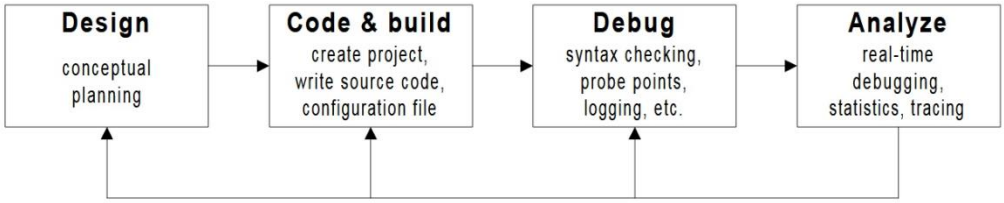
Módulo 0. Introdução ao CCS

OBJETIVO:
Efetivar o primeiro contato com o Code Composer Studio (CCS). Escrever, montar e executar/depurar os primeiros programas em assembly.

DADOS:
Durante o semestre vamos trabalhar muito com o CCS. Ele é um ambiente integrado que oferece facilidades para se trabalhar com as diferentes fases do desenvolvimento de um sistema embarcado:

1. Projeto conceitual (Design),
2. Preparação do Código (Code & Build),
3. Depuração (Debug)
4. Análise (Analyse)



Na etapa de projeto conceitual, o projetista irá decidir qual é a melhor forma de resolver um dado problema. Ele deve escolher os pinos e os periféricos do microcontrolador que irá usar. É tarefa do projetista definir também, com base nas especificações de projeto, as configurações de cada periférico.

A segunda etapa consiste numa tradução das especificações de projeto em código (Assembly ou C). É a partir da segunda etapa que usamos o Code Composer Studio. A IDE fornece ao programador ferramentas para tornar o processo de escrita de um código mais fácil. O CCS possui recursos de auto completar, interpretador de erros do compilador e sugestões para melhorar a performance do seu código com base no microcontrolador que o projetista escolher.

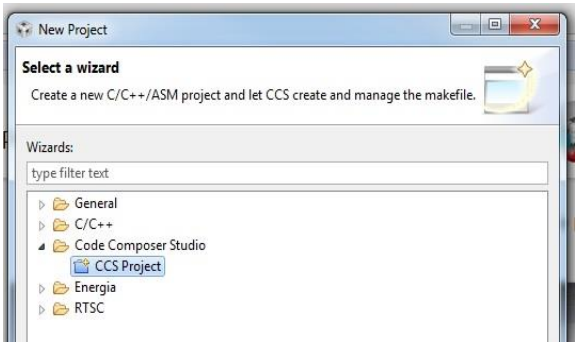
A terceira etapa, também conhecida como depuração ou debug, é fundamental para verificar o correto funcionamento do código. O CCS se conecta com a placa de hardware e permite ao projetista visualizar o conteúdo da memória juntamente com a execução passo-a-passo do software desenvolvido.

Por fim, na etapa de análise, o projetista pode ajustar parâmetros de compilação para melhorar características específicas do sistema dependendo dos requisitos de projeto. Pode compilar com otimizações para privilegiar tamanho do binário gerado ou pode privilegiar a performance de execução ou ainda diminuir o consumo de energia. Algumas launchPads (FR5994 e FR2355) possuem módulos para verificar o consumo energético da sua aplicação embarcada durante a execução do código.

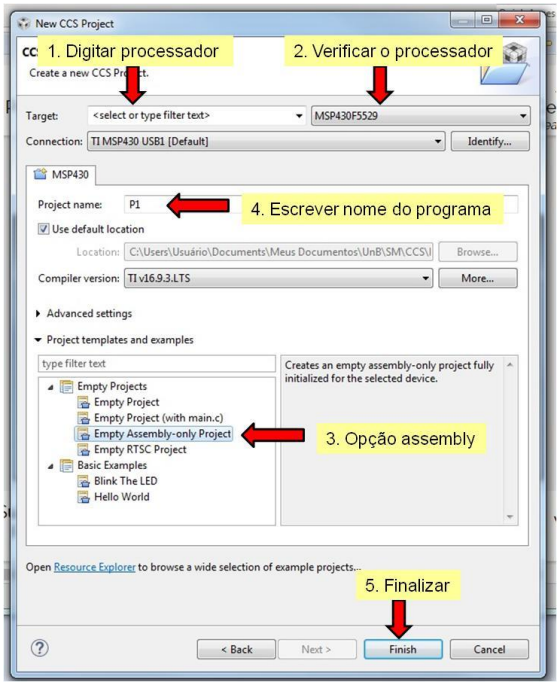
Para criar um projeto, siga as seguintes instruções:

1) Clique nas opções: "File", "New..." e "Project".

