
ScTec
Assessoria e Projetos especiais

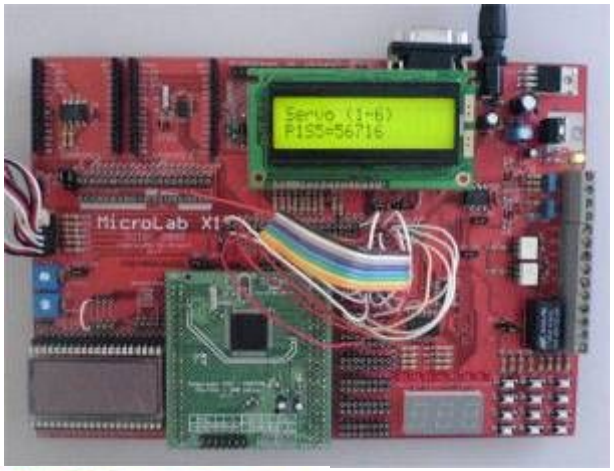
A Empresa

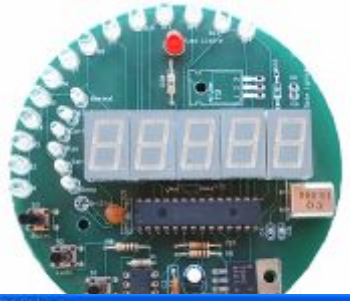
A Sctec Eletrônica foi fundada em abril de 2003 com o objetivo de desenvolver soluções e produtos eletrônicos utilizando a mais moderna tecnologia disponível, atuando no desenvolvimento de hardware e software.

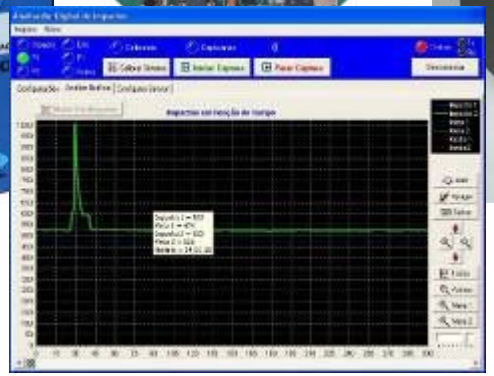
Áreas de Atuação

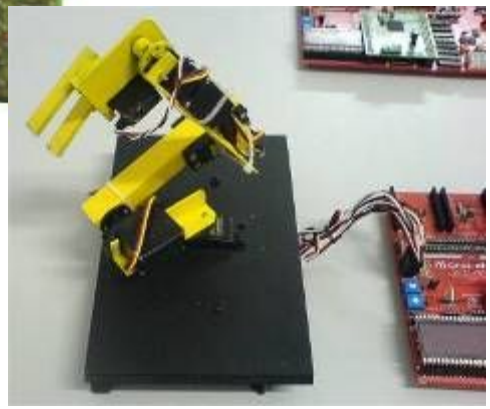
- Produtos Automotivos
- Automação Industrial
- Comunicação com ou sem fio
- Sensoriamento Remoto
- Embedded Systems
- Desenvolvimento de Projetos para Terceiros

Produtos

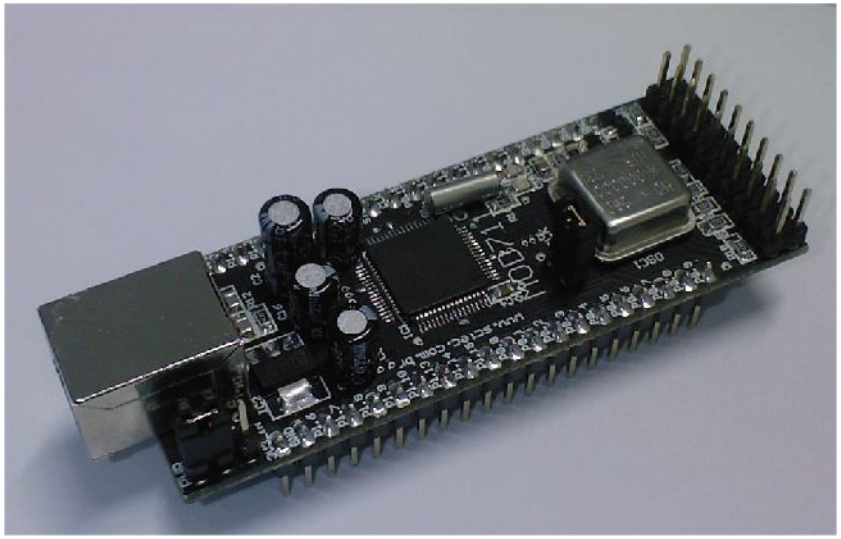








Produtos



MOD711

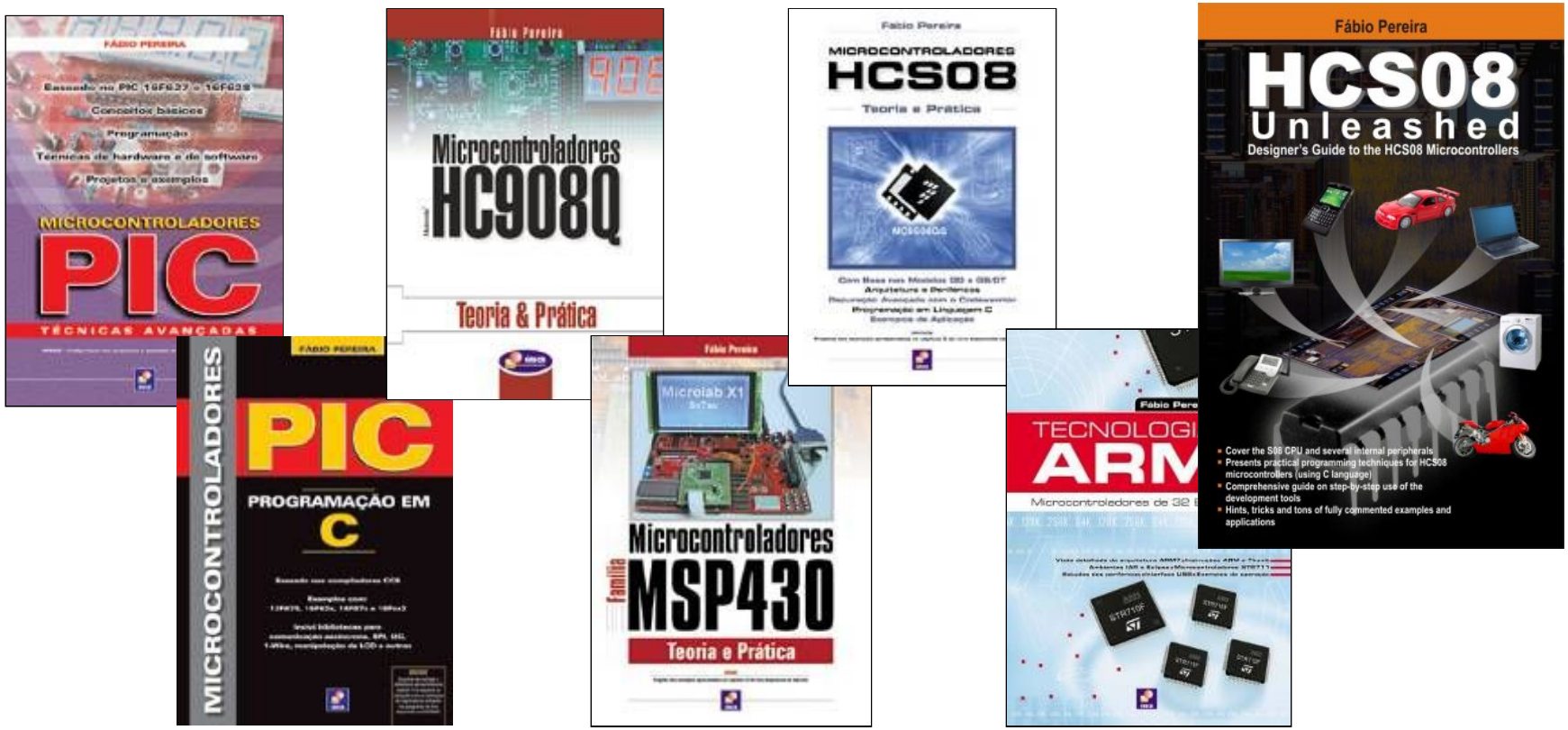
- ARM7
- 256 Kb FLASH
- 64 Kb RAM
- USB 2.0
- JTAG

Projetos

Computador de Bordo – M8 Octopus

Educacional

Livros



Parcerias

freescale™ semiconductor
Design Alliance Program

Texas Instruments

Microchip

Otimização de Programas C para Sistemas Embarcados

Introdução

Sistemas embarcados formam um campo cada vez mais numeroso na área de desenvolvimento de software. No entanto, muitos programadores desconhecem aspectos importantes e essenciais ao desenvolvimento neste tipo de plataforma.

Introdução

Nesta apresentação veremos alguns aspectos importantes da utilização da linguagem C no desenvolvimento de aplicações embarcadas mais eficientes.

Introdução

Aspectos importantes para um programa C eficiente:

1. Conheça o compilador
2. Conheça a CPU
3. Utilize o tipo de dado mais eficiente
4. Evite utilizar funções de biblioteca
5. Programe de forma a auxiliar o trabalho do compilador

Tipos de Dados e Alocação de Memória

Signed x Unsigned

Signed versus unsigned

Qual o tipo ideal para a minha aplicação?

A aplicação necessita de um dado signed?

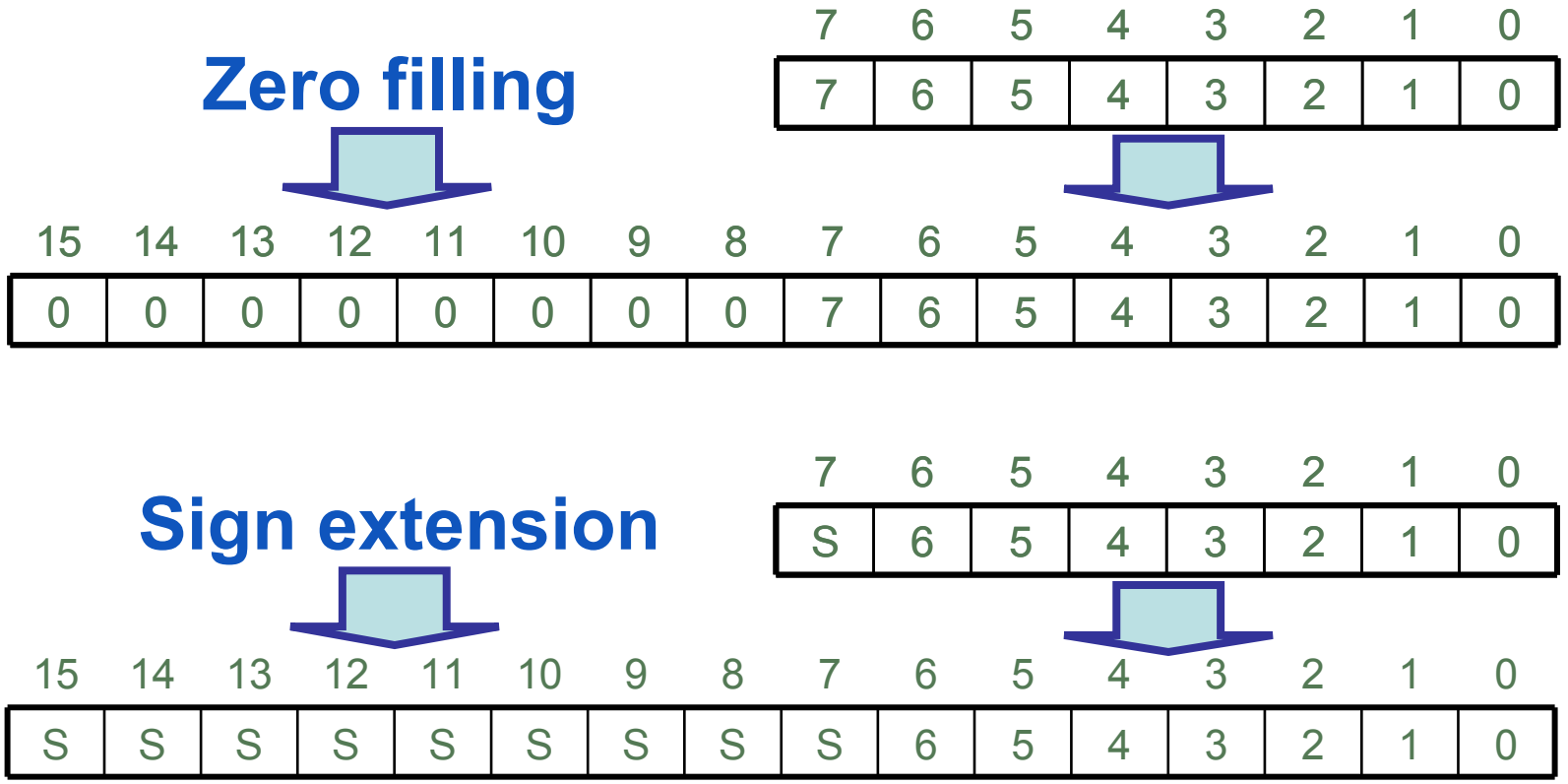
Há diferença de performance?

Signed x Unsigned

Extensão de sinal!

Ocorre sempre nas atribuições entre variáveis sinalizadas e de tamanhos diferentes (por exemplo a atribuição de uma variável signed char para uma variável signed int).

