Fabricado pela INTEL e por vários outros fabricantes sob licença (p.exemplo, Siemens, Philips, Texas, etc)
- Diferentes modelos, que se diferenciam por:
  - Frequência. de relógio
  - Tecnologia de construção (NMOS, CMOS, largura das pistas, etc)
  - Robustez de encapsulamento

REGISTOS

- Registos de uso geral
  - Existe 1 registo privilegiado chamado ACUMULADOR, onde é guardado:
    - →um dos operandos das operações aritméticas
    - →o resultado das operações aritméticas
  - Existem 6 registo de uso geral, que podem ser agrupados 2 a dois para formar "registos" de 16 bits
  - Dois desses registos H e L (High & Low) são usados para gerar endereços. O dado contido no endereço de memória apontado por HL é tratado por algumas instruções como se fosse um registo (chamado M)

A	
B	C
D	E
H	L

Microprocessador 8085

V.Lobo, Escola Naval
v1.6 2007

REGISTOS

- **Program Counter**
  - **Contém o endereço da próxima instrução a ser executada**
- **Stack Pointer**
  - **Contém o endereço do topo do stack (é gerido pelo μP)**
- **STACK**
  - **Estrutura computacional que permite guardar dados numa base LIFO (Last In First Out)**
  - **Comporta-se como uma pilha de livros onde podem ser postos livros, e retirados**
  - **O acesso ao stack é feito APENAS COM DUAS INSTRUÇÕES:**
    - **→PUSH - põe um dado (de 16 bits) no stack**
    - **→POP - retira um dado (de 16 bits) do stack**

Instruções:

PUSH BC	**PUSH DE**	**PUSH HL**	**PUSH PSW**
POP BC	**POP DE**	**POP HL**	**POP PSW**

LXI SP,*(endereço inicial do stack)*

V.Lobo @ EN

FLAGS

- **Existe um registo especial que guarda indicações sobre a última operação efectuada pelo μP.**
- **Esse registo tem 5 bits, ou *FLAGS***
- **Qualquer operação de aritmética ou lógica afecta essas flags**
- **Cada bit pode ser interrogado separadamente por instruções que necessitam de saber se a última operação provocou um *carry* se o resultado foi 0, etc.**

Z	S	CY	P	AC

V.Lobo @ EN

FLAGS

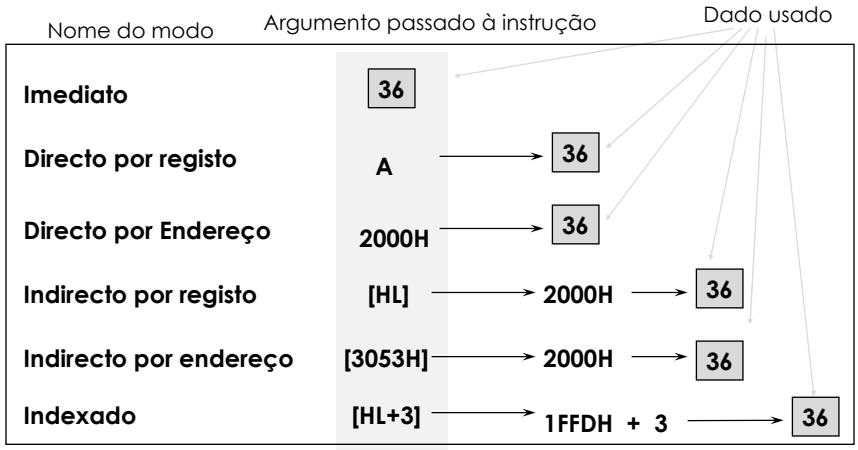
- **As flags são:**
  - **Z Zero**
    - **→1 = O resultado da última operação foi 0**
    - **→0 = O resultado da última operação não foi 0**
  - **S Sign (igual ao bit mais significativo; assume notação de complemento para 2)**
    - **→1 = O resultado da última operação é < 0**
    - **→0 = O resultado da última operação é ≥ 0**
  - **CY Carry**
    - **→1 = Houve Carry**
    - **→0 = Não houve Carry**
  - **P Parity**
    - **→1 = Paridade Par**
    - **→ 0 = Paridade Ímpar**
  - **AC AUXILARY CARRY**
    - **→1 = Houve Carry em BCD**
    - **→0 = Não houve Carry em BCD**

