

# TMS 320C3x

## Introdução

Processamento Digital de Sinal 1



## Tópicos

- | Diagrama de blocos do processador;
- | Arquitectura;
- | Constituição do CPU; registos e mapa de memória;
- | Interrupções;
- | *Instruction set*;
- | Programação em *assembly* e C;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

2

## A família TMS320C3x

- | A família TMS320C3x *floating-point*:
  - | TMS320C30; TMS320C31; TMS320C32;
- | Foi projectada para aplicações em:
  - | Audio;
  - | Comunicação de dados;
  - | Automação industrial;
  - | Controlo de sistemas;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

3

## TMS320C30

### Características 1/2

- | Arquitectura Super Harvard;
- | CMOS *floating point* DSP (32-bit);
- | Executa até 60 MFLOPS e 11 operações numa instrução;
- | XTAL=30 MHz;
- | Ciclo máquina 30 nS; ciclo instrução 60 nS;
- | Portas série bidireccionais;
- | DMA on-chip, 2 TIMER e 2 portas série;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

4

## TMS320C30

### Características 2/2



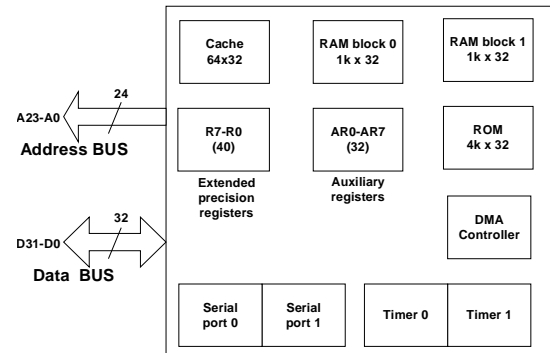
- | Capacidade de endereçamento de 16MB (24 bit) a palavras de dimensão 32 bit;
- | Address BUS a 24 bit; Data BUS a 32 bit;
- | 2 Blocos de RAM (1K, 32 bit);
- | 1 Bloco de ROM (4K, 32 bit);
- | Cache 64x32;
- | External BUS;

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

5

## TMS 320C30 - Diagrama simplificado



11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

6

## Elementos internos do CPU



- | BUS internos (CPU1/CPU2 e REG1/REG2);
- | 32 bit *barrel shifter*;
  
- | Multiplicador inteiro e *floating-point*;
- | ALU;
- | *Auxiliary register arithmetic units (ARAU)*;
- | *Primary register file* (28 registos);

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

7

## Multiplicador



- | Funciona com vírgula fixa e flutuante;
- | Inteiros a 24 bit e vírgula flutuante a 32 bit;
- | Multiplicação de inteiros a 24 bit resulta num inteiro a 32 bit;
- | Multiplicação de vírgula flutuante a 32 bit resulta em 40 bit;

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

8

## ALU

- | Operações num só ciclo sobre inteiros a 32 bit e vírgula flutuante a 40 bit;
- | Resultados da ALU são guardados a 32 bit (inteiros) e 40 bit (vírgula flutuante);
- | Permite multiplicações e adições/subtracções paralelas de quatro inteiros ou vírgula flutuante num só ciclo;
- | Paralelismo com o multiplicador;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

9

## ARAU

- | Duas *auxiliary register arithmetic units* (ARAU0 e ARAU1);
- | Geram dois endereços num ciclo;
- | Operam em paralelo com o multiplicador e a ALU;
- | Suportam 4 tipos de endereçamento: com deslocamentos; com *index registers* (IR0 e IR1); circular; *bit-reversed*;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

10

## Registos

- | R0...R7: Acumuladores ou de uso geral
  - | 40 bits (internamente), 32 bits (externamente)
  - | Float:
    - | b39-b32: expoente; b31: sinal; b30-b0: fracção
  - | Int:
    - | b31-b0: valor binário
- | AR0...AR7: Para endereçamento ou aritmética de inteiros (32 bits) pela ALU ou multiplicador;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