Signed x Unsigned



Signed x Unsigned

Atribuição de char para int nos PIC18:

Unsigned

```
;uiv = ucv
MOVFF ucv,uiv
MOVLB 0
CLRF  uiv+1, BANKED
```

Signed

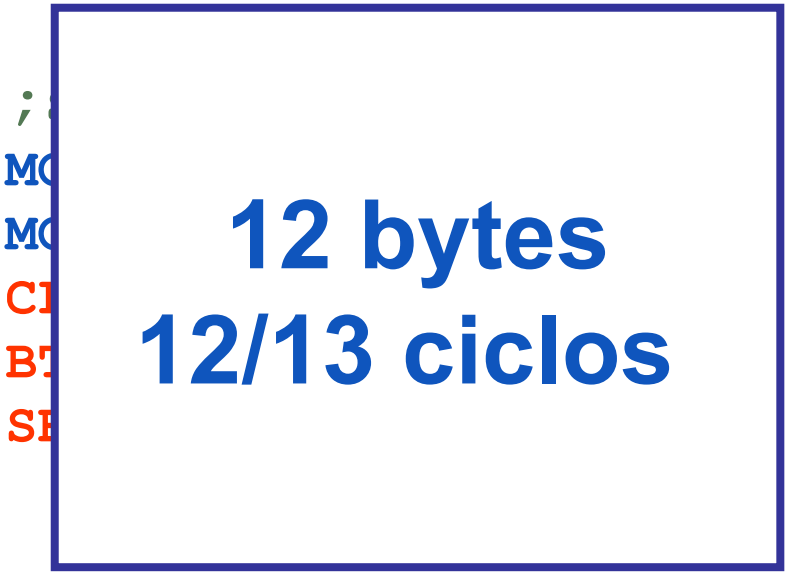
```
;siv = scv
MOVFF scv,siv
MOVLB 0
CLRF  siv+1, BANKED
BTFSC siv,7, BANKED
SETF  siv+1, BANKED
```

Signed x Unsigned

Atribuição de char para int nos PIC18:

Unsigned

Signed



Signed x Unsigned

Atribuição de char para int nos MSP430:

Unsigned

```
;uiv = ucv
MOV.B &ucv,R15
MOV.W R15,&uiv
```

Signed

```
;siv = scv
MOV.B &scv,R15
SXT   R15
MOV.W R15,&siv
```

Signed x Unsigned

Atribuição de char para int nos MSP430:

Unsigned

8 bytes
7 ciclos

Signed

10 bytes
15 ciclos

Signed x Unsigned

Algumas operações aritméticas também podem sofrer impactos negativos pelo uso de tipos signed:

Signed x Unsigned

Divisão de 16 bits nos HCS08

Unsigned	Signed
118 bytes 238 ciclos	160 bytes 356 ciclos

Signed x Unsigned

Divisão de 32 bits nos ARM7

Unsigned	Signed
456 bytes 51 ciclos	472 bytes 48 ciclos

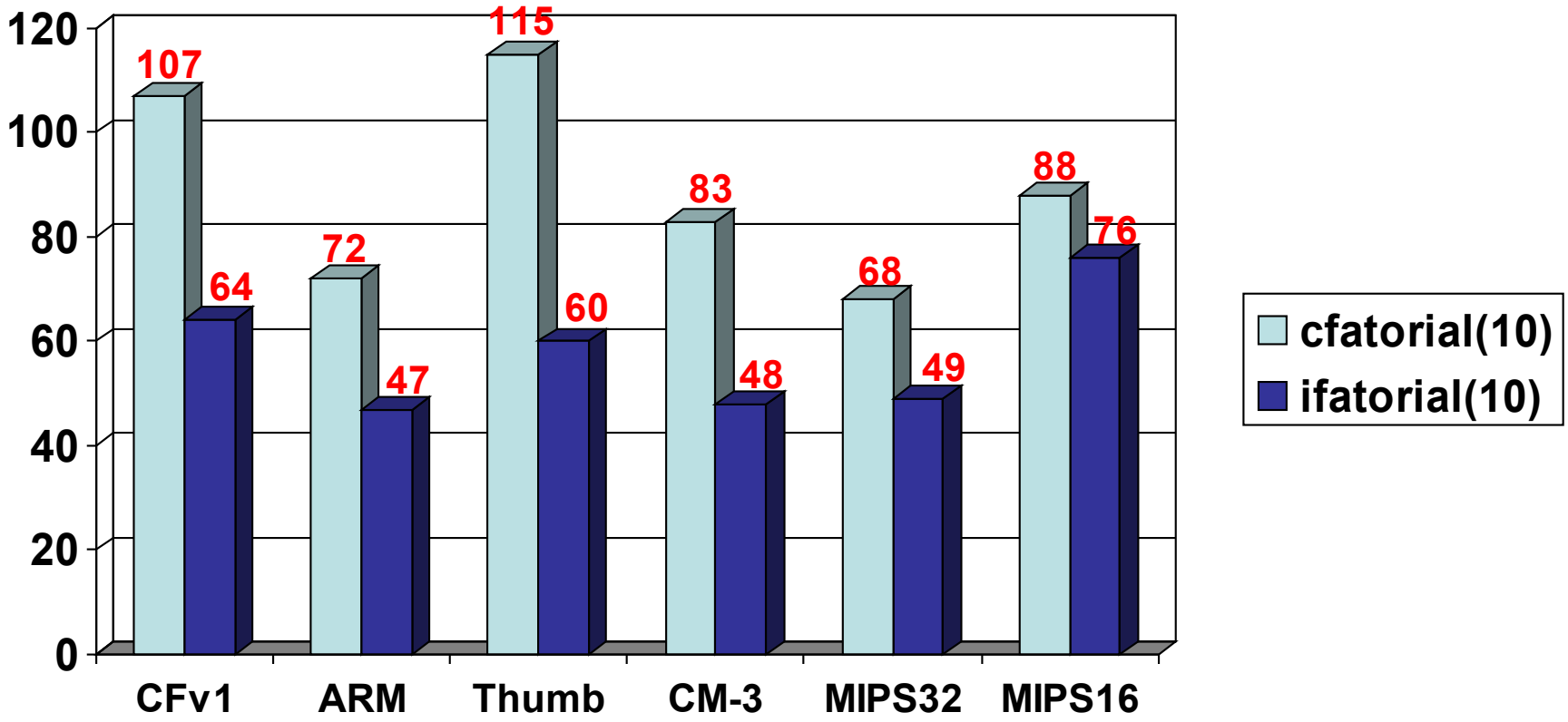
Seleção de Tipos

Tipos nativos tendem a ser mais rápidos:

```
unsigned char cfatorial(unsigned char val)
{
  unsigned char aux, result=1;
  for(aux=1;aux<=val;aux++) result *= aux;
  return result;
}
unsigned int ifatorial(unsigned int val)
{
  unsigned int aux, result=1;
  for(aux=1;aux<=val;aux++) result *= aux;
  return result;
}
```

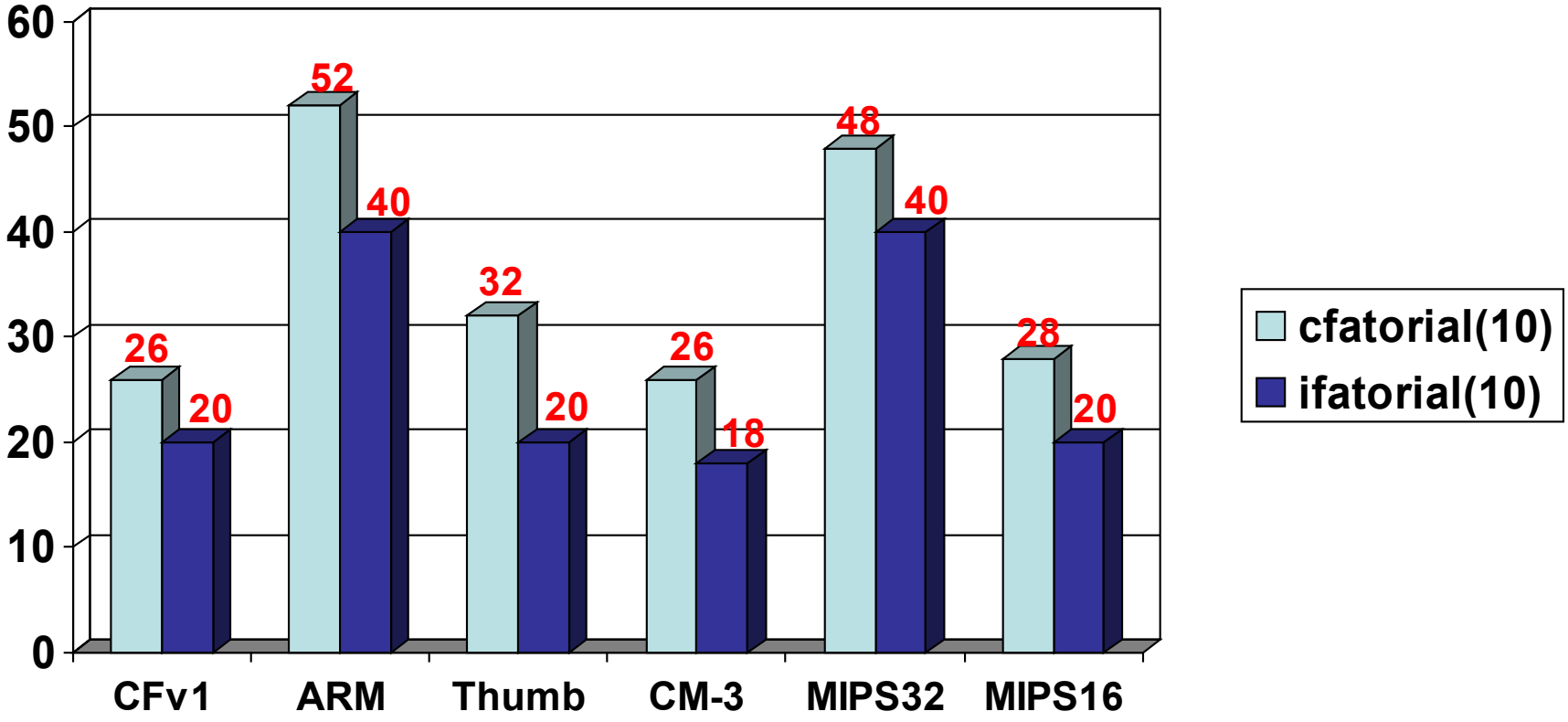
Seleção de Tipos

Velocidade (ciclos de clock):



Seleção de Tipos

Tamanho de código (bytes):



Seleção de Tipos

Variáveis locais devem utilizar preferencialmente tipos nativos da CPU em questão (char para chips de 8 bits, int para chips de 16/32 bits).

Campos de Bits

Campos de bits

Muitas aplicações tendem a utilizar variáveis apenas como flags, assumindo dois valores: verdadeiro ou falso.

A utilização de campos de bits permite reduzir consideravelmente a quantidade de RAM ocupada, mas será o código resultante mais eficiente?

Campos de bits

```
struct
{
  unsigned char bit0 : 1;
  unsigned char bit1 : 1;
  unsigned char bit2 : 1;
} teste;
```

Campos de bits

Atribuições utilizando bits e bytes (HCS08):

Bit:

```
teste.bit0 = 1;

BSET 0,teste
```

Byte:

```
flag = 1;

LDA  #0x01
LDHX #flag
STA  ,X
```

Campos de bits

Atribuições utilizando bits e bytes (HCS08):

Bit:

2 bytes FLASH
1 bit RAM
10 ciclos

Byte:

6 bytes FLASH
1 byte RAM
16 ciclos

Campos de bits

Teste de flag utilizando bits e bytes (HCS08):

Bit:

```
If (teste.bit0)...

BRCLR 0,teste,falso
// verdadeiro
```

Byte:

```
If (flag) ...

LDA teste
BEQ falso
// verdadeiro
```

Campos de bits

Teste de flag utilizando bits e bytes (HCS08):

Bit:

3 bytes FLASH
1 bit RAM
10 ciclos

Byte:

5/6 bytes FLASH
1 byte RAM
12/14 ciclos

Campos de bits

Atribuições utilizando bits e bytes (PIC16/18):

Bit:

```
teste.bit0 = 1;
```

```
BSF teste,0
```

Byte:

```
flag = 1;
```

```
MOVLW 0x01
MOVWF flag
```

Campos de bits

Atribuições utilizando bits e bytes (PIC16/18):

Bit:

1 word FLASH
1 bit RAM
4 ciclos

Byte:

2 words FLASH
1 byte RAM
8 ciclos

Campos de bits

Teste de flag utilizando bits e bytes (PIC16/18):

Bit:

```
If (teste.bit0)...

BTFSS teste,0
GOTO  falso
// verdadeiro
```

Byte:

```
If (flag) ...

MOVF  flag,F
BTFSC STATUS,Z
GOTO  falso
// verdadeiro
```

Campos de bits

Teste de flag utilizando bits e bytes (PIC16/18):

Bit:

2 words FLASH
1 bit RAM
8/12 ciclos

Byte:

3 words FLASH
1 byte RAM
12/16 ciclos

Campos de bits

Atribuições utilizando bits e bytes (MSP430):

Bit:

```
teste.bit0 = 1;
```

```
BIS.B #1,&teste
```

Byte:

```
flag = 1;
```

```
MOV.B #1,&teste
```

Campos de bits

Atribuições utilizando bits e bytes (MSP430):

Bit:

4 bytes FLASH
1 bit RAM
4 ciclos

Byte:

4 bytes FLASH
1 byte RAM
4 ciclos

Campos de bits

Teste de flag utilizando bits e bytes (MSP430):

Bit:

```
If (teste.bit0)...


BIT.B #1,&teste
JNC   falso
// verdadeiro
```

Byte:

```
If (flag) ...


TST.B &flag
JEQ   falso
// verdadeiro
```

Campos de bits

Teste de flag utilizando bits e bytes (MSP430):

Bit:

6 bytes FLASH
1 bit RAM
6 ciclos

Byte:

6 bytes FLASH
1 byte RAM
6 ciclos

Campos de bits

Em máquinas de 32 bits, a utilização de campos de bit causa uma degradação na performance e um aumento no tamanho do código (código menos eficiente).