2) Na janela "New Project". Clique na opção "CCS Project" dentro da pasta “Code Composer Studio” e clique em "Next".



3) Na janela "New CCS Project". Verifique o nome do processador "MSP430F5529". Caso não esteja correto, digite o nome correto à esquerda e selecione o microcontrolador à direita. Selecione a opção "Empty Assembly only Project", defina um nome para o programa (sugestões: “ensaio1” ou “P1”) e clique em finalizar.



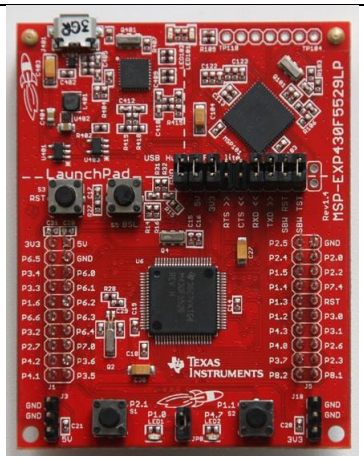
Após essas etapas, deverá surgir a tela de edição com o "esqueleto" do programa mostrado abaixo. Seu código deverá ser digitado logo abaixo do campo "**Main loop here**". Nas próximas aulas comentaremos o porquê das demais linhas de programa que aparecem neste arquivo.

```
;-------------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.

;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer


;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
                                            Seu código
                                            entra aqui.
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
```

Nosso objeto de estudo será o microcontrolador MSP430 (F5529). Seu uso é facilitado com o emprego da placa de desenvolvimento **MSP-EXP430F5529LP**, mostrada ao lado, que deve ser conectada ao computador via cabo USB.

O microcontrolador MSP430F5529 presente nesta placa disponibiliza diversos registradores, denominados R0, R1, ..., R15. Eles são usados para armazenar dados e realizar operações e podem trabalhar com 8 ou 16 bits.

As operações sobre esses registradores são indicadas por meio de instruções assembly. Cada instrução indica os registradores envolvidos e a operação a ser realizada.



Atalhos interessantes do Code Composer:

CTRL + S	Salva o programa
F11	Monta o programa e ativa o ambiente de debug
F5	Executa passo a passo (Step Into);
F6	Executa passo a passo, mas não “entra” nas sub-rotinas (Step Over);
F8	Executa (roda) o programa em modo contínuo.
ALT + F8	Pausa execução
CTRL + SHIFT + R	Soft Reset
CTRL + F2	Termina o debug e volta para o editor
CTRL + SHIFT + B	Insere ou remove um Ponto de Quebra (Break Point)

PEDIDOS:

Programa Ensaio 1:
Escreva, monte e execute com o debug o programa PE1, listado abaixo, que inicializa o conteúdo dos registradores R5 e R6 e depois os soma, guardando o resultado em R6. Note que ao lado de cada mnemônico foi colocado o modificador ".B". Ele indica que a instrução opera em 8 bits.

Utilizamos um montador para gerar o código de máquina a partir de um código assembly. Para montar o seu programa, clique no botão do *martelo* (build)  . Se houver erros, eles serão mostrados na aba de erros. Para entrar em modo de depuração clique no botão do *inseto* (de**bug**)  . Se estiver se sentindo com sorte, (se achar que o código não tem erro) pode encadear as duas operações apertando F11.

Usando F5 (atalho para "step into" ), execute o programa passo a passo. Você pôde ver o cursor se movendo a cada instrução, mas não conseguiu examinar os registradores.

Se a aba "Registers" não estiver visível, na Barra de Ferramentas, selecione as opções "View" → "Registers". Na nova janela, clique no pequeno triângulo à esquerda da linha "Core Registers" para expandir o conteúdo dos registradores. Clicando com o botão da direita sobre o registrador, você pode usar a opção "Number Format" para mudar a apresentação do número. A opção hexadecimal é a padrão.

```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
;
PE1:  MOV.B       #3,R5       ;Colocar o número 3 em R5
      MOV.B       #4,R6       ;Colocar o número 4 em R6
      ADD.B       R5,R6       ;Fazer a operação R6 = R5 + R6
      JMP         $           ;Travar execução num laço infinito
      NOP                     ;Nenhuma operação
```

É preciso comentar alguns pontos importantes sobre um programa assembly. Note que ele é "arrumado" em 4 colunas verticais. Ao escrever seus programas, use a tabulação para separar essas diversas colunas.

