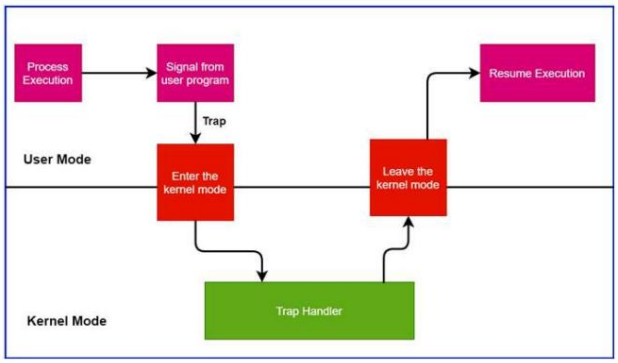
ISR

Interrupções do Arduino
I2C

1

Armadilha

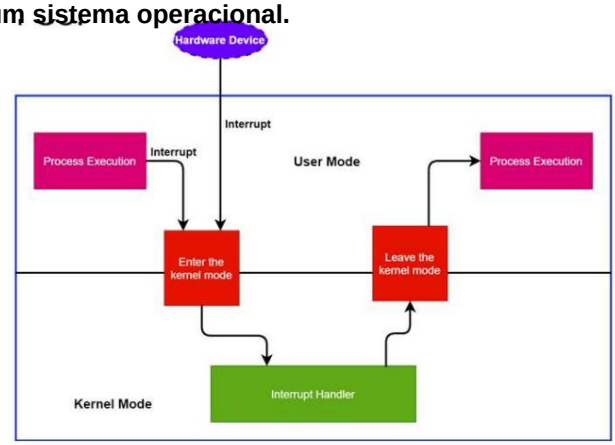
- **Um trap é uma** interrupção **síncrona acionada por uma exceção em um processo do usuário para executar uma funcionalidade (acesso à memória inválido, divisão por zero ou um ponto de interrupção).**



2

Interromper

- **Uma interrupção é um sinal de hardware ou software que exige atenção instantânea de um sistema operacional.**



3

Capturar e Interromper

Trap	Interrupt
It's a signal emitted by a user program	It's a signal emitted by a hardware device
Synchronous process	Asynchronous process
Can occur only from software device	Can occur from a hardware or a software device
Only generated by a user program instruction	Generated by an OS and user program instruction
Traps are subset of interrupts	Interrupts are superset of traps
Execute a specific functionality in the OS and gives the control to the trap handler	Force the CPU to trigger a specific interrupt handler routine

4

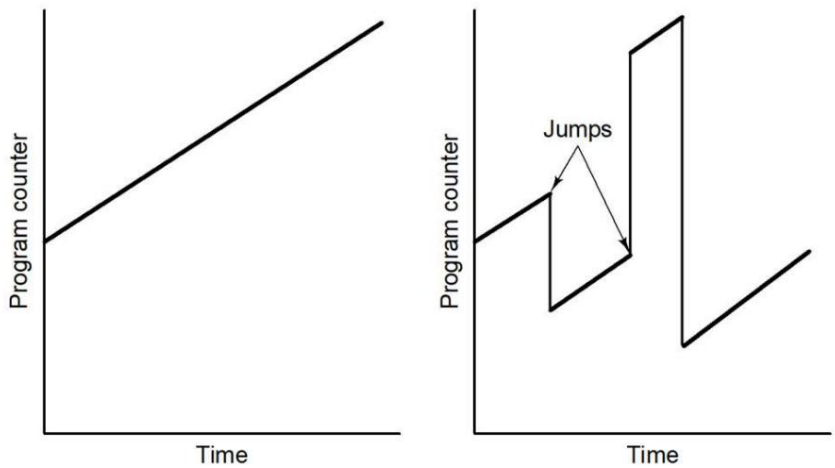
Procedimentos, Armadilhas, Interrupções & Co.