Campos de bits

Atribuições utilizando bits e bytes (Coldfire):

Bit:

```
teste.bit0 = 1;
```

```
BSET #0,x(Ay)
```

Byte:

```
flag = 1;
```

```
MOVE.B #1,x(Ay)
```

Campos de bits

Atribuições utilizando bits e bytes (Coldfire):

Bit:

6 bytes FLASH
1 bit RAM
4 ciclos

Byte:

6 bytes FLASH
1 byte RAM
1 ciclo

Campos de bits

Teste de flag utilizando bits e bytes (Coldfire):

Bit:

```
If (teste.bit0)...

MVZ.B 8(A5),D0
LSL.L D1,D0
LSR.L D1,D0
TST.B D0
BEQ.S falso
// verdadeiro
```

Byte:

```
If (flag) ...

TST.B  11(A5)
BEQ.S  falso
// verdadeiro
```

Campos de bits

Teste de flag utilizando bits e bytes (Coldfire):

Bit:

12 bytes FLASH
1 bit RAM
8 ciclos

Byte:

6 bytes FLASH
1 byte RAM
5 ciclos

Campos de bits

Segunda versão do teste de bit:

Bit:

```
If (teste.bit0)...

MVZ.B 8(A5),D0
BTST  #0,8(A5)
BEQ.S falso
// verdadeiro
```

8 bytes FLASH
1 bit RAM
6 ciclos

Campos de bits

Atribuições utilizando bits e bytes (ARM-ARM):

Bit:

```
teste.bit0 = 1;

LDR  R0,[PC,#offset1]
LDR  R1,[PC,#offset2]
LDR  R1,[R1,#+0]
ORRS R1, R1, #0x01
STR  R1,[R0,#+0]
```

Byte:

```
flag = 1;

LDR  R0,[PC,#+offset3]
MOV  R1, #1
STRB R1,[R0,#+0]
```

Campos de bits

Atribuições utilizando bits e bytes (ARM-ARM):

Bit:

20 bytes FLASH
1 bit RAM
8 ciclos

Byte:

12 bytes FLASH
1 byte RAM
4 ciclos

Campos de bits

Atribuições utilizando bits e bytes (ARM-Thumb):

Bit:

```
teste.bit0 = 1;

LDR R0,[PC,#offset1]
LDR R1,[PC,#offset2]
LDR R1,[R1,#0]
MOV R2, #1
ORR R2, R1
STR R2,[R0,#0]
```

Byte:

```
flag = 1;

LDR  R0,[PC,#offset3]
MOV  R1,#1
STRB R1,[R0,#0]
```

Campos de bits

Atribuições utilizando bits e bytes (ARM-Thumb):

Bit:

12 bytes FLASH
1 bit RAM
9 ciclos

Byte:

6 bytes FLASH
1 byte RAM
4 ciclos

Alinhamento de Dados

Alinhamento de dados

Outro aspecto importante a ser considerado quando se programa um sistema embarcado é o alinhamento de dados.

Plataformas como os ARMs possuem restrições quanto a localização dos dados na memória.

Alinhamento de dados

ARM7:

```
struct
{
   char campo1;
   int campo2;
   short campo3;
} teste;
```

0xuuuuuuuu0	campo1
0xuuuuuuuu1	-
0xuuuuuuuu2	-
0xuuuuuuuu3	-
0xuuuuuuuu4	campo2
0xuuuuuuuu5	
0xuuuuuuuu6	
0xuuuuuuuu7	
0xuuuuuuuu8	campo3
0xuuuuuuuu9	

Alinhamento de dados

ARM7:

Memória RAM útil: 7 bytes
Memória RAM utilizada: 10 bytes!

Alinhamento de dados

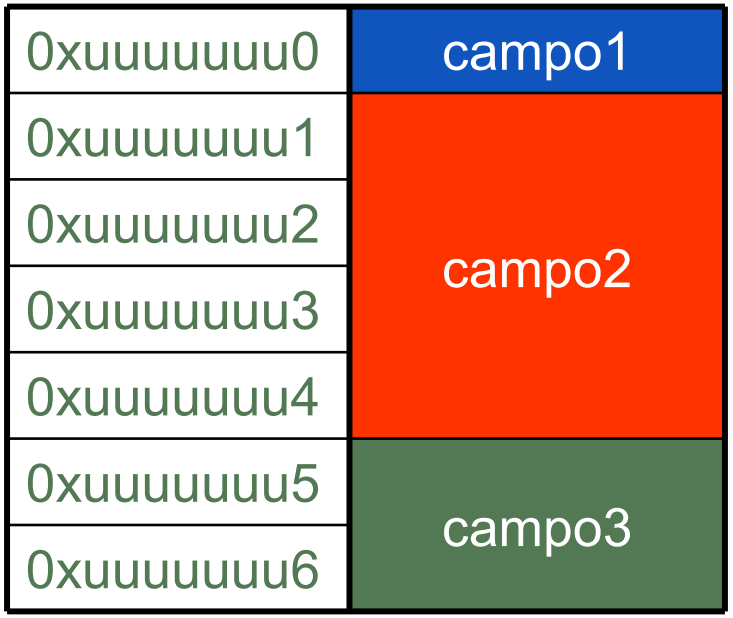
Chips de 8 bits não sofrem de problemas de alinhamento de dados.

Algumas plataformas de 32 bits também permitem otimização da alocação de memória mediante o uso de diretivas especiais:

Alinhamento de dados

Coldfire, Cortex e MIPS:

```
struct
{
   char campo1;
   int campo2;
   short campo3;
} teste;
```



Alinhamento de dados

Coldfire, Cortex e MIPS:

Memória RAM útil: 7 bytes
Memória RAM utilizada: 7 bytes!

Alinhamento de dados

Coldfire, Cortex e MIPS:

IAR: modificador __packed
Codewarrior: diretiva #pragma pack(x)
GCC (MIPS): __attribute__((packed))

Alinhamento de dados

Atenção: acessos não-alinhados à memória podem necessitar de ciclos adicionais de clock para a sua efetivação!

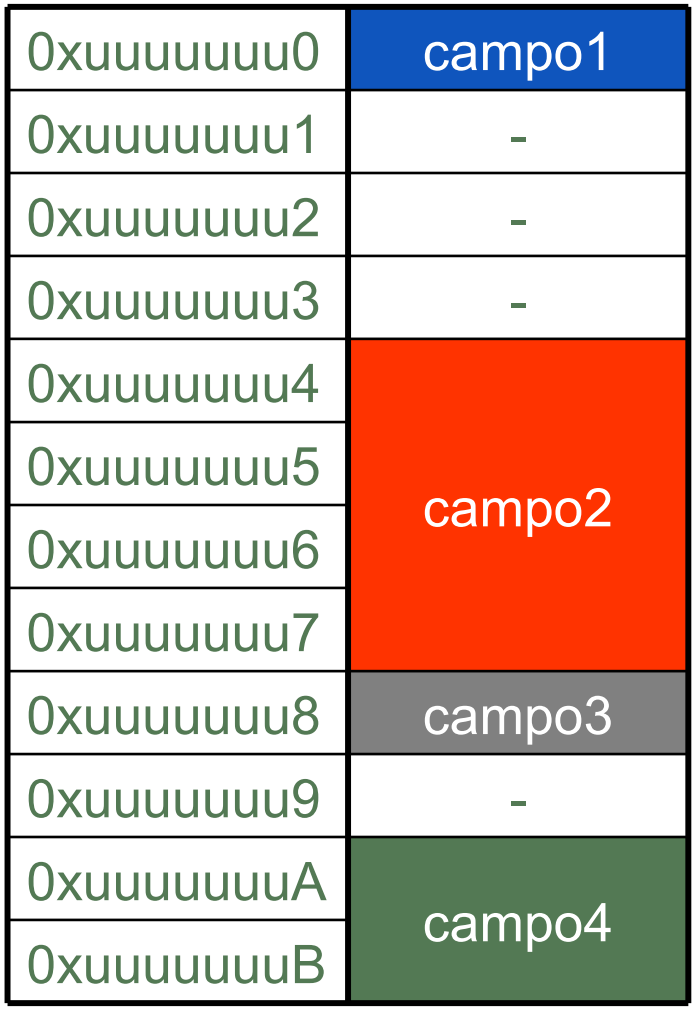
Organização de dados

Também é importante organizar os dados dentro da estrutura de forma a melhorar a distribuição dos mesmos na memória do dispositivo:

Organização de dados

ARM7:

```
struct
{
   char campo1;
   int campo2;
   char campo3;
   short campo4;
} teste;
```



Organização de dados

ARM7:

0xuuuuuuuu0	campo1
0xuuuuuuuu1	-
0xuuuuuuuu2	

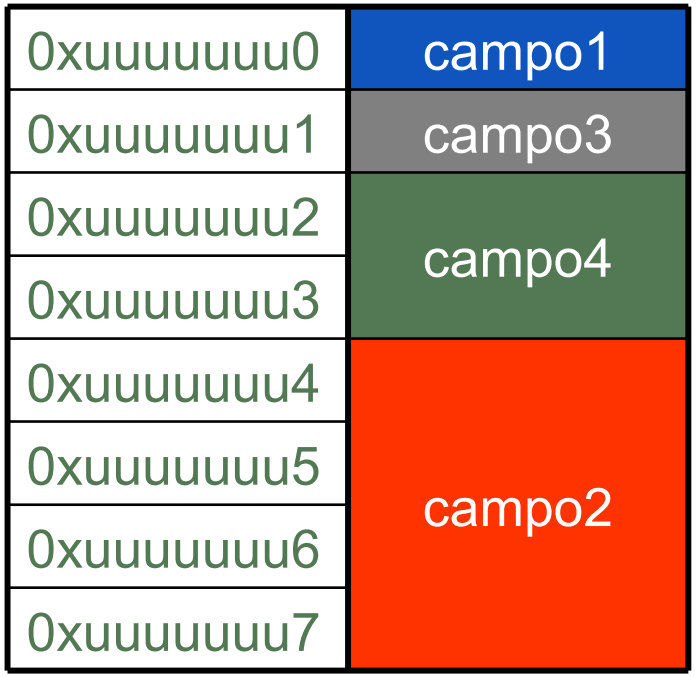
Memória RAM útil: 8 bytes
Memória RAM utilizada: 12 bytes!

```
} teste;
```

0xuuuuuuuuA	campo4
0xuuuuuuuuB	

Organização de dados

```
struct
{
   char campo1;
   char campo3;
   short campo4;
   int campo2;
} teste;
```



Organização de dados

Memória RAM útil: 8 bytes
Memória RAM utilizada: 8 bytes!

```
} teste;
```

Estudo de Casos

Timers e Contadores

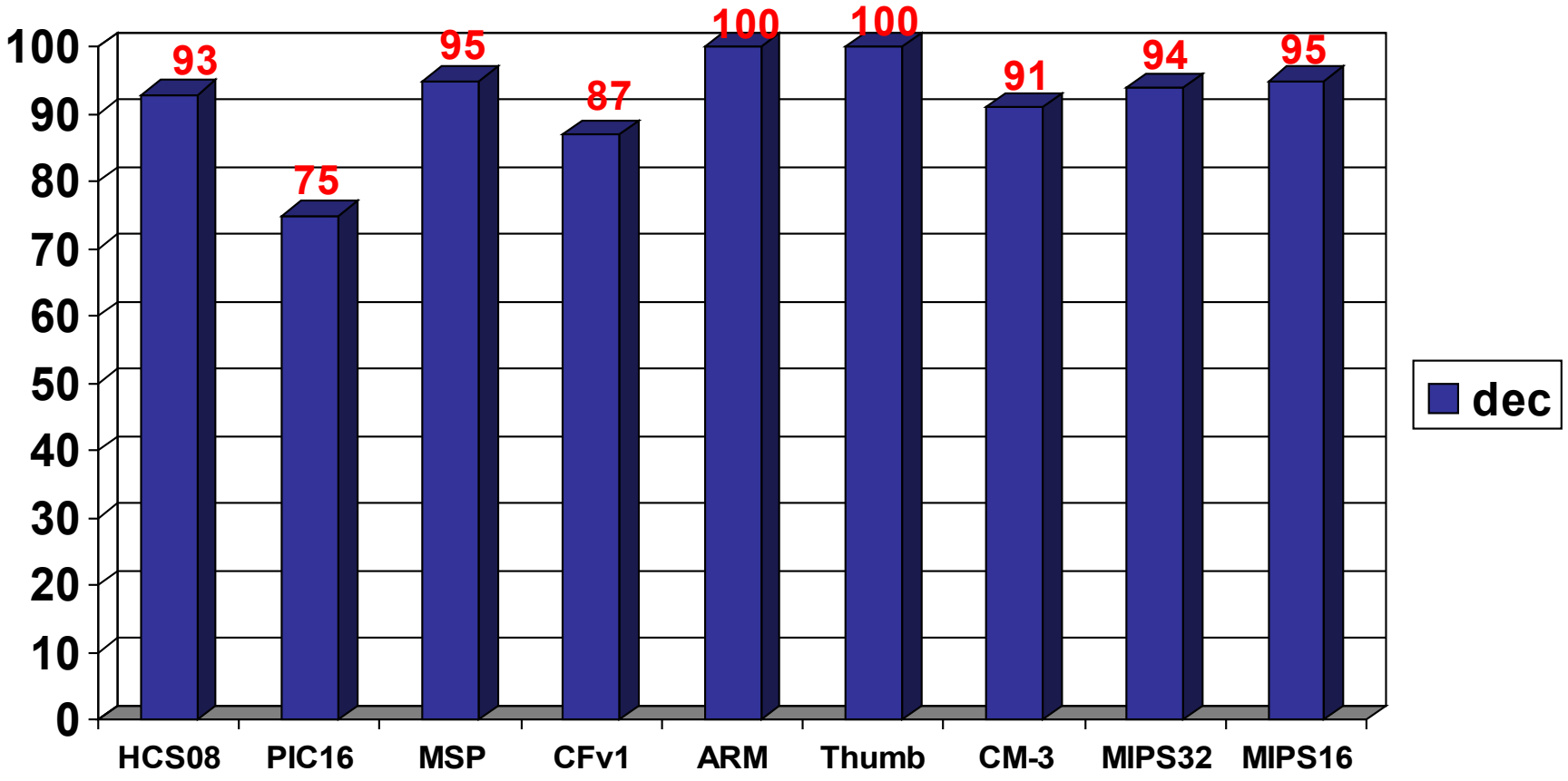
Otimização de timers e contadores:

```
if (va<10) va++;
...
...
```

```
if (va) va--;
...
...
```

Timers e Contadores

Tempos relativos dos contadores decrementais:



Timers e Contadores

Verificação do contador dentro do programa:

```
if (va==VALOR)
...
...

if (!va)
...
...
```

Mais eficiente!