INSTRUCTION SET

- **O 8085 tem um conjunto vasto de instruções**
  - **Era um CISC para a sua época**
- **Cada instrução é um número ou *código máquina***
- **Para facilidade de leitura cada instrução é representada (no papel) por uma *mnemónica*, que se assemelha ao seu significado em inglês**
  - **Ex: A instrução "CFH" faz um "restart" e tem a mnemónica "RST"**
  - **Um programa escrito em mnemónicas diz-se escrito em Assembly Language, ou Assembler**
- **Vamos dividir as instruções em classes**
  - **Mover dados**
  - **Aritmética e lógica**
  - **Controlo de fluxo**
  - **Outras**

MOVER DADOS

- **Modos de endereçamento**
  - **Há várias maneiras de referir o dado que se pretende usar:**



MOVER DADOS

- ***MOV r1,r2***
  - **Move os dados (8 bits) do registo r2 para o registo r1**
  - Ex: MOV A,B ; põe no acumulador o conteúdo de B
    MOV H,D ; põe em H o conteúdo de D

- ***MOV r1,M / MOV M,r1***
  - **Move os dados (8 bits) do endereço apontado por HL para o registo r1, ou vice-versa**
  - Ex: MOV M,A ; põe o conteúdo do acumulador no ; endereço de memória apontado por HL

- **LDAX *rp* / STAX *rp*** **(Load Acumulador x...)**
  - **Move para o acumulador o conteúdo do endereço apontado pelo par de registos rp, ou vice-versa.**
  - Ex: STAX BC; põe o conteúdo do acumulador no ; endereço de memória apontado por BC
    LDAX HL; equivale a MOV A,M

MOVER DADOS

- **MVI *r1, data*$_{(8bits)}$ (move imediate)**
  - **Move para o registo r1 o dado indicado.**
  - Ex: MVI A,00 ; põe no acumulador o valor 0
    MVI B,A0H ; põe em B o valor 160
- **LXI *rp,data*$_{(16bits)}$ (load pair imediate)**
  - **Move para o par de registos rp o dado indicado.**
  - Ex: LXI BC,0000 ; põe no par BC o valor 0
    LXI HL,3F00H ; põe em HL o valor 3F00
- **XCHG (exchange)**
  - **Troca o conteúdo de HL com DE**

MOVER DADOS

- **LDA *addr* (Load Acumulator)**
  - **Mover um dado do endereço ADDR para o acumulador**
  - EX: LDA 1000 ; Carrega o acumulador com o dado ; contido no endereço de memória 1000
- **STA *addr* (Store Acumulator)**
  - **Mover um dado do acumulador para o endereço ADDR**
  - EX: STA FFFF; Carrega no endereço de memória FFFF ; o dado contido no acumulador
- **LHLD *addr* / SHLD *addr* (Load HL Direct/Store HL Direct))**
  - **Mover um dado (16 bits) do endereço ADDR para o par HL**
  - EX: LHLD FFFF ; Carrega o par HL com o dado ; contido no endereço de memória FFFF

OPERAÇÕES ARITMÉTICAS

Microprocessador 8085

- **ADD *r* / ADC *r***
  - **Somar o registo ao acumulador (C/CARRY)**
- **ADD M / ADC M**
  - **Somar a memória ao acumulador (C/CARRY)**
- **ADI *data* / ACI *data***
  - **Somar o dado ao acumulador (C/CARRY)**

Exemplos

```
MVI     A, 255
MVI     B,1
MVI     C,2
ADD     B
ADI     30
ADC     C
```
Agora, qual o valor de A ?

13

V.Lobo @ EN

OPERAÇÕES ARITMÉTICAS

Microprocessador 8085

- **SUB *r* / SBB *r***
  - **Subtrair o registo ao acumulador (C/Borrow)**
- **SUB M / SBB M**
  - **Subtrair a memória ao acumulador (C/Borrow)**
- **SUI *data* / SBI *data***
  - **Subtrair o dado ao acumulador (C/Borrow)**

Exemplos

```
MVI     A,10
SUI     03
MVI     B,AFH
MVI     C,10H
ADD     B
SBB     C
```
Agora, qual o valor de A ?

