



Посебни елементи физичке архитектуре и њихово искоришћење из Цеа



Групписање процесора по намени



- Процесори опште намене
- Наменски процесори
(Процесори посебне намене)
 - **ДСП-ови**
 - Микроконтролери
 - Графички процесори...



```
float aryA[N];  
float aryB[N];  
float aryC[N];  
  
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```





Аритметика у непокретном зарезу



```
1.000 (-1)    int a = 0x8; //1000    _Fract a = -1.0r;  
0.011 (3/8)   int b = 0x3; //0011    _Fract b = 0.375r;
```

```
11.101000    int c = fp_mul(a, b); _Fract c = a * b;  
    << 1  
1.101000
```

```
_Fract  
long _Fract  
_Accum  
long _Accum
```

0.25r, 0.51r, 5.6k, 50.71k, 1.5r



Акумулатори



$$0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 - 0.25 + 0.5 = -0.6$$

$$0.5 + 0.75 = 1.25$$

$$0.5 + 0.75 - 0.3 = 0.95$$

$$0.5 + 0.75 - 0.3 - 0.8 = 0.15$$

$$0.5 + 0.75 - 0.3 - 0.8 - 0.6 = -0.45$$

$$0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 = -0.85$$

$$0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 - 0.25 = -1.1$$

$$0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 - 0.25 + 0.5 = -0.6$$



```
float aryA[N];  
float aryB[N];  
float aryC[N];
```

```
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```

```
_Fract aryA[N];  
_Fract aryB[N];  
_Fract aryC[N];
```

```
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```

```
_conv:  
    a1 = 0  
start:  
    a1 > 30  
    if (T) jmp end:  
        i0 = _aryA + a1  
        x0 = mem[i0]  
        i0 = _aryB + a1  
        y0 = mem[i0]  
        a0 = x0 * y0  
        i0 = _aryC + a1  
        mem[i0] = a0  
        a1 = a1 + 1  
        jmp start  
end:  
    i7 = mem[i6]  
    jmp i7
```



Адресни генератор

```
...ary[0];  
...ary[2];  
...ary[1];  
...ary[2];  
...ary[3];  
p = ary;  
...*p; p+=2  
...*p--;  
...*p++;  
...*p++;  
...*p;
```

```
struct s  
{  
    int a;  
    int b;  
    int c;  
    int d;  
};
```

```
struct s* ps;  
...ps->d;  
...ps->a;  
...ps->c;  
...ps->b;
```

```
struct s  
{  
    int d;  
    int a;  
    int c;  
    int b;  
};
```



```
_Fract aryA[N];  
_Fract aryB[N];  
_Fract aryC[N];
```

```
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```

```
void conv() {  
    int i;  
    _Fract* pA = &aryA[0];  
    _Fract* pB = &aryB[0];  
    _Fract* pC = &aryC[0];  
    for (i = 0; i < N; ++i) {  
        *pC++ = *pA++ * *pB++;  
    }  
}
```

```
_conv:  
    a1 = 0  
start:  
    a1 > 30  
    if (T) jmp end:  
        i0 = _aryA + a1  
        x0 = mem[i0]  
        i0 = _aryB + a1  
        y0 = mem[i0]  
        a0 = x0 * y0  
        i0 = _aryC + a1  
        mem[i0] = a0  
        a1 = a1 + 1  
        jmp start  
end:  
    i7 = mem[i6]  
    jmp i7
```

```
_conv:  
    a1 = 0  
start:  
    a1 > 30  
    if (T) jmp end:  
        x0 = mem[i0]  
        i0 = i0 + 1  
        y0 = mem[i4]  
        i4 = i4 + 1  
        a0 = x0 * y0  
        mem[i1] = a0  
        i1 = i1 + 1  
        a1 = a1 + 1  
        jmp start  
end:  
    i7 = mem[i6]  
    jmp i7
```





```
_Fract aryA[N];  
_Fract aryB[N];  
_Fract aryC[N];
```

```
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```

