Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

Aula 6

Arrays em C e apontadores

- O nome do array vale por um apontador constante:

```
int v[3];    // v tem tipo int*

v[2] = 5;    ⇔  *(v+2) = 5;
```

- Criando dinamicamente e acedendo como array:

```
int *v = (int*)malloc( 3*sizeof(int));
v[2] = 5;
```

Arrays em C e funções

- O parâmetro é um array de dimensão pré-definida:

```
void fun( int vet[3] );
```

NOTA:

```
int myvet[3];
fun( myvet );  ← O que é passado é o endereço!
```

- Mas podemos ter parâmetros arrays sem dimensão pré-definida:

```
void fun( int *vet );
```

ou `void fun( int vet[] );`

Arrays em C e funções

- O nome do array representa o seu inicio
  - Se um argumento é um array, o valor que é passado é o do endereço!

    int myvet[3];

    fun(myvet);

  independentemente da declaração de fun:

    void fun(int v[3]) void fun(int *v)
- Porquê?
- Consequências se passasse todo o array?
  - Memória ocupada? Tempo de execução?

Parâmetros ambíguos

- Mas um array não é um objeto → não tem campo length!

```
void fun( int *vet );
```

  - Vet será um array de int, ou aponta só um int?
  - Qual a dimensão do array (e é vetor, matriz, etc...)?
  - Fun deve ser chamada com um array? De que dimensão?
- Solução habitual: acompanhar o array com a sua dimensão:

```
void fun( int *vet, int vsize );
```

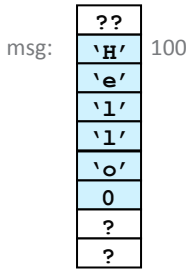
ou `void fun( int vet[], int vsize );`

Strings em C

- Não existe tipo específico para strings.
- Convenção: uma sequência de chars terminada com '\0' (código ASCII 0)

```
char msg[]={'H','e','l','l','o','\0'};
char msg[]= "Hello";
```

símbolo: msg: 100 : 1 byte (char)



- Mas há constantes: `"Hello"`

```
char *msg = "Hello";
```

  - msg é um apontador para a string **constante** "Hello"

Strings em C – string.h

- A biblioteca do C dispõe de várias funções para manipular *strings*, declaradas em string.h
- Exemplos:

```
int strnlen(char *str, int max);
char *strncpy(char *dst, char *src, int len);
char *strndup(char *str, int len);
char *strncat(char *dst, char *src, int len);
int strncmp(char *s1, char *s2, int len);
```

- E versões sem dimensão (sem max ou len):

strlen, strcpy, strdup, strcat, strcmp

Exemplos de utilização

```
char *str1;
char str2[ ] = "Hello";
char str3[100];

str1=str2;
strcat( strcpy(str3,str1), str2 );
str1[strlen(str1)-1] = 'u';
eq=strcmp(str1, str2);
```

Erros comuns

```
char *str1;
char str2[10];
```

- Usar apontadores não iniciados:

```
strcpy( str1, "Hello" );
```

- Esquecer a terminação '\0':

```
char str[5] = "Hello";   ← mas sizeof("Hello") é 6!
          {'H','e','l','l','o','\0'}
```

- Aceder fora do array:

```
strcat( strcpy(str2, "Hello"), "World");
```


Mais erros comuns

```
char *str1; char str2[10];
```

- Copiar apontadores em vez de arrays (usar = em vez de strcpy ou do memcpy):

```
str2 = "hello";    ou   str1 = str2;
```

- Comparar apontadores em vez dos caracteres:

```
str1 == str2
```

em vez de `strcmp( str1, str2 )`

- Julgar que o tamanho do array é o da string e vice versa:

```
strcpy(str2, "Hello");
```

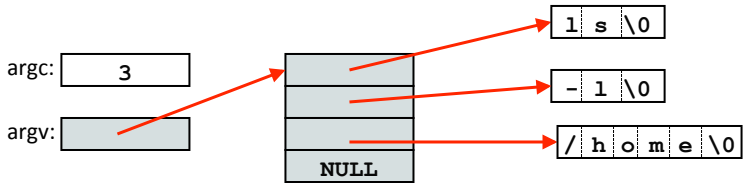
`strlen(str2)` não é 10! É 5!