11

## Registos

```
DP      - 32 bit - Data Page Pointer  
        Os 8 LSBs são usados como apontador para a página  
        de 64K words de dados a aceder em end. directo.  
  
IR0, IR1 - 32 bit - Index Registers  
        Usados para calcular um endereço indexado (Cap 6)  
  
SP      - 32 bit - Active Stack Pointer  
        Contém o endereço do topo do stack. Aponta sempre o  
        último elemento pushed. Push faz pre-incremento de SP. Pop faz  
        pós-decremento. Usa apenas os 24 LSB.  
  
ST      - 32 bit - Status Register  
        Contém informação global sobre o estado do CPU. Ilustram-se  
        os bits mais relevantes:  
        31--- 8 7 6 5 4 3 2 1 0  
        ---- RM OVM - - - UF N Z V C  
  
RM - Repeat mode flag (=1: o PC está em block repeat)  
OVM - Overflow mode (=1: inteiros saturam)  
UF - Floating-point underflow condition flag  
N - Negative condition flag  
Z - Zero condition flag  
V - Overflow condition flag  
C - Carry flag
```

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

12

## Registos

IE	- 32 bit - CPU/DMA Interrupt Enable Register
	31---26-----16 10-----0 -----DMA-----CPU--- 1-enable interrupt; 0-disable interrupt;
IF	- 32 bit - CPU Interrupt Flag Register
	1-set interrupt; 0-not set interrupt; (activado quando ocorre uma interrupção; pode ser posicionado por software para causar interrupção)
IOP	- 32 bit - I/O Flag Register
	Controla as funções dos pinos XF0 e XF1, configurando como input ou output;
RC	- 32 bit - Repeat Counter Register
	Especifica o número de vezes que se repete um bloco de código;
RS	- 32 bit - Repeat Start-Address Register
	Endereço inicial do bloco de código;
RE	- 32 bit - Repeat End-Address Register
	Endereço final do bloco de código;
BK	- 32 bit - Block Size Register
	Dimensão do bloco de dados no endereçamento circular;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

13

## Interrupções

- I No C30 as interrupções são vectorizáveis e *level-triggered*;
- I 64 entradas de interrupção (4 externas);
- I As interrupções internas são geradas por Timer, Porta Série e DMA;
- I Condições de atendimento:
  - I O bit GIE do registo ST deve ser ZERO;
  - I O interrupt deve estar enabled, colocando UM no respectivo bit do IF;

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

14

## Interrupções: tabela

A base da tabela está no endereço 00h

Os endereços dos vectores de interrupção: 00 a 3Fh

00h	RESET	
01h	INT0,	entrada externa
02h	INT1,	entrada externa
03h	INT2,	entrada externa
04h	INT3,	entrada externa
05h	XINT0,	transmissão porta série 0
06h	RINT0,	recepção porta série 0
07h	XINT1,	transmissão porta série 1
08h	RINT1,	recepção porta série 1
09h	TINT0,	timer 0
0Ah	TINT1,	timer 1
.....		

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

15

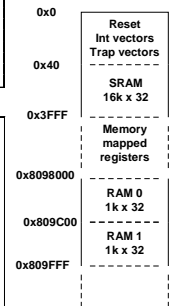
## EVM'30 - Mapa de Memória

```

/* SPECIFY SYSTEM MEMORY MAP FOR TI 'C30 EVM */
MEMORY
{
  VECT: org = 0x00000000 len = 0x00000040 /* INTs TRAP Vectors */
  SRAM: org = 0x00000040 len = 0x00003fc0 /* 64 Kword de SRAM */
  RAM0: org = 0x00809800 len = 0x00000400 /* 1024 Word block 0 */
  RAM1: org = 0x00809c00 len = 0x00000400 /* 1024 Word block 1 */
}

Vectors.asm
.sect ".vecs" ; interrupt and reset vectors
.ref _c_int00 ; compiler defined C initialization reset
.ref _c_int05 ; serial port transmit interrupt routine
.ref _c_int99 ; unexpected interrupt handler (just idles)

reset: .word _c_int00
int0: .word _c_int99
int1: .word _c_int99
int2: .word _c_int99
int3: .word _c_int99
xint0: .word _c_int05
rint0: .word _c_int99
xint1: .word _c_int99
rint1: .word _c_int99
tint0: .word _c_int99
tint1: .word _c_int99
dint: .word _c_int99
    
```



11/7/2002

Paulo Marques e Artur Ferreira Nov2002

16

## Programação

- | Utilizam-se as linguagens de programação *assembly* e C;
- | É possível a interligação de módulos nestas duas linguagens;
- | *Assembly* específico;
  - | Exemplo: <instrução> src,dst