14

V.Lobo @ EN

OPERAÇÕES ARITMÉTICAS

- **DAD rp**
  - **adicionar o par rp ao par HL** **DAD BC**
- **DAA**
  - **Corrigir uma soma /subtracção em BCD** **DAA**
- **INR *r***
  - **incrementar o registo *r*** **INR B**
- **INR M**
  - **Incrementar o conteúdo da posição apontada por HL** **INR M**
- **INX *rp***
  - **Incrementar o par de registos *rp*** **INX BC**
  - **O par *rp* (BC, DE, ou HL), comporta-se como um número de 16 bits**

OPERAÇÕES ARITMÉTICAS

- **DCR *r***
  - **Decrementar o registo *r***
- **DCR M**
  - **Decrementar o conteúdo da posição apontada por HL**
- **DCX *rp***
  - **Decrementar o par de registos *rp***

Exemplos

```
MVI     H,00H
MVI     L,FFH
INX     H
MVI     L,FF
INC     L
DCR     M
DCR     H
```

Agora, qual o valor de H e L ?

OPERAÇÕES ARITMÉTICAS

- **OPERAÇÕES DE COMPARAÇÃO**
  - **Subtraem os operandos mas não guardam o resultado.**
  - **Afectam as flags, que podem depois ser usadas para tomar decisões do tipo "se B=A então..."**
- **CMP *r***
  - **comparar o acumulador com o registo *R***
- **CMP M**
  - **Comparar o acumulador com o conteúdo da posição apontada por HL**
- **CPI data 8**
  - **Comparar o acumulador com o valor indicado**

OPERAÇÕES LÓGICAS

- **ANA *r***
  - **AND lógico entre o acumulador e o registo *r***
- **ANA M**
  - **AND lógico entre o acumulador e a posição apontada por HL**
- **ANI data 8**
  - **AND lógico entre o acumulador e o dado indicado**

Exemplos

```
MVI   A,10H
MVI   B,FFH
MVI   C,10H
CMP   C
ANA   B
CMP   B
```

Agora, qual o valor de A,B,C, e das FLAGS ?

OPERAÇÕES LÓGICAS

- **ORA *r***
  - **OR lógico entre o acumulador e o registo r**
- **ORA M**
  - **OR lógico entre o acumulador e a posição apontada por HL**
- **ORI data 8**
  - **OR lógico entre o acumulador e o dado indicado**
- **XRA *r* / XRA M / XRI I**
  - **OR exclusivo com um registo/memória/dado**

```
MVI     A,F8H
ANI     08H
ORI     02
```
Agora, qual o valor de A ?

INSTRUÇÕES DE CONTROLO DE FLUXO

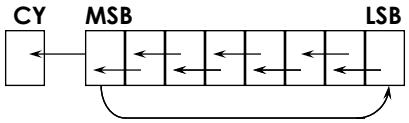
- **Jumps**
  - **Fazem com que a próxima instrução não seja a instrução imediatamente a seguir, mas sim outra qualquer**
- **JMP addr 16**
  - **Salta para o endereço dado**
- **JNZ addr 16**
  - **Salta para o endereço dado se Z=0 (o acumulador ≠ 0)**
- **JZ addr16**
  - **Salta para o endereço dado se Z=1 (o acumulador = 0)**
- **JNC addr 16**
- **JC addr 16**
  - **Salta para o endereço dado se CY=0 / CY=1**

INSTRUÇÕES DE CONTROLO DE FLUXO

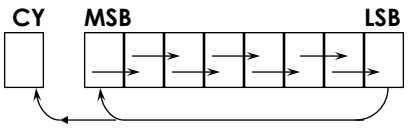
- **JPE addr 16**
  - Salta se P=1 (parity even)
- **JPO addr 16**
  - Salta se P=0 (parity odd)
- **JP addr 16**
  - Salta se S=0 (positive)
- **JM addr 16**
  - Salta se S=1 (minus)
- **PCHL**
  - Põe em "PC" o conteúdo "HL" ⇒ Salta para o endereço contido em HL.
  - É como se fosse JMP M.