Funções

Redução do número de parâmetros de chamada:

```
void lcd_ks0108_write_byte(unsigned char rs, unsigned char
data)
{
  KS0108_RW = 0; KS0108_DATA_PORT = data;
  KS0108_RS = rs; KS0108_EN=1;
  delay(_KS0108_DELAY/2);
  KS0108_EN=0;
}
```

Funções

Redução do número de parâmetros de chamada:

```
void lcd_ks0108_write_data(unsigned char data)
{
  KS0108_RW = 0; KS0108_DATA_PORT = data;
  KS0108_RS = 1; KS0108_EN=1;
  delay(_KS0108_DELAY/2); KS0108_EN=0;
}
void lcd_ks0108_write_cmd(unsigned char data)
{
  KS0108_RW = 0; KS0108_DATA_PORT = data;
  KS0108_RS = 0; KS0108_EN=1;
  delay(_KS0108_DELAY/2); KS0108_EN=0;
}
```

Funções

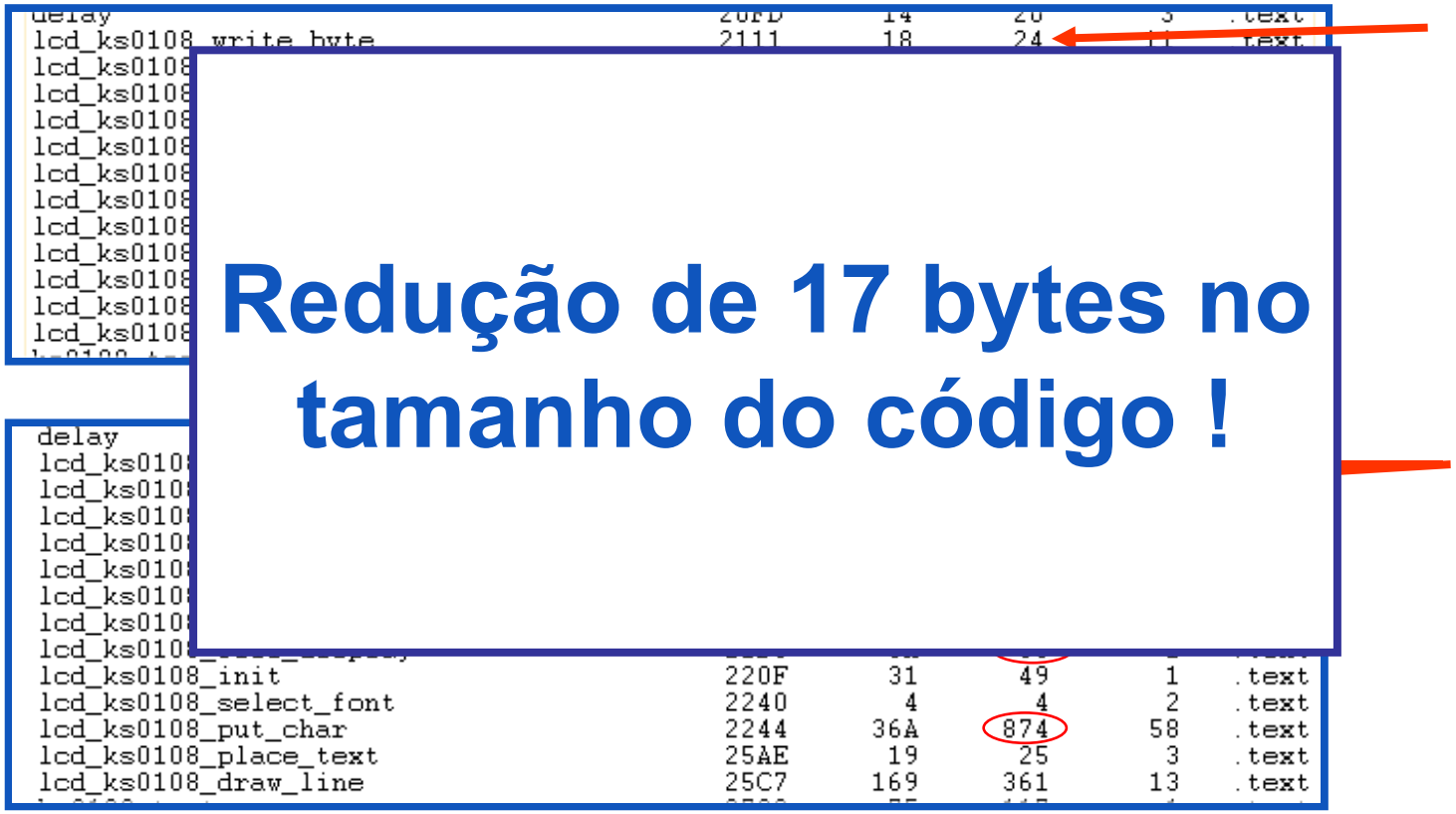
Exame do MAP file:

delay	20FD	14	20	3	.text
lcd_ks0108_write_byte	2111	18	24	11	.text
lcd_ks0108_gotoxy	2129	32	50	8	.text
lcd_ks0108_read_data	215B	17	23	6	.text
lcd_ks0108_display_onoff	2172	1B	27	1	.text
lcd_ks0108_set_start_line	218D	F	15	3	.text
lcd_ks0108_dotxy	219C	41	65	7	.text
lcd_ks0108_fill_display	21DD	3F	63	1	.text
lcd_ks0108_init	221C	31	49	1	.text
lcd_ks0108_select_font	224D	4	4	2	.text
lcd_ks0108_put_char	2251	36E	878	58	.text
lcd_ks0108_place_text	25BF	19	25	3	.text
lcd_ks0108_draw_line	25D8	169	361	13	.text

delay	20FD	14	20	4	.text
lcd_ks0108_write_data	2111	10	16	1	.text
lcd_ks0108_write_cmd	2121	10	16	7	.text
lcd_ks0108_read_data	2131	17	23	6	.text
lcd_ks0108_gotoxy	2148	29	41	8	.text
lcd_ks0108_display_onoff	2171	17	23	1	.text
lcd_ks0108_set_start_line	2188	E	14	3	.text
lcd_ks0108_dotxy	2196	3F	63	7	.text
lcd_ks0108_fill_display	21D5	3A	58	1	.text
lcd_ks0108_init	220F	31	49	1	.text
lcd_ks0108_select_font	2240	4	4	2	.text
lcd_ks0108_put_char	2244	36A	874	58	.text
lcd_ks0108_place_text	25AE	19	25	3	.text
lcd_ks0108_draw_line	25C7	169	361	13	.text

Funções

Exame do MAP file:



Interpolação de 8 bits

Utilizando int:

```
int interpola(int x, int x1, int x2, int y1,
int y2)
{
  if (x==x1) return y1; else
    if (x==x2) return y2; else
    return ((x%(x2-x1))*(y2-y1))/(x2-x1)+y1;

}
```

Interpolação de 8 bits

Utilizando int:

```
int interp                                    x2, int y1,
int y2)
{
   if (x==x
     if (x=
     retur                                    (x2-x1)+y1;

}
```

376 bytes FLASH
1188 ciclos

Interpolação de 8 bits

Utilizando char:

```
char interpola(char x, char x1, char x2, char y1, char
y2)
{
  if (x==x1) return y1; else if (x==x2) return y2;
  else
  {
    return ((x%(x2-x1))*((signed char)y2-
    (signed char)y1))/(x2-x1)+y1;
  }
}
```

Interpolação de 8 bits

Utilizando char:

```
char interp                                    , char y1, char
y2)
{
  if (x==x1                                    return y2;
  else
  {
    return                                     2-
    (signed
  }
}
```

357 bytes FLASH
1060 ciclos

Interpolação de 8 bits

Forçando uchar na operação %:

```
char interpola(char x, char x1, char x2, char y1, char
y2)
{
  if (x==x1) return y1; else if (x==x2) return y2;
  else
  {
    return (((uchar)x%(uchar)(x2-x1))*((signed
    char)y2-(signed char)y1))/(x2-x1)+y1;
  }
}
```

Interpolação de 8 bits

Forçando uchar na operação %:

```
char interp                                    , char y1, char
y2)
{
  if (x==x1                                    return y2;
  else
  {
    return                                     (signed
    char)y2                                    1;
  }
}
```

311 bytes FLASH
802 ciclos

Interpolação de 8 bits

Utilizando apenas uchar:

```
char interpola(char x, char x1, char x2, char y1, char y2)
{
  char aux;
  aux = x%(uchar)(x2-x1);
  if (x==x1) return y1; else if (x==x2) return y2; else
  {
    if (y2>y1) return (((uchar)x%(uchar)(x2-x1))*(uchar)(y2-
y1))/(x2-x1)+y1;
    else return ((uchar)((uchar)(x2-x1)-(uchar)x%(uchar)(x2-
x1))*(uchar)(y1-y2))/(x2-x1)+y2;
  }
}
```

Interpolação de 8 bits

Utilizando apenas uchar:

```
char interpo                                    r y1, char y2)
{
  char aux;
  aux = x%(uc
  if (x==x1)                                    rn y2; else
  {
    if (y2>y1                                   x1))*(uchar)(y2-
y1))/(x2-x1)+
    else retu                                   ar)x%(uchar)(x2-
x1))*(uchar)(y1-y2))/(x2-x1)+y2;
  }
}
```

292 bytes FLASH
628 ciclos

Interpolação de 8 bits

Variação de x constante:

```
char interpola(char x, char x1, char x2, char y1, char y2)
{
  char aux;
  aux = x%20;
  if (x==x1) return y1; else if (x==x2) return y2; else
  {
   if (y2>y1)return ((uchar)aux*(uchar)(y2-y1))/20 + y1;
   else return ((uchar)(20-aux)*(uchar)(y1-y2))/20 + y2;

  }
}
```

Interpolação de 8 bits

Variação de x constante:

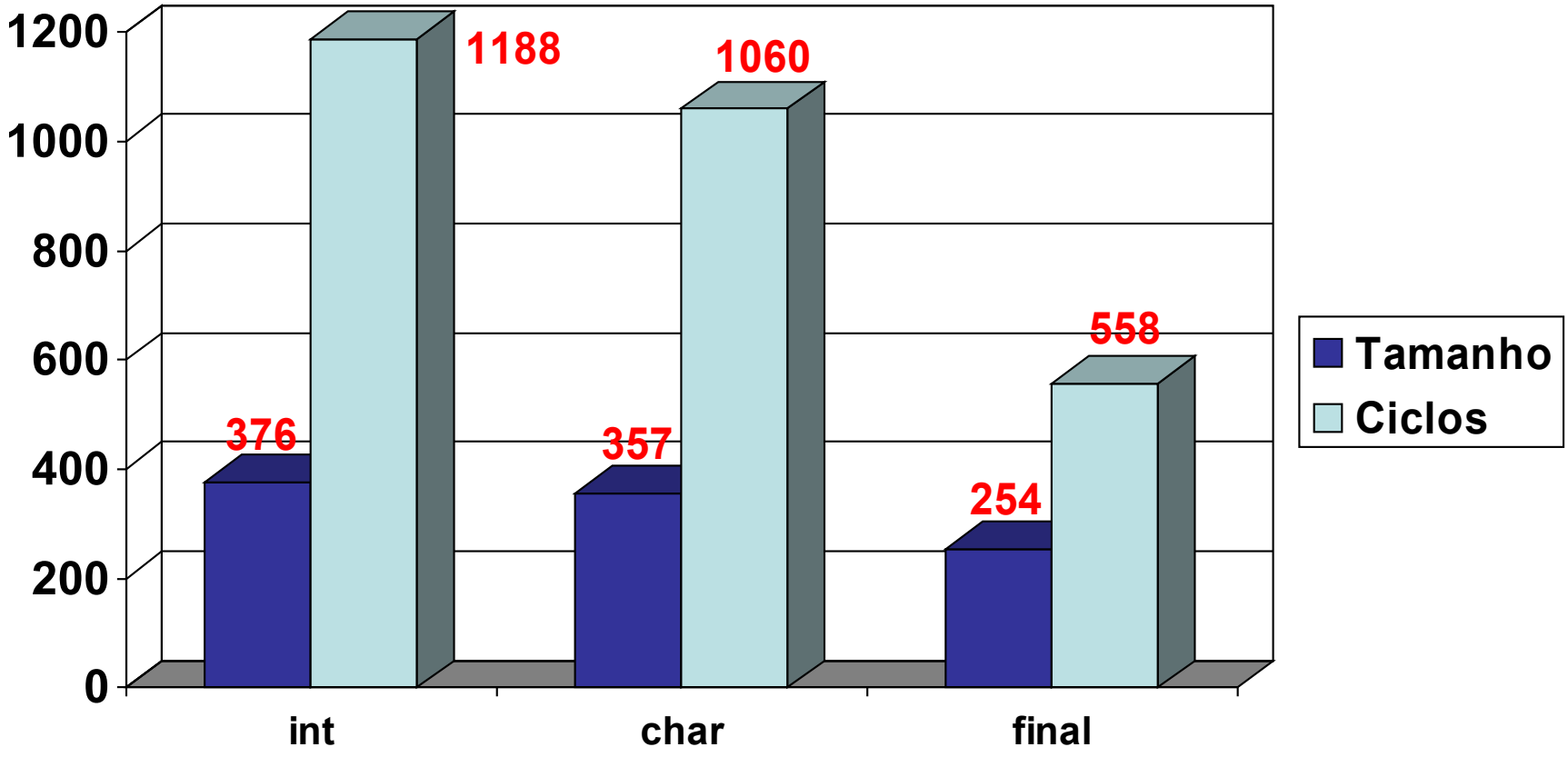
```
char interpo                                    r y1, char y2)
{
  char aux;
  aux = x%20;
  if (x==x1)                                    rn y2; else
  {
   if (y2>y1)                                   1))/20 + y1;
   else retu                                    2))/20 + y2;

  }
}
```