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

17

## Instruções de exemplo 1/3

- | **Load/Store (src, dst)**
  - | LDI 10,R0
  - | LDI @pos,R0
  - | STI R0,\*AR0++
  - | LDP endereço
- | **Aritméticas e Lógicas**
  - | ADDI 1,R0
  - | ADDF 0.3,R1
  - | MPYI 2,R2
  - | MPYF 3,R4
  - | AND 0ffch,r1
  - | XOR IE,IE
- | **Controlo de fluxo**
  - | DBcnd ARn,loop
  - | CMPF G,R (dst-src)
  - | CMPI G,R (dst-src)
  - | BR endereço
  - | BRD endereço
  - | Bcnd endereço
  - | CALL endereço
  - | RETS cnd
  - | RETS endereço

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

18

## Instruções de exemplo 2/3

- | **Conversão de tipos**
  - FIX G,R2; dst=int(src)
  - FLOAT G,R; R=float(G)
  - RND G,R; R=round(G)
- | **Repetição blocos**
  - LDI N-1,RC
  - RPTB fim\_bloco
  - ...
  - fim\_bloco:
  - ultima instrução
  - ...
- | **Repetição 1 instrução**
  - RPTS N-1
  - instrução a repetir

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

19

## Instruções de exemplo 3/3

- | **Aritmética 3 operandos**
  - ADDI3 src2,src1,dst dst=src2+src1
  - ADDF3 src2,src1,dst (FP)dst=src2+src1
  - MPYF3 src2,src1,dst (FP)dst=src2\*src1
- | **Paralelo & 3 operandos**
  - MPYF3 src2,src1,dst1 dst1=src1\*src2
  - || STF src3,dst2 || dst2=src3
  
  - MPYF3 srcA,srcB,dst1 dst1=srcA\*srcB
  - || ADDF3 srcC,srcD,dst2 || dst2=srcC+srcD
  
  - MPYF3 srcA,srcB,dst1 dst1=srcA\*srcB
  - || SUBF3 srcC,srcD,dst2 || dst2=srcD-srcC

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

20

## Condition codes

### --- Unsigned Compares ---

LO - Lower than  
 HI - Higher than  
 EQ - Equal

LS - Lower or Same  
 HS - Higher or Same  
 NE - Not Equal

### --- Signed Compares ---

LT - Less Than  
 GT - Greater Than  
 NE - Not Equal

LE - Less than or Equal  
 EQ - Equal To

### --- Compare to Zero ---

Z - Zero  
 P - Positive

NZ - Not Zero  
 N - Negative

NN - Non Negative

### --- Compare to Condition Flags ---

NN - Non Negative  
 NZ - Not Zero  
 NV - Not Overflow  
 NUF - No underflow FP  
 NC - No carry  
 ZUF - Zero or FP underflow

N - Negative  
 Z - Zero  
 V - Overflow  
 UF - Underflow FP  
 C - Carry

### Exemplo:

```
wait_transmit_0:
    xor    IF,IF
wloop:
    tstb  10h,IF
    bz    wloop
    rets
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

21

## Tipos de endereçamento

### I Directo: @exp

Se DP=81h

```
ADDI @0bcdeh,r7
r7=r7+(81bcdeh)
```

### I Indirecto: \*ARn

\*+ARn(displ)

\*-ARn(displ)

\*++ARn(displ)

\*ARn++(displ)

\*+ARn(IRn)

\*ARn++(IRn)

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

22

## Endereçamento circular

### Registo BK-Block size register

O buffer circular tem que começar num "k bit boundary"

Os k LSB do endereço inicial são 0

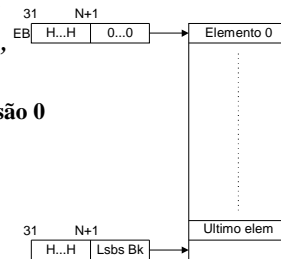
Ex: BK=6

AR0=0

\*AR0++(5)% ;AR0=5

\*AR0++(2)% ;AR0=1

\*AR0--% ;AR0=0



11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

23

## Endereçamento circular: convolução

```
LDI N,BK
LDI COEFS,AR0
LDI TAPS,AR1
```

### NovAmostra:

```
LDF IN_FIR,R3
LDF 0,R0
LDF 0,R2
```

Multiply and accumulate

### Filtro:

```
RPTS N-1
MPYF *AR0++%,*AR1++%,R0
|| ADDF3 R0,R2,R2
ADDF R0,R2
STF R2,OUT_FIR
```



11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

24

## Assembler: directivas de secções

	<code>.bss</code>	simbolo, nwords
<b>simbolo</b>	<code>.usect</code>	“nome secção”, nwords
	<code>.data</code>	
	<code>.text</code>	
	<code>.sect</code>	“nome secção”
<code>.bss</code>	espaço para variáveis não inicializadas.	
<code>.usect</code>	reserva espaço para variáveis na secção referida mas não inicializada.	
<code>.data</code>	indica ao assembler para começar a codificar na zona de dados. Serve usualmente para tabelas, variáveis iniciadas, etc.	
<code>.text</code>	instrução para começar a codificar na zona de texto (código).	
<code>.sect</code>	indica para começar a codificar na secção escolhida.	

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

25

## Assembler: outras directivas

### Directivas de inicialização de memória

	<code>.word</code>	valor1, valor2, ...
	<code>.float</code>	valor1, valor2, ...
<b>simbolo</b>	<code>.set</code>	valor
	<code>.space</code>	nwords
<code>.word</code>	coloca valores em posições de memória consecutivas	
<code>.float</code>	coloca floats em posições de memória consecutivas	
<code>.set</code>	atribui o valor ao símbolo	
<code>.space</code>	reserva espaço para nwords e inicia-o com zeros	

### Outras directivas

	<code>.global</code>	simbolo1, simbolo2
	<code>.end</code>	
<code>.global</code>	torna o símbolo visível a módulos no exterior	
<code>.end</code>	termina o assembler	

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

26

## Exemplo: echo.asm 1/2

```

.global main,wait_int,receive0
.global sysinit,p0_addr,stack_addr,PARMS

.text                ; begin of code
main:
    ldi    @stack_addr, sp    ; load the address into stack pointer
    call  sysinit            ; initialize the Timer, Serial Port, AIC

    ldp  VARS                ; set DP to our data
loop:
    ldi  @rx_ready,r0        ; R0 = @rx_ready
    bz  loop                ; while R0==0 goto loop
    ldi  @rx_data,r0
    sti  r0,@tx_data        ; @tx_data = @rx_data
    ldi  0,r0
    sti  r0,@rx_ready       ; @rx_ready=0
    br  loop                ; goto loop
    
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

27

## Exemplo: echo.asm 2/2

```

receive0:
    push dp
    push ar0
    push r0
    push r1

    ldp  VARS
    ldi  @tx_data,r1
    and  0xffch,r1          ; 2 lsb=0
    ldp  PARMS
    ldi  @p0_addr,ar0      ; get port address
    ldi  *+ar0(12),r0      ; read input
    sti  r1,*+ar0(8)      ; envia output

    ldp  VARS
    sti  r0,@rx_data      ; rx_data=input
    ldi  1,r0
    sti  r0,@rx_ready     ; rx_ready=1

    pop  r1
    pop  r0
    pop  ar0
    pop  dp
    reti

    .data
VARS:
    rx_data .word 0
    tx_data .word 0
    rx_ready.word 0
    
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

28

## Iniciação do sistema

```
sysinit:
    xor    ie,ie           ;disable all interrupts
    xor    if,if           ;clear all interrupts
    ldp    PARSMS           ;load data page pointer to parameters
    ldi    CACHE,st        ;load the status register
    ldi    WAIT0,r0        ;get i/o ready setup
    ldi    @mcntlr0,ar0    ;get memcntl register address
    sti    r0,*ar0         ;set parallel ready
    ldi    WAIT1,r0        ;get i/o ready
    ldi    @mcntlr1,ar0    ;get memcntl reg address
    sti    r0,*ar0         ;set i/o ready
    call   aicreset        ;routine to reset the AIC
    call   aicinit         ;routine to set up the AIC
    or     ENBL_INT0,ie    ;enable interrupt 0
    or     ENBL_GIE,st     ;enable global interrupt
    rets
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

29

## Reset do AIC

Analog Interface Controller

```
aicreset:
    ldi    2,iof           ;xf0 to output, set xf0 to 0
    ;get address of timer control register
    ldi    @t0_ctladdr,ar0
    ldi    1,r1            ;tclk0 will equal h1/2
    sti    r1,*+ar0(8)    ;set the period register to 1
    ldi    @t0_ctlinit,r1 ;get timer 0 setup value
    sti    r1,*ar0        ;set timer 0 to pulse mode

    ldi    @p0_addr,ar0   ;get address of serial port 0
    ldi    111h,r1
    sti    r1,*+ar0(2)    ; set transmit port control
    sti    r1,*+ar0(3)    ; set receive port control

    ;initialize port 0 global control
    ldi    @p0_global,r1
    sti    r1,*ar0

    ;set transmit data to 0
    xor    r1,r1
    sti    r1,*+ar0(8)

    ;wait for 50 timer out clocks
    rpts   99
    nop

    ;set xf0 to 1, !reset AIC
    ldi    6,iof
    rets
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

30

## Iniciação do AIC