```
void conv() {  
    int i;  
    _Fract* pA = &aryA[0];  
    _Fract* pB = &aryB[0];  
    _Fract* pC = &aryC[0];  
    for (i = 0; i < N; ++i) {  
        *pC++ = *pA++ * *pB++;  
    }  
}
```

```
_conv:  
    a1 = 0  
start:  
    a1 > 30  
    if (T) jmp end:  
        x0 = mem[i0]  
        i0 = i0 + 1  
        y0 = mem[i4]  
        i4 = i4 + 1  
        a0 = x0 * y0  
        mem[i1] = a0  
        i1 = i1 + 1  
        a1 = a1 + 1  
        jmp start  
end:  
    i7 = mem[i6]  
    jmp i7
```

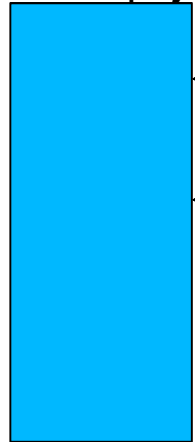
```
_conv:  
    a1 = 0  
start:  
    a1 > 30  
    if (T) jmp end:  
        x0 = mem[i0]; i0 += 1  
        y0 = mem[i4]; i4 += 1  
        a0 = x0 * y0  
        mem[i1] = a0; i1 += 1  
        a1 = a1 + 1  