- Muitas razões para a execução não linear do programa
  - Saltos, galhos
  - Chamadas de procedimento, sub-rotinas, invocação de método
  - Multithreading, processos paralelos, co-rotinas
  - Interrupções de hardware (motivos externos do processador)
  - Traps, interrupções de software (motivos internos do processador)

- A execução não linear do programa é o caso normal!
  - E invalida o conteúdo do cache padrão...
    - Os caches de rastreamento podem ajudar (mais adiante)

5

Execução do programa



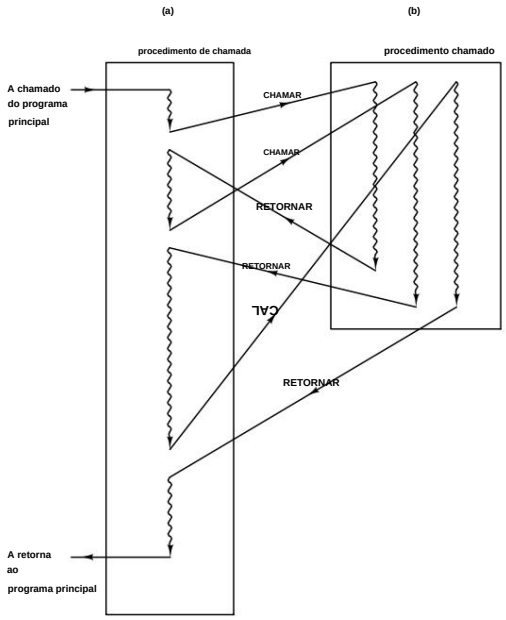
Linear, sem ramificações

Com ramos

6

chamada de procedimento

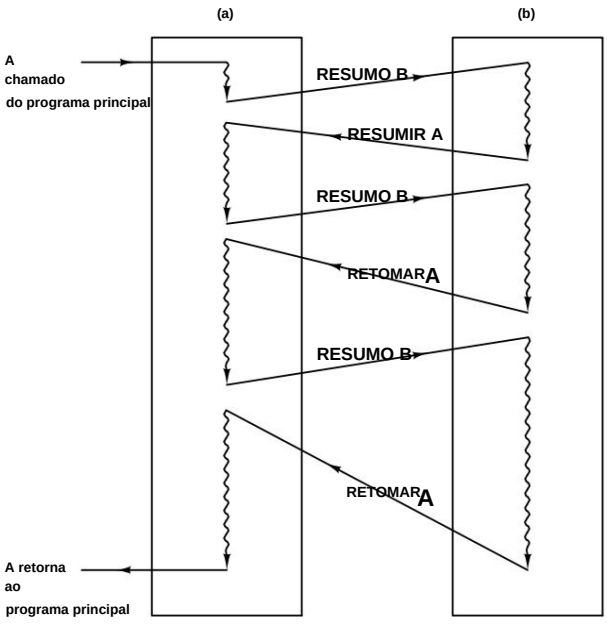
(sub-rotina, método, ...)



7

chamada co-rotina

(processo paralelo, multithreading,...)



8

Como lidar com exceções?

- Durante o tempo de execução podem ocorrer exceções, ou seja, interrupções do fluxo programado de instruções
- Razões
  - Erros no sistema operacional durante a execução de programas aplicativos ou erros no hardware
  - Solicitações de componentes externos para atenção do processador
  - ...
- Situações excepcionais podem exigir a interrupção do atual programa em execução ou até mesmo sua finalização

Manipulação de exceção

- O tratamento de exceções requer rotinas especializadas (Serviço de Interrupção Rotina - ISR)
- Um componente de hardware especializado (sistema de interrupção, controlador de interrupção) normalmente suporta a seleção e ativação de um ISR
- Um ISR tem a mesma estrutura de um subprograma, mas também algumas diferenças

ISR vs. subprograma/sub-rotina

Atividade	Subrotina/subprograma	ISR
Ativação	chamar sub-rotina	Instrução INT ou hardware ativação
Retornar após a conclusão	Instrução RET (retorno da sub-rotina)	Instrução RETI (retorno da interrupção)
Cálculo do endereço inicial	Endereço inicial da sub-rotina chamada escrita no programa de chamada	Endereço inicial do ISR chamado determinado via tabela de interrupção
Salvando status	Chamada de sub-rotina normalmente salva apenas PC em uma pilha	Chamadas ISR salvam o PC e PSW em um pilha

palavra de status do programa (PSW)
rotina de serviço de interrupção (ISR)

11

ISR vs. subprograma/sub-rotina

- O processador sempre executa chamadas de sub-rotina conforme programado.

- No entanto, o processador executa ISR somente se acionado e o
  O bit de habilitação de interrupção no PSW está definido.

- Motivos para o tratamento de exceções
  - Motivos externos (eventos assíncronos): entrada de dados, dispositivo pronto, mouse movimento,...
  - Razões internas (eventos síncronos): chamadas de sistema, depuração, mudança de privilégio,...