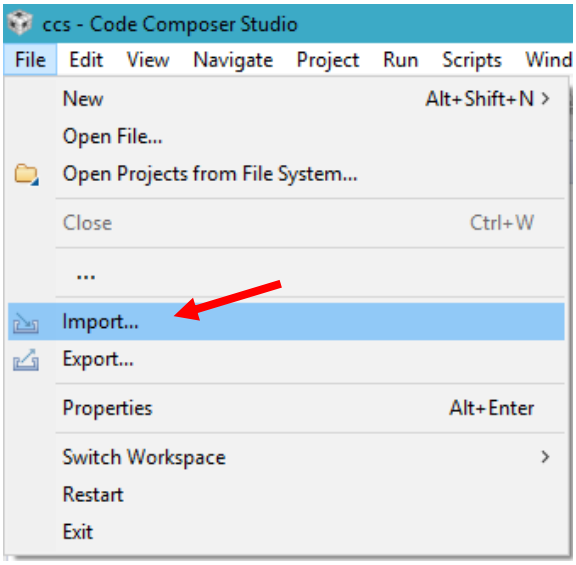
Coluna 1: mais à esquerda, reservada para os rótulos (labels). No programa acima, "PE1" é um label que faz referência ao endereço da instrução **`MOV.B  #3,R5`**. Sempre coloque seus labels nessa primeira coluna. Nunca coloque uma instrução nesta posição, pois ela será interpretada como label.

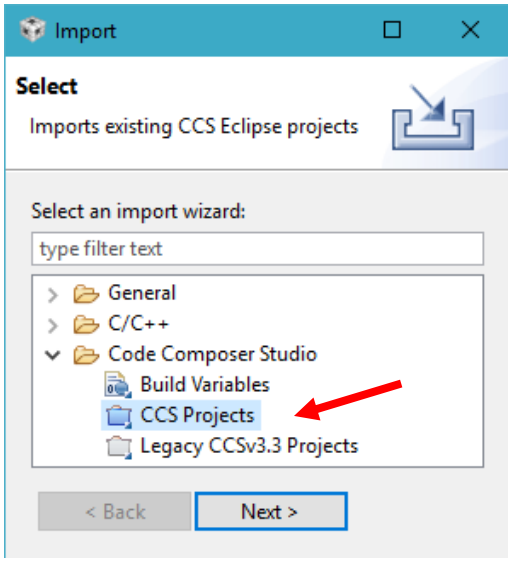
Coluna 2: onde são colocados os mnemônicos das instruções (MOV, ADD, JMP, etc).

Coluna 3: onde são colocados os operandos das instruções (#3,R5; R5,R6, etc). Note que a atribuição é da esquerda para a direita.

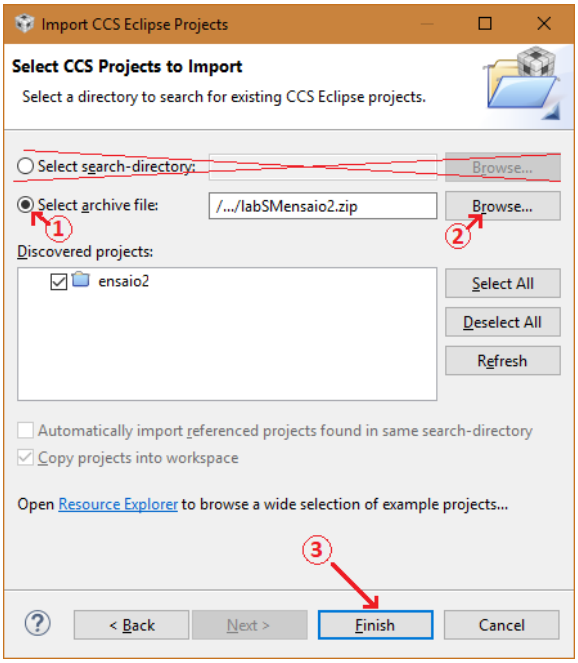
Coluna 4: destinada aos comentários. Tudo que estiver após o ponto e vírgula (;) é ignorado pelo montador assembly (assembler). Note que o comentário pode iniciar na primeira coluna.

Programa Ensaio 2:
Neste ensaio, iremos aprender a importar um projeto existente. Clique em “File” -> “Import” e selecione a opção “CCS Projects” dentro da pasta “Code Composer Studio”.





Em seguida, selecione o arquivo zip contendo o projeto clicando em “Browse” do item “**Select archive file**”. Você irá encontrar o arquivo labSMensaio2.zip na pasta “Downloads”.
Atenção: não usar a opção “Select search-directory”.