ROTAÇÕES

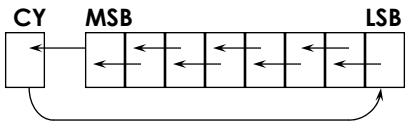
- **RLC**
  - Rotate (acumulator) left to carry



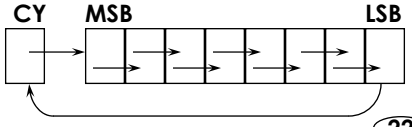
- **RRC**
  - Rotate right to carry



- **RAL**
  - Rotate left through carry



- **RAR**
  - Rotate right trough carry



NEGAÇÕES E "SET"

- **CMA**
  - **Complementa o acumulador**
- **CMC**
  - **Complementa a flag carry**
- **STC**
  - **Set carry flag (faz cy=1)**

METODOLOGIA EM ASSEMBLER

- **DEFINIR PRECISAMENTE OS OBJECTIVOS**
  - **Compreender e explicitar o que se pretende do programa: quais os dados de entrada, e quais os dados de saída**
- **FAZER DIAGRAMA DE BLOCOS OU DIAGRAMA DE ESTADOS**
  - **Definir as sub-tarefas básicas e a interligação entre elas**
- **DEFINIR ESTRUTURA DE DADOS**
  - **Decidir que dados são necessários, onde são guardados, que formato têm, etc.**
  - **Definir a utilização dos registos do μP**
  - **Escrever qual a utilização dos registos e o nome das variáveis (fazer o *léxico de variáveis*), bem como fazer um *mapa de memória***

METODOLOGIA EM ASSEMBLER

- **FAZER FLUXOGRAMA DETALHADO**
  - **Para cada bloco fazer um fluxograma detalhado**
- **FAZER CODIFICAÇÃO**
  - **Escrever uma tabela com as mnenónicas, endereço, código máquina, e nº de bytes usados**

Mnemónicas		Endereço	Código	NºBytes
MVI	A,FFH	4000H	XX XX	2
MVI	B,02H	4002H	XX XX	2
SUB	B	4004H	XX	1
DEC	A	4005H	XX	1

EXEMPLO: Comparar 2 bytes que se encontram em endereços consecutivos. se forem iguais fazer A=1, caso contrário, fazer A=2. Comece o código no endereço 6000H, e compare o conteúdo do endereço 2000H com o seguinte.

25

V.Lobo @ EN

EXEMPLOS:

- **Escreva programas para:**
- **Comparar dois bytes que se encontram em endereços consecutivos. Se forem iguais fazer a=0, se o primeiro for maior, fazer a=1, caso contrário fazer a=2. Assuma que o endereço do primeiro byte é 3010H.**
- **Fazer a soma de 2 números de 3 bytes, que se encontram nos endereços apontados por HL e por DE, deixando o resultado no endereço originalmente apontado por HL**

26

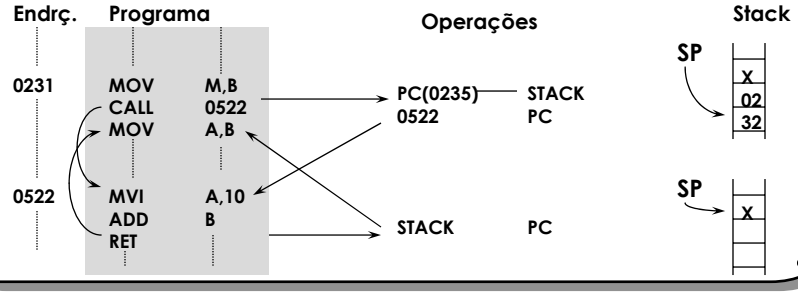
V.Lobo @ EN

SUB - ROTINAS

- **Objectivo**
  - Dividir o programa em tarefas simples e modulares
  - Criar procedimentos que podem ser "chamados" de diversos pontos do programa.
- **Vantagens**
  - Código mais compacto
    - →Se é necessário fazer várias vezes a mesma coisa, existe apenas 1 porção de código para a executar
  - Código mais modular
    - →Permite uma programação mais ordenada e estruturada
  - Menos erros
    - →As sub-rotinas podem ser testadas uma a uma

INSTRUÇÕES CALL/RET

- **CALL addr 16**
  - Chama uma sub-rotina
  - Guarda o endereço contido em "pc" no stack e salta para o endereço "addr"
- **RET**
  - Retorna de uma sub-rotina
  - Vai buscar um endereço ao stack, e salta para esse endereço



Calls condicionais

Microprocessador 8085

- **Calls que são ou não efectuados dependendo das flags**
  - **CZ / CNZ - Call if Zero / if Not Zero**
  - **CM / CP - Call is Minus / if Positive**
  - **CPE / CPO - Call if Parity Even / if Parity Odd**
  - **CC / CNC - Call if Carry / if Not Carry**
- **Exemplo**