12

Razões externas para exceções

- REINICIAR
  - Reset do processador, por exemplo, acionado por um botão, fonte de alimentação, watch dog timer, ...
- PARAR
  - Parar a execução do processador, por exemplo, para evitar conflitos de acesso no barramento do sistema durante o DMA (acesso direto à memória)
- ERRO
  - Chamada de uma rotina de tratamento de erros, por exemplo, devido a erros de barramento
- Interromper
  - Solicitação de interrupção acionada por um dispositivo externo, por exemplo, para anunciar dados recebidos de uma entrada dispositivo
  - 2 tipos:
    - mascarável – interrupção que pode ser desabilitada ou ignorada pelas instruções da CPU
    - não mascarável (NMI) - interrupção que não pode ser desabilitada ou ignorada pelas instruções da CPU

13

Razões internas para exceções

- Interrupções de Software
  - A instrução INT no programa aciona uma interrupção (chamadas do sistema, depuração, ...)
- Armadilhas
  - Exceções causadas por eventos internos (por exemplo, estouro, divisão por zero, estouro de pilha, ...)

14

Etapas típicas de um ISR I

1. Interrompa a ativação 2.

Finalize a instrução atualmente em execução

3. Verifique se há interrupção de software ou interrupção de hardware interno/externo

4. Verifique se o bit de ativação de interrupção está
   definido ÿ permitir interrupção

5. Se for uma interrupção de hardware: localize a origem da interrupção, ative o INTA
   (reconhecimento de interrupção)

6. Salve PSW e PC na pilha

7. Reset Interrupt Enable bit para evitar uma interrupção adicional neste
   estágio

Etapas típicas de um ISR II

8. Calcule o endereço inicial do ISR (por exemplo, com base na tabela de vetores de interrupção) e carregue
   no PC

9. Execute a rotina de interrupção do serviço:
   - Empurre o registro usado na pilha
   - Defina o bit de ativação de interrupção para permitir outras
   - interrupções (ou seja, interrupções podem
     interromper interrupções!) • Faça o trabalho real do ISR
   - Retirar os registradores da pilha
   - Retorno do tratamento de interrupção usando a instrução IRET

10. Restaure PSW e PC e continue com o programa interrompido

- Atenção: se o ISR for muito grande ele bloqueia o computador!

Tabela de vetores de interrupção

- **Normalmente, localizado em um endereço conhecido, por exemplo, na ROM (começando no endereço 0000:0000 para processadores 80X86)**
- **Contém os endereços iniciais dos ISRs**
- **A fonte de uma interrupção cria um número de interrupção apontando para a entrada na tabela de vetores de interrupção**
- **Pode ser bem mais complexo…**

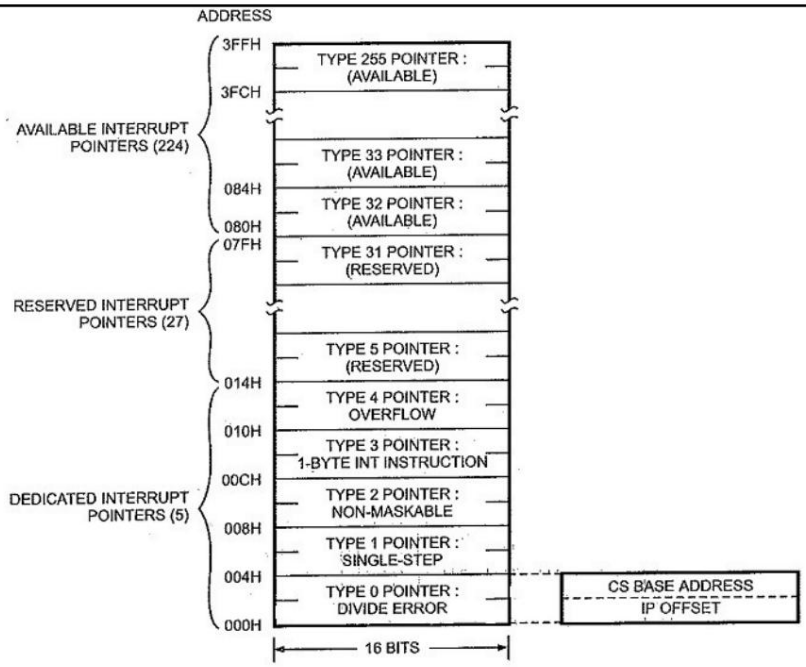


Fig. 9.2 8086 interrupt vector table

Interrupções do Arduino

• attachInterrupt() •

Interrupções são uma maneira de um microcontrolador parar temporariamente o que está fazendo para lidar com

outra tarefa. • O programa atualmente em execução é pausado, uma ISR (rotina de serviço de interrupção) é executada e, em seguida, o programa continua, sem saber.