        jmp start  
end:  
    i7 = mem[i6]  
    jmp i7
```



Харвард архитектура



Меморија



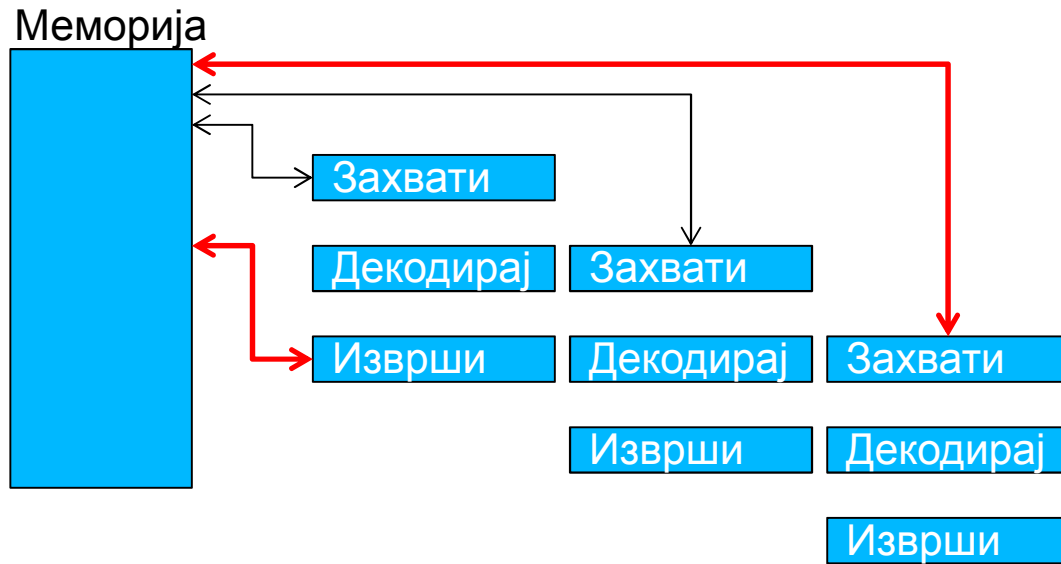
Захвати

Декодирај

Изврши

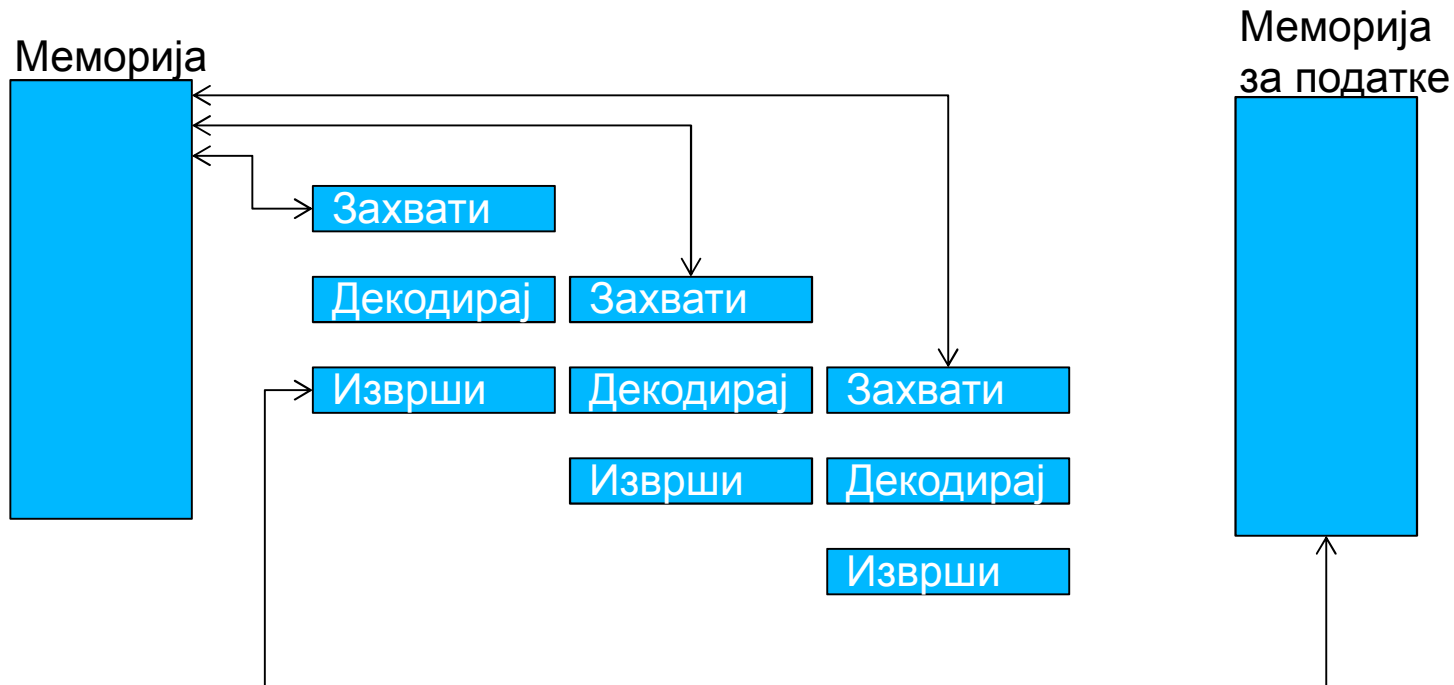


Харвард архитектура





Харвард архитектура

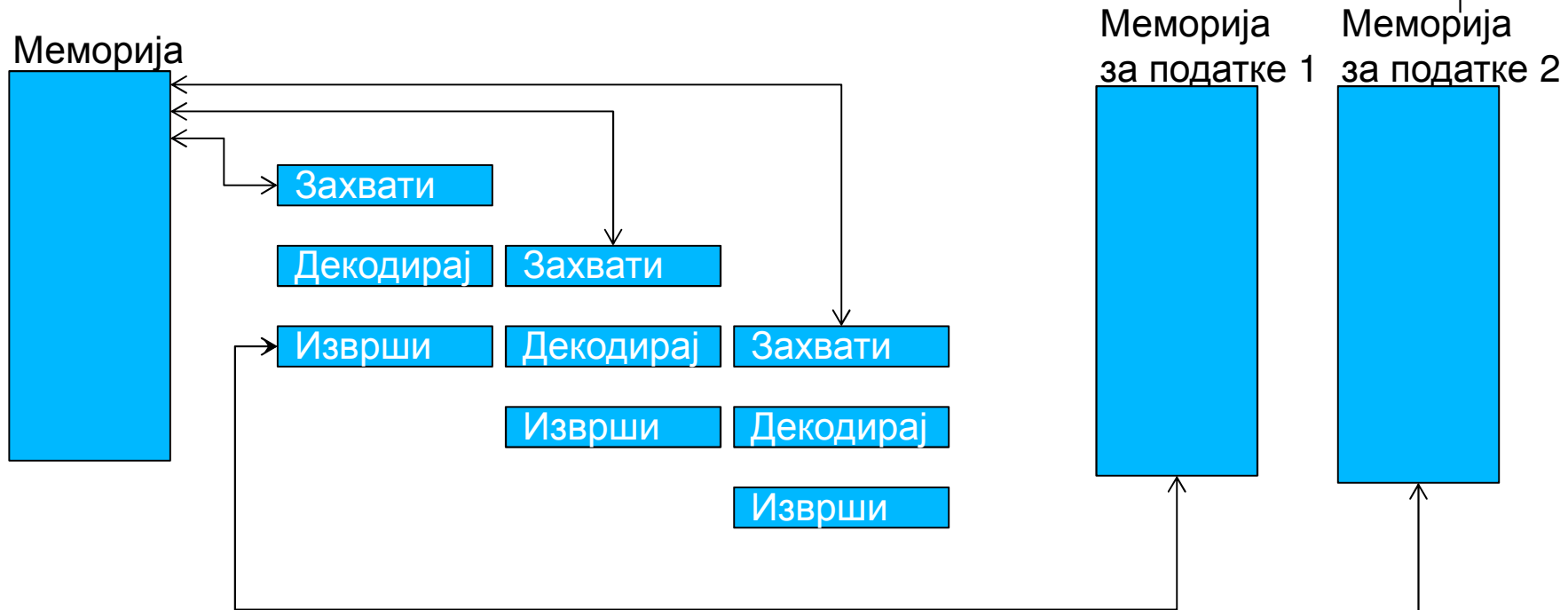


$$c = a * b$$

Колико приступа меморији?



Харвард архитектура



