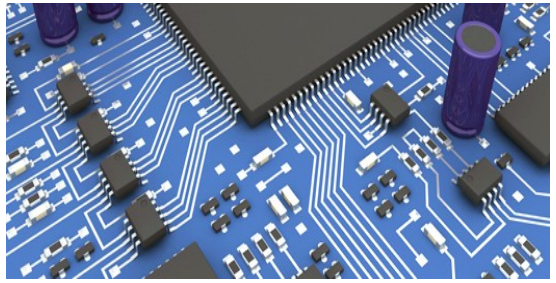
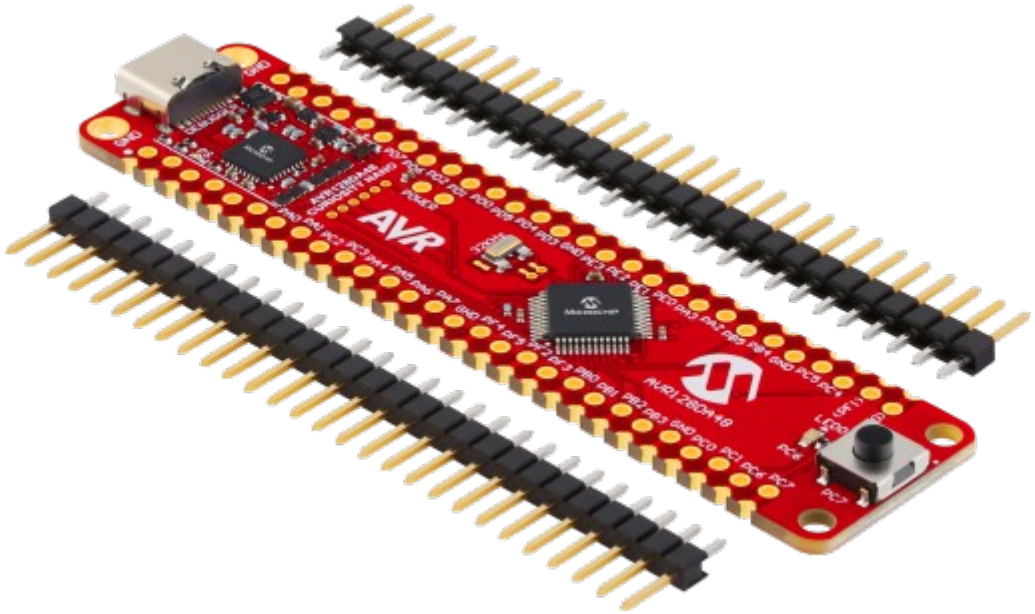
Aplikace Embedded systémů v Mechatronice





MPLAB® X IDE

Michal Bastl

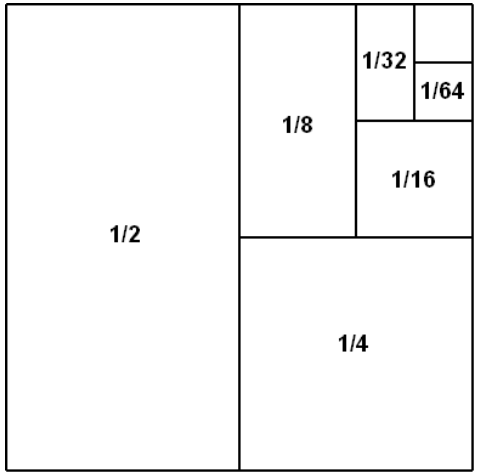
První test

- 4 týden semestru
- Proběhne na cvičení
- Formou testu v e-learningu (student se přihlásí)

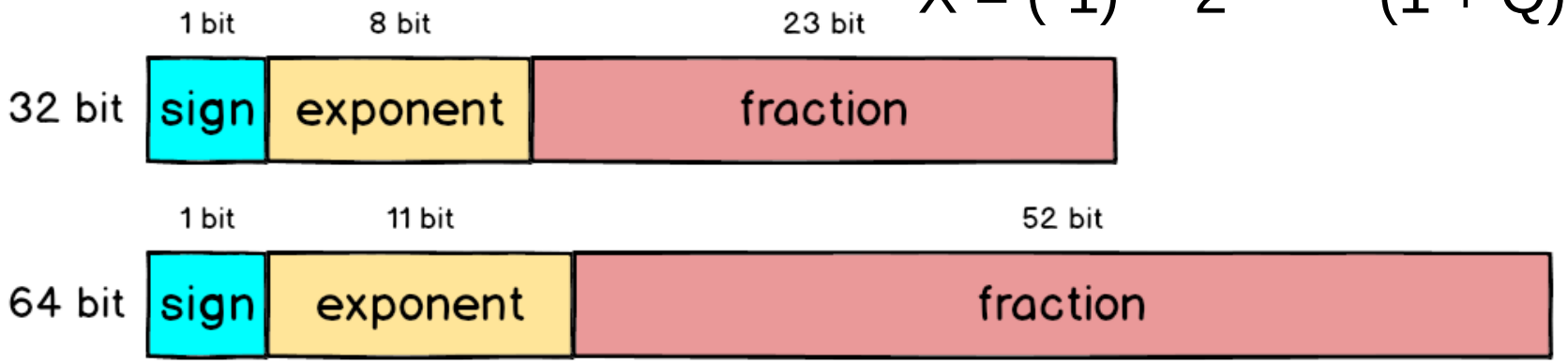
Témata:

- Číselné soustavy (HEX, BIN, i dvojkový doplněk)
- Funkce
- Syntaxe C
- Standardní knihovny (kde je fce printf() apod.)
- Binární operace
- Pointery a pole

Reprezentace IEEE 754



$$X = (-1)^s * 2^{E-127} * (1 + Q)$$



0 10000001 11011000000000000000000

S: kladné
E: 129-127 = 2

Q+1: 1+1/2+1/4+1/16+1/32 = 1,84375

+ 2^2 * 1,84375 = 7,375

Ukázka

-0.875

S: zaporné

Q: 0,111 … 1,11

E : -1 + 127 = 126

1 01111110 11000000000000000000000

Struktury

Struktura je zjednodušeně datový typ, do které uzavřeme další datové typy. Například každý uživatel má jméno, věk atd.

Struktura může uchovávat různé datové typy a pointery i pole.

Strukturu lze vytvořit různým zápisem, ale vřele doporučujeme držet se tohoto zápisu a vytvořit strukturu jako nový datový typ. V příkladu je umístěna do globálního prostoru.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
   char    jmeno[25];
   int     vek;
   int     vyska;
} clovek;

int main() {

   clovek Petr = {"Petr Novak", 25, 178};
   clovek Michal;

   Michal.vek = 16;
   Michal.vyska = 193;

   strcpy(Michal.jmeno, "Michal Novak");

   printf("Petr ma %d let\n",  Petr.vek);
   printf("Michal se jmenuje %s",  Michal.jmeno);

   return 0;
}
```

Ukázka

```
>>Petr ma 25 let
>>Michal se jmenuje Michal Novak
```

Struktury - bitova pole

U proměnné ve struktuře je možné určit rozsah v bitech. Může se tak šetřit místem a nebo využívat omezený rozsah takové proměnné.

Počet bitů se uvádí za dvojtečku. Tato velikost se pak v rámci programu dodržuje.

```
#include <stdio.h>
#include <stdint.h>

typedef struct{
	uint8_t u1:4;
	uint8_t u2:4;
}bitf_t;

void main(void){

	bitf_t my_st;

	my_st.u1 = 1;

	int i;
	for(i=0; i<20;i++){
		printf("%d\n", my_st.u1++);
	}

}
```

Ukázka

Union

Union zabere v paměti pouze tolik místa jako na místo nejnáročnější proměnná. Tyto proměnné se překrývají.

Použití: Přístup k jednotlivým bajtům, například odeslaní typu float přes sběrnici.

```
typedef union{
    int a;
    uint8_t b[4];
}my_un;
```

```
typedef union{
    float f;
    uint8_t b[4];
}myfloat_t;
```

Ukázka

Enum – výčtový typ

Enum je datový typ, který sdružuje konstanty. Může být argumentem funkcí i jako návratová hodnota. Jedná se o datový typ int pro příslušnou platformu. Používá se často jako tzv. flags, jak je demonstrováno v příkladu.

```
typedef enum{
        OK=0,
        NOK,
        ERR,
}STATUS;
```

```
STATUS fun(int a){
        if(a == 0){
                return OK;
        }
        else if(a > 0){
                return NOK;
        }
        else{
                return ERR;
        }
};
```

```
#include <stdio.h>

STATUS fun(int a);

void main(void){

        switch(fun(-1)){
                case OK:
                        printf("OK");
                        break;
                case NOK:
                        printf("NOK");
                        break;
                case ERR:
                        printf("ERR");
                        break;
        }
        printf("\n%d",  ERR);
}
```

Ukázka

Dynamická alokace

Funkce:

- malloc()
- calloc()
- free()
- realloc()

- Dynamická alokace se provádí na haldě Heap
- Ukážeme si malloc a free
- Funkci malloc() si řeknu o alokaci jasně určené velikosti paměti v bajtech
- Funkce malloc() vrací pointer na začátek alokované paměti void*
- Pokud se alokace nezdaří - NULL

```
int main()
{

    int* ptr;
    int n, i;

    n = 5;

    ptr = (int*)malloc(n * sizeof(int));

    //vždy kontrola
    if (ptr == NULL) {
        printf("allocation failed\n");
        return -1;
    }
```

Dynamická alokace

```
char g_str[] = "Nejaky text";

char *p1 = gimme_memory(sizeof(g_str));

if(p1 == NULL){
	return -1;
}

strcpy(p1, g_str);
printf("%s", p1);
free(p1);
char *p2;
 gimme_memory_arg(&p2, sizeof(g_str));

 if(p2 == NULL){
	return -1;
 }

 strcpy(p2, g_str);
 printf("%s", p2);
 free(p2);
```

```
void gimme_memory_arg(char **p, int len){

	*p = (char*)malloc(len * sizeof(char));

}

char *gimme_memory(int len){

	char *ret_p = (char*)malloc(len * sizeof(char));

	if(ret_p == NULL){
		return NULL;
	}
	return ret_p;
}
```

Ukázka