254 bytes FLASH
558 ciclos

Interpolação de 8 bits

Estatísticas de otimização:



Seleção de Tipos

Interpolação de 8 bits no Coldfire v1:

```
uchar interpola(uchar x, uchar x1, uchar x2,
uchar y1, uchar y2)
{
  if (x==x1) return y1; else
     if (x==x2) return y2; else
     return (((uchar)x%(uchar)(x2-
x1))*((signed char)y2-(signed      char)y1))/
(x2-x1)+y1;
}
```

Seleção de Tipos

Interpolação de 8 bits no Coldfire v1:

```
uchar inte                                  , uchar x2,
uchar y1,
{
   if (x==
      if (
      retu                                  -
x1))*((sig                                    char)y1))/
(x2-x1)+y1;
}
```

358 bytes FLASH
222 ciclos

Seleção de Tipos

Interpolação de 32 bits no Coldfire v1:

```
int interpola(int x, int x1, int x2, int y1,
int y2)
{
  if (x==x1) return y1; else
    if (x==x2) return y2; else
    return ((x%(x2-x1))*(y2-y1))/(x2-x1)+y1;

}
```

Seleção de Tipos

Interpolação de 32 bits no Coldfire v1:

```
int interp                                    x2, int y1,
int y2)
{
   if (x==x
     if (x=
     retur                                    (x2-x1)+y1;

}
```

340 bytes FLASH
156 ciclos

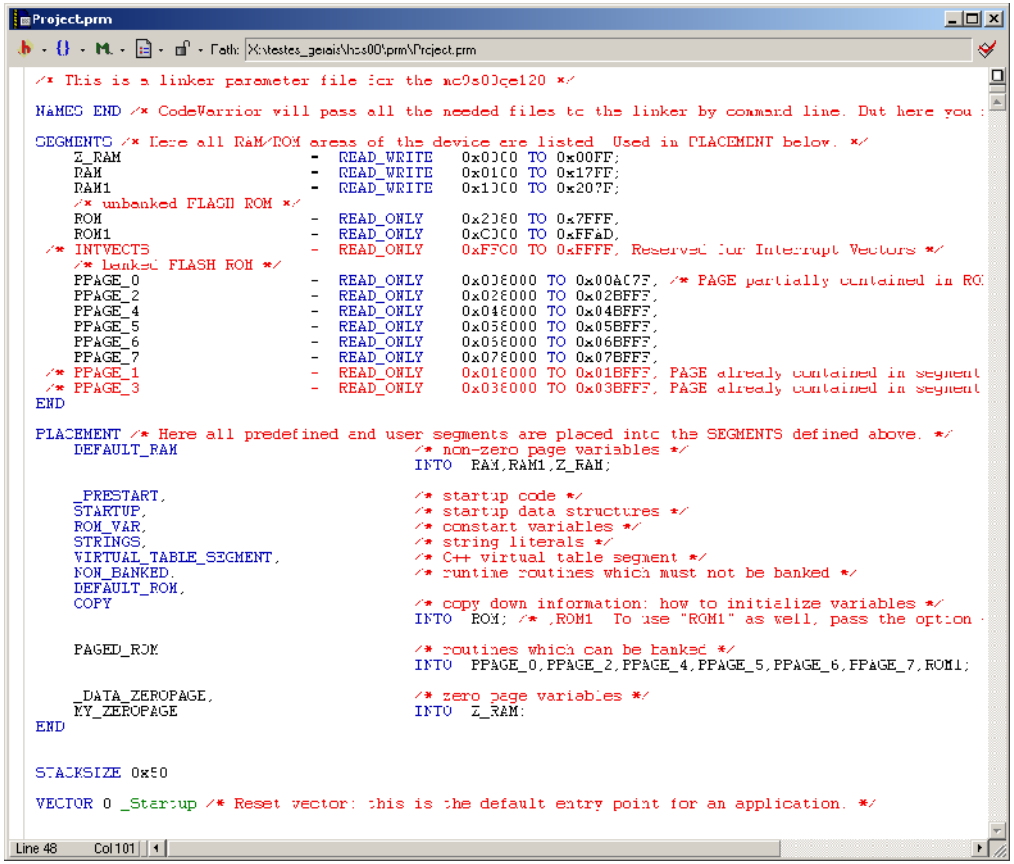
Configuração do Linker

HCS08:

- Utilização da página direta
- Otimização de acessos a memória
- Otimização da pilha

Configuração do Linker

Utilização da página direta nos HCS08:



Configuração do Linker

Utilização da página direta nos HCS08:

```
PLACEMENT /* Here all predefined and user segments are placed into
    DEFAULT_RAM                     /* non-zero page variables
                                    INTO  RAM,RAM1,Z_RAM;

    _PRESTART,                      /* startup code */
```

Project.prm

```
/* This is a linker parameter file for the mc9s08qe128 */

NAMES END /* CodeWarrior will pass all the needed files to the linker by command line. But here you

SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT below. */
    Z_RAM                   = READ_WRITE   0x0080 TO 0x00FF;
    RAM                     = READ_WRITE   0x0100 TO 0x17FF;
    RAM1                    = READ_WRITE   0x1D80 TO 0x207F;
    /* unbanked FLASH ROM */
    ROM                     = READ_ONLY    0x2080 TO 0x7FFF,
    ROM1                    = READ_ONLY    0xC000 TO 0xFFAD,
 /* INTVECTS                = READ_ONLY    0xFFC0 TO 0xFFFF, Reserved for Interrupt Vectors */
    /* banked FLASH ROM */
                              0x008000 TO 0x00AC7F, /* PAGE partially contained in RO
                              0x028000 TO 0x02BFFF,
                              0x048000 TO 0x04BFFF,
                              0x058000 TO 0x05BFFF,
                              0x068000 TO 0x06BFFF,
                              0x078000 TO 0x07BFFF,
                              0x018000 TO 0x01BFFF, PAGE already contained in segment
                              0x038000 TO 0x03BFFF, PAGE already contained in segment

PLACEMENT /* Here all predefined and user segments are placed into the SEGMENTS defined above. */
    DEFAULT_RAM                     /* non-zero page variables */
                                    INTO  RAM,RAM1,Z_RAM;

    _PRESTART,                      /* startup code */
    STARTUP,                        /* startup data structures */
    ROM_VAR,                        /* constant variables */
    STRINGS,                        /* string literals */
    VIRTUAL_TABLE_SEGMENT,          /* C++ virtual table segment */
    NON_BANKED,                     /* runtime routines which must not be banked */
    DEFAULT_ROM,
    COPY                            /* copy down information: how to initialize variables */
                                    INTO  ROM; /* ,ROM1  To use "ROM1" as well, pass the option

    PAGED_ROM                       /* routines which can be banked */
                                    INTO  PPAGE_0,PPAGE_2,PPAGE_4,PPAGE_5,PPAGE_6,PPAGE_7,ROM1;

    _DATA_ZEROPAGE,                 /* zero page variables */
    MY_ZEROPAGE                     INTO  Z_RAM;
END

STACKSIZE 0x50

VECTOR 0 _Startup /* Reset vector: this is the default entry point for an application. */
```

Line 48 Col 101

Explorando MAP files

- Estrutura e disposição das funções e dados na memória
- Verificação de utilização e referências a funções e dados
- Permite conhecer a ocupação da memória do MCU para fins de otimização de código

Explorando MAP files

Project.map

Path: X:\Digifuel\firmware\versao1\bin\Project.map

```
PROGRAM "X:\Digifuel\firmware\versao1\bin\Project.abs"

********************************************************************************************************
TARGET SECTION
--------------------------------------------------------------------------------------------------------
Processor    : Freescale HC08
Memory Model: SMALL
File Format  : ELF\DWARF 2.0
Linker       : SmartLinker V-5.0.34 Build 8120, Apr 30 2008

********************************************************************************************************
FILE SECTION
--------------------------------------------------------------------------------------------------------
digifuel_v02.c.o                        Model: SMALL,        Lang: ANSI-C
RTSHC08.C.o (ansiis.lib)                Model: SMALL,        Lang: ANSI-C
MC9S08QE128.C.o                         Model: SMALL,        Lang: ANSI-C
Start08.c.o                             Model: SMALL,        Lang: ANSI-C
doonstack.asm.o                         Model: SMALL,        Lang: Assembler

********************************************************************************************************
STARTUP SECTION
--------------------------------------------------------------------------------------------------------
Entry point: 0x20FB (_Startup)
_startupData is allocated at 0x2104 and uses 6 Bytes
extern struct _tagStartup {
  unsigned nofZeroOut     2
  _Range   pZeroOut       0x80     2
                          0x100     153
```

Line 1429 Col 46

Explorando MAP files

```
*****************************************************************************
SECTION-ALLOCATION SECTION
Section Name                          Size  Type      From        To        Segment
-----------------------------------------------------------------------------

.data                                   40   R/W       0x100       0x127    RAM
.init                                  132     R      0x2080      0x2103    ROM
.startData                              18     R      0x2104      0x2115    ROM
.rodata                                 68     R      0x2116      0x2159    ROM
.text                                 4165     R      0x215A      0x319E    ROM
.copy                                   46     R      0x31F8      0x3225    ROM
MY_ZEROPAGE                              2   R/W        0x80        0x81    Z_RAM
.abs_section_ffd2                        2     R      0xFFD2      0xFFD3    .absSeg0
...
.abs_section_1870                        2   N/I      0x1870      0x1871    .absSeg184
.bss                                     6   R/W       0x128       0x12D    RAM
.common                                107   R/W       0x12E       0x198    RAM
FLASH_ops                               89     R      0x319F      0x31F7    ROM
.stack                                 128   R/W       0x199       0x218    RAM
.vectSeg185_vect                         2     R      0xFFFE      0xFFFF    .vectSeg185

Summary of section sizes per section type:
READ_ONLY (R):          11B2 (dec:      4530)
READ_WRITE (R/W):        11B (dec:       283)
NO_INIT (N/I):            CF (dec:       207)
```

Explorando MAP files

```
********************************************************************************
SECTION-ALLOCATION SECTION
Section Name                          Size  Type      From        To        Segment
--------------------------------------------------------------------------------
.data                                   40   R/W       0x100       0x127   RAM
.init                                  132     R      0x2080      0x2103   ROM
.startData                              18     R      0x2104      0x2115   ROM
.rodata                                 68     R      0x2116      0x2159   ROM
.text                                 4165     R      0x215A      0x319E   ROM
.copy                                   46     R      0x31F8      0x3225   ROM
MY_ZEROPAGE                              2   R/W        0x80        0x81   Z_RAM
.abs_section_ffd2                        2     R      0xFFD2      0xFFD3   .absSeg0
...
.abs_section_1870                        2   N/I      0x1870      0x1871   .absSeg184
.bss                                     6   R/W       0x128       0x12D   RAM
.common                                107   R/W       0x12E       0x198   RAM
FLASH_ops                               89     R      0x319F      0x31F7   ROM
.stack                                 128   R/W       0x199       0x218   RAM
.vectSeg185_vect                         2     R      0xFFFE      0xFFFF   .vectSeg185

Summary of section sizes per section type:
READ_ONLY (R):          11B2 (dec:      4530)
READ_WRITE (R/W):        11B (dec:       283)
NO_INIT (N/I):            CF (dec:       207)
```

Explorando MAP files

```
**************************************************************************************
OBJECT-ALLOCATION SECTION
     Name                    Module                Addr    hSize    dSize     Ref    Section
--------------------------------------------------------------------------------------------
MODULE:                  -- digituel_v02.c.o --
- PROCEDURES:
     scirx_isr                                     215A       2B       43       1    .text
     write_sci_buffer                              2185       36       54       1    .text
     comm_state_machine                            21DD      1F3      499       1    .text
     co_tx                                         23AE       2B       43       1    .text
     flash_read                                    23D9        2       2        1    .text
     flash_write                                   23DB       82       130      1    .text
     flash_page_erase                              245D       2F       47       1    .text
     save_regs_to_flash                            243B      103       259      1    .text
     read_regs_from_flash                          253E       EA       234      1    .text
     save_password_to_flash                        2628        1       1        1    .text
     do_flash_operations                           2629       46       70       1    .text
     adc_state_machine                             266F       14F      335      23   .text
     rtc_isr                                       27BE       33       51       1    .text
     tpm1ch0_isr                                   27F1       10       16       1    .text
     tpm2ovf_isr                                   2801        C       12       1    .text
     kbi_isr                                       280D       4C       76       1    .text
     hw_init                                       2859       6E       110      1    .text
     get_ntc_table_index                           28C7       34       52       2    .text
     do_temperature_calculations                   28FB       22F      559      14   .text
     do_injection_calculations                     2B2A       2A6      678      10   .text
     update_regs                                   2DD0       33       51       1    .text
     standard_init                                 2E03       1F       31       1    .text
     main                                          2E22       30       48       1    .text
- VARIABLES:
     rtc_curves                                    2116       44       68      13    .rodata
     injection_time_table                           100        B       11       7    .data
     rpm_correction_table                           10B        B       11       7    .data
     air_temp_correction_table                      116        9        9       6    .data
     eng_temp_correction_table                      11F        9        9       6    .data
     checksum.1                                     128        1        1       6    .bss
     cmd.2                                          129        1        1       2    .bss
     reg.3                                          12A        1        1       8    .bss
     regdata.4                                      12B        1        1       7    .bss
     adc_seq.5                                      12C        1        1       9    .bss
     rtc_cnt.6                                      12D        1        1       3    .bss
     flags                                           80        2        2      46    MY_ZEROPAGE
      Vector 22                                     FFD2        2        2       0    .abs sectic
     _Vector_24                                     FFCE        2        2       0    .abs_sectic
     _Vector_4                                      FFF6        2        2       0    .abs_sectic
     _Vector_11                                     FFE8        2        2       0    .abs_sectic
     _Vector_18                                     FFDA        2        2       0    .abs_sectic
     comm_rx                                         12E       12       18       1    .common
     comm_tx                                         140       12       18       1    .common
     comm_rx_state                                   152        1        1       1    .common
     comm_timeout                                    153        1        1       1    .common
     adc_samples                                     154        E       14      21    .common
     adc_avg                                         162        7        7      23    .common
     regs                                            169       20       32      74    .common
     current_injection_time                          189        2        2       5    .common
     rpm_period                                      18B        2        2       4    .common
     rpm_delay_filter                                18D        2        2       1    .common
     last_tps                                        18F        1        1       1    .common
     correction_pulses                               190        1        1       4    .common
     quick_increment                                 191        1        1       3    .common
```