```
__memA int ary1[100];  
__memB int ary2[100];
```



```
_Fract aryA[N];  
_Fract aryB[N];  
_Fract aryC[N];
```

```
void conv() {  
    int i;  
    _Fract* pA = &aryA[0];  
    _Fract* pB = &aryB[0];  
    _Fract* pC = &aryC[0];  
    for (i = 0; i < N; ++i) {  
        *pC++ = *pA++ * *pB++;  
    }  
}
```

```
__memX _Fract aryA[N];  
__memY _Fract aryB[N];  
__memX _Fract aryC[N];
```

```
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```

```
__memX _Fract aryA[N];  
__memY _Fract aryB[N];  
__memX _Fract aryC[N];
```

```
void conv() {  
    int i;  
    __memX _Fract* pA = &aryA[0];  
    __memY _Fract* pB = &aryB[0];  
    __memX _Fract* pC = &aryC[0];  
    for (i = 0; i < N; ++i) {  
        *pC++ = *pA++ * *pB++;  
    }
```

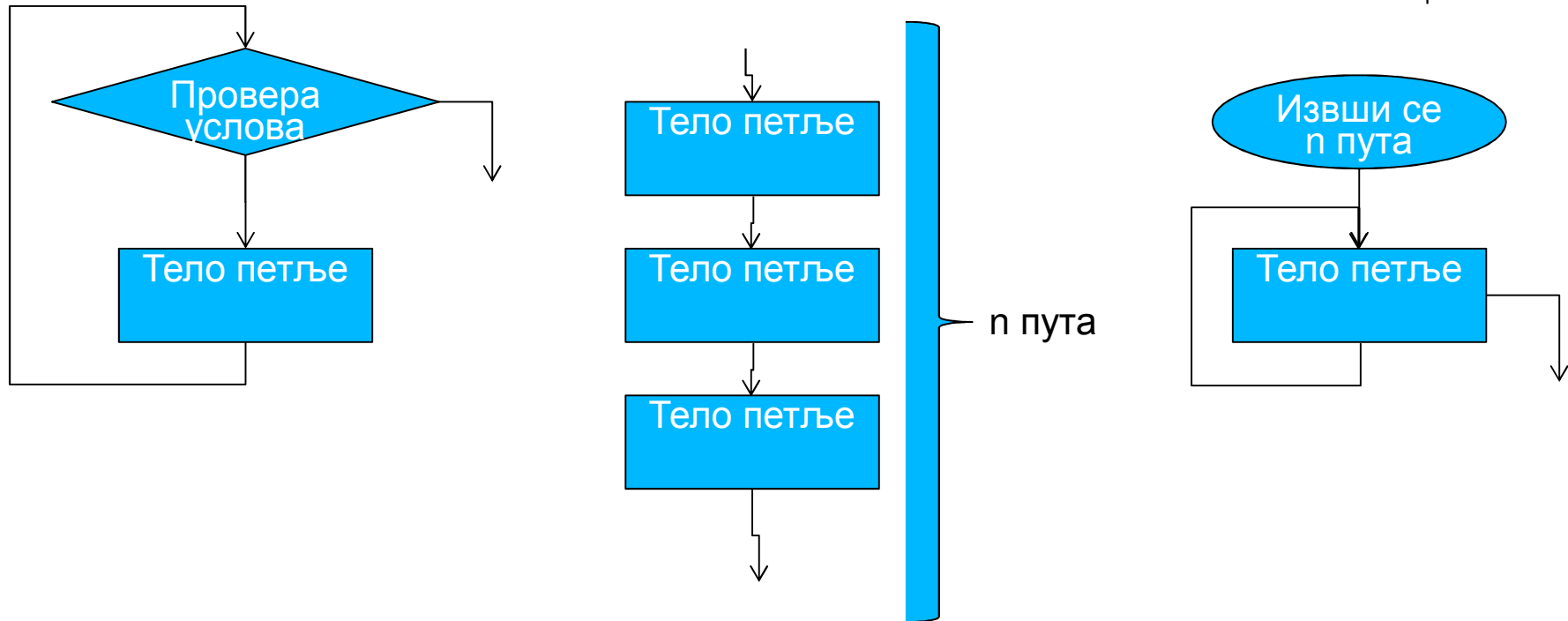


```
_conv:
    a1 = 0
start:
    a1 > 30
    if (T) jmp end:
        x0 = mem[i0]; i0 += 1
        y0 = mem[i4]; i4 += 1
        a0 = x0 * y0
        mem[i1] = a0; i1 += 1
        a1 = a1 + 1
        jmp start
end:
    i7 = mem[i6]
    jmp i7
```

```
_conv:
    a1 = 0
start:
    a1 > 30
    if (T) jmp end:
        x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
        a0 = x0 * y0
        xmem[i1] = a0; i1 += 1
        a1 = a1 + 1
        jmp start
end:
    i7 = mem[i6]
    jmp i7
```



Хардверски подржане петље



- Петље код којих је број итерација познат већ током превођења
- Петље код којих је број итерација познат пре него што петља почне