Exemplo: main

`int main( int argc, char *argv[] )`

- Argv contém uma cópia da linha de comando.
- Argv representa um vetor de strings. Argc o número de strings. Exemplo:

```
$ ls  -l  /home
```



Escrever todos os seus argumentos:

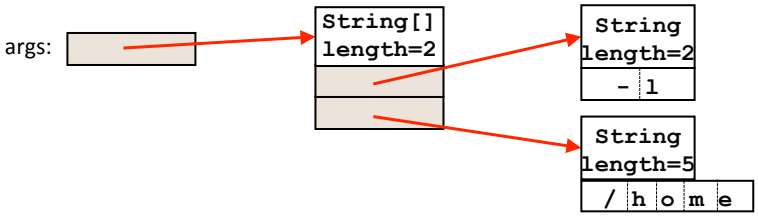
```c
#include <stdio.h>

int main( int argc, char *argv[] ) {
   printf("num args: %d\n", argc );
   for ( int i=0; i<argc; i++ ) {
      printf("%d: %s\n", i, argv[i] );
   }
   return 0;
}
```

Exemplo: main em Java

`public static void main( String[] args )`

- Args contém uma cópia dos argumentos na linha de comando.
- Args representa um vetor de `String`. Exemplo:

```
$ java Ls  -l  /home
```



Escrever todos os seus argumentos:

```
class Prog {

 public static
 void main( String[] args ) {
   System.out.print("num args: ");
   System.out.println( args.length );
   for ( int i=0; i<args.length; i++ ) {
     System.out.printf("%d: %s\n", i, args[i] );
   }
 }
}
```

Código executado pelo CPU?

- Código máquina (instruções do ISA)
- Cada código: padrão de bits que define uma operação elementar, capaz de ser executada pela máquina (interpretada pelo CPU)
- Que informação faz parte do código de cada instrução?
  - Como descrevemos a operação e os dados usados pela operação?
- Que tipo de instruções faz parte da arquitetura?
  - Que operações são implementadas no *hardware*?

Desenho de uma arquitectura

- Definição das instruções: para as boas soluções, contribuem:
  - A definição do conjunto de instruções a suportar
  - E todos os outros elementos da arquitectura: registos, bus, etc
- Uma boa solução é também fruto de compromissos:
  - Tecnologia
  - Custo
  - Funcionalidade
  - Desempenho
  - Leis do mercado

Desenho das instruções do processador

- Os objectivos considerados são muitos:
  - Aproximar das linguagens de alto nível
  - Reduzir o tamanho dos programas
  - Oferecer instruções poderosas
  - Instruções simples, que facilitem a implementação eficiente
  - Oferecer só as instruções mais úteis
  - Oferecer diversos modos de endereçar as variáveis em memória (ex. referências, matrizes, registos)
  - Reduzir o número de acessos a memória central
  - Promover uma abordagem em que instruções mais complexas não são oferecidas, mas podem ser obtidas à custa da várias das existentes

Classes de instruções

- As classes de instruções realmente necessárias são poucas:
  - Aritméticas e lógicas
    - And, Add, ...
  - Transferências de dados (CPU, memória, periféricos)
    - Load, Store, In, Out
  - Controlo da sequência de execução
    - (alterar IP) Jumps

Exemplos de instruções-todas no CPU?

- Que instruções incluir no CPU? Ou em níveis superiores da hierarquia? (p.ex. SO, bibliotecas das linguagens)
  - Multiplicação e divisão
  - Aritmética de reais (FP)
  - Chamada e retorno de funções/métodos
  - Execução eficiente de ciclos tipo *for*
  - Contar e temporizar acções
  - Operar sobre cadeias de caracteres, sobre listas, sobre filas, sobre registos, sobre matrizes
  - Mover e recolocar programas em memória
  - Suspensão e reactivação de programas
  - outras?