Converter o valor de [2000] em negativo se ele for positivo

```
LDA   2000
ANA   A
CP    2A00
STA   2000
```

Rotina que recebe um dado no acumulador e calcula o complemento para 2

```
CMA          ; complementa A
INR   A      ; soma 1
RET
```

29

V.Lobo @ EN

PASSAGEM DE PARÂMETROS

Microprocessador 8085

- **1 - NOS REGISTOS**
  - **Nº limitado de parâmetros**
  - **Rápido e eficiente**
- **2 - EM ENDEREÇOS FIXOS NA MEMÓRIA**
  - **Obriga uma ocupação permanente da memória**
  - **Passagem morosa dos dados**
  - **Não relocável**
  - **Não permite chamadas recursivas**
- **3 - NO STACK**
  - **Nº ilimitado de parâmetros**
  - **Não interfere c/ o resto do programa**
  - **Diversas convenções sobre quem põe e/ou tira os dados do Stack**

30

V.Lobo @ EN

Exemplo de parâmetros

- **Escreva uma rotina para somar dois bytes, e escreva um programa que chame essa sub-rotina.**
  - **Passando parâmetros nos registos:**
    - **→A rotina recebe os dados nos registos B e C, e devolve o resultado no Acumulador**
  - **Passando parâmetros em endereços fixos**
    - **→A rotina recebe os dados nos endereços 20B0H e 20B1H, devolvendo o resultado no endereço 20B2H**
  - **Passando parâmetros no stack**
    - **→Recebendo no stack os dados e devendo-os também no stack**
    - **→Recebendo no stack os dados e o endereço onde deve ser guardado o resultado**

DOCUMENTE ESTAS ROTINAS !

Passagem de Parametros

- **Passagem de parâmetros "à C"**
  - **A rotina que põe os dados no stack antes de chamar a sub-rotina, retira-os após o retorno**
  - **Permite que se use um nº de parâmetros reais inferior ao nº de parâmetros formais**
  - **Cada rotina deixa o stack pointer exactamente na mesma posição em que o recebeu**
  - **Os parâmetros são metidos no stack "da direita para a esquerda"**
- **Passagem de parâmetros "à Pascal"**
  - **A sub-rotina que é chamada retira os parâmetros do stack**
  - **Exige que a rotina que chama e a rotina que é chamada conheçam muito bem o nº de bytes a pôr/retirar do stack**
  - **Os parâmetros são metidos no stack "da esquerda para a direita"**

Documentação das Sub-rotinas

- Há que documentar especialmente bem a INTERFACE da rotina com o resto do sistema
- A documentação deve incluir:
  - quais os parâmetros
  - como são passados os parâmetros (de entrada e saída)
  - quais os registos modificados
  - qual a ocupação de memória
- Usar (e abusar) dos comentários
- Ver exemplos de rotinas do monitor do SDK85

SOFTWARE INTERRUPTS

- Instruções máquina RST (restart)
- Funcionam como CALLS para endereços pré-fixados
- Existem as instruções RST0 a RST7
- Para cada RST há 8 bytes de memória disponível
  - Geralmente, a interrupção tem de ser "vectorizada" para outro local, i.e. nesses 8 bytes há um JMP para o local onde a rotina está implementada.
- Endereço do RST = 8xnº do RST
  - RST0 ⇒ CALL 0000
  - RST1 ⇒ CALL 0008
  - RST2 ⇒ CALL 0010

VANTAGENS DOS SOFT.INT.

- Ocupam só 1 byte
- Permitem chamadas a rotinas independentes da sua implementação(i.e...)
- Podem ser revectorizadas facilmente
  - Podemos substituir as rotinas sem alterar os programas que as chamam
- São normalmente usadas para fazer chamadas às rotinas do sistema operativo
- Normalmente os endereços de RST são ROM, e são revectorizados para RAM
- RST 0 normalmente faz o reset de todo o sistema

INTERRUPÇÕES POR HARDWARE

- Objectivos
  - Forçar o μp a actuar em função de um "evento" externo (que o fará executar um certo programa)
  - O μP não tem que perder tempo a verificar o sistema, pois o sistema avisa-o quando é necessário
  - Eventos urgentes são prontamente atendidos
- Consegue-se sincronismo perfeito
  - Os ciclos de espera e sincronismo por software têm sempre tempos de latência grandes
  - A resposta a interrupções pode ser (QUASE) imediata
- "Acorda" o μP (de "crash", de ciclos de espera, etc.)
- Há 2 tipos de interrupções
  - Umas dependem directamente de pinos do μP, outras necessitam de circuitos externos