Monte e execute com o debug o programa PE2 que você importou. Este programa inicializa o conteúdo dos registradores R5 e R6 e depois os soma, guardando o resultado em R6. Note que os números estão em hexadecimal. A ausência do modificador ".B" indica que as operações são realizadas em 16 bits.

Programa Ensaio 3:
Crie um novo projeto para o ensaio 3. Escreva, monte e execute com o debug o programa PE3, que é idêntico ao anterior, porém R5 é carregado com o número 0xFFFF. Execute o programa **passo-a-passo**, usando a tecla **F5** (step into) ou **F6** (step over). O que você notou de interessante ao executar o esse programa?

```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
;
PE3:        MOV         #0xFFFF,R5        ;Colocar o número 0xFFF em R5
            MOV         #0x4321,R6        ;Colocar o número 0x4321 em R6
            ADD         R5,R6             ;Fazer a operação R6 = R5 + R6
            JMP         $                 ;Travar execução num laço infinito
            NOP                           ;Nenhuma operação
```

Percebeu que o resultado em R6 foi 0x4320? Isso ocorre pois 0xFFFF é a representação de -1 em complemento a 2. Note que no ambiente de debug você sempre vê a representação 0xFFFF. Você pode substituir a instrução **`MOV #0xFFFF,R5`** pela instrução **`MOV #-1,R5`** que o resultado será o mesmo.

Programa Ensaio 4:
Escreva, monte e execute com o debug o programa PE4, listado abaixo. Note que o programa usa a subrotina "SUBROT" que soma 1 ao R5, duas vezes. O número de vezes em que a subrotina é chamada é ditado pelo valor em R6. Isto significa que R6 é o contador do laço (LOOP).

```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
PE4:            CLR         R5          ;Zerar R5
                MOV         #4,R6       ;Colocar o número 4 em R6

LOOP:           CALL        #SUBROT     ;Chamar subrotina "SUBROT"
                DEC         R6          ;Decrementar R6
                JNZ         LOOP        ;Se diferente de zero, ir para LOOP
                NOP                     ;Nenhuma operação
                JMP         $           ;Travar execução num laço infinito
                NOP                     ;Nenhuma operação

SUBROT:         ADD         #1,R5       ;Somar 1 em R5
                ADD         #1,R5       ;Somar 1 em R5
                RET                     ;Retornar
```

1) Rode o programa até o final, usando F5.
2) Faça o Soft Reset (CTRL + SHIFT + R) e rode o programa até o final usando F6.
Você notou alguma diferença?

Vamos agora introduzir o conceito de **ponto de quebra** (**break point**). Quando a execução atinge um ponto de quebra, ela é interrompida e o controle é devolvido para o usuário. Coloque o cursor sobre a instrução **`NOP`**, logo antes do **`JMP $`** e ative um ponto de quebra neste local (CTRL + SHIFT + B). Uma forma alternativa para ativar o ponto de quebra é dar dois cliques no número que aparece à esquerda da instrução. O ponto de quebra é útil para rotinas longas ou demoradas.

3) Faça o Soft Reset (CTRL + SHIFT + R) e rode o programa com F8. Note que o programa é interrompido quando atinge este ponto.

Programa Ensaio 5:
Escreva, monte e execute com o debug o programa PE5, listado abaixo. Note que o programa usa a instrução **`RLA R5`** que desloca o conteúdo de R5 uma vez para a esquerda. A quantidade de rotações é dada por R6.

```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
PE5:              MOV         #1,R5       ;Colocar 1 em R5
                  MOV         #4,R6       ;Colocar o número 4 em R6
LOOP:             RLA         R5          ;Deslocar 1 bit para a esquerda
                  DEC         R6          ;Decrementar R6
                  JNZ         LOOP        ;Se diferente de zero, ir para LOOP
                  NOP                     ;Nenhuma operação
                  JMP         $           ;Travar execução num laço infinito
                  NOP                     ;Nenhuma operação
```

O que aconteceu com o conteúdo de R5 a cada laço de repetição? Você notou alguma coisa interessante?

Ensaio 6 : Exporte o trabalho realizado num arquivo zip:

Ao final de cada módulo, será exigida a entrega de um programa na forma de um arquivo zip contendo o seu projeto CCS. Para exportar o seu trabalho, clique em "File" -> "Export" e selecione a opção **"Archive File"** dentro da pasta "General". Você pode selecionar mais de um projeto para exportar se quiser.