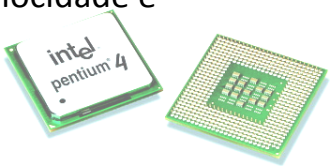
Intel e os circuitos integrados (CI)

- Intel- (Integrated Electronics Corporation)
  - Fundada em 1968 por funcionários vindos da Fairchild Semiconductor
    - Na origem (com a Texas Instruments) dos circuitos integrados (*microchip*)
  - Micro-processador
    - um único circuito integrado contendo todas as funcionalidades da unidade central de processamento (CPU) do computador
      - Intel 4004 – µ-proc. de 4bits para calculadoras

Nota: Já existiam processadores e computadores!

Alguns μ-processadores Intel

- Cada nova geração aumentou a velocidade e capacidade:



  Proc. – dados/endereços
  - 8080 – 8bits/16bits
  - 8086/8088 – 16bits/20bits *(usados nos primeiros IBM/PC)*
  - 80286 – 16bits/24bits *(usado no IBM/AT)*
  - 80386 – 32bits/32bits *(usado no IBM/PS2)*
  - 80486, Pentium (P5), Pentium Pro (P6), …
- Existe compatibilidade
  - continuam a existir instruções com dados de 8bits no 80386 e seguintes

Diagrama do 8086

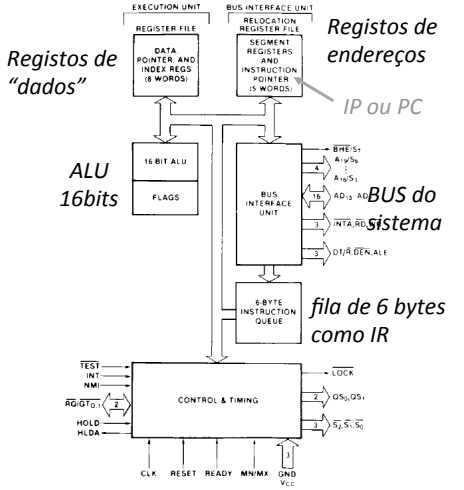
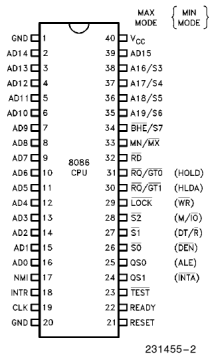


Figure 1. 8086 CPU Block Diagram



40 Lead

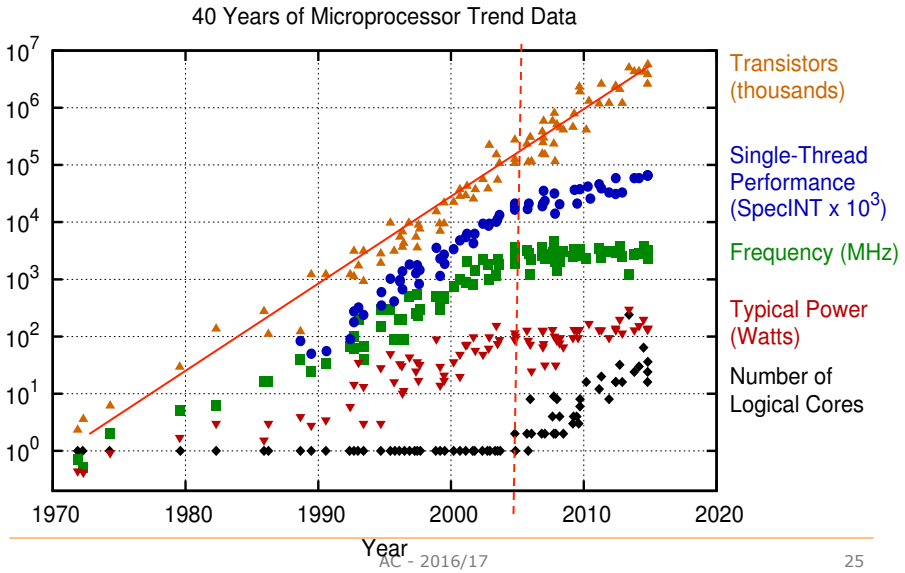
Figure 2. 8086 Pin Configuration

Table 2-1. Processor Performance Over Time and Other Key Features of the Intel Architecture