Table 9-1. Reset and Interrupt Vectors in ATmega48P

Vector No.	Program Address	Source	Interrupt Definition
1	0x000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	PCINT0	Pin Change Interrupt Request 0
5	0x004	PCINT1	Pin Change Interrupt Request 1
6	0x005	PCINT2	Pin Change Interrupt Request 2
7	0x006	WDT	Watchdog Time-out Interrupt
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow
18	0x011	SPI, STC	SPI Serial Transfer Complete
19	0x012	USART, RX	USART Rx Complete
20	0x013	USART, UDRE	USART, Data Register Empty
21	0x014	USART, TX	USART, Tx Complete
22	0x015	ADC	ADC Conversion Complete
23	0x016	EE READY	EEPROM Ready

Sobre rotinas de serviço de interrupção

- **ISRs são tipos especiais de funções que possuem algumas limitações exclusivas que a maioria das outras funções não possui.**

- **Um ISR não pode ter nenhum parâmetro e não deve retornar nada.**

Quando você usaria um?

- **As interrupções podem detectar pulsos breves nos pinos de entrada.**
- **As interrupções são úteis para acordar um processador adormecido.**
- **Interrupções podem ser geradas em um intervalo fixo para repetição em processamento.**

Sintaxe

- **attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);**

Parâmetros:

interrupção: o número da interrupção (int)
pino: o número do pino
ISR: o ISR a ser chamado quando ocorrer a interrupção;
esta função não deve ter parâmetros e não retornar nada.

Às vezes, essa função é chamada de rotina de serviço de interrupção.

Sintaxe

Modo: define quando a interrupção deve ser acionada. Quatro constantes são predefinidas como valores válidos:

- **LOW para acionar a interrupção sempre que o pino estiver baixo,**
- **CHANGE para acionar a interrupção sempre que o pino mudar de valor**
- **RISING para disparar quando o pino vai de baixo para alto,**
- **FALLING para quando o pino vai de alto para baixo.**

```
const byte ledPin = 13;
const byte interrupçãoPin = 2;
estado de byte volátil = BAIXO;

void setup() {
        pinMode(ledPin, OUTPUT);
        pinMode(interruptPin, INPUT_PULLUP);
        attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);

} void loop()
        { digitalWrite(ledPin, estado);
}

void piscar() {
        estado = !estado;
}
```

25

desanexarInterrupção()

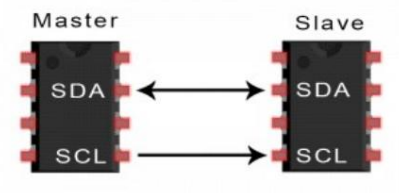
Desativa a interrupção fornecida.

detachInterrupt(digitalPinToInterrupt(pin))

26

Protocolo I2C

- O protocolo I2C envolve o uso de duas linhas para enviar e receber dados:
- SDA (Serial Data) – A linha para o mestre e o escravo enviarem e receber dados.
- SCL (Serial Clock) – A linha que transporta o sinal de clock.



27

COMO FUNCIONA O I2C

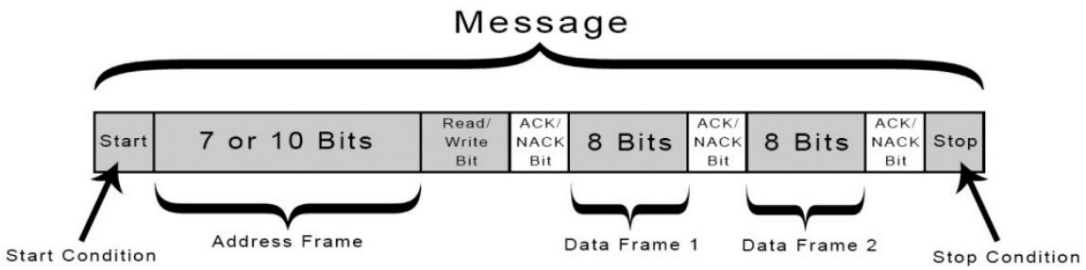
- Com I2C, os dados são transferidos em

mensagens. • As mensagens são divididas em

quadros de dados. • Cada mensagem tem um quadro de endereço que contém o endereço binário do escravo e um ou mais quadros de dados que contêm os dados que estão sendo transmitidos.