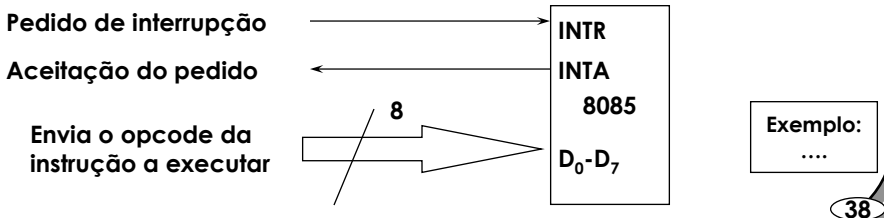
Interrupções directas

- **Pinos TRAP E RST**
  - Existem pinos exteriores que quando actuados forçam o μP a executar uma "instrução" rst
  - Quando um pino é actuado, é feito um CALL para um dado endereço : addr = (nº do Rst) x 8
  - Há 4 pinos:

Pino	Endereço	Tipo de actuação
RST 5.5	2CH	actuado por nível
RST 6.5	34H	actuado por nível
RST 7.5	3CH	actuado por flanco (tem 1 FF)
TRAP (RST 4.5)	24 H	actuado por flanco (MANTIDO)

Interrupções através de INTR

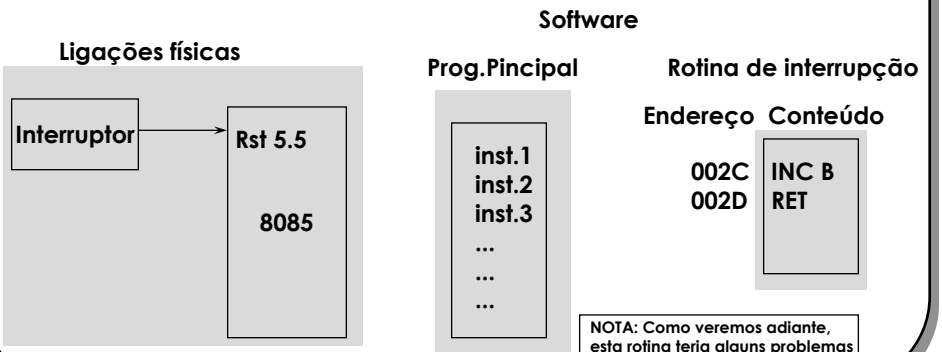
- **Pino INTR**
  - Existe um pino de entrada INTR pelo qual o μP recebe o pedido de interrupção
  - Quando poder atender a interrupção, o μP responde com a activação do pino de saída INTA (interrupt acknolege)
  - Quando é gerado um INTA o up vai gerar um ciclo de "opfetch" (leitura do opcode da memória) mas sem pôr um endereço no bus
    →quem gerou o INTR tem que fornecer um opcode



Exemplo simples

Microprocessador 8085

- **Quero incrementar o registo B sempre que é actuado um sinal**
  - **O sinal pode ser proveniente de um interruptor que gera um impulso curto**

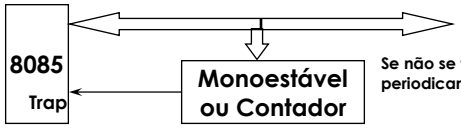


39

Máscara de interrupções

Microprocessador 8085

- **RSTs podem ser "<u>mascarados</u>" , ou seja desactivados ("<u>desablados</u>") por software**
  - **Para evitar que o mP seja interrompido quando está a executar <u>código crítico</u>**
  - **Para que uma rotina de interrupção não se interrompa a si própria**
  - **Para ignorar temporariamente os pedidos de um dado periférico**
- **A linha TRAP não é mascarável**
  - **Está sempre activa**
  - **É usada muitas vezes como WATCH-DOG para evitar "crashes profundos"**



40

Máscara de interrupções

Microprocessador 8085

- **Podemos ligar/desligar todas as interrupções ao mesmo tempo (salvo caso das NMI)**
  - **Instrução DI - Desable Interrupts**
    → **"desliga" as interrupções**
  - **Instrução EI - Enable Interrupts**
    →**"liga" as interrupções**
- **Podemos activar apenas ALGUMAS interrupções:**
  - **Existe um registo (chamado INTERRUPT MASK) que pode ser escrito/lido de modo a seleccionar quais interrupções estão activas**
- **Para que uma interrupção não se interrompa a si própria, a sua chamada faz "automaticamente" DI logo, é necessário fazer EI durante a rotina**
  - **o EI só tem efeito depois de 1 instrução (para possibilitar os RETs)**