Intel Processor	Date of Product Intro-duction	Perfor-mance in MIPs[1]	Max. CPU Frequency at Intro-duction	No. of Transis-tors on the Die	Main CPU Register Size[2]	Extern. Data Bus Size[2]	Max. Extern. Addr. Space
8086	1978	0.8	8 MHz	29 K	16	16	1 MB
Intel 286	1982	2.7	12.5 MHz	134 K	16	16	16 MB
Intel386™ DX	1985	6.0	20 MHz	275 K	32	32	4 GB
Intel486™ DX	1989	20	25 MHz	1.2 M	32	32	4 GB
Pentium®	1993	100	60 MHz	3.1 M	32	64	4 GB
Pentium Pro	1995	440	200 MHz	5.5 M	32	64	64 GB

Fonte: Intel architecture software developer's manual

Observação de Moore ("Lei de Moore")



Capacidades das arquitecturas

- O número de bits nos endereços (p.e. IP) determina a capacidade máxima de memória endereçável
  - Logo o maior programa, de código e dados
  - O tamanho dos apontadores em C
- O número de linhas no bus de endereços determina a capacidade máxima de memória realmente acessível
- O número de bits dos registos gerais determina o tamanho dos dados operados pelas instruções
- O número de linhas no bus de dados determina a capacidade máxima de transferência dos dados em cada ciclo de acesso à memória

Tamanho do endereço

- Palavras de memória que se pode endereçar (teoricamente):
  - 8 bits – 2^8 endereços =256 endereços
  - 16 bits – 2^{16} = 64 K
  - 32 bits – 2^{32} = 4 G
  - 64 bits – 2^{64} = 16 E
- Se memória endereçada ao byte:
  - 32 bits → 4 GBytes
- Se a memória endereçada por palavras de 2 bytes:
  - 32 bits → 8 GBytes

Múltiplos

- Nota sobre os unidades em bases 2 vs base 10 (exemplo com bytes):

Decimal			**Binary**				
Value	Metric		Value	IEC		*JEDEC*	
1000 (10^3)	kB	kilobyte	**1024 (2^{10})**	KiB	kibibyte	***KB***	***kilobyte***
1000^2	MB	megabyte	**1024^2**	MiB	mebibyte	***MB***	***megabyte***
1000^3	GB	gigabyte	**1024^3**	GiB	gibibyte	***GB***	***gigabyte***
1000^4	TB	terabyte	**1024^4**	TiB	tebibyte	***TB***	***terabyte***
1000^5	PB	petabyte	**1024^5**	PiB	pebibyte	***PB***	***petabyte***
1000^6	EB	exabyte	**1024^6**	EiB	exbibyte	***EB***	***exabyte***
1000^7	ZB	zettabyte	**1024^7**	ZiB	zebibyte	***ZB***	***zettabyte***
1000^8	YB	yottabyte	**1024^8**	YiB	yobibyte	***YB***	***yottabyte***

Conclusão

- O resultado desta evolução:
  - Arquiteturas muito complexas e difíceis de compreender e programar
  - Instruções de tamanho variável (de 1 a 18? bytes…)
  - ISA muito complexos, difíceis de interpretar pelo hardware
  - Os programadores (e os compiladores) "refugiam-se" num subconjunto do ISA …
  - Internamente, existe uma "tradução" para micro-código, mais simples. Este é realmente executado pelo CPU
- O desempenho não vem só dos GHz...
  - Pipelines e Paralelismo interno
  - Paralelismo - multi-processadores
- Extensão do IA-32 para 64 bits:
  - AMD com AMD64; Intel com EM64T (x86-64)

Exemplo Intel 80386 / **IA-32** (1985)

- CPU com 32 bits de dados e de endereços, com BUS de 32 bits:
  - dimensão do IP, registos de dados, número de linhas no BUS de dados e de endereços: 32
  - endereça até 4G de bytes de memória
  - transfere e opera até 4bytes de cada vez
- Base para os i486, Pentium/P5 (i586), Pentium Pro/P6 (i686)
  - 486 – FPU interna