Хардверски подржане петље



```
for (i = 0; i < NUMBER_OF_ITERATIONS; i++)  
for (i = NUMBER_OF_ITERATIONS; i > 0; i--)  
for (i = 0; i++ < NUMBER_OF_ITERATIONS; )  
for (i = NUMBER_OF_ITERATIONS; i-- > 0; )  
for (i = 0; i < NUMBER_OF_ITERATIONS; ++i)  
for (i = NUMBER_OF_ITERATIONS; i > 0; --i)  
for (i = 0; ++i < NUMBER_OF_ITERATIONS + 1; )  
for (i = NUMBER_OF_ITERATIONS + 1; --i > 0; )
```



```
__memX __Fract aryA[N];  
__memY __Fract aryB[N];  
__memX __Fract aryC[N];
```

```
void conv() {  
    int i;  
    for (i = 0; i < N; ++i) {  
        aryC[i] = aryA[i] * aryB[i];  
    }  
}
```

```
__memX __Fract aryA[N];  
__memY __Fract aryB[N];  
__memX __Fract aryC[N];
```

```
void conv() {  
    int i;  
    __memX __Fract* pA = &aryA[0];  
    __memY __Fract* pB = &aryB[0];  
    __memX __Fract* pC = &aryC[0];  
    for (i = 0; i < N; ++i) {  
        *pC++ = *pA++ * *pB++;  
    }  
}
```





```
_conv:
    a1 = 0
start:
    a1 > 30
    if (T) jmp end:
        x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
        a0 = x0 * y0
        xmem[i1] = a0; i1 += 1
        a1 = a1 + 1
        jmp start
```

```
end:
    i7 = mem[i6]
    jmp i7
```

```
_conv:
    hw_loop(31), end
        x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
        a0 = x0 * y0
        xmem[i1] = a0; i1 += 1
end:
    i7 = mem[i6]
    jmp i7
```



Хардверски стек позива функција



push pc
jmp addr

pop reg
jmp reg

call addr

ret



```
_conv:
    hw_loop(31), end
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
end:
    i7 = mem[i6]
    jmp i7
```

```
_conv:
    hw_loop(31), end
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
end:
    ret
```

```
i6 += 1
xmem[i6] = pc
jmp conv
```

```
call conv
```



Хардверски подржан кружни бафер



```
__attribute__((align(128)))
```

```
p = ary;  
for (...)   
{  
    ...*p...  
    p = CIRC_INC(p, MOD128, 1);  
}
```



Векторске операције

```
int8_t A[100];
int8_t B[100];
int8_t C[100];
for (i = 0; i < 100; i++)
{
    C[i] = A[i] + B[i];
}
```

```
int8_t __attribute__((vector_size(4))) A[25];
int8_t __attribute__((vector_size(4))) B[25];
int8_t __attribute__((vector_size(4))) C[25];
for (i = 0; i < 25; i++)
{
    C[i] = __builtin_add_qb(A[i], B[i]);
}
```

```
int8_t __attribute__((vector_size(4))) x = {1, 2, 3, 4};
```



Две врсте окружења у којима се Це програми извршавају



Це стандард идентификује две категорије окружења:

- Подржавано (енгл. Hosted)
 - Претпоставља се постојање релативно раскошног ОС-а и имплементација свих библиотечких функција.
 - Функција `main` представља почетну тачку извршавања
- Неподржавано (енгл. Freestanding)
 - ОС може пружати само елементарне ствари, а од библиотека се очекује само: `float.h`, `iso646.h`, `limits.h`, `stdarg.h`, `stdbool.h`, `stddef.h`.
 - Функција `main` нема никакву посебност. Механизам почињана извршавања програма може бити произвољан.
 - Нема (тј. не мора бити) комплексног типа.



Зашто програми не раде добро?





Зашто програми не раде добро?



Једна анализа у домену уграђених система показује:

- 43% проблема настаје због спецификације
- 21% услед промене спецификације након завршетка развоја
- 15% током коришћења и одржавања
- **15% током развоја и имплементације**
- 6% током инсталације и испоруке



MISRA C

(обавезно прочитати поглавља [1,3])



- Rule 1.2 (required): No reliance shall be placed on undefined or unspecified behavior.
- Rule 2.4 (advisory): Sections of code should not be “commented out”.
- Rule 3.1 (required): All usage of implementation-defined behavior shall be documented.
- Rule 5.2 (required): Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.
- Rule 6.1 (required): The plain char type shall be used only for the storage and use of character values.
- Rule 7.1 (required): Octal constants (other than zero) and octal escape sequences shall not be used.
- Rule 8.5 (required) There shall be no definitions of objects or functions in a headerfile.
- Rule 8.6 (required): Functions shall be declared at file scope.
- Rule 16.2 (required): Functions shall not call themselves, either directly or indirectly.