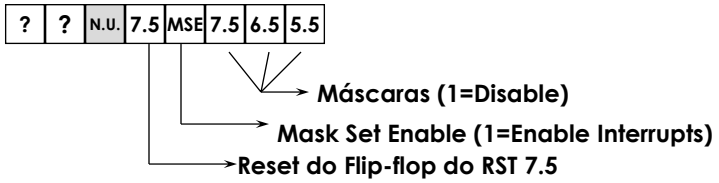
41

V.Lobo @ EN

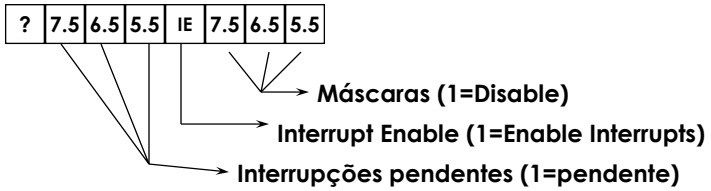
Instruções SIM e RIM

Microprocessador 8085

- **SIM - Set Interrupt Mask**
  - **Põe na máscara de interrupção o conteúdo do acumulador**



- **RIM - Read Interrupt Mask**
  - **Põe no acumulador a máscara de interrupção**



42

V.Lobo @ EN

Prioridades

- **No caso de serem actuadas várias interrupções simultaneamente, existe uma escala de <u>prioridades</u> entre elas:**
  - **os endereços mais baixos são menos prioritários.**
- **TRAP é o mais prioritário**
- **RST 5.5 é menos prioritário que 6.5 e estes dois menos que o 7.5**
- **INTR é o menos prioritário**
- **Existem métodos de controlo de prioridades mais sofisticados (ver 8259)**

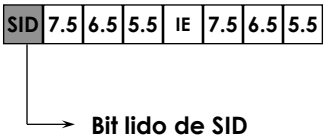
ROTINAS DE INTERRUPÇÃO (INTERRUPT HANDLERS)

- **São <u>assíncronas</u>**
  - **Não é possível saber em que ponto do código é que vão ser chamadas**
  - **logo torna-se necessário <u>salvaguardar o contexto:</u>**
    - **→PC é guardado automaticamente pelo "CALL"**
    - **→FLAGS e ACC têm de ser guardados quase sempre (PUSH PSW)**
    - **→Outras reg. só os que forem usados**
- **Por conveniência são normalmente <u>revectorizadas</u> dos seus endereços originais**
  - **Vector de interrupções original não tem espaço para o código**
  - **Os endereços do vector de interrupções são por vezes ROM**

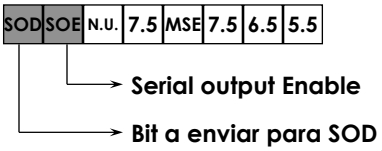
I/O Série com o 8085

- **No 8085 as instruções RIM e SIM também controlam dois pinos de entrada/saída série**
  - **Pinos SOD (Serial Output Data) e SID (Serial Input Data)**
  - **SIM - Send data - Envia um bit do acumulador para SOD**
  - **RIM - Read data - Lê o bit presente em SID para o acumulador**

RIM



SIM



I/O Paralelo com o 8085

- **Espaço de endereçamento de Input/Output (I/O)**
  - **É um espaço de endereçamento similar ao de memória, mas com apenas 256 endereços**
  - **É gerado com um endereço de apenas 8 bits.**
  - **Usado para fazer IO com periféricos, sem "gastar" endereços de memória**
- **IN** *addr8*
  - **Lê do porto IO indicado para o Acumulador**
- **OUT** *addr8*
  - **Escreve do Acumulador para o porto de IO indicado**

```
MVI A,23H ; põe o dado no Acc
OUT 20    ;manda o dado para IO
```

Instrução HLT (HALT)