Explorando MAP files

Com endereçamento extendido:

```
  8:      case 0: // canal ADP17 - tensão da bateria
  9:        adc_samples._vbat = adc_samples._vbat + ADCR - adc_avg.current_vbat;
0020 5500     [4]             LDHX  _ADCR
0022 89       [2]             PSHX
0023 8b       [2]             PSHH
0024 320000   [5]             LDHX  adc_samples
0027 cd0144   [6]             JSR   L144 ;abs = 0144
002a ce0000   [4]             LDX   adc_avg
002d cd012a   [6]             JSR   L12A ;abs = 012a
0030 88       [3]             PULX
0031 960000   [5]             STHX  adc_samples
 10:        adc_avg.current_vbat = adc_samples._vbat / NUM_MEDIA_VBAT;
0034 320000   [5]             LDHX  adc_samples
0037 cd013b   [6]             JSR   L13B ;abs = 013b
003a cf0000   [4]             STX   adc_avg
 11:        ADCSC1 = ADCH16;  // configura o próximo canal do ADC = ADP16
003d 6e1000   [4]             MOV   #16,_ADCSC1
 12:        adc_seq++;
0040 450000   [3]             LDHX  @adc_seq
0043 7c       [4]             INC   ,X
 13:        break;
```

Explorando MAP files

Modificação para endereçamento direto:

```
#pragma DATA_SEG __DIRECT_SEG MY_ZEROPAGE
volatile near struct
{
  unsigned int _vbat;       // amostras da tensão da bateria
  unsigned int _air_temp;   // amostras da temperatura do ar
  unsigned int _eng_temp;   // amostras da temperatura do motor
  unsigned int _tps;        // amostras da posição do TPS
  unsigned int _lambda;     // amostras da sonda lambda
  unsigned int _map;        // amostras do MAP
  unsigned int _in2;        // amostras da entrada IN2
} adc_samples;
volatile near struct
{
  unsigned char current_vbat;      // média da tensão da bateria
  unsigned char current_air_temp;  // média da temperatura do ar
  unsigned char current_eng_temp;  // média da temperatura do motor
  unsigned char current_tps;       // média da posição do TPS
  unsigned char current_lambda;    // média da sonda lambda
  unsigned char current_map;       // média do MAP
  unsigned char current_in2;       // média da IN2
} adc_avg;
unsigned near char regs[MAX_REGS];
#pragma DATA_SEG DEFAULT

unsigned int current_injection_time;
unsigned int tpmlinterval;
unsigned int rpm_period, rpm_delay_filter;
unsigned char last_tps, correction_pulses, quick_increment;
unsigned char air_temp_correction, eng_temp_correction;
unsigned int r_ntc_air, r_ntc_engine;
unsigned char eng_starting;
```

Explorando MAP files

Com endereçamento direto:

```
  8:      case 0: // canal ADP17 - tensão da bateria
  9:        adc_samples._vbat = adc_samples._vbat + ADCR - adc_avg.current_vbat;
0017 5500     [4]             LDHX  _ADCR
0019 89       [2]             PSHX
001a 8b       [2]             PSHH
001b 5500     [4]             LDHX  adc_samples
001d cd0115   [6]             JSR   L115 ;abs = 0115
0020 be00     [3]             LDX   adc_avg
0022 cd00fb   [6]             JSR   LFB ;abs = 00fb
0025 88       [3]             PULX
0026 3500     [5]             STHX  adc_samples
 10:        adc_avg.current_vbat = adc_samples._vbat / NUM_MEDIA_VBAT;
0028 5500     [4]             LDHX  adc_samples
002a cd010c   [6]             JSR   L10C ;abs = 010c
002d bf00     [3]             STX   adc_avg
 11:        ADCSC1 = ADCH16;  // configura o próximo canal do ADC = ADP16
002f 6e1000   [4]             MOV   #16,_ADCSC1
 12:        adc_seq++;
0032 450000   [3]             LDHX  @adc_seq
0035 7c       [4]             INC   ,X
 13:        break;
```

-1 byte/
-1 ciclo

Explorando MAP files

Com endereçamento extendido:

```
Summary of section sizes per section type:
READ_ONLY (R):          1149 (dec:     4425)
READ_WRITE (R/W):        11B (dec:      283)
NO_INIT (N/I):            CF (dec:      207)
```

-105 bytes

```
adc_state_machine                   2653    120    288    12
rtc_isr                             2773     32     50     1
tpm1ch0_isr                         27A5     10     16     1
tpm2ovf_isr                         27B5      C     12     1
kbi_isr                             27C1     4A     74     1
hw_init                             280B     6E    110     1
get_ntc_table_index                 2879     34     52     2
do_temperature_calculations         28AD    21B    539    14
do_injection_calculations           2AC8    287    647    10
```

-47 bytes

-20 bytes

-31 bytes

Pilha

Falhas ocasionadas por overflow da pilha são difíceis de ser diagnosticadas e podem provocar diversos efeitos colaterais, desde funcionamento errático esporádico até crashes completos.

Pilha

Lembre-se de que as variáveis locais que não podem ser armazenadas em registradores da CPU são mantidas na pilha durante a execução da função!

Pilha

Como dimensionar corretamente o tamanho da pilha?

Pilha

Call Graph:

```
****************************************
*                                      *
*             CALL GRAPH               *
*                                      *
****************************************

->Sub-tree of type: Interrupt function tree that does not make
                  : indirect calls
                              CSTACK
  | Stack used (prev) :  00000000
01  trata_timera
    | Stack used (prev) :  00000000
    | + function block  :  00000004
<-Sub-tree of type: Interrupt function tree that does not make
                  : indirect calls
  | Stack used          :  00000004
->Sub-tree of type: Interrupt function tree that does not make
                  : indirect calls
                              CSTACK
  | Stack used (prev) :  00000004
01  trata_compa
    | Stack used (prev) :  00000004
    | + function block  :  00000004
<-Sub-tree of type: Interrupt function tree that does not make
                  : indirect calls
  | Stack used          :  00000008
->Sub-tree of type: Function tree
                              CSTACK
  | Stack used (prev) :  00000008
```

Pilha

Call Graph:

```
02    transmite
      | Stack used (prev) :  0000000A
      | + function block   :  0000000A
03      dcolmhz
        | Stack used (prev) :  00000008
        | + function block   :  00000002
02    inicializa
      | Stack used (prev) :  0000000A
      | + function block   :  00000002
01  main
    | Stack used (prev) :  00000014
    | + function block   :  00000002
<-Sub-tree of type: Function tree
  | Stack used          :  00000016
->Sub-tree of type: Function tree
                          CSTACK
  | Stack used (prev) :  00000016
01  exit
    | Stack used (prev) :  00000016
    | + function block   :  00000002
<-Sub-tree of type: Function tree
  | Stack used          :  00000018
->Sub-tree of type: Function tree
                          CSTACK
  | Stack used (prev) :  00000018
01  __data16_memzero
    | Stack used (prev) :  00000018
    | + function block   :  00000002
<-Sub-tree of type: Function tree
  | Stack used          :  0000001A
```

```
<-Sub-tree of type: Function tree
  | Stack used          :  0000001A
```

Pilha

O MAP file não possui Call Graph ?

Calcule a utilização da pilha utilizando as árvores de dependência presentes no MAP file (juntamente com a observação do código gerado pelo compilador)!

Pilha

Source

X:\Digifuel\firmware\versao1\Sources\digifuel_v02.c Line: 234

```
void do_injection_calculations(void)
{
  int injection_time_temp, aux;
  unsigned char table_index, remainder, rpm_correction;
  // recalcula o tempo atual de injeção

  // Calcula a posição percentual do TPS (corrigido de acordo com os valores de TPS mi
  if (adc_avg.current_tps<=tps_min) tps = 0; else
```

7 bytes!

13 bytes!

Assembly

do_injection_calculations

```
2B21 ADC    #0x00
2B23 RTS
2B24 AIS    #-13
2B26 LDA    regs[13]
2B29 CMP    adc_avg.current_tps
2B2C BCS    do_injection_calculations+13
2B2E CLRA
2B2F BRA    do_injection_calculations+23
2B31 LDA    regs[14]
2B34 CMP    adc_avg.current_tps
```

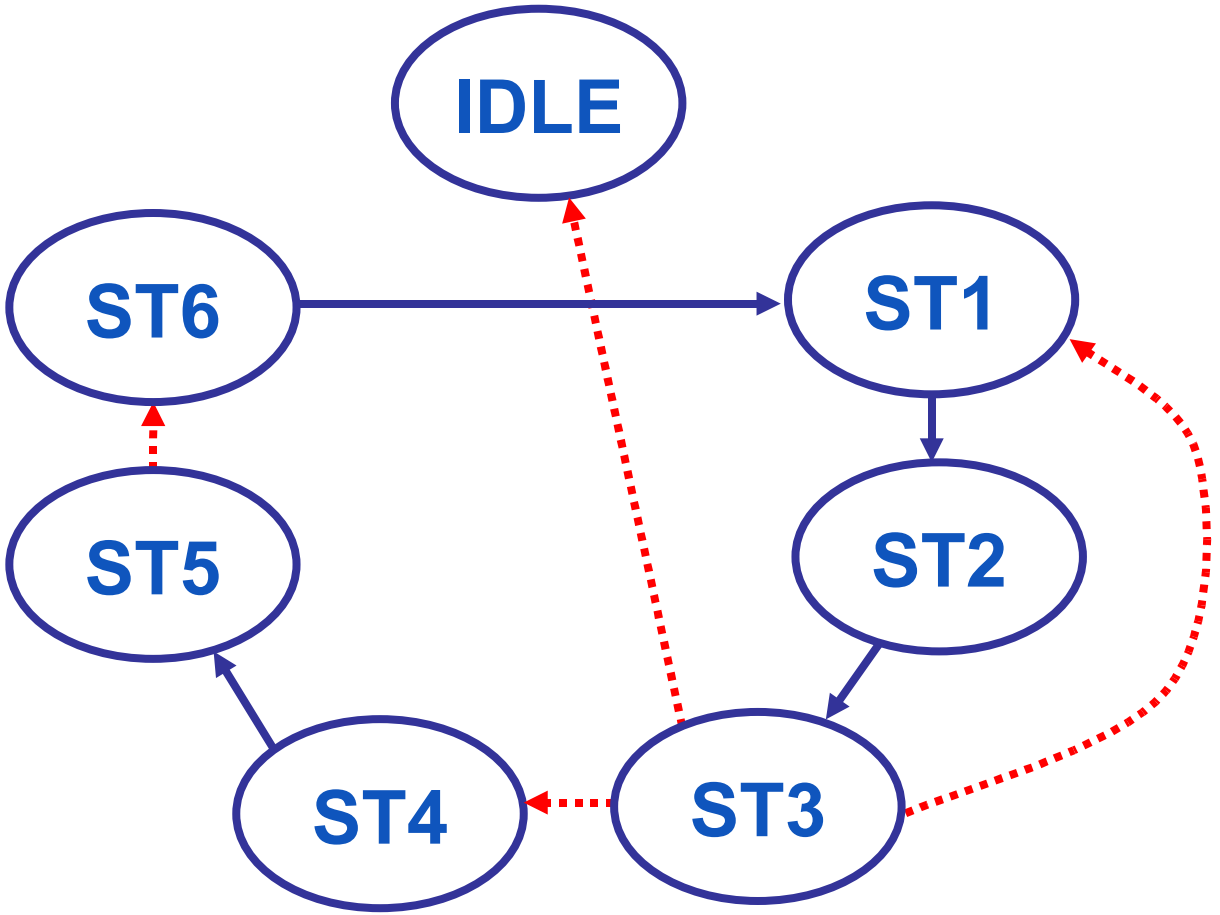
Pilha

Alocação de pilha:

HCS08:	MSP430:	ARM/CM3:
AIS #-X	SUB.W #X,SP	SUB SP,SP,#X

Coldfire:	MIPS16:	MIPS32:
LEA –X(A7),A7	SAVE #X	ADDIU SP,SP,#-X

Máquinas de Estado



Máquinas de Estado

```
void fsm1(unsigned int new_state)
{
  static unsigned int fsm_state;
  if (new_state) fsm_state = new_state;
  if (fsm_state==FSM1_ST1)
  { // estado 1
    va = 10;
    fsm_state = FSM1_ST2;
  } else if (fsm_state==FSM1_ST2)
  { // estado 2
    va = 20;
    fsm_state = FSM1_ST3;
  } else if (fsm_state==FSM1_ST3)
  { // estado 3
    if (va<10) fsm_state = FSM1_ST1; else
      if (va==255) fsm_state = FSM1_IDLE; else
        fsm_state = FSM1_ST4;
  } else if (fsm_state==FSM1_ST4)
  { // estado 4
    va--;
    fsm_state = FSM1_ST5;
  } else if (fsm_state==FSM1_ST5)
  { // estado 5
    if (!va) fsm_state = FSM1_ST6;
  } else if (fsm_state==FSM1_ST6)
  { // estado 6
    va = 100;
    fsm_state = FSM1_ST1;
  }
}
```

```
  if (new_state) fsm_state = new_state;
?cstart_end:
fsm1:
 001118    930C                tst.w   R12
 00111A    2402                jeq     0x1120
  if (new_state) fsm_state = new_state;
 00111C    4C82 0200           mov.w   R12,&fsm_state
  if (fsm_state==FSM1_ST1)
 001120    421F 0200           mov.w   &fsm_state,R15
 001124    931F                cmp.w   #0x1,R15
 001126    2006                jne     0x1134
    va = 10;
 001128    40F2 000A 0208      mov.b   #0xA,&va
    fsm_state = FSM1_ST2;
 00112E    43A2 0200           mov.w   #0x2,&fsm_state
 001132    4130                ret
  } else if (fsm_state==FSM1_ST2)
 001134    932F                cmp.w   #0x2,R15
 001136    2007                jne     0x1146
    va = 20;
 001138    40F2 0014 0208      mov.b   #0x14,&va
    fsm_state = FSM1_ST3;
 00113E    40B2 0003 0200      mov.w   #0x3,&fsm_state
 001144    4130                ret
  } else if (fsm_state==FSM1_ST3)
 001146    903F 0003           cmp.w   #0x3,R15
 00114A    2010                jne     0x116C
    if (va<10) fsm_state = FSM1_ST1; else
 00114C    90F2 000A 0208      cmp.b   #0xA,&va
 001152    2C03                jc      0x115A
    if (va<10) fsm_state = FSM1_ST1; else
 001154    4392 0200           mov.w   #0x1,&fsm_state
 001158    4130                ret
```