```
aicinit:
    call   wait_transmit_0
    ldi    3,r1
    sti    r1,*+ar0(8)
    call   wait_transmit_0
    ; counter A setup value
    ldi    1a34h,r1
    sti    r1,*+ar0(8)
    ldi    *+ar0(12),r1
    call   wait_transmit_0
    ldi    3,r1
    sti    r1,*+ar0(8)
    call   wait_transmit_0
    ; counter B setup value
    ldi    4892h,r1
    sti    r1,*+ar0(8)

    ldi    *+ar0(12),r1
    call   wait_transmit_0
    ldi    3,r1
    sti    r1,*+ar0(8)
    call   wait_transmit_0
    ; control register setup value
    ldi    2a7h,r1
    sti    r1,*+ar0(8)
    ldi    *+ar0(12),r1

    ;clear out all interrupt flags
    xor    if,if
    ;enable serial port 0
    or     @enbl_sp0_r,ie
    rets
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

31

## Exemplo: echo.cmd

```
/* OBJECT FILES TO BE LINKED INTO THE EXECUTABLE MODULE */
echo.obj
sysinit.obj
/* OPTIONS TO BE INCLUDED IN THE LINKING PROCESS */
-e main /* PRIMARY ENTRY POINT FOR THE OUTPUT MODULE */
-o echo.out /* NAMES THE EXECUTABLE OUTPUT MODULE */
-m echo.map
/* SPECIFY THE SYSTEM MEMORY MAP */
MEMORY
{
    INT_V : origin = 0x000000, length = 0x40
    SRAM : origin = 0x000040, length = 0x3FC0
    RAM0 : origin = 0x809800, length = 0x400
    RAM1 : origin = 0x809C00, length = 0x400
}
/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */
SECTIONS
{
    vectors: {} > INT_V
    comdata: {} > SRAM
    .text : {} > SRAM
    .data : {} > SRAM
    .bss : {} > RAM0
    stack : {} > RAM1
}
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

32

## Echo.c (poling) 1/3

```
void main(void)
{
    int inval, outval;
    float tmp, gain = 1.0/20;
    /* global interrupts disable */
    GI_DISABLE;
    /* clear interrupts pending */
    CL_INT_PEND;

    /* 0 wait states on primary bus */
    /* 0 wait states on expansion bus */
    BUS_ADDR->prim_gcontrol = WS_0;
    BUS_ADDR->exp_gcontrol = WS_0;
    CACHE_ON; /* cache on */

    /* set up Timer 0, Serial Port 0, */
    /* start AIC and set its parameters */
    init_aic();

    while (1) {
        WAIT_XRDY;

        inval = SERIAL_PORT_ADDR(
            SERIAL_PORT_ZERO)->r_data;

        tmp = (inval << 16 >> 18) * gain;
        outval = ((int) tmp) << 2;

        SERIAL_PORT_ADDR(SERIAL_PO
            RT_ZERO)->x_data = outval;
    }
}
```

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

33

## Echo.c (poling) 2/3

```
void init_aic(void)
{
    volatile int i;

    /* AIC COMMAND WORD ZERO */
    aic_command_0.command = 0;
    aic_command_0.ra = 13;
    /* ADJUST SAMPLING RATE TO 8 kHz */
    /* AND 3.6 kHz ANTIALIAS FILTER */
    aic_command_0.ta = 13;

    /* ON INPUT HIGHPASS FILTER */
    aic_command_3.highpass = ON;

    /* DISABLE AIC LOOPBACK */
    aic_command_3.loopback = OFF;

    /* DISABLE AUX INPUT */
    aic_command_3.aux = OFF;

    /* DEFAULT AIC COMMAND WORD 1 */
    aic_command_1.command = 1;
    aic_command_1.ra_prime = 1;
    aic_command_1.ta_prime = 1;
    aic_command_1.d_f = 0;

    /* ENABLE SYNC. A/D AND D/A */
    aic_command_3.sync = ON;

    /* SET FOR 1.5 v p-p INPUT */
    aic_command_3.gain = LINE_V;