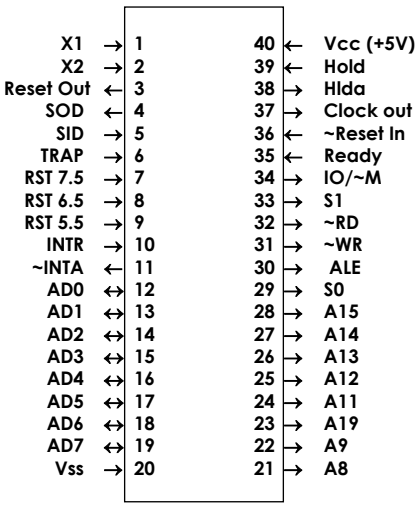
- **Põe o microprocessador num estado de espera**
  - **Pára todo o processamento**
  - **Apenas é possível sair deste estado com um pedido de interrupção**
  - **A resposta a um pedido de interrupção é extremamente rápida porque não é necessário esperar por nada**

Exemplo de espera passiva:

```
L1:     HLT     ; Espera que haja uma interrupção
        JMP L1  ; Volta para a instrução de espera
```

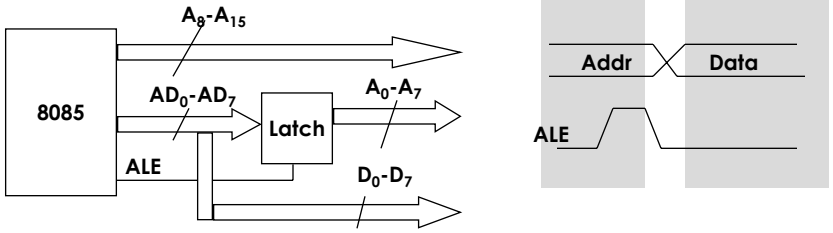

Hardware e ligações externas

- **Existem várias "embalagens"**
  (packaging options)
- **Forma mais vulgar (disponível no laboratório)**
  - **DIP de 40 pinos**
  - **Níveis TTL**
- **Pinout do 8085:**



Multiplexagem no Bus de Dados

- **A fim de poupar pinos no integrado, há pinos que ora servem para DADOS ora servem para ENDEREÇOS**
  - **No princípio do ciclo de leitura/escrita são usados para conter o endereço**
  - **No fim do ciclo de leitura/escrita contêm os dados**
  - **É gerado um pulso no pino ALE (Address Latch Enable) para separar os dois tipos de sinais**



Pinos de temporização/reset/interrupts

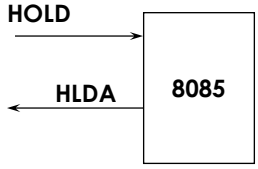
- **X1, X2**
  - **Ligados a um cristal externo para gerar o clock de sistema.**
  - **Montagem típica:**
- **CLK out**
  - **Clock de sistema para os periféricos**
- **~Reset In**
  - **Mantido a 0 durante 4 ciclos de relógio força uma reinicialização**
- **Reset out**
  - **Usado para fazer reset aos outros componentes do sistema**
- **TRAP, RST x.5, INTR, INTA**
  - **Interrupções**

Outros pinos

- **Vcc e Vss**
  - Alimentação (5V cc) e massa
- **SID, SOD**
  - Comunicações série
- **~RD, ~WR, IO/~M**
  - Leitura ou Escrita em memória ou IO
- **S0, S1**
  - Status (para ver o ciclo de BUS)

Pino HOLD e HLDA

- **Por vezes é necessário usar o bus sem que este esteja controlado pelo μP**
  - Sistemas com vários processadores
  - Sistemas com periféricos/sub-sistemas inteligentes (para DMA por exemplo)
- **Funcionamento**
  - Quando alguém quer o BUS, gera um pedido de HOLD ao μP
  - Quando o μP poder prescindir do BUS, põe as suas saídas em Tri-State, e activa o pino HLDA (Hold Acknowledge)



Ciclos de Bus

- Ciclos T e ciclos M
- Há só 7 de ciclos máquina (M) no BUS:
  - OF, MR, MW, IOR, IOW, INA, BI
- Operação de Leitura de um dado em memória (MR)
  - Diagrama temporal completo, e simplificado.
  - Presença de *Wait States*
- Operação de Escrita de um dado (MW)
- Diferenças entre OpFetch (OF), InterruptAck (INA), e MemRead
- Diferenças entre ciclos de I/O (IOW, IOR) e memória
- Ciclo Bus Idle (BI)
- Diagrama de Estados do 8085

V.Lobo @ EN