- A mensagem também inclui condições de início e parada, bits de leitura/ gravação e bits ACK/NACK entre cada quadro de dados.

28



• Condição inicial: A linha SDA muda de um nível de tensão alto para um nível de tensão baixo antes que a linha SCL mude de alto para baixo. • Condição de parada: A linha SDA muda de um nível de tensão baixo para um nível de tensão alto depois que a linha SCL muda de baixo para alto.

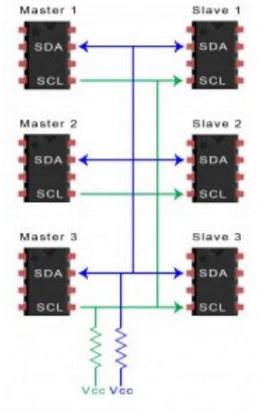
29

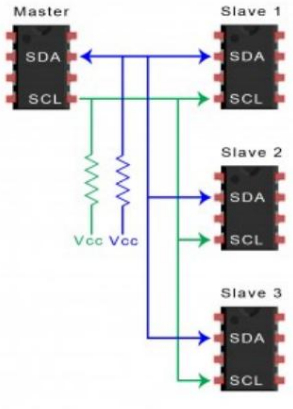
• Quadro de endereço: Uma sequência de 7 ou 10 bits exclusiva para cada escravo que identifica o escravo quando o mestre quer falar com ele.

• Bit de Leitura/Escrita: Um único bit especificando se o mestre está enviando dados para o escravo (baixo nível de tensão) ou solicitando dados dele (alto nível de tensão). • Bit ACK/NACK: Cada quadro em uma mensagem é seguido por um bit de reconhecimento/não reconhecimento. Se um quadro de endereço ou quadro de dados foi recebido com sucesso, um bit ACK é retornado ao remetente do dispositivo receptor.

30

N MESTRE N ESCRAVOS | 1 MESTRE N ESCRAVOS





31

Vantagens

• Usa apenas dois fios •

Suporta vários mestres e vários escravos • O bit ACK/

NACK confirma que cada quadro foi transferido com sucesso

• O hardware é menos complicado do que com UARTs

• Protocolo bem conhecido e amplamente utilizado

32

Mestre do Arduino - Remetente

```
#include <Fio.h>


void setup()
 { Wire.begin(); // junta-se ao barramento i2c (endereço opcional para mestre)
}
byte x = 0;
void loop()
 { Wire.beginTransmission(4); // transmite para o dispositivo
 #4 Wire.write("x is "); // envia cinco bytes
 Wire.write(x); // envia um byte
 Wire.endTransmission(); // para de transmitir
x++;
atraso(500);
}
```


Escravo Arduino - Receptor

```
#include <Fio.h>
void setup() {
 Wire.begin(4); // junta o barramento i2c com o
 endereço #4 Wire.onReceive(receiveEvent); // registra o evento
 Serial.begin(9600); // inicia serial para saída
}
loop void(){
 atraso(100);
}
void receberEvent(int quantos) {
  while(1 < Fio.disponível()) {
 char c = Wire.read(); // recebe o byte como um caractere
 Serial.print(c); // imprime o caractere
 }
int x = Wire.read(); //recebe o byte como um inteiro
Serial.println(x); // imprime o inteiro
}
```

Python – Comunicação com PC e Arduino

- Instale o Python (Windows Store ou em https://www.python.org/downloads/) • Baixe o PySerial • Abra o CMD e digite “pip install pyserial”
- Teste PySerial
  - Abra o CMD, digite “python” e depois “import serial”

código Python

```
serial de importação
arduino = serial.Serial(port='COM4', baudrate=9600, timeout=.1)
def ler():
    dados = arduino.readline()
    dados de retorno
enquanto verdadeiro:
    valor = read().decode('utf-8').strip()
    print(value) # imprimindo o valor
```

Código Arduino

```
intx=0;
void setup()
{ Serial.begin(9600); }

void loop()
{ Serial.println(x++);
atraso(5000);
}
```