    /* DEFAULT AIC COMMAND WORD 2 */
    aic_command_2.command = 2;
    aic_command_2.rb = 36;
    aic_command_2.tb = 36;

    /* SIN x/x CORRECTION FILTER */
    aic_command_3.sinx = ON;
}
```

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

34

## Echo.c (poling) 3/3

```
/* CONFIGURE TIMER 0 TO ACT AS AIC MCLK */
TIMER_ADDR(TIMER_ZERO)->period = 1;
TIMER_ADDR(TIMER_ZERO)->gcontrol = FUNC | HLD_ | GO | CLKSRC;

RESET_AIC;
for(i = 0; i < 50; i++); /* KEEP RESET LOW FOR SOME PERIOD OF TIME */

/* CONFIGURE SERIAL PORT 0 */
SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->gcontrol = 0;
SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->s_x_control = CLKXFUNC | DXFUNC | FSXFUNC;
SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->s_r_control = CLKXFUNC | DXFUNC | FSXFUNC;
SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->gcontrol = XVAREN | RVAREN | FSXP | FSRP |
    XLEN_16 | RLEN_16 | XINT | RINT |
    RRESET | XRESET;
/* clear serial transmit data without pooling */
SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->x_data = 0;

UN_RESET_AIC;

WRITE_COMMAND(*(int *) &aic_command_0);
WRITE_COMMAND(*(int *) &aic_command_3);
}
```

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

35

## Echo.c (interrupção) 1/2

```
void main(void)
{
    GI_DISABLE; /* global interrupts disable */
    CL_INT_PEND; /* clear interrupts pending */
    SET_VECTOR(0, c_int00); /* reset handler */
    SET_VECTOR(5, AIC_InitInterrupt); /* handler to AIC initialization */
    init_aic(); /* Set up Timer 0, initialize Serial Port 0 */
    /* start AIC and set its parameters */

    XINT0_ENABLE;
    GI_ENABLE;

    while (1) {
        /* Process */
    }
}
```

11/7/2002

Paulo Marques e Artur Ferreira Nov2002

36

## Echo.c (interrupção) 2/2

```
#pragma INTERRUPT(AIC_InitInterrupt);
void AIC_InitInterrupt(void)
{
    AIC_sample = (int)SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->r_data << 16 >> 18;
    SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->x_data = AIC_init_seq[AIC_init_cnt++];
    if (AIC_init_cnt == AIC_init_len)
        SET_VECTOR(5, TxRxInterrupt);          /* change vector */
}

#pragma INTERRUPT(TxRxInterrupt);
void TxRxInterrupt(void)
{
    AIC_sample = (int)SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->r_data << 16 >> 18;
    /* process sample */
    SERIAL_PORT_ADDR(SERIAL_PORT_ZERO)->x_data = AIC_sample << 2;
}
```

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

37

## Codificação simultânea em C e assembly

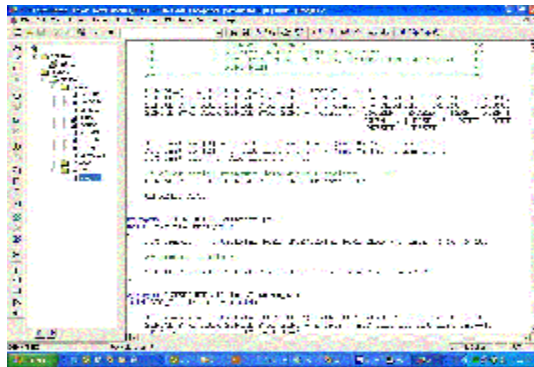
- I Retorno de valores em **R0**;
- I Variáveis a aceder em C são declaradas **.global** em *assembly*. Em C são **extern**;
- I Todos os identificadores a aceder em C têm o prefixo **\_**;
- I **DP** aponta a secção **.bss**

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

38

## Ambiente de desenvolvimento: Code Composer



11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

39

## Links

- I Texas Instruments: <http://www.ti.com>
- I Texas Instruments DSP Developers' Village: <http://www.dspvillage.com>
- I Manuais do processador, assembler, compilador e programas exemplo (eco): [http://www.deetc.isel.ipl.pt/analisedesainai/pdsl/pds1\\_bibliografia.html](http://www.deetc.isel.ipl.pt/analisedesainai/pdsl/pds1_bibliografia.html)

11/7/2002

Paulo Marques e Artur Ferreira Nov/2002

40