Máquinas de Estado

```
void fsm1(unsigned int new_state)
{
  static unsigned int fsm_state;
  if (new_state) fsm_state = new_state;
  if (fsm_state==FSM1_ST1)
  { // estado 1
    va = 10;
    fsm_state = F
  } else if (fsm_
  { // estado 2
    va = 20;
    fsm_state = F
  } else if (fsm_
  { // estado 3
    if (va<10) fs
      if (va==255
        fsm_state
  } else if (fsm_
  { // estado 4
    va--;
    fsm_state = F
  } else if (fsm_
  { // estado 5
    if (!va) fsm_state = FSM1_ST6;
  } else if (fsm_state==FSM1_ST6)
  { // estado 6
    va = 100;
    fsm_state = FSM1_ST1;
  }
}
```

138 bytes FLASH

```
  if (new_state) fsm_state = new_state;
?cstart_end:
fsm1:
 001118    930C                tst.w   R12
 00111A    2402                jeq     0x1120
  if (new_state) fsm_state = new_state;
 00111C    4C82 0200           mov.w   R12,&fsm_state
                                       &fsm_state,R15
                                       #0x1,R15
                                       0x1134
                                       #0xA,&va
                                       #0x2,&fsm_state
                                       #0x2,R15
                                       0x1146
                                       #0x14,&va
                                       #0x3,&fsm_state
                                       #0x3,R15
 00114A    2010                jne     0x116C
   if (va<10) fsm_state = FSM1_ST1; else
 00114C    90F2 000A 0208      cmp.b   #0xA,&va
 001152    2C03                jc      0x115A
   if (va<10) fsm_state = FSM1_ST1; else
 001154    4392 0200           mov.w   #0x1,&fsm_state
 001158    4130                ret
```

Máquinas de Estado

```
#define FSM1_IDLE 0
#define FSM1_ST1  1
#define FSM1_ST2  2
#define FSM1_ST3  3
#define FSM1_ST4  4
#define FSM1_ST5  5
#define FSM1_ST6  6
```

Máquinas de Estado

```
void fsm2(unsigned int new_state)
{
  static unsigned int fsm_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM1_IDLE: // não faz nada
                  break;
  case FSM1_ST1:  // estado 1
                  va = 10; fsm_state = FSM1_ST2;
                  break;
  case FSM1_ST2:  // estado 2
                  va = 20; fsm_state = FSM1_ST3;
                  break;
  case FSM1_ST3:  // estado 3
                  if (va<10) fsm_state = FSM1_ST1; else
                    if (va==255) fsm_state = FSM1_IDLE; else
                      fsm_state = FSM1_ST4;
                  break;
  case FSM1_ST4:  // estado 4
                  va--; fsm_state = FSM1_ST5;
                  break;
  case FSM1_ST5:  // estado 5
                  if (!va) fsm_state = FSM1_ST6;
                  break;
  case FSM1_ST6:  // estado 6
                  va = 100; fsm_state = FSM1_ST1;

  }
}
```

```
  if (new_state) fsm_state = new_state;
fsm2:
 0011A2   930C              tst.w   R12
 0011A4   2402              jeq     0x11AA
  if (new_state) fsm_state = new_state;
 0011A6   4C82 0202         mov.w   R12,&fsm_state
  switch (fsm_state)
 0011AA   421F 0202         mov.w   &fsm_state,R15
 0011AE   831F              dec.w   R15
 0011B0   240B              jeq     0x11C8
 0011B2   831F              dec.w   R15
 0011B4   240F              jeq     0x11D4
 0011B6   831F              dec.w   R15
 0011B8   2414              jeq     0x11E2
 0011BA   831F              dec.w   R15
 0011BC   2422              jeq     0x1202
 0011BE   831F              dec.w   R15
 0011C0   2426              jeq     0x120E
 0011C2   831F              dec.w   R15
 0011C4   242B              jeq     0x121C
 0011C6   4130              ret
                va = 10;
 0011C8   40F2 000A 0208    mov.b   #0xA,&va
                fsm_state = FSM1_ST2;
 0011CE   43A2 0202         mov.w   #0x2,&fsm_sta
 0011D2   4130              ret
                va = 20;
 0011D4   40F2 0014 0208    mov.b   #0x14,&va
                fsm_state = FSM1_ST3;
 0011DA   40B2 0003 0202    mov.w   #0x3,&fsm_sta
 0011E0   4130              ret
```

Máquinas de Estado

```
void fsm2(unsigned int new_state)
{
  static unsigned int fsm_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM1_IDLE: // não faz nada
                  break;
  case FSM1_ST1:

  case FSM1_ST2:

  case FSM1_ST3:

  case FSM1_ST4:

  case FSM1_ST5:
                  break;
  case FSM1_ST6:  // estado 6
                  va = 100; fsm_state = FSM1_ST1;

  }
}
```

```
  if (new_state) fsm_state = new_state;
fsm2:
 0011A2    930C                tst.w   R12
 0011A4    2402                jeq     0x11AA
  if (new_state) fsm_state = new_state;
 0011A6    4C82 0202           mov.w   R12,&fsm_state
  switch (fsm_state)
 0011AA    421F 0202           mov.w   &fsm_state,R15
                               dec.w   R15
                               jeq     0x11C8
                               dec.w   R15
                               jeq     0x11D4
                               dec.w   R15
                               jeq     0x11E2
                               dec.w   R15
                               jeq     0x1202
                               dec.w   R15
                               jeq     0x120E
                               dec.w   R15
                               jeq     0x121C
                               ret
                                  ;
                               mov.b   #0xA,&va
                            te = FSM1_ST2;
                               mov.w   #0x2,&fsm_sta
                               ret
                            va = 20;
 0011D4    40F2 0014 0208      mov.b   #0x14,&va
                            fsm_state = FSM1_ST3;
 0011DA    40B2 0003 0202      mov.w   #0x3,&fsm_sta
 0011E0    4130                ret
```

134 bytes FLASH

Máquinas de Estado

```
unsigned int fsm3(unsigned int new_state)
{
  static unsigned int fsm_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM1_IDLE: // não faz nada
                  break;
  case FSM1_ST1:  // estado 1
                  va = 10; fsm_state = FSM1_ST2;
                  break;
  case FSM1_ST2:  // estado 2
                  va = 20; fsm_state = FSM1_ST3;
                  break;
  case FSM1_ST3:  // estado 3
                  if (va<10) fsm_state = FSM1_ST1; else
                    if (va==255) fsm_state = FSM1_IDLE;
                      else fsm_state = FSM1_ST4;
                  break;
  case FSM1_ST4:  // estado 4
                  va--; fsm_state = FSM1_ST5;
                  break;
  case FSM1_ST5:  // estado 5
                  if (!va) fsm_state = FSM1_ST6;
                  break;
  case FSM1_ST6:  // estado 6
                  va = 100; fsm_state = FSM1_ST1;

  }
  return fsm_state;
}
```

```
  if (new_state) fsm_state = new_state;
fsm3:
 001228    930C                tst.w   R12
 00122A    2402                jeq     0x1230
  if (new_state) fsm_state = new_state;
 00122C    4C82 0204           mov.w   R12,&fsm_state
  switch (fsm_state)
 001230    421F 0204           mov.w   &fsm_state,R15
 001234    831F                dec.w   R15
 001236    240B                jeq     0x124E
 001238    831F                dec.w   R15
 00123A    240F                jeq     0x125A
 00123C    831F                dec.w   R15
 00123E    2414                jeq     0x1268
 001240    831F                dec.w   R15
 001242    241F                jeq     0x1282
 001244    831F                dec.w   R15
 001246    2423                jeq     0x128E
 001248    831F                dec.w   R15
 00124A    2428                jeq     0x129C
 00124C    3C2C                jmp     0x12A6
                va = 10;
 00124E    40F2 000A 0208      mov.b   #0xA,&va
                fsm_state = FSM1_ST2;
 001254    43A2 0204           mov.w   #0x2,&fsm_state
 001258    3C26                jmp     0x12A6

  return fsm_state;
 0012A6    421C 0204           mov.w   &fsm_state,R12
 0012AA    4130                ret
```

Máquinas de Estado

```
unsigned int fsm3(unsigned int new_state)
{
  static unsigned int fsm_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM1_IDLE: // não faz nada
                  break;
  case FSM1_ST1:  // estado 1
                  va = 10; fsm_state = FSM1_S
                  break;
  case FSM1_ST2:  // estado 2
                  va = 20; fsm_state = FSM1_ST3;

  case FSM1_ST3:




  case FSM1_ST4:


  case FSM1_ST5:


  case FSM1_ST6:


  }
  return fsm_state;
}
```

```
                  if (va<10) fsm_state = FSM1_ST1; else
001268    90F2 000A 0208      cmp.b   #0xA,&va
00126E    2819                jnc     0x12A2
                  if (va==255) fsm_state = FSM1_IDLE;
001270    93F2 0208           cmp.b   #0xFF,&va
001274    2402                jeq     0x127A
001276    422F                mov.w   #0x4,R15
001278    3C01                jmp     0x127C
00127A    430F                clr.w   R15
00127C    4F82 0204           mov.w   R15,&fsm_state
                  if (va==255) fsm_state = FSM1_IDLE;
001280    3C12                jmp     0x12A6
```

```
001238    831F                dec.w   R15
                              eq      0x125A
                              ec.w    R15
                              eq      0x1268
                              ec.w    R15
                              eq      0x1282
                              ec.w    R15
                              eq      0x128E
                              ec.w    R15
                              eq      0x129C
                              mp      0x12A6

                              ov.b    #0xA,&va
                               FSM1_ST2;
                              ov.w    #0x2,&fsm_state
                              mp      0x12A6
```

132 bytes FLASH

Máquinas de Estado

```
unsigned int fsm4(unsigned int new_state)
{
  static unsigned int fsm_state;
  unsigned int temp_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM1_IDLE: // não faz nada
                  break;
  case FSM1_ST1:  // estado 1
                  va = 10; temp_state = FSM1_ST2; break;
  case FSM1_ST2:  // estado 2
                  va = 20; temp_state = FSM1_ST3; break;
  case FSM1_ST3:  // estado 3
                  if (va<10) temp_state = FSM1_ST1; else
                    if (va==255) temp_state = FSM1_IDLE;
                      else temp_state = FSM1_ST4;
                  break;
  case FSM1_ST4:  // estado 4
                  va--; temp_state = FSM1_ST5;
                  break;
  case FSM1_ST5:  // estado 5
                  if (!va) temp_state = FSM1_ST6;
                  break;
  case FSM1_ST6:  // estado 6
                  va = 100; temp_state = FSM1_ST1;

  }
  fsm_state = temp_state;
  return fsm_state;
}
```

```
  if (new_state) fsm_state = new_state;
fsm4:
 0012AC    930C              tst.w   R12
 0012AE    2402              jeq     0x12B4
  if (new_state) fsm_state = new_state;
 0012B0    4C82 0206         mov.w   R12,&fsm_state
  switch (fsm_state)
 0012B4    421E 0206         mov.w   &fsm_state,R14
 0012B8    831E              dec.w   R14
 0012BA    240B              jeq     0x12D2
 0012BC    831E              dec.w   R14
 0012BE    240E              jeq     0x12DC
 0012C0    831E              dec.w   R14
 0012C2    2412              jeq     0x12E8
 0012C4    831E              dec.w   R14
 0012C6    241B              jeq     0x12FE
 0012C8    831E              dec.w   R14
 0012CA    241E              jeq     0x1308
 0012CC    831E              dec.w   R14
 0012CE    2422              jeq     0x1314
 0012D0    3C25              jmp     0x131C
                va = 10;
 0012D2    40F2 000A 0208    mov.b   #0xA,&va
                temp state = FSM1 ST2;
 0012D8    432F              mov.w   #0x2,R15
 0012DA    3C20              jmp     0x131C

  fsm state = temp state;
 00131C    4F82 0206         mov.w   R15,&fsm_state
  return fsm state;
 001320    4F0C              mov.w   R15,R12
 001322    4130              ret

                fsm state = FSM1 ST2;
 001254    43A2 0204         mov.w   #0x2,&fsm_state
 001258    3C26              jmp     0x12A6
```

Máquinas de Estado

```
unsigned int fsm4(unsigned int new_state)
{
  static unsigned int fsm_state;
  unsigned int temp_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM1_IDLE: // não faz nada
                  k
  case FSM1_ST1:  /
                  v
  case FSM1_ST2:  /
                  v
  case FSM1_ST3:  /
                  i
                  k
  case FSM1_ST4:  /
                  v
                  k
  case FSM1_ST5:  /
                  i
                  break;
  case FSM1_ST6:  // estado 6
                  va = 100; temp_state = FSM1_ST1;

  }
  fsm_state = temp_state;
  return fsm_state;
}
```

```
  if (new_state) fsm_state = new_state;
fsm4:
 0012AC    930C              tst.w   R12
 0012AE    2402              jeq     0x12B4
  if (new_state) fsm_state = new_state;
                              v.w   R12,&fsm_state

                              v.w   &fsm_state,R14
                              c.w   R14
                              q     0x12D2
                              c.w   R14
                              q     0x12DC
                              c.w   R14
                              q     0x12E8
                              c.w   R14
                              q     0x12FE
                              c.w   R14
                              q     0x1308
                              c.w   R14
                              q     0x1314
                              p     0x131C

                              v.b   #0xA,&va
                  temp_state = FSM1_ST2;
 0012D8    432F              mov.w   #0x2,R15
 0012DA    3C20              jmp     0x131C
```

120 bytes FLASH

Máquinas de Estado

Enumerações

```
enum efsm
{
  FSM2_IDLE, FSM2_ST1, FSM2_ST2, FSM2_ST3,
  FSM2_ST4, FSM2_ST5, FSM2_ST6
};
```

Máquinas de Estado

```
enum efsm fsm5(enum efsm new_state)
{
  static enum efsm2 fsm_state;
  unsigned int temp_state;
  if (new_state) fsm_state = new_state;
  switch (fsm_state)
  {
  case FSM2_IDLE: // não faz nada
                  break;
  case FSM2_ST1:  // estado 1
                  va = 10; temp_state = FSM2_ST2; break;
  case FSM2_ST2:  // estado 2
                  va = 20; temp_state = FSM2_ST3; break;
  case FSM2_ST3:  // estado 3
                  if (va<10) temp_state = FSM2_ST1; else
                    if (va==255) temp_state = FSM2_IDLE; else
                      temp_state = FSM2_ST4;
                  break;
  case FSM2_ST4:  // estado 4
                  va--; temp_state = FSM2_ST5;
                  break;
  case FSM2_ST5:  // estado 5
                  if (!va) temp_state = FSM2_ST6;
                  break;
  case FSM2_ST6:  // estado 6
                  va = 100; temp_state = FSM2_ST1;

  }
  fsm_state = temp_state;
  return fsm_state;
}
```

Máquinas de Estado

```
enum efsm fsm5(enum efsm new_state)
{
  }
  fsm_state = temp_state;
  return fsm_state;
}
```

Lembre-se de que o tipo enum é normalmente um signed int!

Em algumas plataformas de 8 bits pode haver um impacto negativo no uso de enumerações!

Operandos e Operações

HCS08 - separação de operações:

```
if (aux++)   ...

LDA aux
TAX
INCA
STA aux
TSTX
BEQ falso
...
```

```
if (aux)   ...
aux++;

TST aux
BEQ falso
...
INC aux
```

Operandos e Operações

HCS08 - separação de operações:

```
if (aux++)   ...
```

9 bytes FLASH
24 ciclos

```
...
```

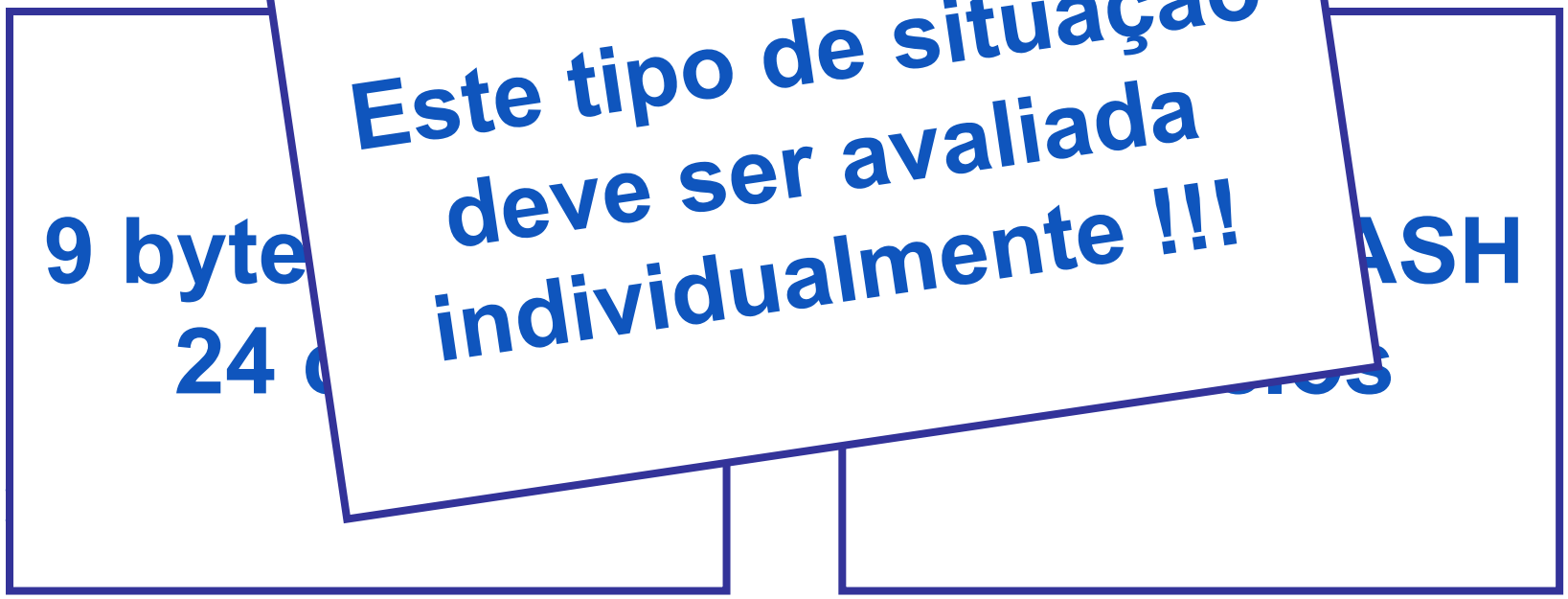
```
if (aux)   ...
aux++;
```

6 bytes FLASH
24 ciclos

Operandos e Operações

HCS08 - separação de operações:

```
if (aux++)   ...
```



Este tipo de situação deve ser avaliada individualmente !!!

```
...
```

Operandos e Operações

ulwos_kernel_hcs08_v04.c

Path: Z:\ULWOS\ULWOS2\Sources\ulwos_kernel_hcs08_v04.c

```
#pragma NO_FRAME
#pragma NO_ENTRY
#pragma NO_EXIT
#pragma NO_RETURN
void task_switcher(void)
{
  save_context();
  RTCSC_RTIF = 1; // apaga flag de interrupção do RTC
  if (ULWOS_CURRENT_TASK<0) ULWOS_CURRENT_TASK=0; else
  {
    if (ULWOS_CURRENT_TASK<(ULWOS_IDLE_TASK))
    {
      task_reg_SP[ULWOS_CURRENT_TASK] = os_data.temp_SP;
      task_reg_H[ULWOS_CURRENT_TASK] = os_data.temp_HX>>8;
      ULWOS_CURRENT_TASK++;
    }
    else
    {
      ULWOS_CURRENT_TASK = os_data.next_task;
    }
  }
  if (ULWOS_CURRENT_TASK>=(ULWOS_IDLE_TASK)) ULWOS_CURRENT_TASK = 0;
  if (task_status[ULWOS_CURRENT_TASK]==TASK_DELAY)
  {
    if (task_timer[ULWOS_CURRENT_TASK])
    {
      task_timer[ULWOS_CURRENT_TASK]--;
      os_data.next_task = ULWOS_CURRENT_TASK+1;
      ULWOS_CURRENT_TASK = ULWOS_IDLE_TASK;
    } else
    {
      task_status[ULWOS_CURRENT_TASK]=TASK_READY;
    }
  }
```

Line 64 Col 1

Operandos e Operações

`if (CURRENT_TASK<(IDLE_TASK))`

```
 84     52:    if (ULWOS_CURRENT_TASK<0) ULWOS_CURRENT_TASK=0; else
 85  000c 3d08     [4]             TST   os_data:8
 86  000e 9004     [3]             BGE   L14 ;abs = 0014
 87  0010 3f08     [5]             CLR   os_data:8
 88  0012 2028     [3]             BRA   L3C ;abs = 003c
 89  0014          L14:
 90  0014 cd00b4   [6]             JSR   LB4 ;abs = 00b4
 91     53:    {
 92     54:
 93     55:      if (ULWOS_CURRENT_TASK<(ULWOS_IDLE_TASK))
 94  0017 9320     [3]             BLE   L39 ;abs = 0039
 95     56:      {
 96     57:        task_reg_SP[ULWOS_CURRENT_TASK] = os_data.temp_S
 97  0019 be08     [3]             LDX   os_data:8
 98  001b 58       [1]             LSLX
 99  001c 8c       [1]             CLRH
100  001d b600     [3]             LDA   os_data
101  001f 87       [2]             PSHA
102  0020 b601     [3]             LDA   os_data:1
103  0022 87       [2]             PSHA
104  0023 9ee602   [4]             LDA   2,SP
105  0026 d70000   [4]             STA   @task_reg_SP,X
106  0029 86       [3]             PULA
107  002a d70001   [4]             STA   @task_reg_SP:1,X
108     58:      task_reg_H[ULWOS_CURRENT_TASK] = os_data.temp_HX
109  002d be08     [3]             LDX   os_data:8
```

```
215  00b4          LB4:
216  00b4 b608     [3]             LDA   os_data:8
217  00b6 87       [2]             PSHA
218  00b7 48       [1]             LSLA
219  00b8 4f       [1]             CLRA
220  00b9 a200     [2]             SBC   #0
221  00bb be07     [3]             LDX   os_data:7
222  00bd 87       [2]             PSHA
223  00be 9f       [1]             TXA
224  00bf 87       [2]             PSHA
225  00c0 48       [1]             LSLA
226  00c1 4f       [1]             CLRA
227  00c2 a200     [2]             SBC   #0
228  00c4 87       [2]             PSHA
229  00c5 8a       [3]             PULH
230  00c6 88       [3]             PULX
231  00c7 afff     [2]             AIX   #-1
232  00c9 9ee602   [4]             LDA   2,SP
233  00cc 87       [2]             PSHA
234  00cd 9ee602   [4]             LDA   2,SP
235  00d0 87       [2]             PSHA
236  00d1 9f       [1]             TXA
237  00d2 8b       [2]             PSHH
238  00d3 88       [3]             PULX
239  00d4 cd0000   [6]             JSR   _ICMP
240  00d7 a702     [2]             AIS   #2
241  00d9 81       [6]             RTS
```

Operandos e Operações

```
if ((char)CURRENT_TASK<(char)(IDLE_TASK))
```

```
 84     52:    if (ULWOS_CURRENT_TASK<0) ULWOS_CURRENT_TASK=0; else
 85   000c 3d08     [4]             TST   os_data:8
 86   000e 9004     [3]             BGE   L14 ;abs = 0014
 87   0010 3f08     [5]             CLR   os_data:8
 88   0012 202a     [3]             BRA   L3E ;abs = 003e
 89   0014          L14:
 90     53:    {
 91     54:      if ((char)ULWOS_CURRENT_TASK<(char)(ULWOS_IDLE_TASK))
 92   0014 b607     [3]             LDA   os_data:7
 93   0016 4a       [1]             DECA
 94   0017 b108     [3]             CMP   os_data:8
 95   0019 2320     [3]             BLS   L3B ;abs = 003b
 96     55:      {
 97     56:        task_reg_SP[ULWOS_CURRENT_TASK] = os_data.temp_SP;
 98   001b be08     [3]             LDX   os_data:8
 99   001d 58       [1]             LSLX
100   001e 8c       [1]             CLRH
101   001f b600     [3]             LDA   os_data
102   0021 87       [2]             PSHA
103   0022 b601     [3]             LDA   os_data:1
104   0024 87       [2]             PSHA
```

Operandos e Operações

```
- PROCEDURES:
    task_switcher          2116    E5    229    4    .text
    tpm1ch1_isr            21FB     3      3    1    .text
    swi_isr                21FE     3      3    1    .text
    rtc_isr                2201     3      3    1    .text
    task_idle              2204     3      3    1    .text
    task_set_entry         2207    57     87    1    .text
    ulwos_task_create      225E    18     24    3    .text
    ulwos_task_delay       2276     F     15    8    .text
    ulwos_start            2285    10     16    1    .text
    tpm3ovf_isr            2295     B     11    1    .text
    read_key               22A0    5A     90    1    .text
    task_led_seq           22FA    35     53    1    .text
    mcu_init               232F    1A     26    1    .text
    main                   2349    1A     26    1    .text

- PROCEDURES:
    task_switcher          2116    C4    196    3    .text
    tpm1ch1_isr            21DA     3      3    1    .text
    swi_isr                21DD     3      3    1    .text
    rtc_isr                21E0     3      3    1    .text
    task_idle              21E3     3      3    1    .text
    task_set_entry         21E6    57     87    1    .text
    ulwos_task_create      223D    18     24    3    .text
    ulwos_task_delay       2255     F     15    8    .text
    ulwos_start            2264    10     16    1    .text
    tpm3ovf_isr            2274     B     11    1    .text
    read_key               227F    5A     90    1    .text
    task_led_seq           22D9    35     53    1    .text
    mcu_init               230E    1A     26    1    .text
    main                   2328    1A     26    1    .text
```

Referências

C:

- ISO/IEC 9899:TC3 – ISO C Standard
  http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf
- The Firmware Handbook

ARM:

- ARM Architecture Reference Manual
- ARM DDI 0405A-01 ARM v7-M Architecture Application Level Reference Manual
- ARM DDI 0337D Cortex-M3 Technical Reference Manual
- Tecnologia ARM: Microcontroladores de 32 bits
- AN34 – ARM – Writing Efficient C for ARM
- AN36 – ARM – Using C Global Data

Referências

Coldfire:

- Coldfire Family – Programmer's Reference Manual

HCS08:

- HCS08 Unleashed – Designer's Guide to the HCS08 Microcontrollers
- HCS08RMv1/D – HCS08 Family Reference Manual

MSP430:

- Microcontroladores MSP430: Teoria e Prática
- MSP430 IAR C/C++ Compiler Reference Guide

Referências

PIC:

- MPLAB C32 C Compiler User's Guide
- DS61113B – PIC32MX Family Reference Manual
- MD00249-2B-M4K-SUM – MIPS32 M4K Software User's Manual
- MD00076 IV-a – MIPS16e Extension to the MIPS32 Architecture

Obrigado !

fabio@sctec.com.br